Opinnäytetyö (AMK)

Tietotekniikka

Sulautetut ohjelmistot

2016

Tero Hyvönen

MIDI-STUDIOSOVELLUS

– Teoria, suunnittelu ja ohjelmointi

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Sulautetut ohjelmistot

2016 | 64

Ohjaaja: TkL Jari-Pekka Paalassalo

Tero Hyvönen

MIDI-STUDIOSOVELLUS

- Teoria, suunnittelu ja ohjelmointi

Tämän opinnäytetyön tarkoituksena oli tutkia pienen studion syntetisaattorien ohjauksessa käytettävää tekniikkaa. Tämä opinnäytetyö toimii perusoppaana syntetisaattorin MIDI-ohjaukseen käytettävän sovelluksen suunnitteluun ja ohjelmointiin.

Teoriaosuus tutkii MIDI-standardia ja Windowsin multimediakirjastoa ja kerää informaatiota Unity-pelimoottorin C# skriptien ohjelmointia varten.

Ohjelmisto-osuudessa esitellään MIDI-viestejä vastaanottavan ja lähettävän sovelluksen kehittäminen.

ASIASANAT:

MIDI, C#, platform invoke, Unity 3D

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology | Embedded Software

2016 | 64

Instructor: M.Sc Jari-Pekka Paalassalo, Lic.Sc (Tech.)

Tero Hyvönen

MIDI STUDIO APPLICATION

- Theory, design and programming

The purpose of this thesis is to study technology needed to control synthesizers in a small studio. This thesis works as a basic programming and design guide for developing an application to control synthesizers via MIDI.

Theory partition studies MIDI standard and Windows multimedia libraries and collects information needed for programming C# scripts in Unity game engine.

Software partition introduces the development of the application that can send and receive MIDI messages.

KEYWORDS:

MIDI, C#, platform invoke, Unity 3D

SISÄLTÖ

LIITTEET

KAAVAT

KUVAT

KUVIOT

TAULUKOT

KÄYTETYT LYHENTEET JA SANASTO

API	Application Programming Interface. Ohjelmointirajapinta on kokoelma rutiineja, joita kutsumalla sovellus voi käskeä API:a suorittamaan halutun toiminnon ilman, että sovelluksen ohjelmoijan pitää tietää, miten tehtävä suoritetaan.
MIDI	Musical Instrument Digital Interface. Standardi elektronisten instrumenttien ja studiolaitteiden ohjaus- ja synkronointimenetelmä.
P/Invoke	Platform invoke on menetelmä, jossa frameworkin virtuaalikoneessa ajettavasta ns. manageroidusta koodista käsin kutsutaan frameworkin ulkopuolista manageroimattomassa koodissa määriteltyä funktiota.
Sävelaskeltransponointi	Soitettavan nuotin, soinnun tai sävelkuvion perussäveleen lisätään määrätty määrä sävelaskelia, jolloin soitettava sävelkuvio soitetaan eri sävellajissa siten, että sävelten keskinäinen harmonia säilyy ennallaan sävelkorkeuden muuttuessa.
Tempo	Musiikkitermi esityksen soittonopeudelle. Nopeuden yksikkönä käytetään minuutissa soitettavien neljäsosaiskujen määrää – bpm (beats per minute).
VST	Virtuaalinen studioteknologia. Instrumenttien tuottamaa ääntä mallinnetaan ohjelmallisesti matemaattisilla malleilla, jotka kuvaavat oskillaattorien, suodattimien ja modulaattoreiden ominaisuuksia. Äänityssovelluksissa mallinnetaan analogisten nauhureiden ja efektilaitteiden signaalinkäsittelyä.

1 JOHDANTO

Omien instrumenttien suunnittelu ja rakentaminen on mahdollista kaikissa musiikin genreissä. Elektronisessa musiikissa instrumentit voidaan toteuttaa kokonaan analogisilla komponenteilla, mikä edellyttää elektroniikan osaamista, tai instrumenttien ohjaukseen voidaan sulauttaa digitaalisia ohjauspiirejä, joiden hallinta vaatii jonkinlaisen ohjelmallisen sovelluksen. Kolmas tapa on luoda ohjelmallisesti kokonaan virtuaalinen instrumentti. Virtuaalisessa studioteknologiassa (VST) ohjelmasovellus jäljittelee joko olemassa olevan fyysisen instrumentin ominaisuuksia tai voidaan luoda kokonaan kustomoitu virtuaalinen kokoonpano. Virtuaalinen ratkaisu tulee kyseeseen vaihtoehtona, kun musiikin tuottamiseen käytetään instrumentteja, joiden hankkiminen ei ole mahdollista vähäisen laitekannan tai korkean hinnan takia.

Musical Instrument Digital Interface, eli MIDI, on vuonna 1983 julkaistu standardiohjauskieli syntetisaattorien hallintaan. Opinnäytetyössä tarkastellaan pienen kotistudion syntetisaattorien ohjaukseen käytettävän MIDI-ohjelmasovelluksen kehitystyötä.

Musiikin äänitystä, sekä nuottidatan tallennusta ja toistoa varten on saatavilla useita kaupallisia ja ilmaisia sovelluksia. Kaupallisissa sovelluksissa on yleensä paljon ominaisuuksia, joista suuri osa voi jäädä käyttämättä ja jotka vaikeuttavat ohjelmiston käytön opettelua. Tässä opinnäytetyössä suunnitellaan ja toteutetaan tietokonepohjainen sekvensserisovellus pienen kotistudion instrumenttien MIDI-ohjausta varten. Tässä työssä esitetyt asiat toimivat ohjelmointi- ja suunnitteluoppaana ohjelmointia aloittavalle harrastajalle sekä sellaiselle musiikin harrastajalle, joka haluaa tutustua elektronisen musiikin standardina käytettävän MIDI-ohjausteknologian perusteisiin. Työssä käytettävät työkaluohjelmistot on valittu saatavuuden (open source- ja ilmaisohjelmat) sekä helppokäyttöisyyden (laaja dokumentointi) perusteella.

Teoriaosassa esitellään joitakin tämän kaltaisen sovelluksen toteutuksessa mahdollisesti käytettäviä kehitystyökaluja. MIDI-standardia tarkastellaan sovelluksen vaatimusten näkökulmasta.

Ohjelmisto-osassa esitellään sovelluksen toiminta jaettuna loogisiksi osiksi. Osat kootaan Unity-pelimoottoriin, ja koko sovelluksen rakenne ja osien väliset yhteydet selvitetään havainnollisesti.

2 TAVOITTEET

Tämän opinnäytetyön tavoitteeksi asetettiin laatia sekvensserisovellus ohjaamaan kotistudion syntetisaattoria. Sovelluksen suunnittelun pohjana käytetystä standardista ja sovelluksen suunnittelusta laadittu dokumentaatio toimii ohjelmointioppaana, jossa esitellään sovelluksessa käytettyjä ohjelmointitekniikoita ja kehitystyökaluja. Tavoitteeksi asetettiin, että käyttökelpoisen sovelluksen kehittämisen lisäksi tässä työssä esitettyjä tekniikoita käyttämällä olisi mahdollista vähäisellä ohjelmointikokemuksella kehittää vastaavia ohjelmistoja.

Tässä työssä laaditaan sekvensserisovellus, jolla voidaan:

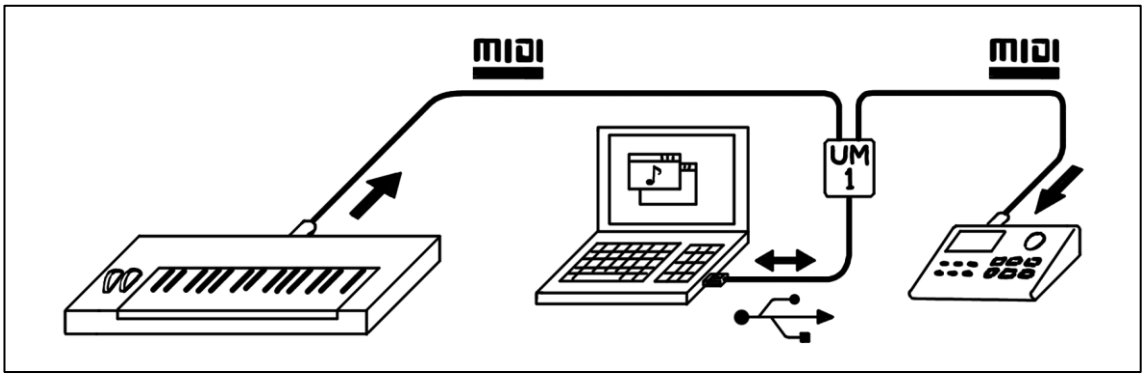
- Lukea ohjelmaan laitteelta saapuva MIDI-viesti
- Käsitellä viesti
- Luoda ajastettuja toimintoja
- Lähettää MIDI-viesti laitteelle

MIDI-ohjaus toteutettiin käyttämällä Windows käyttöjärjestelmän multimedia API:n winmm.dll-kirjastoa. Instrumenttien ohjauskielenä käytettävä MIDI-standardi esitellään perusteellisesti The MIDI Manufacturers Associationin ylläpitämässä dokumentissa ”The Complete MIDI 1.0 Detailed Specification”, ja sovelluksen toiminta perustuu tähän standardiin.

C#-ohjelmointiopaskirjoissa ei yleensä käsitellä manageroidun .NET frameworkin ja käyttöjärjestelmän manageroimattomien funktioiden välisessä liikenteessä käytettyjä platform invoke järjestelmäkutsuja. Platform invoke -kutsuilla on mahdollista tavoittaa toimintoja, joihin .NET framework ei anna suoraa pääsyä frameworkin sisältä, ja siksi menetelmän osaaminen on hyödyllistä tehokkaampien ohjelmien kehitystyössä.

3 TYÖKALUT

Studiosovellus kehitettiin ohjaamaan pienen kotistudion syntetisaattoreita ja rumpukonetta. Sovellus korvaa järjestelmässä fyysisen sekvensserilaitteen. Tietokoneella toimivan sekvensserin nuottidatan ohjelmointi voidaan suunnitella helppokäyttöisemmäksi kuin fyysisellä laitteella. Vaihtoehtoisesti tietokonesovelluksella voidaan jäljitellä tarkasti vaikeasti saatavilla olevan tai kalliin fyysisen sekvensserin toimintaa. Sovellus suunniteltiin järjestelmään, missä MIDI-koskettimistolta tulevaa dataa voidaan lukea tietokoneen sovelluksella ja samalla sovelluksella ohjataan analogista syntetisaattoria tai rumpukonetta. MIDI-laitteet kytkettiin tietokoneeseen Edirol UM-1EX USB – MIDI-sovittimen välityksellä. Kytkentä esitellään kuiviossa 1.



Kuvio 1. Järjestelmän laitteiden kytkentä.

Tässä luvussa esitellään joitakin mahdollisia työkaluja sovelluksen ohjelmointiin. Sopivan työkalun valinta riippuu käyttötarkoituksesta. Työkalun esittelyssä kerrotaan lyhyesti yleinen kuvaus työkalun ominaisuuksista.

3.1 C++-ympäristö

C++-kielelle toteutettu RtMIDI tarjoaa yhtenäiset ohjelmistoluokat, jotka muodostavat ohjelmointirajapinnan (API) Linuxille (ALSA & JACK), Macintosh OS X:lle (CoreMIDI & JACK) ja Windowsille (Multimedia Library) [1]. C++:n etuna voidaan pitää hyvää suoritusnopeutta ja RtMIDIn etuna erikseen ohjelman käännettävyyttä eri alustoille, sekä tarvetta vain yhdelle otsikkotiedostolle lähdekoodissa

Kääntäjälle ilmoitetaan käyttöjärjestelmäkohtainen API ja sama lähdekoodi kääntyy asianomaiselle käyttöjärjestelmälle toimivaksi. Tässä opinnäytetyössä kehitettiin sovellus Windows -ympäristöön, joten tarvetta portattavalle järjestelmälle ei ollut. RtMIDI esitellään vaihtoehtona .NETille. C++:aa käytetään ohjelmointikielenä esimerkiksi Unreal 4 -pelimoottorin skriptien ohjelmoinnissa. C++:lla voi myös laatia Windowsin ikkuna- ja konsolisovelluksia aivan kuten C#:lla.

RtMIDIn löytämiseen käytettiin hakusanoja: MIDI programming c++

RtMIDIn voi ladata osoitteesta:

https://www.music.mcgill.ca/~gary/rtMIDI/

3.2 .NET-ympäristö

MIDI.NET toimii samanlaisella periaatteella kuin RtMIDI, eli tarjoaa käyttäjälle valmiit luokat MIDIn käyttämiseen. MIDI.NETin kotisivulla kerrotaan MIDI.NETin antavan ohjelmistonkehittäjälle mahdollisuuden käyttää MIDIn suorituskykyä tarvitsematta tehdä järjestelmäkutsuja (P/Invoke) [2].

MIDI.NETin löytämiseen käytettiin hakusanoja: MIDI programming .net

MIDI.NETin voi ladata osoitteesta:

https://MIDInet.codeplex.com/

3.3 P/Invoke -järjestelmäkutsu

.NET frameworkissa ohjelmat suoritetaan ajoympäristössä (Common Language Runtime, CLR), joka tarjoa ohjelman suorittamista varten virtuaalikoneen, jossa ajettava ohjelmakoodi käännetään reaaliaikaisesti käyttöjärjestelmän käyttämään binäärimuotoon [3].

Frameworkissa ajettavaa ohjelmakoodia kutsutaan manageroiduksi koodiksi, sillä virtuaalikone pitää huolen sovelluksen muistinhallinnasta ja virheiden käsittelystä. Frameworkin ulkopuolella ajettava ohjelmakoodi on suoraan tekemisissä käyttöjärjestelmän ja laitteiston kanssa ilman virtuaalikoneen välitystä. Tätä koodia

kutsutaan manageroimattomaksi koodiksi ja P/Invoke -järjestelmäkutsu tarjoaa mahdollisuuden kutsua tässä ohjelmakoodissa määriteltyjä funktioita .NET frameworkin sisältä.

Edellä mainituissa kehittäjille tarjottavissa kirjastoissa toimintaperiaate on sama, eli valmiiksi ohjelmoidut luokat hoitavat liikennöinnin sovelluskehittäjän ja käyttöjärjestelmän multimediarajapinnan välillä. Tämän menetelmän etuna voidaan pitää ohjelman helppoa portattavuutta eri käyttöjärjestelmille.

Käytettävästä menetelmästä riippumatta ohjelman on jossakin vaiheessa pystyttävä lähettämään viesti järjestelmään asennetulle laiteajurille ja vastaanottamaan laiteajurin lähettämät viestit. Katsottiin, että viestiliikenteen toteuttaminen valmiiksi ohjelmoituja luokkia käyttäen ei toteuta tavoitteeksi asetettua oppimistavoitetta, joten tässä työssä päätettiin käyttää järjestelmäkutsuja, jotka keskustelevat suoraan multimediaohjelmointirajapinnan kanssa. Windowsissa tämän rajapinnan kanssa toimivat funktiot on koottu winmm.dll linkkikirjastoon.

Winmm.dll toimitetaan Windowsin mukana versiosta Windows 2000 alkaen ja Windows 7:ssa se löytyy kansiosta:

\Windows\System32

3.4 Unity 3D

Käyttöliittymää ja ohjelman muuta toimintaa varten tarvittiin kehitysympäristö. Yleinen tapa on kehittää Windows Forms -sovellus, jolla ohjelman käyttöliittymä voidaan toteuttaa standardeilla Windowsin graafisen käyttöliittymän elementeillä.

Tässä opinnäytetyössä käyttöliittymä ja sovelluksen muu rakenne päätettiin toteuttaa Unity-pelimoottoria käyttäen. Tällä tavalla käyttöliittymän vaatimat rakenteet voitiin toteuttaa Unityn omilla elementeillä siten, että niiden toteuttamiseen vaadittiin mahdollisimman vähän ohjelmakoodia.

Unity 3D on kaupallisessa käytössä maksullinen pelinkehitysympäristö. Tässä työssä käytettiin yksityis- ja opiskelukäyttöön lisensoitua maksutonta 64 bittistä Unity Personal -versiota. Kaikki Unityn toiminnalliset ominaisuudet ovat käytössä kaikissa lisenssiversioissa.

Sovelluksen kaikista osa-alueista ohjelmointiin testiversiona C#-konsolisovellukset Xamarin Studiolla. Testauksen jälkeen testiversioista tehtiin C# skripti Unityyn.

Opinnäytetyössä käytetty Unityn versio 5.3.4 ladattiin osoitteesta:

https://Unity 3D.com/get-unity/download

Unityn päivittyessä uudempaan versioon vanhemmat versiot voi ladata osoitteesta:

https://Unity 3D.com/get-unity/download/archive

3.5 Visual Studio

Visual Studio on Microsoftin kehittämä ohjelmistonkehitysympäristö, josta on harjoittelukäyttöön saatavilla ilmainen Express Edition -versio. Visual Studiossa on tuki useille ohjelmointikielille, kuten C, C++, C# ja Visual Basic. Visual Studio soveltuu erityisesti Windows-sovellusten ohjelmointiin ja Visual Studio Community 2015 asennettiin Unityn asennuksen yhteydessä. Skriptien ohjelmointiin voidaan käyttää myös Visual Studion muita versioita.

Visual Studion eri versiot voi ladata osoitteesta:

https://www.visualstudio.com/

3.6 MonoDevelop

MonoDevelop on Visual Studion kaltainen ohjelmistonkehitysympäristö, joka on tarkoitettu ohjelmistojen kehitykseen erityisesti Mono ja .NET ohjelmointiympäristöissä. MonoDevelop on avoimen lähdekoodin kehitysympäristö, ja sillä voidaan kirjoittaa ohjelmistoja Linux, OS X ja Windows käyttöjärjestelmille.

Tässä opinnäytetyössä esitellyt lähdekoodit on kirjoitettu C#:lla MonoDevelopilla, joka asennettiin Unityn yhteydessä automaattisesti. Skriptien prototyyppejä testattiin Unityn ulkopuolella MonoDevelopia käyttävässä Xamarin Studio 6.0 -kehitysympäristössä (build 5174).

MonoDevelop löytyy hakusanalla: monodevelop.

Opinnäytetyössä käytetty versio ladattiin osoitteesta: www.monodevelop.com

3.7 Hex editor Neo

MIDI-viestit koostuvat 8-bittisistä tavuista. Tavun sisältö voi olla kokonaisluku väliltä 0–255. Kun musiikkiesitys tallennetaan tiedostoksi tietokoneen kiintolevylle tai muulle tallennuslaitteelle, MIDI-viestit tallennetaan tiedostoon lukuarvoina. MIDI-tiedoston tunnistaa tiedostopäätteestä .mid, sekä tiedoston alussa olevista heksadesimaalitavuista: 4d 54 68 64 (ASCII-merkkijonona: MThd), jotka ilmaisevat MIDI-otsikkotiedon (header) aloituskohdan.

Koska MIDI-tiedosto on binääritiedosto, eivät tekstin muokkaukseen tarkoitetut ohjelmat pysty näyttämään tiedoston sisältöä oikeassa muodossa. MIDI-ja muiden binääritiedostojen lukemiseen ja muokkaamiseen tietokoneella tarvitaan ohjelma, joka pystyy näyttämään tiedoston sisältämän numeerisen tiedon. Tällaisia muokkausohjelmia kutsutaan yleisesti heksaeditoreiksi.

Tässä opinnäytetyössä ei käsitellä MIDI-tiedostojen sisältöä, mutta mikäli sovelluksen ominaisuusvalikoimaa kehitettäisiin edelleen lisäämällä nuottitiedon tallennus tai nuotinnoksen teko MIDI-tiedostoon tallennetusta informaatiosta, heksaeditori on tärkeä työkalu tiedoston sisällön tutkimiseen. Tarkoitukseen sopiva heksaeditori on esimerkiksi HHD Softwaren Free Hex Editor Neo.

Free Hex Editor Neo löytyy hakusanoilla: hex editor

Free Hex Editor Neo ladattiin osoitteesta: http://www.hhdsoftware.com/free-hex-editor

3.8 MIDI-standardi

Jotta voidaan olla varmoja siitä, että sovellus pystyy ohjaamaan kaikkien valmistajien syntetisaattoreita, on sovellus suunniteltava noudattamaan yhtenäistä standardia. The MIDI Association ylläpitää MIDI-standardin dokumentaatiota internetsivustollaan ja kaikki standardiin liittyvät dokumentit ovat ladattavissa sieltä. Tämän opinnäytetyön sovellusesimerkki perustuu MIDI 1.0 -standardiin. Vaikka ensimmäinen versio standardista julkaistiin jo vuonna 1983, on 1.0-standardi edelleen ajankohtainen, sillä

MIDIn vuodesta 2005 alkaen kehitteillä oleva HD-Protocol tulee edelleen olemaan alas yhteensopiva versioon 1.0 asti [4].

The Complete MIDI 1.0 Detailed Specification v.96.1 pdf-dokumentti ladattiin osoitteesta: https://www.MIDI.org/specifications

4 MIDI-STANDARDI

Syntetisaattori tarvitsee äänen tuottamiseen tietoa soitettavasta äänestä. Analogisen syntetisaattorin äänen tuottamisesta vastaa jännitteellä ohjattava oskillaattori (VCO), tai digitaalisesti ohjattu oskillaattori (DCO). Oskillaattorin ulkopuolelta tuotu analoginen ohjausjännite tai digitaalisesta signaalista oskillaattorin sisällä muunnettu ohjausjännite määrittää oskillaattorin värähtelytaajuuden. Analogisissa syntetisaattoreissa käytetään kahta standardia sävelkorkeuden määrittämiseen.

Voltti / oktaavi -järjestelmässä oskillaattorin sävelkorkeus kasvaa oktaavilla, eli signaalin taajuus kaksinkertaistuu ohjausjännitteen jokaista volttia kohden. Tämän järjestelmän etuna on kahden sävelen intervallin pysyminen samansuuruisena jänniteasteikolla oktaavista riippumatta. Voltti / oktaavi -järjestelmä vakiintui 1960-luvulla, ja sitä käyttävät valmistajat, kuten Roland, Moog, Sequential Circuits, Oberheim ja ARP.

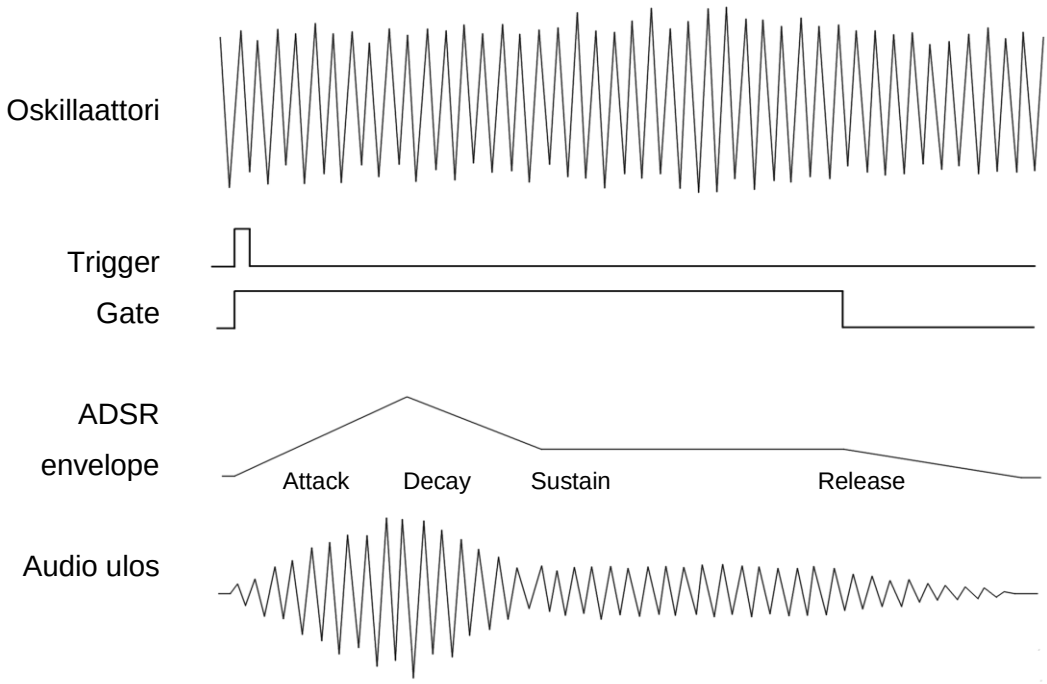
Hertsiä / voltti -järjestelmässä signaalin taajuus seuraa suoraan jännitettä. Oktaavin nousu sävelasteikolla vaatii kaksinkertaisen jännitteen. Hertsiä / voltti -järjestelmää käytetään useimmissa Korgin ja Yamahan valmistamissa analogisissa syntetisaattoreissa.

Kumpikin järjestelmä tarvitsee laitteen ohjausjännitteen muodostamiseksi. Yleensä ohjainlaitteena käytetään koskettimistoa, joka tuottaa oikeanlaisen ohjausjännitteen. Väärän järjestelmän ohjausjännite pystyy ohjaamaan oskillaattorin sävelkorkeutta, mutta oskillaattoria ei pysty virittämään soimaan oikeassa sävellajissa. Tämän lisäksi eri valmistajat käyttivät järjestelmän sisällä eri jännitetasoja, kuten esimerkiksi -5 V–+5 V, 0 V–+5 V tai 0 V–+10 V. Tämän lisäksi eri valmistajien käyttämät liittimet eivät olleet yhteensopivia, joten suuren syntetisaattorijärjestelmän rakentaminen eri valmistajien instrumenteista oli vaikeaa.

Sävelkorkeuden määrittämisen lisäksi syntetisaattorin on tiedettävä, milloin nuotti alkaa, ja milloin se päättyy. Oskillaattori tuottaa ohjausjännitteen mukaista signaalia jatkuvasti. Kosketinta painettaessa vahvistin avaa äänen. Äänen avaamista ja nuotin kestoa ohjaa porttisignaali (gate), joka ohjaa vahvistimen toimintaa ohjaavan verhokäyrägeneraattorin (envelope) tuottamaa ADSR-ohjausjännitettä. Porttisignaalin

lisäksi voidaan luoda myös liipaisinsignaali (trigger), jota varsinkin modulaarisissa syntetisaattoreissa voidaan käyttää eri toimintojen käynnistämiseen.

Verhokäyrägeneraattorin ohjaukseen käytettävien signaalien toiminta esitetään kuviossa 2.



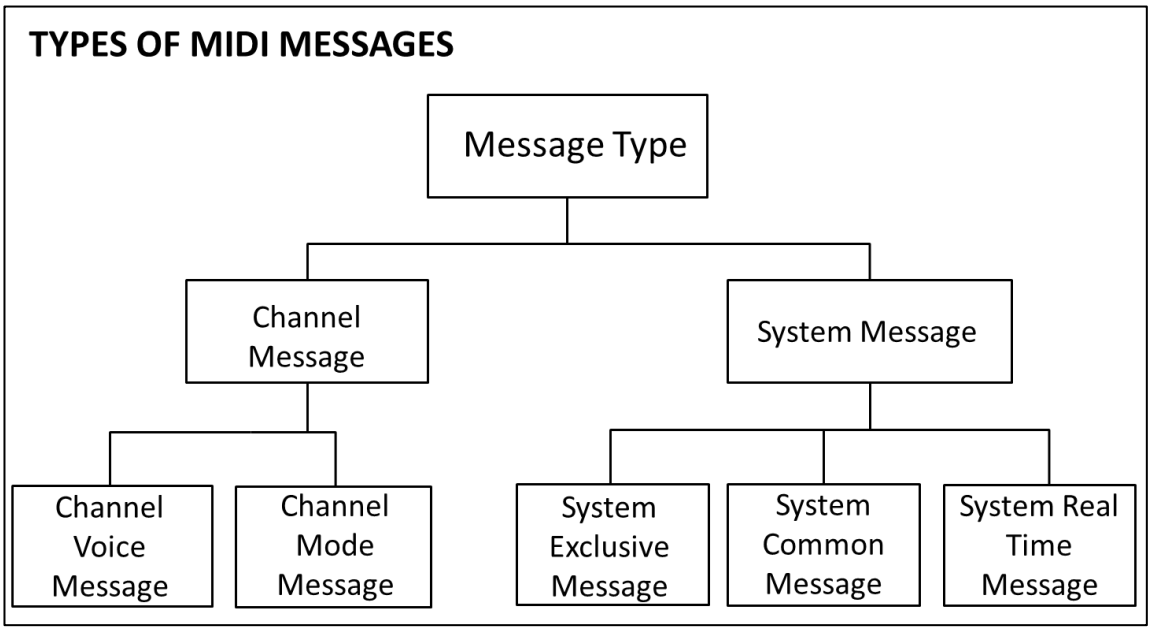
Kuvio 2. Analogisen syntetisaattorin ohjaussignaaleja.

Eri valmistajien vaihtelevista käytännöistä johtuen tarvittiin yhtenäinen tapa ohjata laitteita keskitetysti. Vuonna 1981 Rolandin perustajan Ikutaro Kakehashin aloitteesta syntetisaattorivalmistajat Roland, Oberheim ja Sequential Circuits yhdessä Yamahan, Korgin ja Kawain kanssa alkoivat kehittää yhteistä ohjausstandardia, joka mahdollistaisi eri valmistajien syntetisaattorien ohjaamisen yhteisellä rajapinnalla riippumatta eri valmistajien analogisten piirien yksilöllisistä ohjausjännitteistä. Vuonna 1983 NAMM messuilla (National Association of Music Merchants) esiteltiin Roland JP-6 ja Sequential Circuits Prophet 600 -syntetisaattorien välillä toimiva MIDI-yhteys. MIDI-standardi 1.0 julkaistiin elokuussa 1983. Ikutaro Kakehashi ja Sequential Circuitsin Dave Smith palkittiin Grammy-palkinnolla teknisestä kehitystyöstä MIDI-standardin luomiseksi vuonna 2013. Kyseisen teknisen Grammy-palkinnon ovat voittaneet mm. Ray Dolby, Clarence Fender, Robert Moog ja Thomas Alva Edison. [4]

MIDI-ohjaus toimii käyttämällä useasta tavusta muodostuvia viestejä. Viesti koostuu statustavusta ja sitä seuraavista yhdestä tai kahdesta datatavuista. Reaaliaikaiset ja System Exclusive -viestit voivat muodostaa poikkeuksen. [5]

4.1 Viestityypit

MIDI-viestit jaetaan kahteen kategoriaan: kanavaviesteihin (Channel) ja järjestelmäviesteihin (System). Nämä viestit esitellään tässä opinnäytetyössä, koska näiden viestien tunnistaminen MIDI-viestien joukosta on tärkeää suunniteltaessa studio-ohjelmiston viestinkäsittelijää. Kuviossa 3 esitetään viestiluokat.



Kuvio 3. MIDI-viestityypit [5].

Kanavaviestissä statustavun 4 ylintä bittiä ilmaisevat komennon ja 4 alinta bittiä ilmoittavat MIDI-kanavan 0–15. Komennosta riippuen statustavua seuraa 0–2 datatavua komennon parametreina.

Instrumentti voi ottaa vastaan MIDI-viestejä useammalla kanavalla. Kanavaa, jolla laite ottaa vastaan pääkomennot, kuten esimerkiksi käytettävän ohjelman ja instrumentin tilan, kutsutaan peruskanavaksi. Lisäkanavia kutsutaan äänikanaviksi (Voice Channel).

4.1.1 Kanavaääniviesti

Yleisimmät lähetetyt MIDI-viestit ovat kanavaääniviestejä (channel voice), joilla ohjataan kuultavaa musiikkia. Statustavun neljä eniten merkitsevää biittiä määräävät viestin tarkoituksen ja neljä vähiten merkitsevää bitti määräävät MIDI-kanavanumeron. Kanavaääniviestit on lueteltu taulukossa 1.

Taulukko 1. Kanavaääniviestit [5].

Status	Toiminto	
8n	Nuotti pois päältä	
9n	Nuotti päälle	
An	Polyfoninen jälkipaine	
Bn	Kontrollerin muutos	(0–119)
Cn	Ohjelman muutos	
Dn	Kanavan jälkipaine	
En	Pitch Bend muutos	

Kanavaääniviesti Kontrollerin muutos (Bn) mahdollistaa erilaisten analogisten ohjainten arvojen välittämiseen MIDI-laitteelle. Ohjainten 0 – 31 resoluutio on 14 b siten, että säätöarvon 7 eniten merkitsevää bittiä määritellään viestillä: ”1011nnnn 000ccccc 0mmmmmmm”, missä ”nnnn” on MIDI-kanavan numero, ”ccccc” on ohjaimen numero 0–31 ja ”mmmmmmm” on säätöarvon 7 eniten merkitsevää bittiä. Tilanteessa, jossa tarvitaan korkeampaa resoluutiota, lähetetään viesti: ”1011nnnn 001ccccc 0lllllll”, missä ”ccccc” on sama kuin edellisessä viestissä ilmaisten ohjaimia 32–63 ja ”lllllll” määrää säätöarvon 7 vähiten merkitsevää bittiä.

Laitteiden lähettämällä ohjaindatalla saattaa olla valmistajakohtaisia eroja. Analysoitaessa testikokoonpanon Fatar MIDI 72 koskettimiston modulaatio-ohjainpyörän lähettämää viestivirtaa tarkoitukseen ohjelmoidulla konsolisovelluksella, huomattiin vastaanotettujen MIDI-viestien tulostuksesta ohjaimen lähettävän Bn 01 dd -viestejä, eli karkearesoluutioista eniten merkitseviä tavuja.

Ohjaimet 64:stä ylöspäin ovat matalamman resoluution yksitavuisia ohjaimia, joista ohjaimet 64–69 on oletuksena varattu kytkintyyppisille ohjaimille. Ohjaimet 91–95 ohjaavat audioefektien voimakkuutta.

4.1.2 Kanavan tilaviesti

Kanavan tilaviesti (channel mode) lähetetään samalla statustavulla kuin kontrollerin muutos (Bn), mutta sitä seuraavan datatavun arvo 120–127 ilmaisee viestin olevan kanavan tilaviesti. Taulukossa 2 esitetään tilaviestien koodit.

Taulukko 2. Kanavan tilaviesti [5].

Status (hex)	Data (dec)	Toiminto
Bn	120	Kaikki äänet pois päältä
Bn	121	Nollaa kaikki ohjaimet
Bn	122	Paikallinen ohjaus päälle/pois
Bn	123	Kaikki nuotit pois päältä
Bn	124	Omni pois päältä
Bn	125	Omni päälle
Bn	126	Monofoninen tila
Bn	127	Polyfoninen tila

Standardissa suositellaan, että laite käynnistetään omni-tila päälle. Omni-tilassa laite vastaa kaikilla MIDI-kanavilla saapuviin viesteihin. Omni-tilan ollessa pois päältä laite pitää asettaa määrätylle kanavalle ja laite vastaa vain näillä kanavilla saapuviin viesteihin. Omni pois päältä -tilaa käytetään silloin, kun ohjataan useampaa syntetisaattoria samaan aikaan.

Paikallinen ohjaus voidaan kytkeä pois päältä tilanteessa, jossa paikallisten säätimien ja kontrollerien ei haluta häiritsevän MIDIn ohjauksessa olevaa syntetisaattoria.

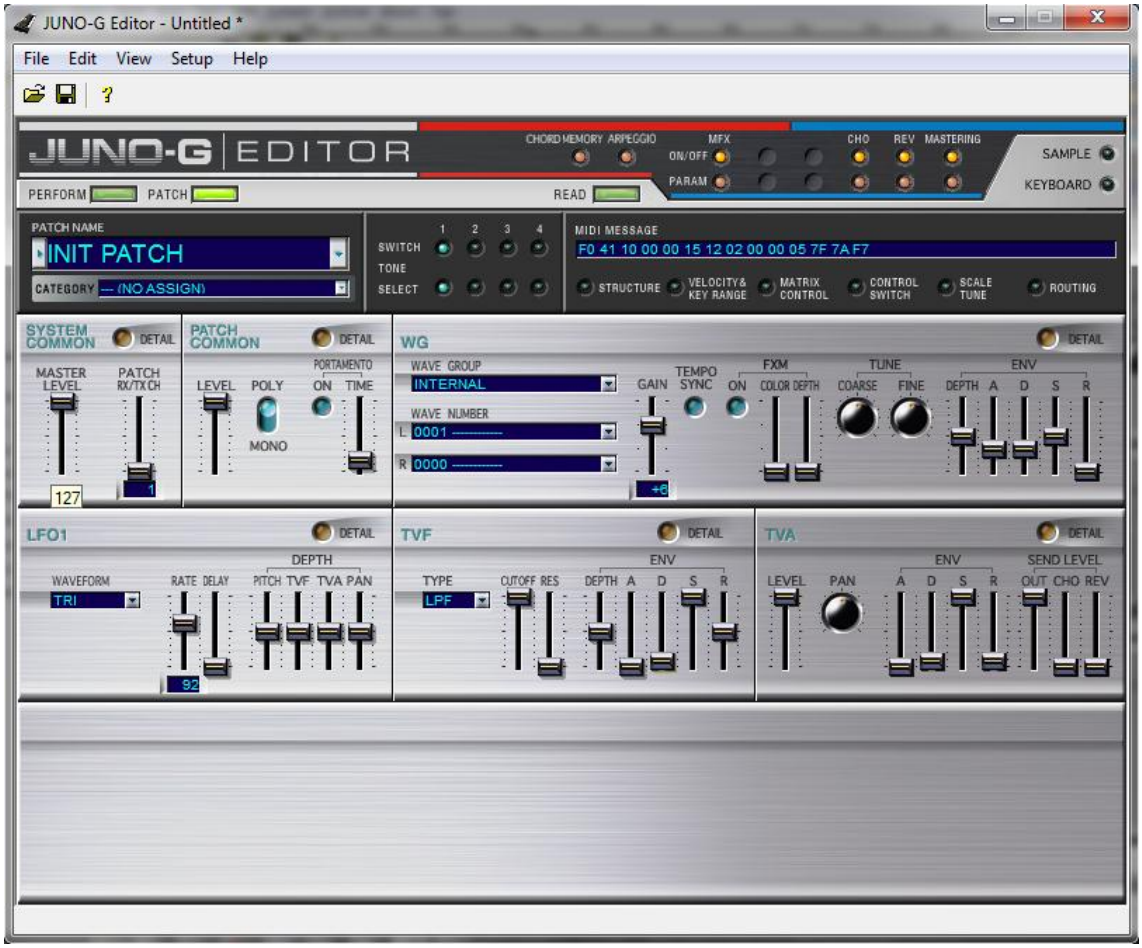
Kanavaviestin lisäksi toinen viestityyppi on järjestelmäviesti. Järjestelmäviestejä ei yksilöidä kanavanumeroilla, vaan ne on tarkoitettu ohjaamaan koko järjestelmää. Järjestelmäviestejä on kolmea tyyppiä.

4.1.3 System exclusive viesti

System exclusive on erikoisviesti, joka voi sisältää minkä tahansa määrän datatavuja. SysEx-viesti päätetään End of Exclusive -viestillä (EOX), tai keskeytetään toisen viestin

statustavulla (ei Real-Time). SysEx-viesti sisältää valmistajan tunnuskoodin (ID). Mikäli vastaanottava laite ei tunnista koodia, viestin dataosuus jätetään huomioimatta.

Yleinen SysEx-viestin käyttötarkoitus on syntetisaattorin ohjelmointi. Useimmissa syntetisaattoreissa ohjelmointi perustuu parametrien syöttämiseen etupaneelin numeronäppäimistöllä käyttöliittymän valikkorakenteeseen. Tämä ohjelmointitapa on hidas ja työläs. Aikaisemmin ohjelmoinnin apuna käytettiin laitevalmistajan tarkoitukseen kehittämää ohjelmointilaitetta. Ohjelmointilaitteen paneelissa on säätimet syntetisaattorin parametrien säätämiseen ja säätöarvot lähetetään syntetisaattorille system exclusive -viesteinä MIDIn välityksellä. Ohjelmointilaitteet on tarkoitettu rajoitetulle valikoimalle syntetisaattoreita ja eri tuoteryhmän syntetisaattorit tarvitsevat niille sovitetun SysEx-standardin mukaisen ohjelmointilaitteen. Tietokoneiden multimediaominaisuuksien yleistyttyä laitevalmistajat ovat kehittäneet ohjelmointisovelluksia, joissa syntetisaattorin ohjelmointi tapahtuu tietokoneen avulla graafisessa käyttöliittymässä. Tietokoneella muokatut ohjelmaparametrit lähetetään MIDI-liitännän välityksellä reaaliaikaisina system exclusive -viesteinä samalla periaatteella kuin fyysisellä ohjelmointilaitteella. Kuva 1 seuraavalla sivulla esittää Roland Juno-G -syntetisaattorin ohjelmointisovelluksen käyttöliittymän ulkoasua. Kuvassa ”Master level” -äänenvoimakkuus on säädetty maksimiarvoonsa (127). Komentoa vastaava SysEx-viesti: ”F0 41 10 00 00 15 12 02 00 00 05 7F 7A F7”, näkyy ”MIDI message”-kentässä. Ensimmäinen tavu F0 ilmoittaa SysEx-viestin statuksen MIDI-kanavalla 0. Seuraava tavu 41 on Rolandin standardin mukainen valmistajan ID, jonka perusteella vastaanottava laite pystyy tunnistamaan itseään koskevat SysEx-viestit. Kolmanneksi viimeinen datatavu 7F on säätimen lukuarvo 127 heksadesimaalilukuna, ja viimeinen tavu F7 on komentojonon päättävä EOX-viesti.



Kuva 1. Kuvaruutukaappaus ohjelmointisovelluksesta.

Jotta kolmannen osapuolen sovellusten kehittäjät voisivat kehittää sovelluksensa toimimaan eri valmistajien instrumenttien kanssa, valmistajien on julkaistava SysEx-viestiensä dataformaatti. Vain valmistaja voi määrittää ja päivittää dataformaattia.

Täydellinen lista valmistajien ID-koodeista on lueteltu MIDI-standardissa.

4.1.4 System common -viesti

Common, eli yhteisviestit on tarkoitettu kaikille vastaanottajille. Yhteisviestejä käytetään järjestelmän synkronointiin ja kappalevalintaan. Esimerkiksi oskillaattorien herkkyys lämpötilan vaihteluille ja virityksen muuttuminen on analogisten syntetisaattorien ominaisuus ja syntetisaattorin saattaa joutua virittämään uudelleen käytön aikana. Mikäli analogisessa syntetisaattorissa on MIDI-liitäntä ja automaattinen oskillaattorien

viritystoiminto, yhteisviestillä F6 voidaan käskeä järjestelmään kytkettyjä laitteita virittämään oskillaattorinsa.

Taulukossa 3 esitetään yhteisviestien koodit.

Taulukko 3. System common -viestit [5].

Status (hex)	Toiminto
F1	MIDI aikakoodi ¼ kehys
F2	Kappaleen paikkaosoitin
F3	Kappalevalinta
F6	Virityspyyntö
F7	EOX. SysEx loppumerkki

4.1.5 Reaaliaikaviesti

Yhteisviestillä synkronoitaessa paikkamerkki lähetetään kaikille laitteille yhteisesti. Ohjattavissa laitteissa voi olla oma sisäinen sekvensseri ja laite noudattaa omaa ohjelmaa. Reaaliaikaviestejä (Real-Time) käytetään tällaisten ajastettujen laitteiden synkronointiin. Nämä viestit koostuvat pelkästään statustavuista ilman datatavuja ja ne voidaan lähettää milloin tahansa. Mikäli Real-Time-viesti ei koske vastaanottavaa laitetta, se jätetään huomioimatta, mutta se voidaan muuten suorittaa milloin tahansa, jopa toisen suoritettavan viestin tavujen välillä, jolloin laite viestin suoritettuaan palaa takaisen kesken jääneen viestin suorituksen tilaan. Reaaliaikaviestit esitetään taulukossa 4.

Taulukko 4. Reaaliaikaviestit [5].

Status (hex)	Toiminto
F8	Ajastin
FA	Käynnistä
FB	Jatka
FC	Pysäytä
FE	Laitetunnistus
FF	Järjestelmän nollaus

Laitetunnistuspyyntöä FE käytetään silloin, kun muuta MIDI-liikennettä ei lähetetä. Laitetunnistuspyynnöllä ja siihen saapuvilla vastausviesteillä pidetään yllä yhteyttä järjestelmän laitteisiin ja viestien loppumisesta voidaan tunnistaa irrotettu MIDI-kaapeli, tai muusta syystä katkennut yhteys.

4.2 Tietotyypit

MIDIn välityksellä lähetetään kahden tyyppisiä tavuja: statustavuja ja datatavuja. MIDI-standardissa tavu tunnistetaan statustavuksi tai datatavuksi sen eniten merkitsevän bitin (2^7) perusteella. Statustavun eniten merkitsevä bitti on 1 ja datatavun eniten merkitsevä bitti on 0.

4.2.1 Statustavu

Statustavun ensimmäinen bitti määrittelee tietotyypin statustavuksi. Statustavu määrittelee sitä seuraavien datatavujen merkityksen. Statustavu määrää laitteen uuteen tilaan riippumatta siitä onko aikaisempi viesti suoritettu.

Yksittäisen nuotin MIDI-viesti käsittää kolme tavua, joiden lähettäminen 31,25 (+/- 1%) kb/s nopeudella vie aikaa 960 µs. Useiden samanaikaisten nuottien lähettäminen voi kestää useita millisekunteja ja aiheuttaa havaittavan viiveen musiikkiin. Ongelmaa voidaan vähentää käyttämällä niin sanottua jatkuvaa statusta (running status).

Jatkuvaa statusta voidaan käyttää erityisesti lähetettäessä pitkiä sarjoja nuotti päälle ja nuotti pois päältä viestejä. MIDI-laite säilyttää viimeisimmän vastaanotetun statuksen, jolloin statustavu voidaan jättää lähettämättä. Nuotti pois päältä -viesti voidaan korvata antamalla nuotti päälle -viestin parametrina iskuvoimakkuudeksi 0, jolloin viestin vaikutus on sama, kuin nuotin sammuttavalla viestillä.

MIDI-laitteen virran kytkemisen ja sammuttamisen yhteydessä laitevalmistajan tulee huolehtia, ettei virheellisiä viestejä lähetetä. Laitteiden ja ohjelmistojen suunnittelussa tulee ottaa huomioon virheviestien mahdollisuus ja standardin mukaan virheelliset viestit tulee jättää huomiotta, etteivät ne häiritse järjestelmän toimintaa [5].

4.2.2 Datatavu

Statustavua seuraa yksi tai kaksi datatavua, joissa viestin parametrit välitetään laitteelle. Datatavu tunnistetaan eniten merkitsevän bitin arvosta 0. Datatavujen määrä määräytyy statustavun perusteella.

Eniten merkitsevän bitin varaamisesta tunnistuskäyttöön seuraa se, että MIDI-parametrille on varattu 128 tasoa (0–127). MIDI sävelasteikko käsittää 10½ oktaavia alkaen C0:sta (8,176 Hz) jatkuen G10:een (12543,854 Hz). Klassisen musiikin sävelasteikoiden virityksessä käytetään asteikon keskimmäistä A-nuottia (440 Hz), joka on MIDI-nuotti numero 69.

Kun tarvitaan parempaa erottelutarkkuutta, esimerkiksi sävelkorkeutta viritettäessä, datatavulla voidaan esittää enemmän tasoja järjestämällä data ylä- ja alatavuksi. Tällä menetelmällä parametrin 7 eniten merkitsevää bittiä esitetään ylätavulla ja 7 vähiten merkitsevää bittiä alatavulla. Datatavujen eniten merkitsevät bitit pysyvät 0:na, mutta datatavut yhdistämällä saadaan parametrille 16384 tasoa.

5 STUDIOSOVELLUS

Tässä luvussa esitellään sovelluksen toiminta ja Unity skriptien lähdekoodi. Sovelluksen toiminta jäljittelee kuvassa 2 esitetyn analogisen sekvensserin kaltaisen laitteen toimintaa. Sovellus ohjaa järjestelmän MIDI out -porttiin kytkettyä syntetisaattoria sovelluksen säätimien asennon määräämällä tavalla.

Kuva 2. Analoginen sekvensseri. [6]

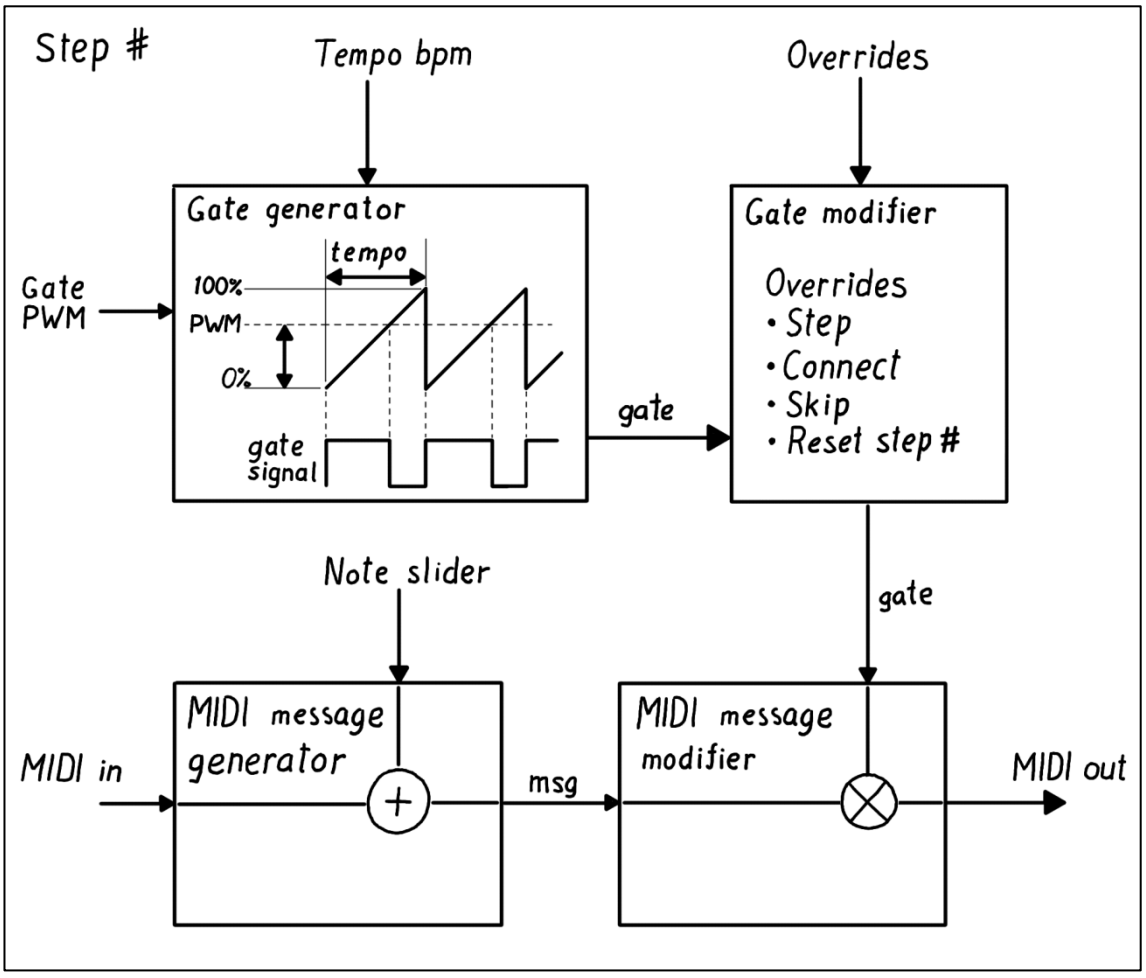
Sovelluksessa sävelkuvio voidaan käynnistää joko tietokoneen näppäinkomennolla tai järjestelmän MIDI in -porttiin kytketyn koskettimiston koskettimen painalluksella. Sävelkuvion toistonopeutta säädetään sovelluksen ”Tempo bpm” -säätimellä. Soitettavan nuotin painalluksen pituutta säädetään sovelluksen ”Gate PWM” -säätimellä, jolla määrätään prosenttiosuus tahdin neljäsosaiskusta, jonka ajan nuottia pidetään alaspainettuna.

Soitettavan nuotin sävelkorkeus määrätään sovelluksen ”Note” -säätimellä. MIDI-sisääntuloon kytketyn koskettimiston lähettämän nuotin sävelkorkeus lasketaan yhteen säätimen osoittaman sävelkorkeuden kanssa, jolloin saadaan aikaan sävelaskeltransponointi käyttäen vertailunuottina keski-C-nuottia 231,626 Hz (MIDI 60).

Sävelkuvion etenemistä hallitaan valintasäätimellä, joka toteuttaa yhden neljästä valintatoiminnosta.

1. Step — Askel soitetaan normaalissa järjestyksessä ajastimen määräämässä tahdissa.
2. Connect — Nuotti liitetään seuraavaan siten, että nuottia ei sammuteta askelten välillä.
3. Skip — Soiva nuotti soitetaan normaalisti loppuun. Askeleen nuotti jätetään soittamatta ja sen paikalla soitetaan tauko.
4. Reset — Merkitsee sävelkuvion viimeisen nuotin. Soiva nuotti soitetaan loppuun, askellaskuri nollataan, ja sävelkuvion soittaminen aloitetaan uudelleen ensimmäisestä askeleesta.

Sävelkuvion yhden askeleen toiminta esitetään kuviossa 4.



Kuvio 4 Sekvensserisovelluksen yhden askeleen toimintaperiaate.

5.1 Ajastimet

Sekvensserin toimintaperiaate on suorittaa toimintoja määrätyin väliajoin.

Sovelluksen on reagoitava säädettyihin tempon ja pulssinleveyden säätöjen muutoksiin sovelluksen käydessä. Käytettävän ajastimen tyyppi valitaan tähän tarkoitukseen sopivaksi. Erilaisia ajastimia on valittavana useita tyyppejä käyttötarkoituksen mukaan. Tarkasteluun valittiin kolme toimintaperiaatteeltaan erilaista ajastinta: .NET-frameworkin System.Timers.Timer -ajastin ja System.Diagnostics.Stopwatch, sekä Unityn pelimoottorin UnityEngine.Time.deltaTime.

5.1.1 System.Timers.timer

Kun käyttöjärjestelmän ajastimeen määritelty lukuarvo (Timer.Interval) täyttyy, ajastin aiheuttaa Timer.ElapsedEvent -tapahtuman. Tämä tapahtuma toimii keskeytyksen tavoin ja tapahtumaan liitetty metodi suoritetaan automaattisesti. Järjestelmän ajastimen Interval-arvon muokkaaminen ajastimen käydessä on mahdollista, mutta tässä työssä käytetään havainnollisempaa tapaa, jossa kulunutta aikaa verrataan annettuun raja-aikaan. Järjestelmän ajastin on tarkoitettu ajastettujen toimintojen käynnistämiseen, eikä käynnissä olevan ajastimen laskuria voi seurata. Ajastimen voi asettaa uudelleen, mutta tämä nollaa ajastimen ja tästä syystä järjestelmän ajastin ei ole käyttökelpoinen sovelluksen käydessä.

5.1.2 System.Diagnostics.Stopwatch

Ajanottoajastimen laskuria voidaan lukea ajastimen käydessä. Järjestelmän laskurin toiminnasta poiketen päätös ajastettavan toiminnon käynnistämisestä tehdään sovelluksen hallitsemana. Ajastimen nollaus, käynnistys ja sammutus tehdään sovelluksesta käsin. Sovellus ohjelmoidaan reagoimaan vain raja-arvon ylitykseen riippumatta siitä, ollaanko aikaviivettä lyhentämässä vai pidentämässä, joten ajanottoajastin on sopiva vaihtoehto sovelluksen käyttöön.

5.1.3 UnityEngine.Time.deltaTime

Unity-pelimoottorissa on oma ajastinjärjestelmä, joka mittaa skriptin päivityskierrokseen käyttämän ajan. Edellä mainituista ajastimista poiketen deltaTime ei ole juokseva arvo, vaan se ilmoittaa pelkästään edellisestä päivityksestä kuluneen ajan sekunteina. Sovelluksessa on huolehdittava käytetyn ajan keräämisestä muuttujaan vertailua varten. Yleinen periaate on, että päivityskierroksen kesto on sidottu sovelluksen ruudunpäivitystaajuuteen, joka voidaan joko määrittää sovelluksessa (yleensä 30 tai 60 kertaa sekunnissa), tai jättää määrittelemättä, jolloin ruudunpäivitys tapahtuu niin nopeasti kuin Unity pystyy suorittamaan.

Päivityskierroksen aikana suoritetaan joitakin tehtäviä, kuten fysiikan laskemiseen tarkoitettu FixedUpdate()-metodi, joka pyritään suorittamaan tasaisin määrätyin väliajoin. FixedUpdate() suoritetaan useamman kerran normaalin päivityskierroksen aikana, joten tässä metodissa tehty laskurin tarkkailu mahdollistaa tarkemman ajan mittauksen. FixedUpdate()-metodin sallittu ajankäyttö määritellään Unityn time managerissa, joka löytyy valikosta Edit – Project Settings – Time – Fixed Timestep. Mikäli FixedUpdate()-metodia kuormitetaan liian suurella tehtävämäärällä, sen suoritusnopeus alkaa hidastua ja ajanseuranta muuttuu epätarkemmaksi. Yksinkertaisesta ajastimen arvon seuraamisesta FixedUpdate() suoriutuu tyypillisesti 1000 kertaa sekunnissa ilman vaikeuksia.

5.1.4 Ajan seuranta

Unityn deltaTime-arvon käyttäminen lisäisi sovellukseen tarpeettoman muuttujan päivityksen, joten sovelluksen ajanseurantaan valittiin System.Diagnostics.Stopwatch, jonka edistymistä seurataan ajankäyttöä käsittelevän C#-skriptin FixedUpdate()-metodissa. Skripti lukee sovelluksen toimintalogiikkaa käsittelevästä skriptistä seuraavaksi tarkasteltavan aikarajan ja ilmoittaa aikarajan täyttymisestä takaisin toimintalogiikkaskriptille. Ajanottoajastimen arvoa seurataan lukemalla Stopwatch-luokan ElapsedMilliseconds kenttää, joka palauttaa ajastimen arvon millisekunteina long -tyyppisenä muuttujana.

Musiikissa tempon yksikkö on neljäsosaiskua minuutissa. Musiikin tahtilaji ei vaikuta esitysnopeuteen, vaan 4/4 ja 9/8 tahtilajit esitetään samalla nopeudella. Edellä valittiin

ajan laskentamenetelmäksi ajanottoajastimen millisekuntilaskuri. Sekvensserin ajastuksessa seurattava laskurin arvo saadaan siis jakamalla minuutissa olevien millisekuntien määrä (60000) tempon arvolla ♩=n

$$t_{ms} = \frac{60000}{n}$$

Kaava 1. Sekvenssin askeleen pituus.

Jakolaskuja sisältävien kaavojen ohjelmoinnissa C#:lla tulee suunnittelussa kiinnittää huomiota lopputuloksen tarkkuusvaatimukseen. Laskettaessa jakolaskuja kokonaislukumuuttujilla, kuten tämän työn ajastimen ohjelmoinnissa tehdään, laskun lopputuloksessa menetetään desimaaliosa. Desimaaliosan leikkautuminen aiheuttaa lopputulokseen maksimissaan 1 ms:n eron, mikä on pienempi kuin sovelluksessa käytettävän ajastimen resoluutio.

Suurempaa tarkkuutta vaativissa tilanteissa, missä oikealla pyöristyksellä on merkitystä, voidaan käyttää .NET matematiikkakirjaston pyöristystä:

```
Math.Round(liukulukuarvo, MidpointRounding.AwayFromZero);
```

Tai Unityn pyöristysluokkaa:

```
Mathf.RoundToInt (liukulukuarvo);
```

Murtolukujen laskentaa varten kokonaislukumuuttuja voidaan väliaikaisesti muuttaa liukuluvuksi tyyppimuunnoskomennolla:

```
(float)kokonaislukumuuttuja
```

Laskutoimituksiin syötettävä tasaluku käsitellään oletuksena kokonaislukuna, mutta se voidaan määrätä liukuluvuksi esittämällä se muodossa:

```
1.0f
```

Pyöristys suoritetaan laskutoimitusten lopputulokseen.

Musiikkiesityksessä sama nuotti voidaan esittää eri tavoin. Tavallisesti säveltäjä merkitsee normaalista poikkeavan nuotin esitystavaksi joko *staccato* (♩̇ , suom. terävästi), tai *legato* (♩⁀♩ , suom. sitoen). Mikäli esitystapaa ei ole nuottikirjoitukseen muuten merkitty, kahden nuotin välissä olevan tauon pituus riippuu esittäjän

tulkinnasta. Syntetisaattorin ohjaussignaalin kannalta ajateltuna tämä tarkoittaa sitä, että osan askeleen keston ajasta nuotti on alas painettuna ja askeleen loppuosan ajaksi nuotti päästetään vaimentumaan syntetisaattorin sustain-asetuksen mukaisesti.

Tempon lisäksi sovelluksen ajastinta ohjaa PWM-säädin (Pulse Width Modulation), joka ilmoittaa nuotin aktiivisen ja passiivisen vaiheen välisen suhteen p_w prosentteina. Laskemalla nuotin aktiivisen ajan osuus askeleen kokonaiskestosta, selvitetään nuotin vapauttamisen ajankohta t_r.

$$t_r = \frac{60000 * p_w}{n * 100}$$

Kaava 2. Nuotin vapautusaika.

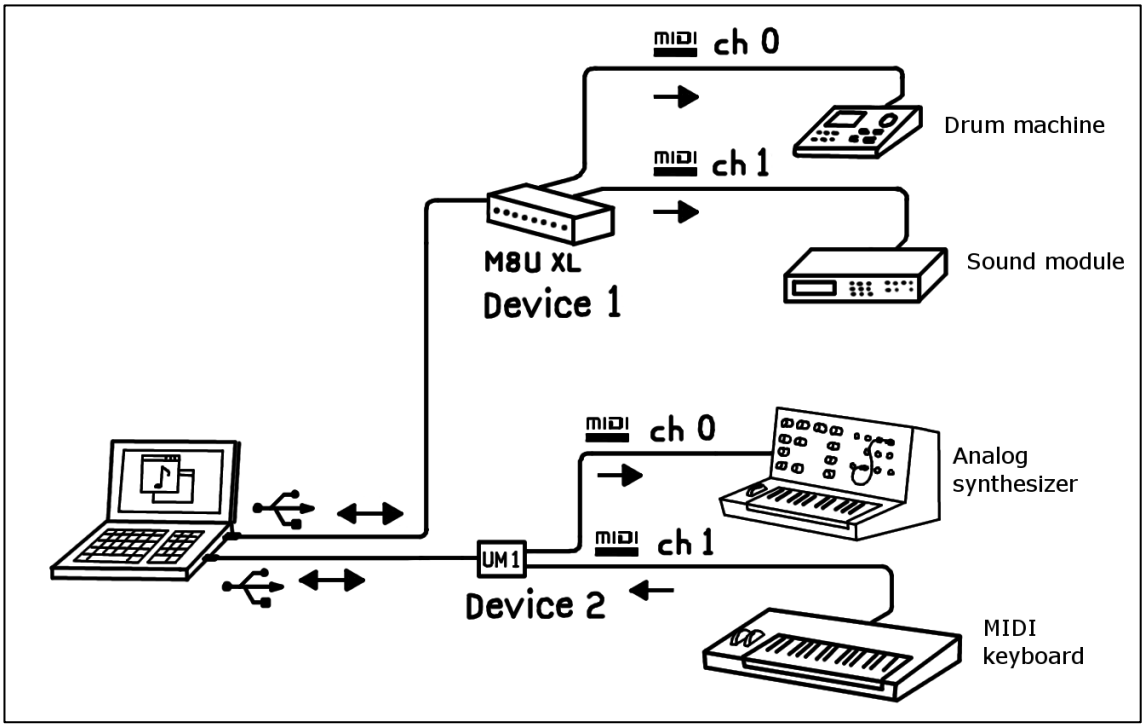
Nuotti voidaan merkitä myös fermaatilla (), pidäkemerkillä, jolloin nuotin soittamista pidennetään esittäjän tulkinnan mukaan. Pidäkkeellä merkityn nuotin soittoaikaa ei ole täsmällisesti määritelty, eikä sitä yleensä käytetä sekvensserillä tuotettavissa sävelkuvioissa. Käytännössä pidäke voidaan toteuttaa pysäyttämällä manuaalisesti kaikkia järjestelmään kytkettyjä laitteita ohjaava synkronointisignaali ja lähettämällä MIDI-reaaliaikaviesti FC (pysäytä), ja pidäkkeen loputtua lähettämällä FB (jatka).

5.2 Järjestelmän alustukset

Jotta sovellus kykenisi lähettämään ja vastaanottamaan MIDI-viestejä, on sovellukseen aluksi luotava lista järjestelmään kytketyistä laitteista. Winmm.dll on ohjelmointirajapinta (API, Application Programming Interface), jonka kautta sovelluksella on pääsy MIDI-laitteen toimintoihin.

Ensimmäisenä toimenpiteenä selvitetään järjestelmään kytkettyjen MIDI-laitteiden lukumäärä. Ulkoiset funktiot MidiInGetNumDevs() ja MidiOutGetNumDevs() palauttavat järjestelmään kytkettyjen laitteiden lukumäärän. Sovellukseen luodaan listat laitetunnuksille ja laitteiden nimille. Laitetunnus (handle) on yksilöllinen tunnus, kuin radiokutsu tai puhelinnumero, jolla voidaan osoittaa toiminto määrätylle laitteelle.

MIDI-kanavien määrä rajoittaa ohjattavien laitteiden määrää. Kytkemällä järjestelmään useampia MIDI-laitteita, voidaan ohjattavien instrumenttien määrää lisätä. Kuviossa 5 esitetään laitteiden ja kanavien määrän lisäämisen logiikka.



Kuvio 5. Sama kanava eri laitteella.

MIDI-laitteiden numerointi alkaa 0:sta siten, että ensimmäinen MIDI out -laite (0) on tietokoneen äänipiirin MIDI out. Tietokoneiden multimedialaitteissa on valmistajakohtaisia eroja. Esimerkiksi tapauksessa, jossa tietokoneen äänipiirissä ei ole MIDI in -porttia, kuten kuvan 5 tapauksessa, tietokoneen MIDI out on laite 0, M8U XL MIDI -liittymä on ulostulolaite 1 ja UM1 on ulostulolaite 2. M8U XL MIDI sisääntulot näkyvät MIDI in -laitteena 0 ja UM1 sisääntulo MIDI in -laitteena 1.

MIDI-laitteiden käyttöönotto aloitetaan luomalla MidiInProc- ja MidiOutCaps-objektit new-komennolla. MidiInProc-delegaatti saa parametriksi viestinkäsittelymetodin nimen. MidiOutCaps ei tarvitse parametria. Callback-funktiolle ja viestinkäsittelijän statuslipuille määritellään vakioarvot, jotta näihin voidaan viitata suoraan nimellä lukuarvon sijasta. Luotavat vakiot ovat:

```
private const int CALLBACK_FUNCTION = 0x30000;
private const int MIM_DATA = 0x3C3;
private const int MIM_ERROR = 0x3C5;
private const int MIM_LONGDATA = 0x3C4;
```

Seuraavaksi MidiOutGetDevCaps-funktiolle toimitetaan parametriksi tiedusteltava laitenumero, muistiosoitin MidiOutCaps-structiin ja structin koko etumerkittömänä int -tyyppiseksi muuttujaksi käännettynä. Winmm.dll-funktiot palauttavat int -tyyppisen muuttujan, joka ilmaisee suoritetun funktion onnistuneen palauttamalla arvon 0, tai virhetilanteessa jonkin muun arvon.

Funktiot voivat palauttaa vain yhden arvon kerrallaan, ja paluuarvo on varattu virhekoodille. Tästä syystä MidiOutGetDevCaps-funktiolle annettiin parametriksi muuttujan sijasta osoitin muuttujan varaamaan osoitteeseen tietokoneen muistissa. Funktio kirjoittaa halutun tiedon suoraan muuttujan käyttämään muistiosoitteeseen ja funktion onnistuneen suorittamisen jälkeen muuttujan uusi arvo, tässä tapauksessa MIDI-laitteen nimi on pääohjelman käytettävissä normaaliin tapaan.

Laitetunnus, eli handle on tärkein MIDI-ohjauksessa käytettävä tieto. Laitetunnuksella viestit osoitetaan halutulle laitteelle. Handle saadaan tietää suorittamalla MidiInOpen- ja MidiOutOpen-funktiot. Parametriksi annetaan handlen osoitin, laitteen ID-numero, callback-delegaatin nimi, instanssi ja mahdolliset ohjausliput. Funktion suorittamisen jälkeen handle on varattu kyseisen laitteen käyttöön, kunnes se vapautetaan sulkemalla yhteys.

Sovelluksen kehitysvaiheessa yhteyden asianmukainen sulkeminen on tärkeää. Unityssä yhteyden sulkeminen tapahtuu skriptin OnApplicationQuit()-metodissa. Mikäli yhteyttä ei suljeta, menetetään muuttujaan tallennettu handle-tieto ja yhteyden avaaminen uudelleen on mahdotonta avonaisen yhteyden ollessa edelleen varattuna Unityn käyttöön. Tapauksessa, jossa handle-tieto menetetään, ainoa vaihtoehto avata yhteys uudelleen on sulkea ja käynnistää Unity uudelleen.

5.3 MIDI-viestin lukeminen

Alustuksissa luotiin valmiudet MIDI-viestien lukemiselle. Viestien kuuntelu käynnistetään suorittamalla MidiInStart-funktio ja antamalla funktion parametriksi laitteen handle.

Kun käyttöjärjestelmän laiteajuri saa laitteelta MIDI-viestin, se nostaa ohjelmistokeskeytyksen (event). MidiInProc-delegaatti välittää delegaatin esittelyssä määritellyt parametrit viestinkäsittelymetodiin.

Käsittelymetodissa tutkitaan saapunut viesti. Metodin parametreina saadaan handle, saapuvan viestin tyyppi, instanssin numero, MIDI-viesti ja viestin aikaleima. Alustuksissa tehtyjen vakioiden määrittelyjen perusteella callback- funktiolle saapuva viestin tyyppi voi olla jokin taulukossa 5 määritellystä tyypistä.

Taulukko 5. MIDI IO -status [7].

MIM_DATA	MIDI-laite on vastaanottanut viestin.
MIM_ERROR	MIDI-laite on vastaanottanut virheellisen viestin.
MIM_LONGDATA	SysEx-bufferi on täynnä ja lähetetään sovellukselle.

5.3.1 Tiedon tulkinta viestistä

Oletetaan, että sovellus vastaanottaa MIDI-viestin 8329368 10-järjestelmän kokonaislukuna. Tietokoneelle viestin esitysmuoto on merkityksetön, mutta käyttäjän kannalta ihanteellisempaa on esittää viesti heksadesimaalilukuna 7F1898.

On huomattava, että x86- ja x86-64 -arkkitehtuurit käyttävät little endian -tavujärjestystä, missä yli tavun mittaiset muuttujat tallennetaan muistiin alkaen vähiten merkitsevästä tavusta. Esimerkkitapauksessa vastaanotetun MIDI-viestin statustavu on 98, eli nuotti päälle kanavalla 8. Ensimmäinen datatavu, eli MIDI-nuotin järjestysnumero on 18, joka on standardin pianokoskettimiston alin C. Toinen datatavu, lyöntivoimakkuus, on 7F (dec 127), eli nuotti soitetaan suurimmalla voimakkuudella.

Palautetaan mieleen teoriaosuudessa esitelty MIDI-viestien jaottelu esittämällä viestien statustavut taulukossa 6.

Taulukko 6. MIDI-viestien statustavut.

Kanavaviestit		Järjestelmäviestit		
Ch voice	Ch mode	SysEx	Common	Real Time
8n	Bn	F0	F1	F8
9n			F2	FA
An			F3	FB
Bn			F6	FC
Cn			F7	FE
Dn				FF
En				

Huomataan, että mikäli tavun arvo on pienempi kuin F0 (dec 240), kyseessä on kanavaviestiä merkitsevä statustavu. Mikäli tavun arvo on pienempi kuin 80 (dec 128), kyseessä on oltava datatavu.

5.3.2 Bittimaski ja siirto-operaatiot

Statustavun erotteluun viestistä käytetään C#-kielen loogista JA-bittioperaattoria (&). JA-operaattoria käytetään bittimaskina, jolla eliminoidaan luvusta sellaiset bitit, joiden ei haluta näkyvän tuloksessa. JA-bittioperaatiossa binäärimuotoista muuttujaa vertaillaan binäärimuotoiseen vertailulukuun. Mikäli luvun positiossa verrattu ja vertaava bitti on 1, tuloksen positiossa vastaava bitti on 1, muuten 0.

Statustavu erotetaan ohjelmakoodissa komennolla:

```
tulos = (verrattava & 0xff);
```

Esimerkkitapaus antaa operaation tulokseksi dec 152, joka on välillä 128–240, eli tiedämme statustavun olevan kanavaviesti. Kanava selvitetään vertaamalla viestiä lukuun F, mikä antaa kanavanumeroksi 8. Vertaamalla lukuun F0, saadaan dec 144, joka tarkoittaa statustavun tarkoittavan nuotin soittamista.

Standardin perusteella tiedetään nuotin soittokäskyn muodostuvan kolmesta tavusta. Nuotin numeron selvittämiseen tarvitaan bittimaskin lisäksi bittisiirto-operaattoria, jolla siirretään viestin sisältöä yhden tavun, eli 8 bittiä oikealle. Nuotin numero selvitetään komennolla:

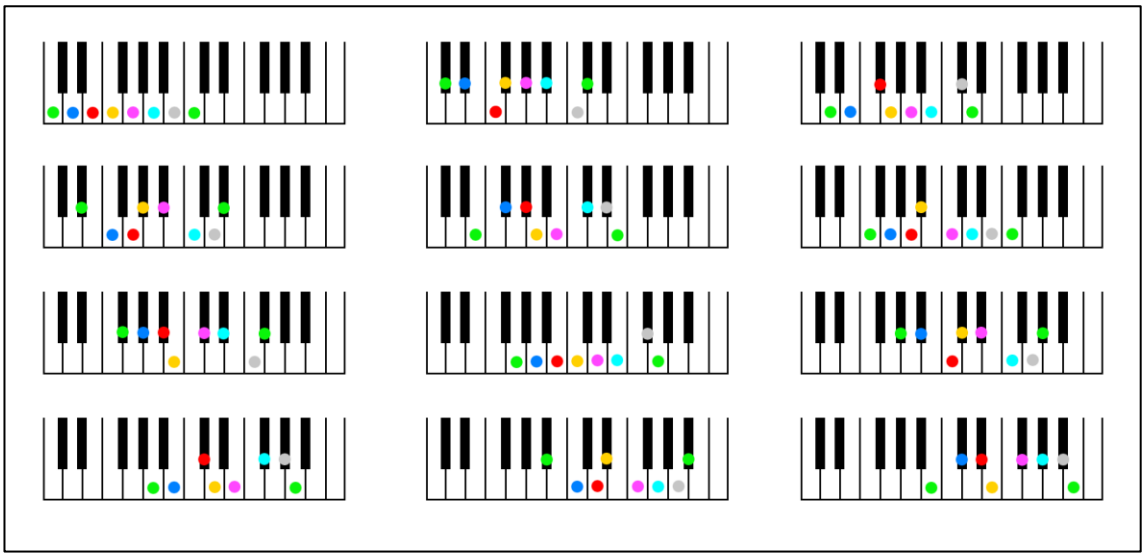
```
tulos = (verrattava & 0xff00) >> 8;
```

Tulokseksi saadaan dec 24, joka on suoraan soitettavan nuotin numero. Nuotin soittovoimakkuus saadaan käyttämällä vertailulukuna 0xFF0000 ja 16 bitin siirtoa oikealle. Nämä operaatiot antavat soitettavan nuotin voimakkuudeksi 127.

5.4 MIDI-viestin transponointi

Sovelluksen käyttöliittymän säätimissä sävelkuvio on asetettu määrätylle sävelkorkeudelle. Jotta sävelkuviota voitaisiin käyttää musiikkiesityksen osana, on se

pystyttävä toistamaan esitykseen sopivassa sävellajissa. Nuottien uudelleensäätäminen käyttöliittymän paneelin säätimillä ei ole käytännöllistä. Sävelkuvion sävellajia muutetaan lisäämällä tai vähentämällä säädettyjen nuottien sävelkorkeutta tietyllä määrällä sävelaskelia. Tätä toimenpidettä kutsutaan transponoinniksi. Perusnuottiin lisättävien sävelaskelten vaikutus transponoitavaan skaalaan esitetään kuviossa 6.



Kuvio 6. Transponoidut sävellajiskaalat.

Perussäätöön laskettavien sävelaskelten määrä selvitetään vertaamalla koskettimistolta luetun MIDI-viestin nuotin numeroa määrättyyn nuottiin. Transponoinnin nollatasona käytetään A 440 Hz -nuotin sijaan keski-C-nuottia (MIDI 60) 261,626 Hz. Tämä noudattaa yleistä C-duuri sävelasteikkoon perustuvaa transponointikäytäntöä, missä C-nuotille transponoitu sävelkuvio soitetaan samassa sävellajissa. Oktaavien numeroinnissa on olemassa kolme eri standardia. Nuotin 60 oktaavin järjestysnumeroksi merkitään valmistajasta riippuen joko 3, 4 tai 5.

C3 -merkintä noudattaa Yamahan yleisesti käyttämää standardia, missä standardin 88-koskettimisen pianokoskettimiston alimman C–B-oktaavin numerointi alkaa 0:sta. Rolandin käyttämässä standardissa saman oktaavin numerointi alkaa 1:sta. Ennen elektronisen musiikin aikakautta tämä numerointi oli riittävä pianon äänialalle.

MIDI-standardissa sävelasteikko alkaa kahta oktaavia alempaa, joten MIDI-asteikon alin oktaavi olisi merkinnältään -1 tai -2. MIDI noudattaa ohjelmointistandardiksi

kutsuttua järjestelmää, jota käytetään musiikkiohjelmistoissa kuten esimerkiksi Cakewalk Sonarissa. Ohjelmointistandardissa keski-C on C5.

Nimeämiskäytännöllä on merkitystä vain käyttöliittymässä näytettävän nuotin kannalta. Tämän työn sovelluksessa transponoinnissa käytettävä menetelmä noudattaa musiikkiohjelmistoissa käytettävää ohjelmointistandardia.

Sovelluksesta ulos lähetettävän nuotin n_o numero saadaan lisäämällä käyttöliittymään säädetyn nuotin n_s arvoon koskettimistolta luetun nuotin n_k ja nuotin 60 erotus.

$$n_o = n_s + n_k - 60$$

Kaava 3. Nuotin järjestysnumeron muokkaaminen.

Nuotin muokatun arvon kelvollisuus on tarkistettava ennen MIDI-viestin muodostamista. Jos nuotin arvo on negatiivinen, lisätään nuotin arvoon yksi oktaavi, eli 12 puolisävelaskelta, kunnes nuotin arvo asettuu sallitulle lukualueelle. Samoin 127 ylittävistä arvoista vähennetään yksi oktaavi, kunnes arvo asettuu sallitulle alueelle.

5.5 MIDI-viestin muodostaminen

MIDI-viestin muodostaminen muistuttaa toimenpiteenä vastaanotetun viestin analysointia. Viesti rakennetaan statustavusta ja vaadittavasta määrästä datatavuja samalla periaatteella loogisilla bittioperaatioilla ja bittisiirto-operaatioilla. Oikeaan positioon siirto-operaattorilla (<<) siirretyt tavut voidaan kirjoittaa nollaksi alustettuun muuttujaan loogisella TAI-bittioperaattorilla (|).

Oletetaan, että esimerkkitapauksessa MIDI-kanavalle 10 asetetulle rumpukoneelle lähetetään viesti, joka soittaa tamburiinin iskun *mezzo forte* -voimakkuudella, eli melko voimakkaasti lukualueella dec 70–80. Oletetaan edelleen, että rumpukoneen äänikonfiguraatio noudattaa standardia General MIDI -lyömäsoitinmääritystä, jossa tamburiini on asetettu nuotille dec 54.

MIDI-viesti rakennetaan statustavusta hex 9A, sekä datatavuista hex 36 (nuotti) ja hex 4B (voimakkuus). Edelleen muistetaan, että viesti on rakennettava noudattamaan little endian -tavujärjestystä. Viestin sisältämä muuttuja alustetaan nollaksi ja viestin tavut kirjoitetaan viestiin komennolla:

```
viesti += (0x9a | (0x36 << 8) | (0x4b << 16));
```

Lopputuloksena saadaan viesti-muuttujan arvoksi 4B369A (dec 4929178), joka on laitteelle lähetettävä MIDI-viesti.

5.6 MIDI-viestin lähettäminen

Edellä muodostettu viesti lähetetään laitteelle MidiOutShortMsg-funktiolla. Funktio tarvitsee parametreiksi vastaanottavan laitteen handle-tunnuksen, sekä itse viestin. Laitteen handle saatiin alustuksessa tehdyn MidiOutOpen-funktion suorittamisen jälkeen luettua muuttujasta.

Viestin lähettäminen laitteelle ei vaadi toimiakseen Callback-funktiota, vaan MidiOutShortMsg-funktio kirjoittaa viestin välittömästi lähetettäväksi laitteelle.

6 SOVELLUKSEN RAKENNE

Tässä luvussa esitetään C#-skriptien lähdekoodi sekä objektiluokkien hierarkia Unityssa.

Unity-projekti koostuu olio-ohjelmoinnin tapojen mukaisesti ohjelmaluokkien muodostamasta rakenteesta. Yksittäiset luokat toteuttavat luokan ohjelmakoodissa määrätyn toiminnallisen kokonaisuuden ja tietoa voidaan siirtää luokkien välillä ominaisuusfunktioiden get; ja set; avulla, tai osoittamalla kysely suoraan luokassa määriteltyyn staattiseen muuttujaan.

Edellisessä luvussa esiteltiin sovelluksen tarpeellisten osa-alueiden toiminta. Tässä luvussa esitellään sovelluksen skriptien C#-lähdekoodi ja sovelluksen rakenne.

6.1 MIDI-määritykset

MIDI in- ja MIDI out -luokat ylläpitävät listaa järjestelmään kytketyistä laitteista. Luokat lähettävät ja vastaanottavat pelkästään nuotin soittamiseen tarvittavaa tietoa. Tiedot järjestelmään kytketyistä laitteista pidetään luokan sisällä ja sillä ei ole vaikutusta muun sovelluksen toimintaan.

Määrityksissä esitellään winmm.dll:n p/invoke-funktiot, joita käytetään yhteyksien avaamiseen ja tiedon siirtoon MIDI-laitteiden ja sovelluksen välillä. Saapuvien viestien käsittelemiseen käytettävä MidiInProc-delegaatti esitellään muuttujien yhteydessä ilman luontia alustamalla se tyhjällä arvolla ”null”. Tällä tavalla delegaatti on näkyvissä skriptin kaikille metodeille, vaikka se luodaan vasta Start()-metodissa.

Skriptin FixedUpdate()-metodi tarkkailee lähetettävän nuotin sisältävän muuttujan arvoa. Positiivinen muuttujan arvo ilmaisee lähetettävän nuotin, jolloin FixedUpdate() kirjoittaa nuotin numeron Moottori-luokan midiKey-muuttujaan.

`MidiIn.cs`

```
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Runtime.InteropServices;

public class MidiIn : MonoBehaviour
{
```

```
//MIDI in eventin rakenne.
private delegate void MidiInProc
(int handle, int msg, int instance, int param1, int param2);

//MIDI in laitteen ominaisuudet.
[StructLayout(LayoutKind.Sequential)]
struct MidiInCaps
{
    public ushort wMid;
    public ushort wPid;
    public uint vDriverVersion;     // MMVERSION
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 32)]
    public string szPname;
    public uint dwSupport;
}

//Multimediarajapinnan käyttämät funktiot.
[DllImport("winmm.dll")]
static extern uint midiInGetNumDevs();

[DllImport("winmm.dll")]
private static extern int midiInGetDevCaps
(UIntPtr uDeviceID, ref MidiInCaps caps, uint cbMidiInCaps);

[DllImport("winmm.dll")]
private static extern int midiInOpen
(ref int handle, int deviceId, MidiInProc proc,
    int instance, int flags);

[DllImport("winmm.dll")]
private static extern int midiInStart(int handle);

[DllImport("winmm.dll")]
private static extern int midiInStop(int handle);

[DllImport("winmm.dll")]
private static extern int midiInClose(int handle);

//Tapa esittää numerotieto selkotekstinä lähdekoodissa.
private const int CALLBACK_FUNCTION = 0x30000;
private const int MIM_DATA = 0x3C3;
private const int MIM_ERROR = 0x3C5;
private const int MIM_LONGDATA = 0x3C4;

//Muuttujat sovelluksen muiden skriptien käyttöön.
public static int midiInNuotti;
//Muuttujat skriptin omaan käyttöön.
public static string[] inNimi = new string[8];
public static int[] inHandle = new int[8];
public static int nytInAuki;
private MidiInProc messageHandler = null;
public Text inInfo;
int inNums;
bool mouseHover = false;
bool inDevMiss = true;

void Start ()
{
    //Saapuvan MIDI-viestin käsittelijämetodin määrittely.
    messageHandler = new MidiInProc(InViesti);
```

```
        //In laitteiden tunnistus MidiInCaps structin avulla.
        MidiInCaps InLaite = new MidiInCaps();
        inNums = (int)midiInGetNumDevs();
        for (int i = 0; i < inNums; i++)
        {
            if (inNums == 0)
            {
                inInfo.color = Color.red;
                inInfo.text="Ei MIDI in laitetta!" +
                    "\nKytke laite!";
                inDevMiss = true;
                break;
            }
            midiInGetDevCaps((UIntPtr)i, ref InLaite,
                (uint)Marshal.SizeOf(InLaite));
            inNimi[i] = InLaite.szPname;
            midiInOpen(ref inHandle[i], i, messageHandler,
                0, CALLBACK_FUNCTION);
            midiInStart(inHandle[i]);
            inDevMiss = false;
        }
        nytInAuki = 0;
        midiInNuotti = 0;
    }

    void FixedUpdate()
    {
        //Positiivinen arvo ilmaisee koskettimistolta saapuneen nuotin.
        if (midiInNuotti > 0)
        {
            Moottori.midiKey = midiInNuotti;
        }
        //Käsittelyn jälkeen järjestelmä palautetaan odottamaan uutta nuottia.
        midiInNuotti = 0;
    }

    void Update ()
    {
        if (inDevMiss == true)
        {
            inInfo.color = Color.red;
            inInfo.text="Ei MIDI in laitetta!" +
                "\nKytke laite!";
            return;
        }
        else
        {
            int mWh = Mathf.RoundToInt
                ((Input.GetAxis("Mouse ScrollWheel")) * 10.0f);
            VaihdaInLaite(mWh);
            string inla = inNimi[nytInAuki] + "\n"
                + inHandle[nytInAuki].ToString() + "\n"
                + (nytInAuki + 1).ToString() + "/"
                + inNums.ToString();
            inInfo.color = Color.green;
            inInfo.text = inla;
        }
    }

    void OnApplicationQuit()
```

```
{
    for (int i = 0; i < inHandle.Length; i++)
    {
        midiInStop(inHandle[i]);
        midiInClose(inHandle[i]);
    }
}

public void mEnter()
{
    mouseHover = true;
}

public void mLeave()
{
    mouseHover = false;
}

void VaihdaInLaite(int mWh)
{
    if (mouseHover == true)
    {
        nytInAuki = nytInAuki + mWh;
        if (nytInAuki > (inNums-1))
        {
            nytInAuki = (inNums-1);
        }
        else
        {
            if (nytInAuki < 0)
            {
                nytInAuki = 0;
            }
        }
    }
}

void InViesti(int handle, int errmsg, int ins, int paraA, int paraB)
{
    switch (errmsg)
    {
        case MIM_DATA:
            //Normaalitilanne
            AvaaViesti(paraA);
            break;
        case MIM_ERROR:
            //Virheen debuggausviesti (mutta näitähän ei tule).
            Debug.Log("Virhe! Koodi: " + errmsg.ToString());
            break;
        case MIM_LONGDATA:
            //Varaus SysEx viestien käsittelyyn.
            Debug.Log("SysEx viesti!");
            break;
        default:
            break;
    }
}

void AvaaViesti(int msg)
{
```

```
        //Vain statustavulla 0x9n saapuneet viestit ovat
        //sovellukselle merkittäviä ja tulkitaan.
        int status = msg & 0xf0; //MIDI-kanavanumero suodatetaan
        if (status == 0x90)
        {
            midiInNuotti = ((msg & 0xff00) >> 8);
        }
    }
}
```

MidiOut-skriptin FixedUpdate()-metodi tarkkailee lähetettävän MIDI-viestin sisältämän muuttujan arvoa. Nollaa suuremmat arvot tulkitaan lähetettäväksi viestiksi ja ne annetaan midiOutShortMsg()-funktion parametriksi aktiivisen MIDI out -laitteen handlen kanssa. Viestin lähettämisen jälkeen arvo palautetaan nollaksi odottamaan seuraavaa lähetettävää viestiä.

MidiOut.cs

```
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Text;
using System.Runtime.InteropServices;

public class MidiOut : MonoBehaviour
{
    //MIDI out laitteen ominaisuudet
    [StructLayout(LayoutKind.Sequential)]
    public struct MidiOutCaps
    {
        public UInt16 wMid;
        public UInt16 wPid;
        public UInt32 vDriverVersion;
        [MarshalAs(UnmanagedType.ByValTStr,
            SizeConst = 32)]
        public String szPname;
        public UInt16 wTechnology;
        public UInt16 wVoices;
        public UInt16 wNotes;
        public UInt16 wChannelMask;
        public UInt32 dwSupport;
    }

    //Multimediarajapinnan käyttämät funktiot.
    [DllImport("winmm.dll")]
    private static extern long mciSendString(string command,
        StringBuilder returnValue, int returnLength, IntPtr winHandle);

    [DllImport("winmm.dll")]
    private static extern int midiOutGetNumDevs();

    [DllImport("winmm.dll")]
    private static extern int midiOutGetDevCaps(Int32 uDeviceID,
        ref MidiOutCaps lpMidiOutCaps, UInt32 cbMidiOutCaps);
```

```
[DllImport("winmm.dll")]
private static extern int midiOutOpen(ref int handle,
    int deviceID, MidiCallBack proc, int instance, int flags);

[DllImport("winmm.dll")]
protected static extern int midiOutShortMsg(int handle,
    int message);

[DllImport("winmm.dll")]
protected static extern int midiOutClose(int handle);

private delegate void MidiCallBack(int handle, int msg,
    int instance, int param1,
    int param2);

//Muuttujat sovelluksen muiden skriptien käyttöön.
public static int midiOutViesti;
//Muuttujat skriptin omaan käyttöön.
static string[] outNimi = new string[8];
static int[] outHandle = new int[8];
static int nytOutAuki;
public Text outInfo;
int outNum;
bool mouseHover;

void Start ()
{
    //MIDI out laitteiden tunnistus MidiOutCaps structin avulla.
    MidiOutCaps OutCaps = new MidiOutCaps();
    outNum = midiOutGetNumDevs();
    for (int i = 0; i < outNum; i++)
    {
        midiOutGetDevCaps(i, ref OutCaps,
            (uint)Marshal.SizeOf(OutCaps));
        outNimi[i] = OutCaps.szPname;
        midiOutOpen(ref outHandle[i], i, null, 0, 0);
    }
    if (outNimi.Length == 0)
    {
        outInfo.color = Color.red;
        outInfo.text = ("Ei MIDI out laitetta!" +
            "\nTarkista järjestelmän" +
            "\najureiden asennus!");
    }
    nytOutAuki = 0;
    mouseHover = false;
}

void FixedUpdate()
{
    if (midiOutViesti > 0)
    {
        midiOutShortMsg(outHandle[nytOutAuki], midiOutViesti);
        midiOutViesti = 0;
    }
}

void Update ()
{
    int mWh = Mathf.RoundToInt
```

```
            ((Input.GetAxis("Mouse ScrollWheel")) * 10.0f);
        VaihdaOutLaite(mWh);
        string outla = outNimi[nytOutAuki] + "\n"
                       + outHandle[nytOutAuki].ToString() + "\n"
                       + (nytOutAuki + 1).ToString() + "/"
                       + outNum.ToString();
        outInfo.color = Color.green;
        outInfo.text = outla;
    }

    void OnApplicationQuit()
    {
        for (int i = 0; i < outHandle.Length; i++)
        {
            midiOutClose(outHandle[i]);
        }
    }

    public void mEnter()
    {
        mouseHover = true;
    }

    public void mLeave()
    {
        mouseHover = false;
    }

    void VaihdaOutLaite(int mWh)
    {
        if (mouseHover == true)
        {
            nytOutAuki = nytOutAuki + mWh;
            if (nytOutAuki > (outNum-1))
            {
                nytOutAuki = outNum-1;
            }
            else
            {
                if (nytOutAuki < 0)
                {
                    nytOutAuki = 0;
                }
            }
        }
    }
}
```

MIDI-määrityksistä vastaavat skriptit asetetaan sovelluksen käyttöliittymän InLaite- ja OutLaite-objektien komponenteiksi. Objekteihin liittyviin tekstikenttiin liitetään Unityn Event trigger- tapahtumankäsittelijät, joiden avulla sovelluksen käyttämää MIDI-laitetta voidaan vaihtaa käyttämällä hiiren vieritysrullaa. Event trigger -käsittelijät tunnistavat hiiren osoittimen saapumisen tekstikentän alueelle ja osoittimen poistumisen alueelta. Tapahtumiin liitetään skriptin mEnter()- ja mLeave()-metodit, jotka päivittävät

tapahtumaa seuraavia mouseHover totuusarvoja. Muuttujan totuusarvon ollessa ”true”, hiiren rullaa vierittämällä voidaan muuttaa tekstikenttään liitetyn muuttujan lukuarvoa ja skriptin MIDI-viestien käsittelyssä käytettävää aktiivista handlea.

6.2 Ajastimien määrittäminen

Sovelluksen käyttämät ajastimet kuvaavat soittimen alas painetun koskettimen toimintaa. Start()-metodissa käyttöliittymän liukusäätimeen lisätään Unityn tapahtumakuuntelija, joka suorittaa skriptin LaskeAika()-metodin aina säätimen arvon muuttuessa. LaskeAika()-metodi laskee ajan laskennassa seurattavan aikarajan millisekunteina. Laskettu aika kirjoitetaan Moottori-luokan stepAika-muuttujaan.

MIDI-määritysskriptien tapaan tekstikenttiin lisätään tapahtumakäsittelijät tunnistamaan tekstikentän päällä oleva hiiren osoitin ja mahdollistamaan lukuarvon muutokset hiiren vieritysrullaa käyttämällä.

Tempo.cs

```
using UnityEngine;
using UnityEngine.UI;

public class Tempo : MonoBehaviour
{
    //Muuttujat sovelluksen muiden skriptien käyttöön.
    public static int stepAikaraja;
    public static int tempo;
    //Muuttujat skriptin omaan käyttöön.
    public Text tempoData;
    public Slider tempoSet;
    bool mouseHover;

    void Start ()
    {
        tempo = (int)tempoSet.value;
        LaskeAika(tempo, 0);
        //Säätimen arvon seuraaminen käyttää Unityn event- järjestelmää.
        tempoSet.onValueChanged.AddListener
            (delegate
                {
                    LaskeAika((int)tempoSet.value, 0);
                }
            );
    }

    //Ylikuormitetun metodin sijaan käytetään käyttötarkoitukseen
    //sovitettua parametrijärjestelmää.
    void LaskeAika(int tempo, int mWh)
    {
        if (mouseHover == true)
```

```
        {
            tempo += mWh;
            if (tempo > tempoSet.maxValue)
            {
                tempo = (int)tempoSet.maxValue;
            }
            else
            {
                if (tempo < tempoSet.minValue)
                {
                    tempo = (int)tempoSet.minValue;
                }
            }
        }
        tempoSet.value = tempo;
        tempoData.text = tempo.ToString();
        stepAikaraja = 60000 / tempo;
        Moottori.stepAika = stepAikaraja;
    }

    public void mEnter()
    {
        mouseHover = true;
    }

    public void mLeave()
    {
        mouseHover = false;
    }

    void Update ()
    {
        int mWh = Mathf.RoundToInt
            ((Input.GetAxis("Mouse ScrollWheel")) * 10.0f);
        tempo = (int)tempoSet.value;
        LaskeAika(tempo, mWh);
    }
}
```

Askeleen pulssinleveyssuhteen laskeva skripti toimii samalla periaatteella Tempo-skriptin kanssa. Pwm-aikaraja lasketaan prosenttiosuutena askeleen kokonaisaikarajasta.

PulseWidth.cs

```
using UnityEngine;
using UnityEngine.UI;

public class PulseWidth : MonoBehaviour
{
    //Muuttujat sovelluksen muiden skriptien käyttöön.
    public static int pulseAikaraja;
    //Muuttujat skriptin omaan käyttöön.
    public Text pwmData;
    public Slider pulseSet;
```

```
    int pwm;
    bool mouseHover;

    void Start ()
    {
        pwm = (int)pulseSet.value;
        LaskeAika(pwm, 0);
        pulseSet.onValueChanged.AddListener
        (delegate
            {
                LaskeAika((int)pulseSet.value, 0);
            }
        );
    }

    void LaskeAika(int pwm, int mWh)
    {
        if (mouseHover == true)
        {
            pwm += mWh;
            if (pwm > pulseSet.maxValue)
            {
                pwm = (int)pulseSet.maxValue;
            }
            else
            {
                if (pwm < pulseSet.minValue)
                {
                    pwm = (int)pulseSet.minValue;
                }
            }
        }
        pulseSet.value = pwm;
        pwmData.text = (pwm.ToString() + "%");
        pulseAikaraja = (int)((60000 * pwm) / (Tempo.tempo * 100.0f));
        Moottori.pwmAika = pulseAikaraja;
    }

    public void mEnter()
    {
        mouseHover = true;
    }

    public void mLeave()
    {
        mouseHover = false;
    }

    void Update ()
    {
        int mWh = Mathf.RoundToInt
            ((Input.GetAxis("Mouse ScrollWheel")) * 10.0f);
        pwm = (int)pulseSet.value;
        LaskeAika(pwm, mWh);
    }
}
```

6.3 Nuotin määrittäminen

Sävelkuvion jokaiselle yksittäiselle nuotille on sovelluksessa määritelty oma liukusäädin. Objekti muodostuu valintapainikkeesta, jolla askel voidaan asettaa aktiiviseksi hiiren painalluksella, ja joka ilmaisee aktiivisen askeleen vihreällä värillä. Valintapainikkeen teksti asetetaan sijoittamalla käyttöliittymän asettelun yhteydessä Unityn inspector -asetuslehdellä määrätty julkisen askel-muuttujan arvo painikkeen tekstiominaisuudeksi.

Skriptissä määritetty tapahtumakäsittelijä seuraa säätimen arvoa ja suorittaa ArvoMuuttunut()-metodin, kun säädin saa uuden arvon. Metodi kirjoittaa säätimen arvon Moottori-luokan ylläpitämään nuotit[ ]-taulukkoon.

Objektissa on kaksi tekstikenttää näyttämässä askeleen säädettyä MIDI-nuottia ja sävelkuvion toimintaa ohjaavaa override-arvoa. Ylempi kenttä näyttää nuotin nimen ja oktaavin. Alempi kenttä näyttää askeleen override-arvon. Kenttien arvoa muutetaan vierittämällä hiiren rullaa. Tekstikenttien sisältö määritellään C#-kielen Dictionary-luokan avulla, jolla voidaan määrittää esimerkiksi lukuarvoja vastaavia merkkijonoja.

Muokattavan kentän valinta tapahtuu välittämällä kenttien tapahtumaseuraajien toimesta valintametodille parametriksi muokattavaa kenttää vastaava arvo.

Nuotin nimeä vastaava lukuarvo x_n lasketaan MIDI-nuotin järjestysnumerosta n_m modulaariaritmetiikan avulla jäännösluokassa mod 12.

$$x_n \equiv n_m \ (mod\ 12)$$

Kaava 4. Nuotin kongruenssi jäännösluokassa.

Oktaavin ilmaiseva kertaluku lasketaan lattiafunktion avulla kaavalla:

$$f(x) = \lfloor x/12 \rfloor$$

Kaava 5. Oktaavin lattiafunktio.

Suunnittelussa sävelaskeleen loogiselle ohjaukselle määrättiin neljä ominaisuutta.

- Next (nxt). Nuotti soitetaan normaalisti painamalla ja vapauttamalla tempo- ja PWM-aikarajojen määräämässä tahdissa.

- Connect (con). Nuottia ei vapauteta PWM-aikarajan täyttyessä, vaan nuotin soittaminen jatkuu, kunnes seuraavan nuotin soittaminen on aloitettu.
- Skip (skp). Askeleen nuotti jätetään soittamatta ja askel toimii sävelkuviossa taukona.
- Reset (rst). Sävelkuvion loppumerkki, jolla kuvio pakotetaan välittömästi palaamaan takaisin ensimmäiseen askeleeseen ja aloittamaan alusta.

Askeleen nuotti- ja override-arvot kirjoitetaan Moottori-luokan vastaaviin taulukkomuuttujiin.

StepControl.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;

public class StepControl : MonoBehaviour
{
    //Muuttujat sovelluksen skriptien väliseen liikenteeseen.
    public int overRArvo;       // -> Moottori.overit[]
    public int midiNuotti = 60; // -> Moottori.nuotit[]
    //Muuttujat skriptin omaan käyttöön.
    public int askel;
    public Slider potikka;
    int potikanArvo;
    public Text nuotti;
    public Text oRide;
    public bool mHoverNote;
    public bool mHoverOver;
    public Button asetaStep;

    Dictionary<int, string> MidiToAscii = new Dictionary<int, string>();
    Dictionary<int, string> overR = new Dictionary<int, string>();

    void Start ()
    {
        //Numeroita vastaavat nuottien nimet.
        MidiToAscii.Add(0, "C"); MidiToAscii.Add(1, "C#");
        MidiToAscii.Add(2, "D"); MidiToAscii.Add(3, "D#");
        MidiToAscii.Add(4, "E"); MidiToAscii.Add(5, "F");
        MidiToAscii.Add(6, "F#"); MidiToAscii.Add(7, "G");
        MidiToAscii.Add(8, "G#"); MidiToAscii.Add(9, "A");
        MidiToAscii.Add(10, "A#"); MidiToAscii.Add(11, "B");
        //Override tilat.
        overR.Add(0,"nxt"); overR.Add(1,"con");
        overR.Add(2,"skp"); overR.Add(3,"rst");
        //Valintapainikkeen askel- lukuarvo.
        asetaStep.GetComponentInChildren<Text>().text = askel.ToString();
        potikka.value = midiNuotti;
        Moottori.nuotit[askel -1] = midiNuotti;
        Moottori.overit[askel -1] = overRArvo;
        //Säätimen arvoa seuraava tapahtumakäsittelijä
        potikka.onValueChanged.AddListener(delegate
            {
```

```
            ArvoMuuttunut();
        });
}

void OnGUI()
{
    //Käyttöliittymän piirtämiseen tarkoitettu metodi
    //vaihtaa aktiivisen askeleen valintapainikkeen
    //värin vihreäksi.
    if (Moottori.aktiivinenStep == askel)
    {
        ColorBlock merkkivalo = asetaStep.colors;
        merkkivalo.normalColor = new Color32(0, 255, 0, 255);
        asetaStep.colors = merkkivalo;
    }
    else
    {
        ColorBlock merkkivalo = asetaStep.colors;
        merkkivalo.normalColor = new Color32(255, 0, 0, 255);
        asetaStep.colors = merkkivalo;
    }
}

void Update ()
{
    int mMod = Mathf.RoundToInt
        ((Input.GetAxis("Mouse ScrollWheel")) * 10.0f);
    potikanArvo = (int)potikka.value;
    Moottori.nuotit[askel-1] = (int)potikka.value;
    nuotti.text = MtA(potikanArvo, mMod);
    Moottori.overit[askel-1] = overRArvo;
    oRide.text = AdjustOverride(overRArvo, mMod);
}

public void mEnter(int para)
{
    switch (para)
    {
        case 1:
            mHoverNote = true;
            break;
        case 2:
            mHoverOver = true;
            break;
        default:
            break;
    }
}

public void mLeave(int para)
{
    switch (para)
    {
        case 1:
            mHoverNote = false;
            break;
        case 2:
            mHoverOver = false;
            break;
        default:
```

```
            break;
        }
    }

    public string MtA(int midiN, int mMod)
    {
        if (mHoverNote == true)
        {
            midiN = midiN + mMod;
            if (midiN < 0)
            {
                midiN = 0;
            }
            else
            {
                if (midiN > 127)
                {
                    midiN = 127;
                }
            }
        }
        potikka.value = midiN;
        int nuotti, oktaavi;
        string nNimi = "";
        //Nuotin nimi lasketaan nuotin numerosta modulaariaritmetiikan
        //avulla jäännösluokassa mod 12. Oktaavin numero määräytyy
        //jakolaskun perusteella lattiafunktiolla.
        nuotti=(int)(midiN % 12);
        oktaavi = Mathf.FloorToInt(midiN / 12);
        nNimi = nNimi + MidiToAscii[nuotti] + oktaavi.ToString();
        midiNuotti = midiN;
        return nNimi;
    }

    public string AdjustOverride(int arvo, int mMod)
    {
        if (mHoverOver == true)
        {
            arvo = arvo + mMod;
            if (arvo < 0)
            {
                arvo = 3;
            }
            else
            {
                if (arvo > 3)
                {
                    arvo = 0;
                }
            }
        }
        overRArvo = arvo;
        string ortxt = overR[arvo];
        return ortxt;
    }

    public void AsetaAktiivinen()
    {
        Moottori.aktiivinenStep = askel;
    }
```

```
    void ArvoMuuttunut()
    {
        Moottori.nuotit[askel-1] = (int)potikka.value;
    }
}
```

6.4 Sekvenssin toimintalogiikka

Skripteistä kerätyt tiedot käsitellään Moottori-luokassa. FixedUpdate()-metodi on Unityssa pääasiassa fysiikan laskemiseen tarkoitettu metodi. Muista metodeista poiketen se suoritetaan useammin kuin muut metodit ja tiheämmän suoritustaajuutensa ansiosta se soveltuu erinomaisesti sovelluksen ajastimen seurantaan.

FixedUpdate()-metodin suoritustaajuutta säädetään Unityn projektiasetusten Time Managerin Fixed Timestep -arvoa säätämällä. Oletusarvona Fixed Timestep oli opinnäytetyöprojektissa säädetty tapahtumaan 0,02 sekunnin välein, mutta arvon voi huoletta nostaa, tai ajankäytön tapauksessa, laskea tapahtuvaksi ainakin 200–500 kertaa sekunnissa. Sovelluksen kaltaisessa käytössä Fixed Timestepin liika kuormittaminen ei aiheuta haittaa ajastimen toiminnalle, mutta esimerkiksi pelifysiikan laskennassa tasaisin väliajoin tapahtuvaksi laskennaksi suunnitellun ajoituksen viiveet saattavat aiheuttaa ennustamatonta toimintaa.

Sovelluksessa seurataan järjestelmädiagnostiikan stopwatch-ajastinta, josta voidaan lukea ajastimen käydessä ElapsedMilliseconds-arvoa ja verrata sitä StepControl-luokassa laskettuihin raja-arvoihin.

Aikarajan täyttyessä suoritetaan askeleen vaiheen perusteella nuotin soitto-, nuotin vapautus-, tai askeleen vaihtometodi. Näissä metodeissa oikea toiminto valitaan askeleen override-arvon perusteella Switch-rakenteen avulla.

Switch-lauseissa rakennetaan tapauskohtaisesti kokonaislukumuuttujaan arvo, joka lähetetään MidiOut-luokan midiOutViesti-muuttujan arvoksi. MidiOut-luokka lähettää muuttujan arvon ja aktiivisen MIDI-laitteen handlen parametrina midiOutShortMsg()-funktiolle ja laiteajuri lähettää MIDI-viestin ulkoiselle laitteelle.

Sekvensserin käynnin ohjaus tapahtuu käyttöliittymän ”Play/pause” -painikkeella, johon liitetty tapahtumaseuraaja suorittaa painalluksen tapahtuessa Moottori-luokan PressPlay()-metodin, joka pysäyttää tai käynnistää ajanottokellon.

Moottori.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Diagnostics;

public class Moottori : MonoBehaviour
{
    //Muuttujat sovelluksen skriptien väliseen liikenteeseen.
    //Tänne muut skriptit kirjoittavat tietonsa.
    public static int midiKey;                  //MidiIn
    int viesti;                                 //MidiOut
    public static int stepAika;                 //Tempo
    public static int pwmAika;                  //PulseWidth
    public static int aktiivinenStep;           //StepControl
    public static int[] nuotit = new int[36];   //StepControl
    public static int[] overit = new int[36];   //StepControl
    //Muuttujat skriptin omaan käyttöön.
    int nytSoi = 1;
    public bool running;
    Button toista;
    Stopwatch sKello = null;
    int nxtAction; //Estää envelope spammauksen.

    void Start ()
    {
        running = false;
        aktiivinenStep = 1;
        sKello = new Stopwatch();
    }

    void FixedUpdate()
    {
        //Jos MIDI-koskettimiston kosketinta painetaan
        if ((midiKey > 0) && (running == false))
        {
            sKello.Reset();
            LaskeNuotit(midiKey);
            sKello.Start();
            running = true;
            midiKey = 0;
        }
        if ((midiKey > 0) && (running == true))
        {
            LaskeNuotit(midiKey);
        }
        //Kun aikaraja saavutetaan.
        if ((sKello.ElapsedMilliseconds >= pwmAika) && nxtAction == 1)
        {
            EnvelopeRelease();
        }
        if ((sKello.ElapsedMilliseconds >= stepAika) && nxtAction == 2)
        {
            EnvelopeEnd();
```

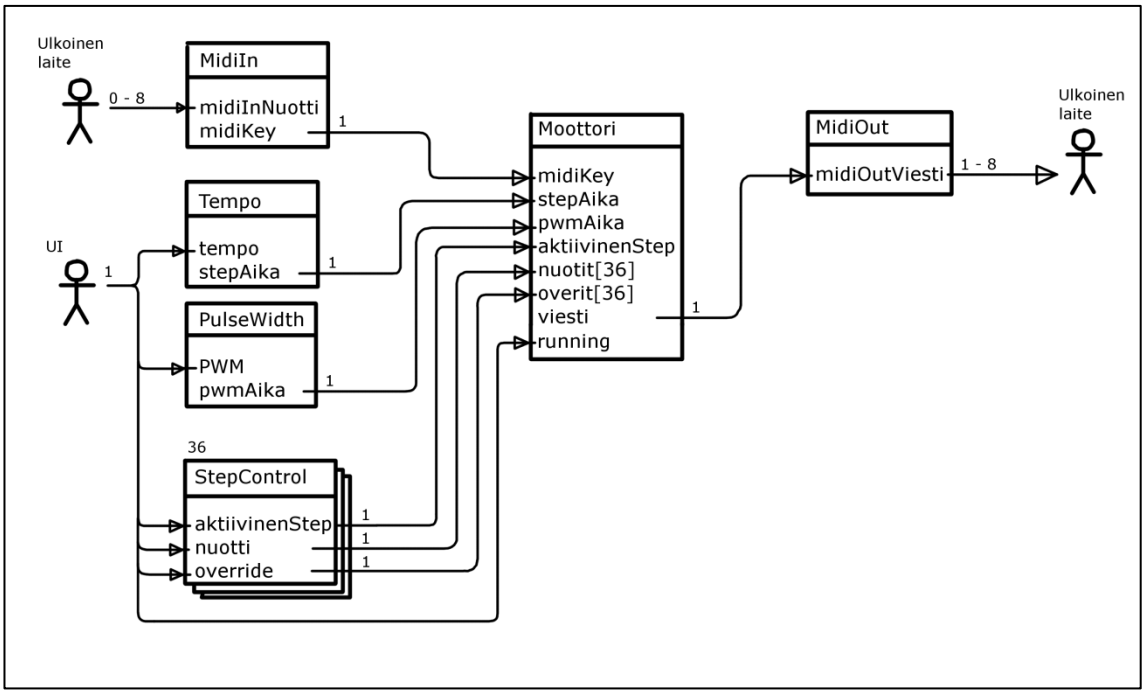
```
            sKello.Reset();
            sKello.Start();
        }
    }

    public void PressPlay()
    {
        if (running == true)
        {
            running = false;
            sKello.Stop();
        }
        else
        {
            if ((running == false) && (nxtAction == 0))
            {
                running = true;
                sKello.Start();
                nxtAction = 1;
                EnvelopeAttack(nuotit[aktiivinenStep-1]);
            }
            else
            {
                running = true;
                sKello.Start();
                nxtAction = 1;
            }
        }
    }

    //Laske nuottilista säätimien ja Midi in nuotin avulla
    void LaskeNuotit(int Key)
    {
        for (int i = 0; i < 36; i++)
        {
            nuotit[i] = nuotit[i] + Key-60;
            while (nuotit[i] > 127)
            {
                nuotit[i] = nuotit[i]-12;
            }
            while (nuotit[i] < 0)
            {
                nuotit[i] = nuotit[i] + 12;
            }
        }
    }

    //Askeleen alku.
    void EnvelopeAttack(int nuotti)
    {
        viesti = 0;
        switch (overit[aktiivinenStep-1])
        {
            case 0: //nxt
                if (nytSoi > 0) //Edellinen con
                {
                    if (nytSoi == nuotit[aktiivinenStep-1])
                    {
                        break;
                    }
```

```
            else
            {
                viesti += (0x90 | (nytSoi << 8));
                MidiOut.midiOutViesti = viesti;
                viesti = 0;
                viesti += (0x90 |
                ((nuotit[aktiivinenStep-1]) << 8) | 0x7f0000);
                MidiOut.midiOutViesti = viesti;
                nytSoi = nuotit[aktiivinenStep-1];
            }
        }
        else
        {
            viesti += (0x90 |
                ((nuotit[aktiivinenStep-1] << 8) | 0x7f0000));
            MidiOut.midiOutViesti = viesti;
            nytSoi = nuotit[aktiivinenStep-1];
        }
        break;
    case 1: //con
        if (nytSoi > 0) //Edellinen con
        {
            if (nytSoi == nuotit[aktiivinenStep-1])
            {
                break;
            }
            else
            {
                viesti += (0x90 | (nytSoi << 8));
                MidiOut.midiOutViesti = viesti;
                viesti = 0;
                viesti += (0x90 |
                ((nuotit[aktiivinenStep-1]) << 8) | 0x7f0000);
                MidiOut.midiOutViesti = viesti;
                nytSoi = nuotit[aktiivinenStep-1];
            }
        }
        else
        {
            viesti += (0x90 |
                ((nuotit[aktiivinenStep-1]) << 8) | 0x7f0000);
            MidiOut.midiOutViesti = viesti;
            nytSoi = nuotit[aktiivinenStep-1];
        }
        break;
    case 2: //skp
        if (nytSoi > 0) //Edellinen con
        {
            viesti += (0x90 | (nytSoi << 8));
            MidiOut.midiOutViesti = viesti;
            nytSoi = 0;
        }
        break;
    case 3: //rst
        if (nytSoi > 0)
        {
            viesti += (0x90 | (nytSoi << 8));
            MidiOut.midiOutViesti = viesti;
            nytSoi = 0;
        }
```

```
            break;
        default:
            break;
    }
    if (overit[aktiivinenStep-1] == 3)
    {
        //Asetettu rst pakottaa kuvion alkamaan alusta heti
        EnvelopeEnd();
        sKello.Reset();
        sKello.Start();
    }
}

//PwmAikaraja saavutettu. Askeleen keskiosa.
void EnvelopeRelease()
{
    nxtAction = 2;
    viesti = 0;
    switch (overit[aktiivinenStep-1])
    {
        case 0: //nxt
            viesti += (0x90 | (nytSoi << 8));
            MidiOut.midiOutViesti = viesti;
            nytSoi = 0;
            break;
        case 1: //con
            break;
        case 2: //skp
            break;
        case 3: //rst
            //Tänne ei pitäisi joutua.
            break;
        default:
            break;
    }
}

//StepAikaraja saavutettu. Askeleen loppu.
void EnvelopeEnd()
{
    nxtAction = 1;
    //Tehdään päätös seuraavasta askeleesta. Nuotit ovat joko
    //sammutettuja tai jatkuvat askeleen rajan yli.
    switch (overit[aktiivinenStep-1])
    {
        case 0: //Overridetapaukset nxt, con ja skp.
        case 1: //Sävelkuvion toisto jatkuu normaalisti
        case 2: //seuraavaan askeleeseen.
            {
                aktiivinenStep++;
                if (aktiivinenStep > 36)
                {
                    aktiivinenStep = 1;
                }
                break;
            }
        case 3: //Overridetapaus rst. Viimeinen askel saavutettu.
            aktiivinenStep = 1;
            break;
        default:
```

```
                break;
            }
            EnvelopeAttack(nuotit[aktiivinenStep-1]);
        }

        void Update ()
        {
            //Testausvihje:
            //UnityEngine ja System.Diagnostics sisältävät molemmat
            //Debug- luokan. Käskyssä on eriteltävä, kumpaa
            //nimiavaruutta käytetään.
            //UnityEngine.Debug.Log(aktiivinenStep.ToString());
        }
}
```

Moottori-luokan tehtävänä on suorittaa muiden luokkien tuottaman informaation perusteella määrätty looginen toiminto säädetyllä ajan hetkellä. Kuviossa 7 esitettävä informaation kulku on suunniteltu välttämään turhaa edestakaista liikennettä luokkien välillä.



Kuvio 7. Sovelluksen tiedonkulku ja käyttötapauskaavio.

6.5 Sovitus Unityyn

Edellä esitettiin sovelluksen luokkien sisältö ja toiminta. Tässä luvussa esitetään luokkien paikat Unityn rakenteessa. Objektit, joiden komponentteja ohjelmaluokat ovat, muodostavat Unityssa hierarkkisen puurakenteen.

Hierarkkisuus tarkoittaa sitä, että objektin tietyt ominaisuudet periytyvät ketjussa seuraaville objekteille. Käyttöliittymässä tämä tarkoittaa sitä, että ylimmällä tasolla olevan Canvasin Transform -informaatio (sijainti, kierto ja mittakaava) toimivat lähtöpisteenä. Alemmalla tasolla käyttöliittymän säätimen sijaintitieto ei ole säätimen absoluuttinen sijainti Canvas-tasolla, vaan suhteellinen sijainti Canvas-tason nollapisteeseen.

Sovelluksessa hierarkian ylimmällä tasolla rinnakkain ovat käyttöliittymän Canvas ja Unityn Event System. Event System on tapahtumatarkkailija, jonka tehtävänä on ilmoittaa, milloin sovelluksessa tapahtuu jotakin merkittävää. Sovelluksessa tapahtumatarkkailijaa käytetään tunnistamaan hiiren osoittimen osuminen määrättyjen käyttöliittymän kohteiden päälle.

Tapahtumatarkkailija toimii koko sovelluksen laajuisesti, joten tarvitaan vain yksi tapahtumatarkkailija.

Jotta kohde voidaan tunnistaa, se pitää määritellä kohteeksi merkitsemällä kohteen Raycast Target valintaruutu, sekä asettamalla painikkeiden ja säätimien Interactable-arvo. Tunnistettavaan kohteeseen liitetään Event Trigger -skripti, jonka työkaluvalikoimasta sovelluksessa valitaan Pointer Enter- ja Pointer Exit -tapahtumat.

Komponentit tarvitsevat tiedon siitä, mihin sovelluksen objektiin niiden toiminta liittyy. Objektiviittaus voidaan tehdä skriptissä ohjelmallisesti, mutta Unity tarjoaa objektiviittausten tekemiseen myös kehitysympäristön graafisessa käyttöliittymässä toimivan tavan.

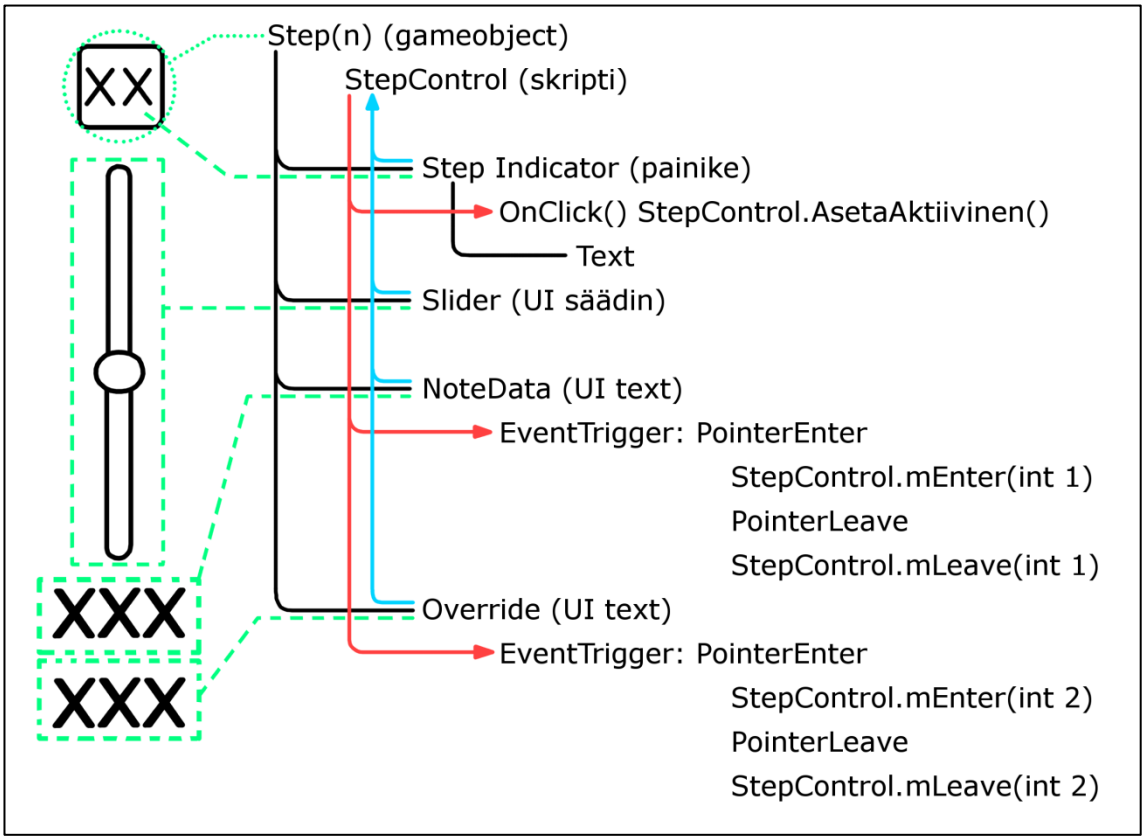
Esimerkkinä käytettävässä Event Triggerissä viittausta varten on objektikenttä, johon hierarkiavälilehdeltä vedetään objekti, jossa Event Triggerin tuottama informaatio käsitellään.

Ohjelmallisessa ratkaisussa skriptiin luodaan muuttuja, joka vastaanottaa ulkoiselta komponentilta skriptissä käsiteltävän informaation. C#-kielessä muuttuja voidaan esittelyn yhteydessä varustaa näkyvyysmäärittelyllä. Oletusarvoltaan ilman näkyvyysmäärettä esitelty muuttuja, luokka tai metodi on näkyvyydeltään private-tyyppinen, jolloin sitä voidaan havainnoida ja käsitellä vain siinä lohkossa, jossa se on luotu. Lohkot ovat C#-lähdekoodissa aaltosulkeiden { } rajaamia alueita.

Muuttujat, joiden on tarkoitus käsitellä ulkoisten objektien tuottamaa informaatiota, määritellään näkyvyysmääreellä public, jolloin Inspectorissa näkyvän skriptin

kuvaukseen ilmestyy muuttujan nimellä otsikoitu tyhjä kenttä. Kentän oikealla puolella näkyvästä painikkeesta avautuvassa valintaikkunassa näkyvät kaikki objektit jotka kelpaavat kenttään objektiviittaukseksi. Objektiviittaus voidaan vetää kenttään myös Hierarchy-lehdeltä.

Kuviossa 8 esitetään sovelluksen sekvensserin yhden askeleen rakenne, jossa näkyy erilliset komponentit ja niiden isäntähierarkia sekä komponenttien välisten objektiviittausten suunta nuolin esitettynä.



Kuvio 8. Objektin isännyys- ja viittaussuhteet.

Kuviossa musta viiva esittää isännyyshierarkian rakenteen objektissa. Tapahtumatarkkailijoiden tarvitsemat viittaukset skriptiin kuvataan punaisella ja skriptin muuttujien tarvitsemat viittaukset informaation tuottaviin objekteihin kuvataan sinisellä.

Sovelluksessa skriptien on lähetettävä tietoa muiden skriptien käytettäväksi. Helpoin tapa suorittaa tämä on luoda kohteeseen public static -tyyppinen muuttuja. Luokan ja metodin, jossa muuttuja esitellään, on myös oltava näkyvyysmäärittelyltään public.

Muuttujan arvo kirjoitetaan kohteeseen komennolla:

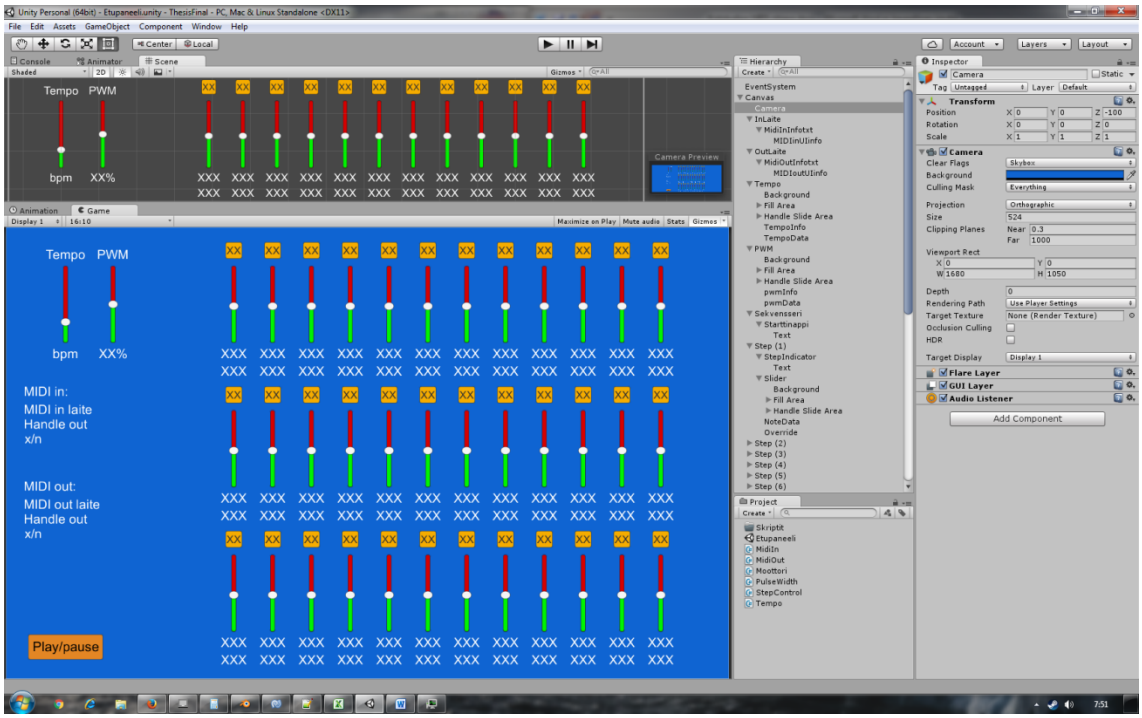
```
Kohdeluokka.kohdemuuttuja = lähdemuuttujaTaiLukuarvo;
```

Sovelluksessa esiintyy objektiviittauksen monimutkaisempi versio, missä skripti tarvitsee pääsyn objektin useampaan komponenttiin. Unityssä käyttöliittymän painikkeen teksti on erillinen objekti, joka on painikkeen alainen hierarkiassa. Step Control -skripti käyttää jo painikkeen värіominaisuuksia ilmaisemaan askeleen aktiivisuutta, joten painikkeeseen on luotu objektiviittaus.

Objektin alaisen komponentin ominaisuuksiin päästään skriptissä kiinni komennolla:

```
viittaus.GetComponentInChildren<Ominaisuus>().ominasuudenNimi = arvo;
```

Kun sovelluksen kaikki osat on määritetty, osat asetellaan Canvas-tasolle haluttuun järjestykseen, kuten kuvassa 3 esitetään, ja tehdään skriptien vaatimat objektiviittaukset.



Kuva 3. Kuvaruutukaappaus valmiista projektista.

Unity valvoo sovelluksen toimintaa ja näyttää virheilmoitukset viestikonsolissa. Skriptien kirjoittaminen tehtiin ulkoisessa Monodevelop-kehitysympäristössä. Skriptin

muokkaamisen jälkeen muutokset on tallennettava Monodevelopissa, jotta ne tulisivat voimaan Unityssa. Tallentamisen jälkeen Unity kääntää lähdekoodin automaattisesti. Tämä kestää tämän opinnäytetyön kokoisen sovelluksen tapauksessa muutamia sekunteja.

Onnistuneen kääntämisen jälkeen konsolissa ei pitäisi olla kriittisiä punaisella merkittyjä virheilmoituksia. Keltaisella merkityt ilmoitukset koskevat esimerkiksi käyttämättömiä muuttujia ja muita huomiota vaativia vähemmän kriittisiä asioita, eivätkä nämä estä sovelluksen käynnistämistä.

Valmiista projektista rakennetaan suoritettava exe-tiedosto File–Build Settings-valikosta asettamalla vaadittavat parametrit.

Tämä opinnäytetyö käyttää multimediakirjastoja, jotka toimivat vain Windows-käyttöjärjestelmässä. Valmis projekti käännettiin windowsin exe-tiedostoksi. Opinnäytetyö on kokonaisuudessaan laadittu ja testattu toimivaksi 64b Windows 7 professional N Service Pack 1 versiolla 2,66 GHz Intel Core2 Quad Q9400 prosessorilla 8 Gt:n muistilla ja nVidia GeForce GTX 660 -näytönohjaimella. Lisäksi käännetty exe-tiedosto testattiin HP ProBook 6560b:ssa 64 b Windows 7 professional Service Pack 1 -versiolla 2,3 GHz Intel Core i5-2410M -prosessorilla 8 Gt:n muistilla.

Testauksessa käytettiin Rolandin valmistamaa Edirol UM-1 EX USB–MIDI-sovitinta ja ohjattavana laitteena käytettiin analogista Korg MS-20 mini -syntetisaattoria. MIDI-koskettimistona käytettiin Roland SH-01 Gaia -syntetisaattoria.

Testauksessa todettiin sovelluksen toimivan suunnitellulla tavalla. Lisäksi havaittiin, että käytettäessä äänilähteenä ulkoista fyysistä syntetisaattoria, koskettimen painallusta seuraava viive ennen nuotin soimista lyheni lähes huomaamattomaksi. On ilmeistä, että tietokoneen äänipiirin MIDI-syntetisaattorin äänenmuodostuksessa on lukuisista ohjelmistorajapinnoista johtuvaa viivettä, joka puuttuu varta vasten studiokäyttöön suunnitelluista soittimista, jotka reagoivat MIDI-viesteihin välittömästi.

Tietokonekäyttöön on suunniteltu ohjelmallisia ratkaisuja, jotka ohittavat raskaan multimediarajapinnan ja kontrolloivat äänipiiriä suoraan laitetasolla. Tällainen ratkaisu on esimerkiksi saksalaisen Steinberg GmbH:n kehittämä ASIO (Audio Stream Input / Output).

7 LOPPUTULOKSEN ARVIOINTI

Opinnäytetyön toteutuksessa havaittiin, että ohjelmasovelluksen kehittäminen MIDI-laitteiston hallitsemiseksi on käyttökelpoinen ratkaisu. Windowsin multimediarajapinnan ohjelmointi on mahdollista toteuttaa kaikilla luvussa 3 esitellyillä työkaluilla.

Tämän opinnäytetyön toteutustavaksi valitun menetelmän etuna todettiin olevan Unity-pelimoottorin tarjoama helppokäyttöinen käyttöliittymän suunnittelu ja tehokas tapahtumakäsittelyjärjestelmä.

.NET frameworkin Windows Forms- ja konsolisovellusympäristöissä käytettäviä menetelmiä voidaan käyttää myös Unity-pelimoottorissa.

Opinnäytetyössä käytettiin Platform Invoke -järjestelmäkutsua ohjaamaan winmm.dll -multimediakirjastossa määriteltyjä MIDI-ohjausfunktioita. Winmm.dll pitää sisällään lukuisia muita multimediatoimintoja. Tärkeimpänä voidaan mainita audio input -laitteita käyttävät waveIn-funktiot äänen tallennukseen, audio output -laitteita käyttävät waveOut-funktiot äänen toistoon ja mixer-funktiot äänenvoimakkuuden säätöön.

P/invoke-menetelmä mahdollistaa esimerkiksi näppäimistön tehokkaamman hyödyntämisen kuin :NET framworkin sisällä muuten on mahdollista. Tämä menetelmä mahdollistaa varsinkin PC-peliohjelmoinnissa määrittää pelitoimintoja esimerkiksi näppäimille Ctrl, Shift, Alt ja Caps Lock, joiden käsittely .NET frameworkissa ei ole täysin suoraviivaista.

Koska pelimoottorin käyttäminen kehitysympäristönä osoittautui käyttökelpoiseksi menetelmäksi, kannattaa tutkia myös muiden vastaavien ratkaisujen käyttökelpoisuutta. Unreal 4 -pelimoottori on vuodesta 2014 lähtien käyttänyt skriptien ohjelmointikielenä C++:aa. C++:n etuna on sillä kirjoitettujen ohjelmien parempi suoritusnopeus. Pitkä kehityshistoria vuodesta 1983 lähtien tekee C++:sta hyvin tunnetun ja dokumentoidun ohjelmointikielen.

Sovelluksen jatkokehityksessä kannattaa tutkia menetelmiä MIDI-nuottitiedon tallentamista useille rinnakkaisille muistiraidoille ja tämän tiedon muokkausta. Nuottitiedon tallentamisen lisäksi kannattaa tutkia mahdollisuutta tallentaa sekvensserin ohjaamien syntetisaattorien tuottamaa audiosignaalia äänitiedostoksi

hyvälaatuisella äänilaitteistolla, mikä tekisi sovelluksesta monipuolisen studiosovelluksen.

LÄHTEET

[1] Scavone, G. P. 2016. RtMIDI Tutorial. McGill University. [www-dokumentti]. Saatavilla: https://www.music.mcgill.ca/~gary/rtMIDI/ [Luettu: 29.5.2016].

[2] obiwanjacobi. 2015. MIDI.NET. Canned Bytes. [www-dokumentti]. Saatavilla: https://MIDInet.codeplex.com/ [Luettu 2.6.2016].

[3] Wikipedia. 2015. NET Framework. [www-dokumentti]. Saatavilla: https://fi.wikipedia.org/wiki/.NET_Framework [Luettu: 3.6.2016].

[4] Wikipedia. 2016. MIDI [www-dokumentti]. Saatavilla: https://en.wikipedia.org/wiki/MIDI [Luettu 7.8.2016].

[5] MIDI Manufacturers Association. 2014. The Complete MIDI 1.0 Detailed Specification v.96.1. La Habra CA: MMA. [pdf] Saatavilla: https://www.MIDI.org/specifications [Ladattu: 20.4.2016].

[6] Analog sequencer. Google kuvahaku. [www-dokumentti]. Saatavilla: http://4.bp.blogspot.com [Luettu 9.9.2016].

[7] Microsoft. 2016. MIDI IO status. Saatavilla: https://msdn.microsoft.com/en-us/library/windows/desktop/dd798458(v=vs.85).aspx [Luettu: 6.11.2016].

UNITY-FUNKTIOIDEN SUORITUSJÄRJESTYS