Marko Wiik

MIDI-tiedostojenlukija sähkörumpu-peliin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

31.3.2025

Tiivistelmä

Tekijä:	Marko Wiik
Otsikko:	MIDI-tiedostojenlukija sähkörumpupeliin
Sivumäärä:	36 sivua
Aika:	28.4.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaajat:	Lehtori Miikka Mäki-Uuro

Tämän insinöörityön tarkoituksena oli kehittää MIDI-tiedostojen lukija Godot-pelimoottorilla tehtävään sähkörumpupeliin. Tiedostojen lukija avaa MIDI-tiedoston, lukee sen ja tallentaa MIDIn rumpunuotteja sisältävän raidan listaksi, jota peli voi käyttää. MIDI-tiedostojen lukemisessa käytettiin bittioperaatioita tavujen lukemiseen ja heksadesimaalisessa muodossa olevan tiedon tulkitsemista.

MIDI-tiedostojen lukija saatiin tunnistamaan MIDI-tapahtumat ja lukemaan MIDI-tiedostoja onnistuneesti. MIDIstä tunnistettiin MIDI-otsikko ja MIDI-raidat, joista palautettiin rumpunuotteja sisältävät MIDI-tapahtumat listana ohjelman koodin käsiteltäväksi.

MIDI-tiedostojen lukijaa on tarkoitus kehittää vielä eteenpäin, jotta pelissä voidaan monipuolisemmin käyttää hyödyksi koko MIDI-tiedostoa. Mahdollisena tavoitteena on, että MIDIstä luetaan ja tallennetaan kaikki raidat, joita voidaan haluttaessa käyttää taustamusiikin tuottamiseen pelaamisen aikana.

Avainsanat: MIDI, MIDI-tapahtumat, Godot, bittioperaatiot

Abstract

Author:	Marko Wiik
Title:	MIDI-file reader for an electronic drums -game
Number of Pages:	36 pages
Date:	28th April 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Game Applications
Supervisors:	Miikka Mäki-Uuro, Senior Lecturer

The purpose of this engineer thesis was to create a MIDI-file reader for an electronic drums game. The game is being developed using Godot engine. The file reader opens a MIDI-file, reads it and saves the track which contains the drum notes into a list that the game can use. The MIDI reader uses bitwise operations to read bytes in the MIDI file and interprets data in hexadecimal format.

The MIDI reader was able to recognize MIDI-events and read MIDI-files successfully. MIDI reader recognized the MIDI header and MIDI tracks and returned a list containing the drum events.

The MIDI reader will be developed further, so MIDIs can be used in the game more fully. The MIDI reader will possibly read and save all MIDI tracks, which can be used to produce music during gameplay.

Keywords: MIDI, MIDI-events, Godot, bitwise operations

This thesis has been inspected with Turnitin Originality Check program.

AI has been used during this thesis. OpenAI ChatGPT, GPT-4 Turbo has been used as assistance in creating code, music vocabulary, shaping some pharagraphs and word choices. Another AI program used for shaping pharagraphs and helping with word choices is Mistral AI, Mistral 7B. Writer of this thesis takes full responsibility of the contents and shaping of this work.

Sisällys

Lyhenteet

BPM: *Beats per minute.* Kappaleen nopeus eli tempo, tarkoittaa ‘iskua minuutissa’, eli montako neljäsosatahtia mahtuu minuuttiin.

MIDI: *Musical instrument digital interface*. Kansainvälinen elektroninen musiikinsiirtotietojärjestelmärajapinta.

SMPTE: *Society of Motion Picture and Television Engineers timecode* -standardi. SMPTE:tä käytetään synkronisoimaan ääntä ja kuvaesityksiä, kuten videoita. Projektin MIDI-lukija toimii toistaiseksi MIDIn toisella, TPQN-ajanmittaustavalla.

TPQN: *Ticks per quarter note.* Sykäystä neljäsosanuottia kohden, on toinen ajanmittausvaihtoehto MIDIssä SMTPE:n sijaan.

VLQ: *Variable length quantity.* Vaihtuvan pituinen määrä tarkoittaa, että tietoa sisältäviä arvoja voidaan koota useammasta tavusta yhdistämällä tavujen tietoa sisältävät bitit.

1 Johdanto

Tämän insinöörityön tavoitteena oli luoda Godot-pelimoottoriin MIDI-tiedostojen lukija kehitteillä olevaan sähkörummuilla pelattavaan rytmipeliin. MIDI-tiedostojen lukijan tehtävänä oli lukea musiikkikappaleen MIDI-tiedosto ja kääntää siinä oleva rumpusoitinraita muotoon, jota voidaan käyttää pelissä. MIDI-lukija tunnistaa MIDI-musiikkitiedoston otsikon sisällön ja raitojen tapahtumat. Pelissä musiikkikappaleen rumpunuotit ilmaantuvat ylhäältä ja liikkuvat kohti alareunan maaliviivaa omilla linjoillaan. Peli havaitsee sähkörumpujen rumpuinstrumenttien iskut ja tarkistaa, osuttiinko oikeaan rumpunuottiin sen ollessa maalialueella.

Insinöörityön toisessa luvussa käsitellään lyhyesti MIDI-rajapinnan käsitettä, MIDIn toimintaperiaatetta, MIDIn historiaa ja sitä, miksi MIDI-standardi kehitettiin. MIDI-tiedoston rakennetta käsitellään kolmannessa luvussa. Luvussa esitellään MIDI-otsikon ja MIDI-raitojen sisältämät tiedot. Kolmannessa luvussa kerrotaan myös MIDIn vaihtuvan pituisen tietomäärän lukemisesta, joka perustuu tavun merkitsevimmän bitin hyödyntämiseen. MIDI-raidan tapahtumia on myös esitelty heksadesimaaliarvoineen. Luvussa neljä esitellään esimerkkejä samankaltaisista peleistä, jotka toimivat sähkörumpupelin esimerkkeinä ja jota varten MIDI-lukija kehitettiin. Luvussa annetaan myös lyhyt kuvaus insinöörityöhön liittyvästä rumpupelistä. Viidennessä luvussa esitellään osia MIDI-lukijan koodista ja toiminnallisista ratkaisuista. Lisäksi esitellään lyhyesti muita ohjelmistoja, joita käytettiin MIDI-lukijan kehityksessä.

2 Musical Instrument Digital Interface (MIDI)

MIDI on tunnettu musiikkiin liittyvä nykykäsite. MIDIstä puhuttaessa viitataan yleensä MIDI-musiikkitiedostoon tai tapaan luoda MIDI-musiikkia.

2.1 MIDI lyhyesti

MIDI (Musical instrument digital interface) on musiikki-instrumenttien digitaalinen rajapinta. MIDI on yleisesti käytetty standardi, jonka ansiosta eri laitteet voivat kommunikoida keskenään.

Rajapinta on tapa, jolla eri ohjelmat tai laitteet voivat keskustella keskenään. Toinen ohjelma antaa tietoja tai pyyntöjä rajapinnan määräämällä tavalla. Nämä tiedot ja pyynnöt rajapinta välittää eteenpäin ohjelmalle, joka myös osaa tulkita rajapintaa. Ohjelmistojen tai laitteiden ei tarvitse tietää, miten rajapinta toimii, eivätkä ne itse häiriinny, jos rajapinnan sisäistä toimintaa muutetaan, kunhan rajapinnan välittämät viestit eivät muutu.

Sähkörummut, digitaaliset pianot ja muut MIDI-yhteensopivat laitteet voidaan kytkeä MIDI-yhteydellä MIDIä tukevaan laitteeseen. MIDI-laitteet voidaan liittää toisiinsa eri tavoilla: MIDI-liittimellä, joka on pyöreä ja voi olla joko läpimeno, sisääntulo tai ulosmeno. Kuvassa 1 soittimen oikeassa laidassa kolme suurinta pyöreää liitinporttia ovat MIDIlle.

Kuva 1. Novation 49 SL MK III -soittimen takaosa liitinportteineen [49 SL MK III MIDI-kontrolleri].

Myös USB-B-porttia voi käyttää MIDI-laitteen yhdistämiseen, jos MIDI-laitteessa on USB-B-ulostulo. Kuvassa 2 näkyy USB-A - USB-B-kaapeli, eli niin sanottu

printteripiuha, jolla projektin sähkörummut liitettiin tietokoneeseen, sekä Yamaha DTX402 -sähkörumpujen rumpumoduuli, jossa on USB-B-portti.

Kuva 2. Projektissa käytetty USB-A - USB-B -kaapeli, sekä Yamaha DTX402 sähkörumpujen rumpumoduuli [Yamaha DTX402].

MIDI-tiedoston pääte on .mid. MIDI-tiedosto on kokoelma tallennettuja MIDI-viestejä eli ohjeita. MIDIn välittämät viestit ymmärretään jokaisessa vastaanottavassa laitteessa samanlaisina, mutta niihin voidaan reagoida eri lailla. MIDI-tiedosto ei sisällä ääntä vaan ohjeita, kuinka ääntä tulisi tuottaa. Kaikki MIDI-standardia noudattavat ohjelmistot osaavat tulkita ohjeet ja mukauttaa toimintaansa niiden perusteella. Ohjeiden avulla voi tuottaa esimerkiksi ääniä, graafista esitystä tai nuottimerkintöjä nuottitiedostoon. Tallennettuja MIDI-tiedostoja voi muokata jälkikäteen, minkä ansiosta MIDI on hyvä tallennusmuoto musiikkituotannossa, koska MIDI-raidan instrumentin voi esimerkiksi vaihtaa toiseen, kuin mikä se on alun perin ollut. [Huber 1, 2020, Luku 1, kohta 1.]

MIDI-laitteet ja -ohjelmistot voivat vastaanottaa, muuttaa sekä lähettää MIDI-viestejä; esimerkiksi esityksen aikana laite voi muuttaa saamansa MIDI-nuotin sävelkorkeutta ja lähettää sen eteenpäin seuraavalle laitteelle. [Huber 1, 2020. Luku 1, kohta 1.]

2.2 MIDIn historiaa

Elektronisten instrumenttien yleistyessä useimpien laitteiden toiminta perustui sähkövirran volttien vaihteluun. Musiikkilaitteiden nuottien sävelkorkeus tulkittiin volttien perusteella, mutta tämä tapa vanhentui nopeasti. Elektronisten

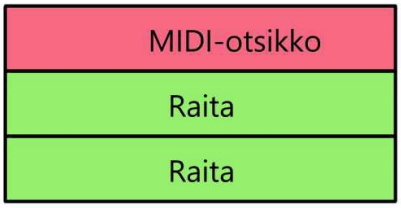
musiikkilaitteiden valmistajien myyntitulokset olivat heikompia, kuin ne olisivat voineet olla, koska eri valmistajien laitteet eivät toimineet hyvin toistensa kanssa. [Huber 2, 2020, Luku 1, kohta 2.]

Monet musiikkialan valmistajat kehittivät omia keinojaan, mutta yhteistä standardia ei ollut. Keskenään sopimattomat laitteet eivät kyenneet pysymään samassa tahdissa kovinkaan pitkään, eli eivät pysyneet synkronoituneina. Vuonna 1981 Chet Wood julkaisi tulevasta MIDIstä ensimmäisen version nimellä Universal Synthesizer Interface (USI). USI:n ansiosta eri valmistajien laitteet sattoivat toimia yhdessä. Muutaman vuoden kuluessa, yhteistyössä suurimpien alan valmistajien kanssa USI:sta kehitettiin Musical Insrument Digital Interface eli MIDI 1.0. [Huber 2, 2020, Luku 1, kohta 2.]

Vuonna 2020 julkaistiin MIDI 2.0, joka käyttää 32-bittisiä tietopaloja (data chunks). MIDI 2.0:ssa MIDI-laitteet voivat keskustella keskenään, ja ne ovat taaksepäin yhteensopivia MIDI 1.0:n kanssa. MIDI 1.0:ssa tieto kulkee vain yhteen suuntaan. Tämän projektin MIDI-tiedostojen lukijassa ei ole MIDI 2.0 -toiminnallisuutta. [Huber 2, 2020, Luku1, kohta 2.]

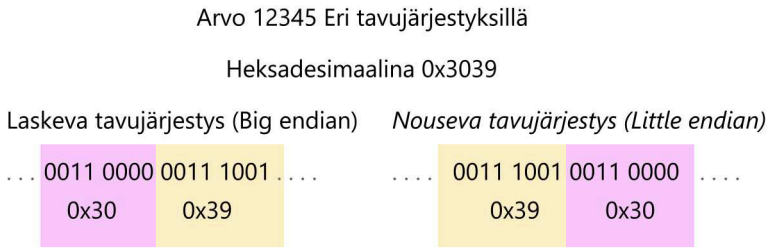
3 MIDI-tiedostot

MIDI-tiedosto koostuu MIDI-otsikosta sekä yhdestä tai useammasta raidasta. Jokainen raita sisältää raidan otsikon ja tapahtumia omine aikatunnisteineen. Tapahtumat voivat olla muun muassa kappaleen nuotteja tai tempon muutoksia. Kuva 3 esittää MIDI-tiedoston rakennetta. Ensimmäisenä on otsikko, jonka lisäksi tiedostossa on kaksi raitaa.



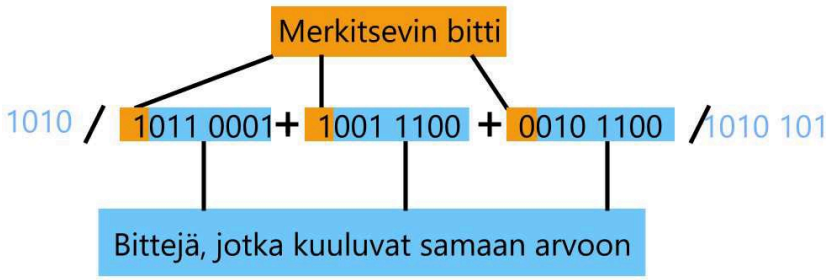
Kuva 3. MIDI-tiedoston rakenne.

MIDIn tallennettu tieto noudattaa laskevaa järjestystä (big endian), eli suurin merkitsevä bitti on tavun vasemmassa reunassa, kuten normaalissa kirjoitetussa tekstissä olevat numerot [What is Endianness? Big-Endian & Little-Endian]. Kuvassa 4 arvo 12345 on esitetty heksadesimaaliarvona 3039, sekä laskevalla ja nousevalla tavujärjestyksellä bitteinä.



Kuva 4. Laskeva ja nouseva tavujärjestys.

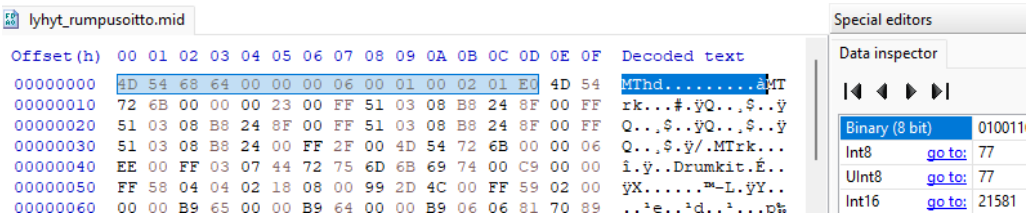
MIDIn tiedostot noudattavat joissakin kohdissa vaihtuvan pituisia tavumääriä (variable-length-quantity, VLQ). Aikatunnistetta lukiessa MIDIn tietotavut koostuvat kahdeksasta bitistä, joissa seitsemän pienintä bittiä sisältävät tietoa, ja suurin bitti ilmaisee, kuuluuko lukuun yhdistää seuraavastakin tavusta seitsemän tietotavua. Jos suurin merkitsevä bitti on 1, luetaan mukaan seuraavakin tavu. Tavuja yhdistetään, kunnes suurin merkitsevä bitti on 0, jolloin luetun tavun jälkeen ei enää lisätä seuraavaa tavua kyseiseen lukuun. Kuvassa 5 luetaan lukuun kolme tavua tietoa. Kolmannen tavun merkitsevin bitti on nolla, joten sen jälkeen lopetetaan kyseiseen arvoon lisääminen.



Kuva 5. Vaihtuva pituus ja tietoa sisältävät bitit.

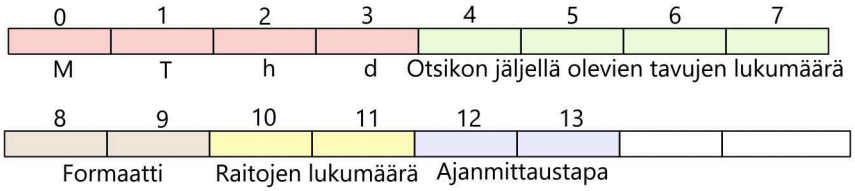
3.1 Otsikko

MIDI-tiedosto alkaa aina otsikko-osalla (header chunk). Otsikko alkaa neljän tavun tunnisteella, josta muodostuu ASCII-merkkijono *”MThd”* (kuvat 6 ja 7 esittävät otsikko-osaa). Tunnisteosan avulla tiedetään kyseessä olevan MIDI-tiedosto.



Kuva 6. MIDI-tiedosto avattuna ja otsikko valittuna heksaeditorissa.

Otsikon seuraavat neljä tavua kertovat MIDI-otsikossa jäljellä olevien tavujen määrän (yleensä kuusi tavua). MIDI-formaatti ilmoitetaan kahdella tavulla, mahdolliset arvot ovat 0, 1 tai 2. Raitojen lukumäärä ilmoitetaan kahdella tavulla. Formaatin 0 MIDI-tiedostossa on aina yksi raita. Formaattien 1 ja 2 MIDI-tiedostoissa voi olla useampia raitoja. Näissä formaateissa ensimmäinen raita on niin sanottu tietoraita, joka kertoo muun muassa tempon muutoksista. Otsikon viimeiset kaksi tavua ilmaisevat ajanmittaustavan. Ajanmittaustapoja on kaksi vaihtoehtoa. Ensimmäinen mittaustapa on SMTPE (Society of Motion Picture and Television Engineers), joka tavallaan ilmoittaa, mitä SMTPE-standardia käytetään. SMTPE määrittää, montako kuvaa sekunnissa lasketaan. Toinen ajanmittaustapa on TPQN (Ticks per quarter note). [Back. 1999.] Projektin MIDI-lukijan kehityksessä kaikki MIDI-tiedostot ovat käyttäneet TPQN-ajanmittaustapaa. Kuva 7 esittää MIDI-otsikon rakennetta tavuina.



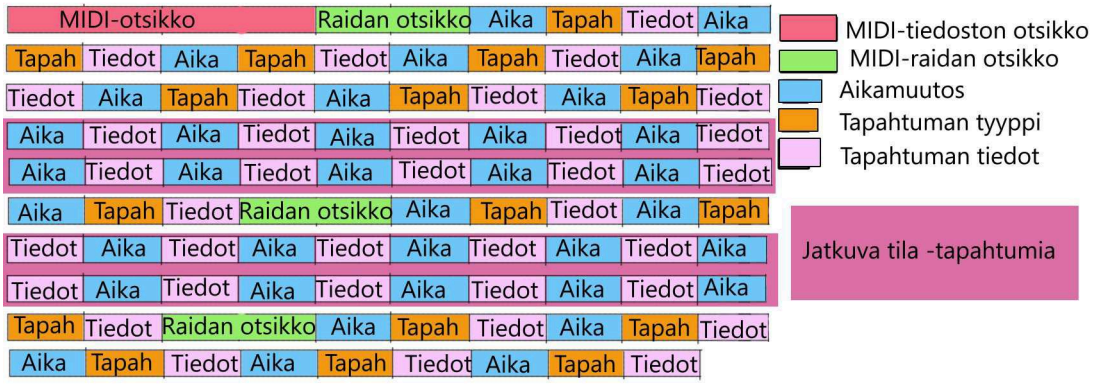
Kuva 7. Otsikko-osan tavut.

3.2 Raita

Jokainen MIDI-tiedoston raita (track chunk) alkaa otsikolla Raidan otsikko on neljä tavua ja muodostaa ASCII-merkkijonon ”*MTrk*”. Otsikon jälkeen seuraa MIDI-tapahtumia (event), jotka sisältävät aikatunnisteen ja tapahtuman tietoja.

Tapahtuman alussa oleva aikatunniste ilmaisee montako sykäystä (tick) on kulunut edellisestä tapahtumasta. Aikatunnistetta lukiessa käytetään aiemmin mainittuja MIDIn vaihtuvan pituisia tietotavuja (VLQ). Aikatunniste voi vaihtuvan pituisen arvon ansiosta olla melkein minkä pituinen tahansa biteissä mitattuna.

Aikatunnisteen jälkeen seuraa tapahtuman tyyppi (status byte), jota ilmaistaan yhdellä kahdeksanbittisellä tavulla. MIDIssä voidaan noudattaa niin sanottua jatkuvaa tilaa (running status), jolloin luetun tavun suurin merkitsevä bitti kertoo, muuttuuko tapahtuman tyyppi, vai käytetäänkö jo voimassa olevaa tapahtumatyyppiä. Jos suurin merkitsevä bitti on 1, on tapahtuma uusi tapahtuma. Kun suurin merkitsevä bitti on 0, käytetään jo voimassa olevaa tapahtuman tyyppiä ja luettu tavu käsitelläänkin tietotavuna, joka sisältää tietoa tapahtumasta. Kaikki MIDI-tiedostot eivät käytä jatkuvaa tilaa, mutta se on kyettävä tunnistamaan, jotta tiedosto osataan lukea oikein. Kuvassa 8 on esitetty tietojen järjestystä MIDI-tiedostossa



Kuva 8. MIDI-tiedostossa mahdollisesti olevaa tietoa.

Alussa on MIDI-otsikko, jota seuraa ensimmäisen raidan otsikko. Raidan otsikon jälkeen seuraa tapahtumia, jotka koostuvat aikatunnisteesta, tapahtuman tyypistä ja tapahtuman tiedoista. Osassa raidan tapahtumista käytetään jatkuvaa tilaa, joka sisältää vain aikatunnisteen ja tapahtuman tiedot. Kuvan 8 MIDI-tiedostossa on yhteensä kolme raitaa.

Tapahtumia on monenlaisia, esimerkiksi tietyn instrumentin nuotin soittamisen aloittaminen ja lopettaminen tai tempon muutos. Jokaisen tapahtuman jälkeen tulee tietotavuja, jotka antavat tapahtumalle eri arvoja. Tietotavut voivat kertoa soitettavan soittimen nuotin sävelkorkeuden tai iskun voimakkuuden, jonka perusteella voidaan määrittää kuinka lujalla äänellä nuotin tulee soida. Kuva 9 esittää MIDI-tapahtuman rakenteen, kun tapahtumakin ilmoitetaan, eli ei ole käytetty jatkuvaa tapahtumaa.

Kuva 9. MIDI-raidan tapahtuman rakenne.

3.3 MIDI-tapahtumat

MIDI-tapahtuman tyyppi kerrotaan yhdellä tavulla (8 bittiä). Yleensä neljä ensimmäistä bittiä kertovat tapahtuman tyypin, ja sen jälkeiset neljä bittiä kertovat MIDI-kanavan. Seuraavaksi on lueteltu MIDI-tapahtumat, jotka MIDI-lukijan on osattava tunnistaa.

```
Heksa-arvo 0x8+kanavanumero nuottinumerotavu nopeustavu.
```

Nuotti pois päältä (note off) -tapahtuma tarvitsee lisäksi kaksi tietotavu: nuotin numeron sekä irtipäästämisnopeuden, joka kuvaa kuinka nopeasti soivasta nuotista päästetään irti.

```
Heksa-arvo 0x9+kanavanumero nuottinumerotavu iskunopeustavu.
```

Nuotti päälle (note on) -tapahtuma tarvitsee kaksi tietotavua: nuotin numeron ja iskunopeuden, joka kuvastaa kuinka voimakkaasti soittimen tulee soida.

```
Heksa-arvo 0xA+kanavanumero nuottinumerotavu voimakkuustavu
```

Nuotin jälkikosketus (note aftertouch) -tapahtuma kuvastaa soivan nuotin painalluksen voimakkuutta. Tämä tapahtuma tarvitsee kaksi tietotavua: nuotin numeron ja uuden painalluksen voimakkuuden.

```
Heksa-arvo 0xB+kanavanumero ohjainnumerotavu arvotavu
```

Ohjaintapahtuma (controller event) kertoo MIDI-kanavan muutoksesta. Ohjaintapahtuma saa kaksi tietotavua: ohjaimen numeron ja uuden asetuksen arvon.

```
Heksa-arvo 0xC+kanavanumero uusisoitintavu
```

Ohjelman muutos (program change) -tapahtuma ilmaisee käytettävän soittimen äänen yhdellä tietotavulla tapahtumakoodin jälkeen.

```
Heksa-arvo 0xD+kanavanumero uusiarvotavu
```

Kanavan jälkikosketus (channel aftertouch) tarvitsee yhden tietotavun ilmaisemaan koko kanavan nuottien painalluksen voimakkuuden muutoksen.

```
Heksa-arvo 0xE+kanavanumero pieninmerkitsevätavu suurinmerkitsevätavu
```

Sävelkorkeuden muutos (pitch bend) saa tapahtuman lisäksi kaksi tavua, jotka yhdistämällä ilmoitetaan sävelkorkeuden muutoksesta koko kanavan nuoteille.

```
Heksa-arvo 0xF0 viestinpituustavu …viestitavuja… 0xF7
```

Järjestelmän omat tapahtumaviestit (system exclusive) ovat laitteen valmistajien omia viestejä, jotka voivat olla vaihtelevan pituisia. Viestityypin jälkeinen tavu kertoo viestin pituuden tavuina. Järjestelmän oma tapahtumaviesti päättyy heksadesimaaliin `0xF7`.

Metatapahtumat

Jotkut MIDI-tiedoston tapahtumista ovat metatapahtumia. Metatapahtumat (meta event) alkavat Heksadesimaalisella-arvolla `0XFF`.

```
Heksa-arvo 0xFF 00 02(viestinpituustavu) tietotavu1 tietotavu2 tai
0xFF 00 00(viestinpituustavu)
```

Jos sarjan numero (sequence number) -tapahtuma on MIDIssä, sen on tapahduttava raidan alussa. Jos viestin pituustavu on 02, tietotavut 1 ja 2 muodostavat 16-bittisen arvon. Tämän tapahtuman avulla voidaan yhdistää useita MIDI-tiedostoja yhdeksi kokonaisuudeksi. Tyyppien 0 ja 1 MIDI-tiedostoissa sarjan numero -tapahtuma on ensimmäisellä raidalla. Tyypin 2 MIDI-tiedostoissa jokaisella raidalla voi olla oma sarjan numero -tapahtumansa.

Moni metatapahtumista sisältää merkkijonomuotoista tekstiä. Merkkijonoja sisältävän tapahtuman viestityyppitavu kertoo, minkä tyyppinen viesti on. Tekstin tyypin jälkeen seuraa tavu, joka ilmaisee viestin pituuden tavuissa. Loput tapahtumasta on viestiä. Nämä tapahtumat ovat rakenteeltaan samanlaisia, joten ne voidaan käsitellä yhtenevällä periaatteella.

```
Heksa-arvo 0xFF viestityyppitavu viestinpituustavu …viestitavuja…
```

Teksti (text) (viestityyppitavu `0x01`), sisältää tekstiä, jonka ohjelma voi käsitellä soittoajankohtana.

Tekijänoikeusviesti (copyright) (viestityyppitavu `0x02`) kertoo tekijänoikeuksista.

Raidan nimi (track name) (viestityyppitavu `0x03`), kertoo mikä on raidan nimi. Jos MIDI-tiedosto on moniraitainen (MIDI-tyypit 1 ja 2), ensimmäisen raidan nimi kertoo koko MIDI-teoksen nimen. [MIDI Events.]

Soittimen nimi (instrument name) (viestityyppitavu `0x04`) kertoo tekstimuodossa, mitä soitinta on tarkoitus käyttää raidassa. Jos soitin käyttää eri soitinnumeroita, kuin MIDI-standardissa, voidaan tämän avulla kertoa, mitä soitinääntä on tarkoitus käyttää.

Lyriikka (lyrics) (viestityyppitavu `0x05`) sisältää laulettavaa tekstiä, tai tietyllä ajankohdalla laulettavan osan siitä.

Merkki (marker) (viestityyppitavu `0x06`) ilmaisee tekstin tietylle ajankohdalle.

Vihjeteksti (cue point) (viestityyppitavu `0x07`) kertoo vihjetekstin, esimerkiksi mitä ajankohdalla tulisi tapahtua.

```
Heksa-arvo 0xFF 20 01(viestinpituustavu) kanavatavu
```

Kanavan määritys (channel prefix) kertoo, mille soitinkanavalle sitä seuraavat metatapahtumat ja järjestelmän omat tapahtumaviestit tulevat. Viestit tulevat kyseiselle kanavalle, kunnes seuraava viesti on tavallinen viesti, kuten nuotti päälle -viesti. Kanavan määritys -tapahtuma katsotaan vanhentuneeksi. [MIDI Events.]

```
Heksa-arvo 0xFF 21 01 väylätavu
```

Väylän määritys (MIDI port prefix) katsotaan myös vanhentuneeksi. Tämä tapahtuma saattaa olla heti raidan alussa ilmaisemassa, mitä MIDI-väylää raidan tulee käyttää.

```
Heksa-arvo 0xFF 2F 00
```

Raidan loppu (end of track) ilmaisee raidan päättymisen. Tätä tapahtumaa käytetään jokaisen raidan lopussa.

```
Heksa-arvo 0xFF 51 03(viestinpituustavu) tempotavu1 tempotavu2 tempo-
tavu3
```

Tempon asettaminen (set tempo) ilmaisee uuden tempon asettamisen. Tempotavut yhdistetään 32-bittiseksi arvoksi, joka kertoo, montako mikrosekuntia yhtä neljäsosatahtia kohden kuuluu kulua aikaa. Tempo saadaan laskutoimituksella *60 000 000 / tempotavujen arvo*. MIDI-formaatissa 1 tempon asettamiset tapahtuvat ensimmäisellä raidalla. MIDI-formaatissa 0 on vain yksi raita, joten tempon asettamiset tapahtuvat siinä. Jos tempon asettamistapahtumaa ei ole, oletustempo on 120 iskua minuutissa (120 bpm).

```
Heksa-arvo 0xFF 54 05(viestinpituustavu) tuntitavu minuuttitavu sekun-
titavu kuvaasekunnissatavu sadasosakuvaasekunnissatavu
```

SMTPE-siirto (SMTPE offset) kertoo, mihin soittoaikaan raidan tulee alkaa. Viestinpituustavu kertoo, että sitä seuraa viisi tavua tietoa. Tuntitavun toinen ja kolmas bitti ilmaisevat myös kuvanopeuden (frame rate) joka on 24, 25, 29.97, tai 30. Kuvaasekunnissa-tavun suurin mahdollinen arvo perustuu kuvanopeuden arvoon: esimerkiksi jos kuvanopeus on 25, silloin suurin mahdollinen arvo on 24.

```
Heksa-arvo 0xFF 58 04(viestinpituustavu) osoittajatavu nimittäjätavu
sykäystätavu kolmaskymmeneskahdesosanuottiatavu
```

Tahtilaji (time signature) kertoo tahtilajin. Osoittajatavu kertoo tahtilajin osoittajan. Tahtilajin nimittäjä voidaan saada laskemalla luku 2 nimittäjätavun potenssiin, esimerkiksi jos nimittäjätavu on 3 saadaan $2^3 = 8$. `Sykäystätavu` ilmaisee montako sykäystä yksi neljäsosanuotti kestää. `Kolmaskymmenskahdes-osanuottitavu` kertoo, montako kolmaskymmeneskahdesosanuottia menee yhteen neljäsosanuottiin.

```
Heksa-arvo 0xFF 59 02(viestinpituustavu) sävellajitavu mollitavu
```

Sävellaji (key signature) ilmaisee MIDI-kappaleen sävellajin. Arvo 0 tarkoittaa C-sävelkorkeutta. Sävellaji nousee positiiviseen suuntaan ja madaltuu negatiiviseen suuntaan, enintään seitsemän askelta. `Mollitavu` ilmaisee onko sävellaji duurissa (englanniksi major) (arvo 0) vai mollissa (englanniksi minor) (arvo 1).

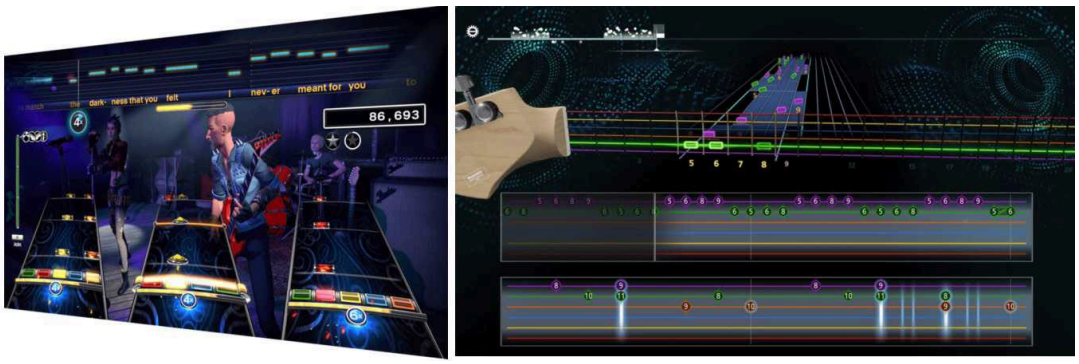
```
Heksa-arvo 0xFF 7F viestinpituustavu …viestitavuja…
```

Sekvensserikohtainen viesti (sequencer specific) on laitteen tai valmistajan viesti. `Viestipituustavun` jälkeinen viestitavu voi ilmaista tietyn tunnetun valmistajan tunnuksen, jonka viestirakennetta viestin kuuluu noudattaa. Laitteet, joille viesti ei ole tarkoitettu, eivät reagoi viestiin.

4 Sähkörumpupeli

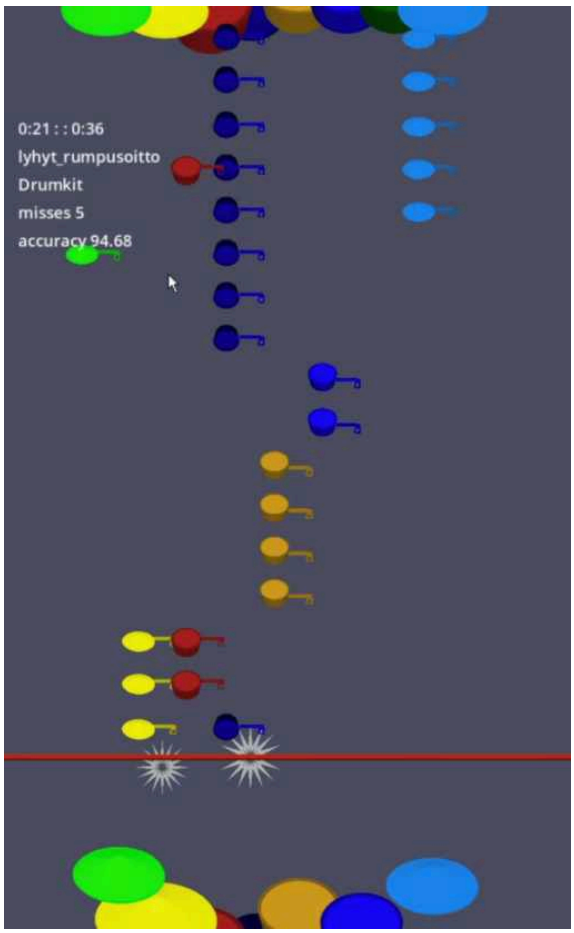
Peli, johon MIDI-tiedostonlukija tehtiin, on sähkörummuilla pelattava peli. Pelissä on tarkoitus soittaa musiikkikappaletta, jonka rumpunuotit liikkuvat kohti maalialuetta. Hyviä esimerkkejä pelityypistä ovat Guitar Hero- ja Rock Band -pelisarjojen tuotteet. Guitar Heroa pelataan peliä varten kehitetyllä kitaraa muistuttavalla ohjaimella. Kitaraohjaimen kaulassa on nappeja kuvastamassa muutamaa kitaran nauhaa ja rungossa on läppämäinen vipu kohdassa, josta kitaraa soitetaan. Rock Band -pelissä pystyy lisäksi soittamaan peliä varten tehdyllä rumpusetillä tai oikeilla sähkörummuilla.

Projektin rumpupelin innoittajana on toiminut myös Rocksmith-pelisarja, jonka periaate on sama kuin Guitar Hero -pelissä, mutta siinä musiikkikappaleita voi soittaa oikeilla kitaralla, bassolla tai mahdollisesti kosketinsoittimilla. Rocksmith-pelissä ruudussa liikkuvat tabulatuurimerkinnät, joita käytetään yleisesti musiikin kirjalliseen esittämiseen nuottien sijaan. Kuvassa 10 vasemmalla on pelitilanne RockBand -pelistä, jossa pelaa kolme pelaajaa. Pelissä jokaisella pelaajalla on oma nuottirata, jossa musiikkia ilmaisevat merkit ilmaantuvat ja liikkuvat kohti pelaajaa. Oikealla on Rocksmith+-pelin pelitilanne, jossa yläpuoliskossa on nuottirata, jota pitkin tabulatuurimerktinöjä esittävät pelimerkit liikkuvat kohti pelaajaa. Rocksmith+-pelin alapuoliskossa on kaksi vaakasuoraa, perinteistä tabulatuurinäkymää, jossa liikkuva pystyviiva osoittaa soitettavaa kohtaa.



Kuva 10. Rock Band -peli ja Rocksmith+ -peli [Rock Band; Rocksmith].

Projektin rumpupelissä voi valita MIDI-tiedoston. Peli etsii MIDI-musiikkitiedostosta rumpuraidan ja tallentaa siitä nuotit listaan aikaleimoineen, jonka jälkeen se lähettää listan eteenpäin pelin käyttöön. Peliä pelatessa rumpunuotit ilmaantuvat peliruudun ylälaidasta ja liikkuvat oman rumpusoittimen väylää pitkin kohti alareunaa, jossa sijaitsee oikeaa lyöntiajankohtaa osoittava maaliviiva. Peli havaitsee sähkörummun MIDI-viestin rumpuja soitettaessa ja tarkistaa, lyötiinkö rumpuja oikeaan aikaan (jos sopiva nuotti on maalialueella). Jos oikeaa rumpusetin osaa lyötiin hyväksyttävällä aikavälillä oikeaan aikaan, isku merkitään osuneeksi. Jos rumpunuotti ehtii mennä maaliviivan ohi, on tuloksena ”huti” ja soiton tarkkuutta osoittava luku pienenee. Kuvassa 11 näkyy millainen peli on: yläreunasta tulee rumpunuotteja esittäviä merkkejä, jotka liikkuvat kohti alareunaa. Jokaisen soittimen nuotit kulkevat omalla linjallaan. Pelissä näytetään kulunut aika, osumatarkkuus ja muuta tietoa.



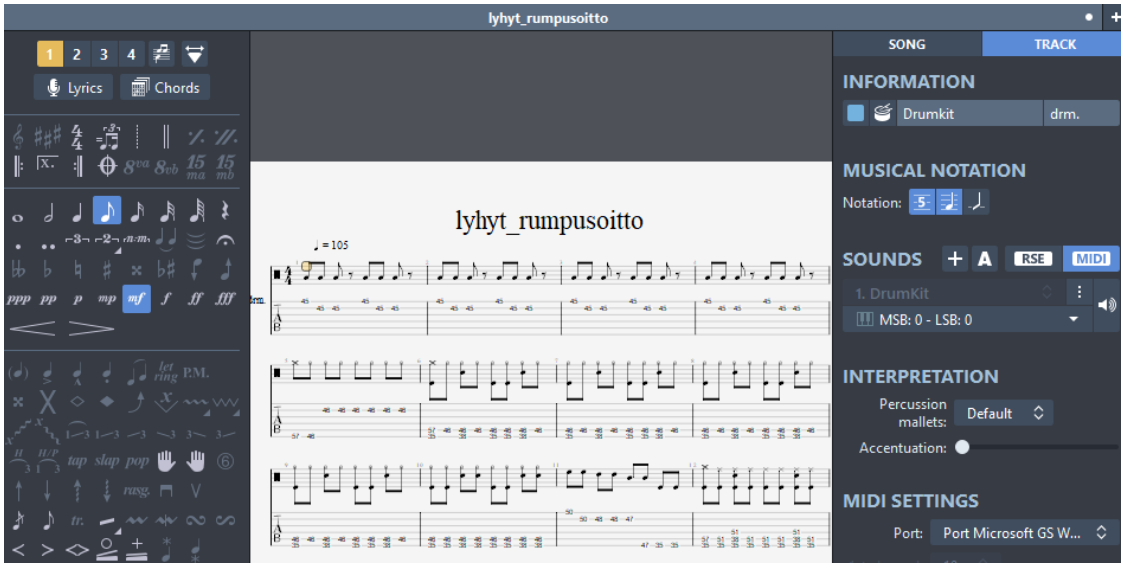
Kuva 11. Projektin rumpupeliä

5 Projektin toteutus

Projektin rumpupelin kehityksessä käytetään yhtä Yamaha DTX402 -sähkörumpusettiä ja pelinkehitysalustana toimii Godot-pelimoottori.

5.1 Projektissa käytetyt ohjelmat

Rumpupeliä kehitetään Godot-pelimoottorilla [Godot Engine]. Pelin ohjelmointikieli on GDScript, joka muistuttaa syntaksiltaan Python-ohjelmointikieltä. Pääsyy Godot-pelimoottorin käyttämiseen pelin kehityksessä oli sen valmis tuki MIDI-tapahtumien vastaanottamiselle MIDI-laitteesta. [InputEventMIDI.] MIDI-tapahtumat, kuten sähkörumpujen tai kosketinsoittimien soittaminen, havaitaan Godot-pelimoottorissa helposti lisäämällä yksinkertaisia koodirivejä pelin koodiin. Toinen pelimoottori, johon olisi ollut helposti lisättävä MIDI-tapahtumien lukija, on Unreal-pelimoottori. Oletuksena oli, että Godot on nopeampi ja kevyempi käyttää testikehitysalustana kuin raskaampi Unreal-pelimoottori.



Kuva 12. Guitar Pro -ohjelmassa avattu kappale.

MIDI-lukijan testaamiseen käytettiin MIDI-musiikkitiedostoja eri lähteistä. Osa musiikkitiedostoista haettiin verkosta, osa luotiin Guitar Pro 7 tai Mixcraft 9 -ohjelmilla. Guitar Pro 7 on nuotitus- ja tabulatuuriohjelma, joka osaa myös avata ja

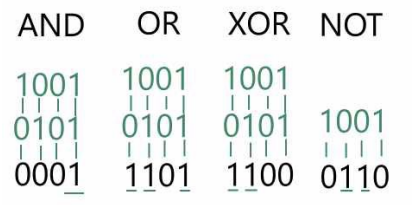
tallentaa MIDI-tiedostoja. Kuvassa 12 on Guitar Pro -ohjelmalla avattu, projektia varten luotu rumpuraita, josta näytetään perinteiset nuotit ja rumpusoittimilla mahdollisesti vaikeampilukuiset tabulatuurit. Tabulatuurien numerot vastaavat rumpuäänien MIDI-numeroita. Mixcraft 9 on musiikinluontiohjelma, joka osaa avata MIDI-tiedostoja ja jolla voi luoda MIDI-ääniä käyttävää musiikkia, jonka voi tallentaa MIDI-muotoon. MIDI-tiedostojen yksityiskohtaiseen tutkimiseen on käytetty HxD-heksaeditoria, joka on ilmainen.

Rumpusettinä pelissä on käytetty Yamaha DTX402 -sähkörumpuja. Kuvassa 13 on DTX402-rumpusetti, johon kuuluu hi-hat, virveli, bassorumpupoljin, hi-hatpoljin – joka voi toimia myös toisena bassorumpupolkimena – yksi aksenttisymbaali (crash), yksi komppisymbaali (ride) ja kolme tom-rumpua.

Kuva 13. Yamaha DTX402 -sähkörummut [Yamaha DTX402].

5.2 Bittioperaatiot

Projektin koodissa käytetään bittioperaatioita (bitwise operation). Kuvassa 14 esitellään `AND`, `OR`, `XOR` ja `NOT` -bittioperaattorien toimintaperiaatteita.



Kuva 14. Bittioperaattorien toimintoja.

Bittioperaatioissa bittioperaattorit vertaavat ja muokkaavat tavujen bittejä:

- AND-operaattorilla (`&`) verrataan kahta bittiä tai bittijonoa. Jos bitti on päällä, eli 1, palautetaan tuloksena 1, muutoin palautetaan 0.
- OR-operaattori (`|`) palauttaa tulokseksi arvon 1, jos jompikumpi vertailtavista biteistä on päällä. Jos kumpikaan biteistä ei ole päällä, palautetaan 0.
- XOR-operaattorilla (`^`) verrataan bittien eroja: jos vertailtavista biteistä vain toinen on 1 ja toinen on 0, eli bitit ovat eriarvoiset, palautetaan tuloksena 1. Jos molemmat bitit ovat arvoltaan samoja, palautetaan tuloksena 0.
- NOT-operaattori (`~`) kääntää bittien arvot. Jos arvo on 1, siitä tulee 0. Jos arvo on 0, siitä tulee 1.
- Siirto-operaattori (`<<`) siirtää bittejä vasemmalle annetun arvon verran (esimerkkikoodi 1).
- Siirto-operaattori (`>>`) siirtää bittejä oikealle annetun arvon verran.

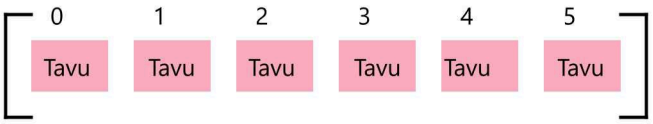
Paikkoihin, mistä bitti on siirto-operaattorilla siirretty pois, tulee arvoksi 0. Bittejä siirrettäessä voi tapahtua ylivuoto, jos aktiiviset bitit siirtyisivät niille varatun paikan yli, esimerkiksi jos kahdeksanbittisen arvon suurin bitti siirtyisi vielä vasemmalle, se katoaisi, eikä sitä enää laskettaisi arvoon. Uuden muuttujan voi luoda siirto-operaattorilla. Esimerkkikoodissa 1 on pseudokoodiesimerkki, jossa16-bittinen arvo luodaan siirto-operaattorin avulla.

```
var muuttuja_8bit = 0b01010101 #0b = binääriarvo

var muuttuja_16bit = muuttuja_8bit << 8 #siirretään 8 bittiä

# muuttuja_16bit on: b01010101 00000000 -siirrettyjen bittien pai-
koilla on 0
```

Esimerkkikoodi 1. Pseudokoodiesimerkki siitä, kuinka voidaan luoda 16-bittinen muuttuja siirto-operaattorin avulla kahdeksanbittisestä muuttujasta.

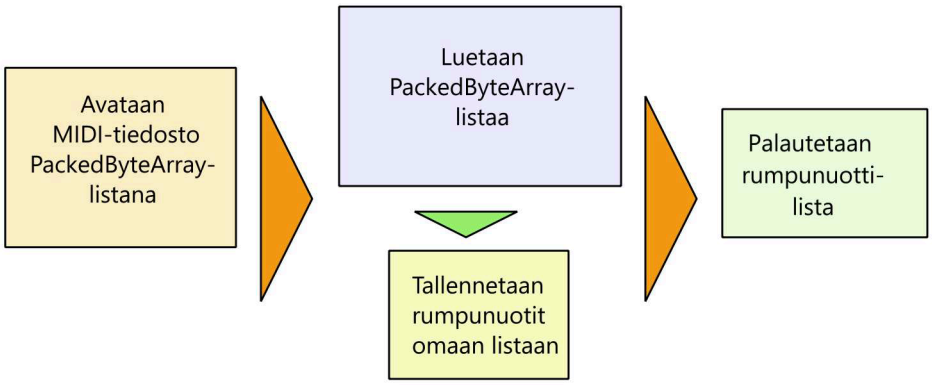
5.3 Koodin rakenne

MIDI-lukija aktivoituu, kun sen käsketään luoda rumpuraita. MIDI-lukija hakee pelin alkuvalikossa valitun kappaleen MIDI-tiedoston ja tekee siitä Godotin pakatun tavulistan (PackedByteArray). Pakatussa tavulistassa tietotavut ovat tiiviisti peräkkäin, mikä tehostaa lukemista ja säästää tilaa. Pakatussa tavulistassa tieto on indeksittäin tavuina, jolloin MIDIn tietoa voidaan lukea indeksi kerrallaan tavusta tavuun. Kuvassa 15 on tavuja indeksinumeroineen listassa.



Kuva 15. Tavuja listassa indeksoituina.

Ohjelma lukee ensin MIDI-tiedoston otsikon. Otsikon lukemisen jälkeen toistetaan raidanlukufunktiota, kunnes kaikki raidat on luettu. Raidanlukufunktio itsessään lukee ensiksi raidan otsikon, jonka jälkeen se lukee raidan tapahtumat. Raitoja lukiessa tallennetaan raidan mahdolliset rumpunuotit listaan. Kuvassa 16 esitetään MIDI-lukijan toimintaperiaate. Ensin avataan MIDI-tiedosto PackedByteArray-muodossa, joka luetaan. Lukemisen aikana tallennetaan rumpunuotit omaan listaan. Lukemisen jälkeen rumpunuottilista palautetaan eteenpäin.



Kuva 16. MIDI-lukijan toimintaperiaate.

MIDI-tiedoston otsikon lukeminen

MIDI-lukijan ohjelmoinnissa käytettiin tukena useita verkkolähteitä. Tärkeimpänä apuna voidaan pitää Youtube-videota *Programming MIDI*, jossa luotiin MIDI-lukija C++ -ohjelmointikielellä. [javidx9, 2020.]

Lukemisen alussa MIDI-tiedoston lukeva funktio tarkistaa, muodostuuko neljästä ensimmäisestä tavusta merkkijono ”*MThd”*, eli MIDI-tiedostoa tarkoittava lyhenne. Lyhennettä seuraavat neljä tavua kertovat, montako tavua otsikkoa on jäljellä, yleinen arvo on kuusi. Seuraavaksi luetaan MIDIn tyyppi (indeksit 8 ja 9). Tyyppi on 0, 1 tai 2. MIDIn tyypin jälkeen luetaan raitojen lukumäärä, jota merkitään kahdella tavulla (indeksit 10 ja 11). Kuvassa 17 näkyy lokitekstiä MIDI-lukijan lukiessa MIDIn otsikon.

```
reading header
4D 54 68 64
M T h d

size: 6

format type (0,1 0r 2): 01
number of tracks: 2

time division is ticks per quarter note (TPQN)
time division 480 ticks_per_beat(same as per quarter note)
```

Kuva 17. Lokitekstiä MIDIn otsikon lukemisen aikana.

MIDI-tiedoston ajanmittaustapa aloitetaan lukemalla tavun suurin merkitsevä bitti indeksistä 12. Jos suurin merkitsevä bitti on 1, ajanmittaustapa pohjautuu SMPTE -järjestelmään. Jos suurin merkitsevä bitti on 0, ajanmittaustapa on ”sykähdystä per neljäsosanuotti” eli TPQN. Ajanmittaustapa vaikuttaa laskutoimitukseen, jonka perusteella ohjelma merkitsee aktivoituvien nuottien aikaleimat. Esimerkkikoodi 2 näyttää ajanmittaustavan lukemisen.

```
     var timediv = 0
     var first_int = file[12]
     var top_bit : int = (first_int & 0x80) >> 7

     #combine a bytes into one time division value
     timediv = file[12]
     timediv = (timediv << 8) | (file[13])

     if top_bit == 1:
          tpqn = false #don't use ticks per quarternote
          print("time division is SMPTE, frames per second")
          frames_per_second = -(timediv >> 8) #top byte is negative...
          ticks_per_frame = timediv & 0xFF #low byte is ticks/frame
          print("frames per second ", frames_per_second)
          print("ticks per frame ", ticks_per_frame )

     elif top_bit == 0:
          tpqn = true
          print("time division is ticks per quarter note (TPQN)")
          ticks_per_beat = timediv & 0x7FFF #dont read the top bit
          print("time division ", ticks_per_beat, " ticks_per_beat(same
as per quarter note)")
```

Esimerkkikoodi 2. Ajanmittaustavan määrittelevä koodiosa.

MIDI-raitojen lukeminen

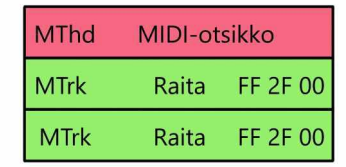
MIDI-tiedoston otsikon lukemisen jälkeen ohjelma alkaa lukea tiedoston raitoja toistolausekkeella. Lukemista toistetaan, kunnes viimeinen raita ja samalla koko MIDI-tiedosto on luettu.

Raidan lukeminen aloitetaan lukemalla raidan otsikko. Raidan otsikon neljä ensimmäistä tavua muodostavat merkkijonon ”*MTrk”*. Seuraavat neljä tavua ilmaisevat raitaan kuuluvien tavujen määrän (esimerkkikoodi 3).

```
var amount_of_bytes = 0
for i in range(4):
     #amount_of_bytes
     amount_of_bytes = (amount_of_bytes << 8) | file[index]
     index+=1
```

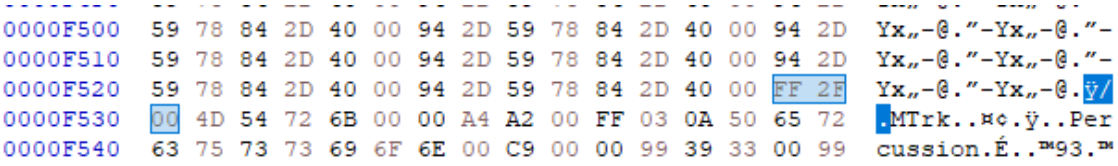
Esimerkkikoodi 3. Raitaan kuuluvien tavujen määrä.

Tavumäärän lukemisen jälkeen raidan tapahtumat luetaan omalla funktiollaan. Funktio on pääasiassa toistolause, joka jatkuu, kunnes sen käsketään lopettaa. Raidanlukufunktiossa (`func read_track_data()`) on kolme muuttujaa: `Boolean`-muuttuja `track_continues` kertoo jatkuuko vai loppuuko raita. Kun raidan tapahtumaksi luetaan kokonaisuudessaan heksa-arvot *`FF 2F 00`*, kyseinen raita on päättynyt. Kuvassa 18 esitetään MIDI-tiedoston otsikon ja raitojen alkujen osoittavat koodit sekä raitojen päättymisestä ilmoittavat koodit.



Kuva 18. MIDI-tiedoston rakennetta.

Raidan alkamisesta ja päättymisestä ilmoittavat koodit voi löytää merkkijonomuodossa myös heksaeditorissa avatusta MIDI-tiedostosta. Kuvassa 19 näkyy heksaeditorissa merkittynä MIDI-tiedoston kohta *`FF 2F 00`*, joka tarkoittaa raidan päättymistä. Päättyneen raidan jälkeen alkaa uusi raita, josta ilmoittaa merkkijono *MTrk*.



Kuva 19. Raita päättyy, jonka jälkeen alkaa uusi raita.

Aikamuunnosta varten on muuttuja `delta_ticks`, johon tallennetaan kulunut aika tapahtumien välillä. Kolmas muuttuja on `status`, johon tallennetaan raidassa käytettävä senhetkisen tapahtuman tila. Tila ei aina muutu MIDIn jatkuvan tilan takia (running status), joten se on hyvä tallentaa raitaa lukiessa.

Raidan tapahtumat luetaan `while`-toistolauseella, joka jatkuu niin kauan, kun koko tiedoston loppua ei ole saavutettu ja raidan jatkumisesta kertova *boolean* muuttuja `track_continues` on tosi.

Tapahtumia luettaessa silmukassa tulee aina ensimmäisenä aikatunniste, joka voi olla myös nolla. Tiedoston tavujen lukemisessa on käytetty apufunktiota `read_value()` (Esimerkkikoodi 4), joka lukee tavun vuorossa olevasta listan indeksistä, siirtää indeksiä yhdellä eteenpäin ja palauttaa luetun tavun arvon.

```
func read_value():
     var value = file[index]
     index+=1
     return value
```

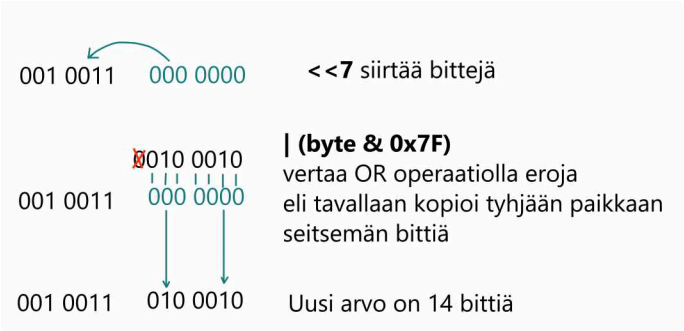
Esimerkkikoodi 4. indeksinlukufunktio.

Aikatunniste on vaihtuvan pituinen tietoarvo. Se luetaan normaalisti kahdeksanbittisinä tavuina, joissa suurin merkitsevä bitti kertoo, luetaanko arvoon seuraavakin tavu. Jos suurin merkitsevä bitti on 1, luetaan seuraavakin arvo. Kun suurin merkitsevä bitti on 0, luetaan kyseinen tavu, mutta ei enää seuraavaa. Arvoon ei lisätä suurinta merkitsevää bittiä, vaan sen jälkeiset seitsemän pienempää bittiä. Aikatunniste muutetaan laskutoimituksen avulla sekunneiksi, joka osoittaa kappaleen senhetkistä soittokohtaa (esimerkkikoodi 5).

```
var byte = read_value()
                    #ignore the top byte, add 7 to the delta time
               delta_ticks = (delta_ticks << 7) | (byte & 0x7F)
                    #if the top byte is 0, break, delta_ticks-data
                    ended
               if byte & 0x80 == 0:
                    break
               #add delta time to total time
               tracktime_in_seconds += delta_ticks * microsecs_per_tick
     / 1000000.0
```

Esimerkkikoodi 5. Aikamuutoksen lukeminen.

Koodissa suoritetaan bittioperaatioita: (`delta_ticks << 7`) siirtää bittejä seitsemän paikkaa ja koodin osa: `| (byte & 0x7F`) lisää arvoon seitsemän pienintä bittiä TAI-operaattorilla ja jättää suurimman bitin pois. Näin arvoa voi kasvattaa yli seitsemän bitin suuruiseksi. Kuvassa 20 luodaan bittioperaatioilla 14-bittinen arvo.



Kuva 20. Bittioperaatioesimerkki.

Tapahtuman tila (Status)

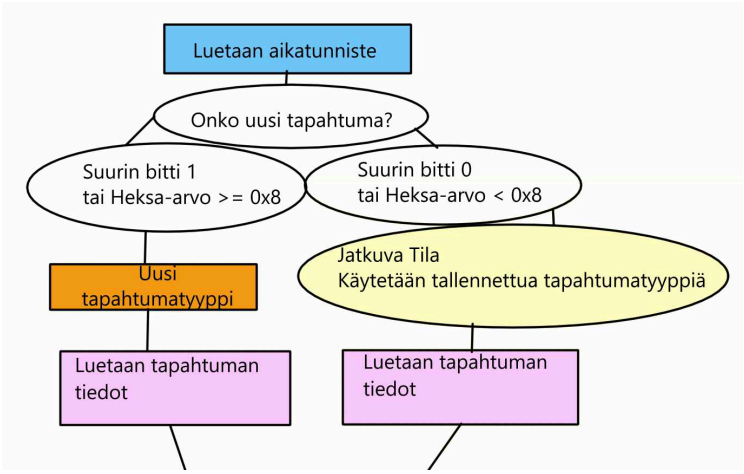
Aikamuutoksen jälkeen luetaan tapahtuman tila (status). Tila ei välttämättä ole muuttunut, joten se tallennetaan aluksi `new_status` -muuttujaan (esimerkkikoodi 6).

```
var new_status = file[index]
          #running status
          if new_status < 0x80:
               index -=1
          else:
               status = new_status
```

Esimerkkikoodi 6. Tapahtuman tilan lukeminen.

Se, onko tila muuttunut vai pysyykö, samana, eli onko tila ”jatkuva tila” (running status), voidaan tarkistaa eri tavoilla. Jos tapahtuman tilaa merkitsevän tavun suurin merkitsevä bitti on 1, on tila muuttunut. Jos tavu on 0, aikaisempi tila jatkuu ja kyseinen tavu on tietotavu (data byte). Tila voidaan myös lukea heksadesimaalina, jolloin katsotaan, onko tavun arvo alle 80 heksadesimaaliarvona, eli `<0x80`. Jos arvo on alle 0x80, jatkuu aikaisempi tila, ja tavu on tietotavu.

Kuvassa 21 Kuvassa esitetään MIDI:n tapahtuman lukeminen. Ensin luetaan aikatunniste, sitten tarkastetaan onko tapahtuma muuttunut, eli onko suurin merkitsevä bitti 1, joka tarkoittaisi, että tapahtuma on muuttunut. Jos suurin merkitsevä bitti on 0 jatketaan jo käytettyä tapahtuman tyyppiä. Viimeisenä luetaan tapahtuman tiedot



Kuva 21. Ajan ja tapahtuman lukeminen silmukassa.

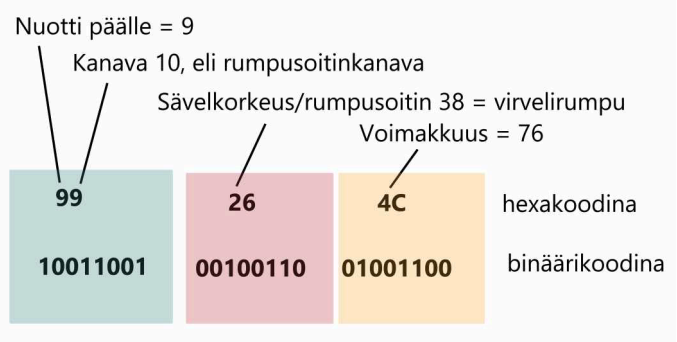
Tapahtuman tila koostuu kahdesta osasta, alkuosasta ja loppuosasta. Alkuosa on tilatavun neljä suurinta bittiä. Molemmat voivat olla väliltä 0−15.

Vaikka peliä varten halutaan tietää vain joidenkin tapahtumien sisältämät tiedot, on kaikki tapahtumat luettava ja siirrettävä indeksiä oikea määrä eteenpäin tiedostossa. Tapahtumien lukeminen suoritetaan GDScriptin `match`-lauseella, joka vastaa joissain kielissä käytettäviä `switch`-lauseita. Kyseinen lause käy läpi kaikki mahdolliset tapahtumat. Ohjelmassa tapahtumille on nimetty omaan listaan tapahtumien nimet ja niitä vastaavat heksadesimaaliarvot (esimerkkikoodi 7)

```
enum EVENTNAME
{
     note_off = 0x8,
     note_on = 0x9,
     aftertouch = 0xA,
     control_change = 0xB,
     program_change = 0xC,
     channel_pressure = 0xD,
     pitch_bend = 0xE,
     system_exlusive = 0xF
}
```

Esimerkkikoodi 7. Listatut tapahtumat.

Esimerkiksi nuotti pois päältä -viesti ilmaistaan tilatavun heksadesimaalisella alkuosalla 0x8. Tilatavun toinen osa ilmaisee kanavaa. Numero 9 tarkoittaa kanavaa 10, eli rumpusoitinkanavaa. Rumpukanavan soittimen päältä poislaittaminen alkaisi heksa-arvolla `0x89`. Nuotti pois päältä -viestin tieto-osuus on kaksi kahdeksanbittistä tavua: ensimmäinen tavu kertoo sävelkorkeuden ja toinen tavu kuvastaa soittimen päältä poislaittamisen nopeutta. Kuvassa 22 on nuotti päälle -tapahtuman osat esitettynä heksakoodina ja binäärikoodina. Heksakoodi 99 kertoo nuotti päälle tapahtumasta, sekä kanavan arvoksi 10 eli rumpukanava. Heksa-arvo 26 kertoo rumpusoittimen numeroksi 38, eli virvelirumpu. Heksakoodi 4C kertoo voimakkuudeksi 76. Samat arvot näkyvät myös binäärisarvoina.



Kuva 22. Nuotti päälle -tapahtuman mahdolliset arvot.

Tilan alkuosan ollessa 9 on tilan tapahtumana nuotti päälle, eli nuotin soittamisen aloittaminen. Nuotti päälle -tapahtuma luetaan tarkkaan (esimerkkikoodi 8), koska se voi sisältää rumpunuotin.

```
     EVENTNAME.note_on:
                    var channel = status & 0x0F #4 lowest bits
                    index +=1
                    var note = read_value()
                    var velocity = read_value()
                    #if velocity is 0, it's actually note off
                    if velocity > 0:
                         if channel == 9:
                              create_a_note(tracktime_in_seconds, note)
                              #print("appened note")
                         if print_note_ons:
                              print(" time ",tracktime_in_seconds,",
deltaticks: ", delta_ticks, " note on " + "%X" % event_type, " channel
" + "%X" % channel, " note ", note, " velocity ",velocity )
                    else:
                         if print_note_offs:
                              #note off event here..#must add note off
to list here too
                              print(" time ",tracktime_in_seconds, "
note on " + "%X" % event_type, " channel " + "%X" % channel, " note ",
note, " velocity ",velocity )
```

Esimerkkikoodi 8. Nuotti päälle -tapahtuman käsittely.

Nuotti päälle -tapahtuman neljä pienintä bittiä kertovat soittimen kanavan. Kymmenes kanava (koodin lukemana 9) on tarkoitettu rumpusoittimille, eli koodi `0x99` tarkoittaa rumpunuotin aktivointia. Nuotti päälle -tapahtumaa seuraa kaksi tietotavua. Ensimmäinen tietotavu kertoo tyypillisesti soittimen sävelkorkeuden, mutta rumpusoittimissa ensimmäinen tietotavu kertoo soittimen tyypin (virveli, tom, symbaali, jne.). Toinen tietotavu kertoo iskunopeuden. Jos iskunopeus on nolla, tulkitaan tapahtumaksi nuotti pois päältä -tapahtuma. Jos nuotti päälle -tapahtuman soittimena on rummut, lisätään tapahtuma raidan rumpunuottilistaan. Muiden soitinkanavien nuotteja ei käsitellä enempää.

0xFF-koodi ilmaisee tapahtuman olevan metatapahtuma. Jotkut metatapahtumat sisältävät pelille hyödyllistä tietoa, osa tärkeätä tietoa tiedoston lukemisen kannalta. Esimerkkikoodi 9 näyttää listatut metatapahtumat. Yksi tärkeä metatapahtuma on aikaisemminkin mainittu raidan päättyminen, jonka heksadesimaaliarvo on FF 2F 00.

```
enum METAEVENT
{
     sequence = 0x00,
     text = 0x01,
     copyright = 0x02,
     track_name = 0x03,
     instrument_name = 0x04,
     lyrics = 0x05,
     marker = 0x06,
     cuepoint = 0x07,
     channel_prefix = 0x20,
     port_prefix = 0x21,
     endOfTrack = 0x2F,
     set_tempo = 0x51,
     smpte_offset = 0x54,
     time_signature = 0x58,
     key_signature = 0x59,
     sequencer_specific = 0x7F
}
```

Esimerkkikoodi 9. Listatut metatapahtumat.

Moni metatapahtumista sisältää merkkijonon, esimerkiksi raidan nimen (Track Name, 0x03) tai soittimen nimen (Instrument Name, 0x04). Esimerkkikoodi 10 käsittelee soittimen nimi -tapahtuman.

```
METAEVENT.instrument_name:
     print("reading instrument")
     index +=1
     var text = read_string()
     if track_name == "":
          track_name = text
```

Esimerkkikoodi 10. Koodi käsittelemään soittimen nimi -tapahtuma.

Merkkijonoja sisältävät tapahtumat kertovat merkkien lukumäärän. Kyseiset tapahtumat luetaan omalla apumetodilla `read_string()` (esimerkkikoodi 11). Apumetodi palauttaa merkkijonon, ja jos tapahtuma oli raidan tai instrumentin nimi, se voidaan tallentaa muuttujaan. Ainoastaan rumpusoittimen nuotteja sisältävän raidan tallennettu merkkijono näkyy pelissä − jos sellainen on löytynyt MIDI-tiedoston lukemisen aikana.

```
func read_string():
     var sentence_length = read_value()
     print("sentence length: ", sentence_length)

     var sentence = ""
     for i in range(sentence_length):
          sentence += char(read_value())
     print("sentence: ", sentence)
     return sentence
```

Esimerkkikoodi 11. Metodi merkkijonojen lukemiseen.

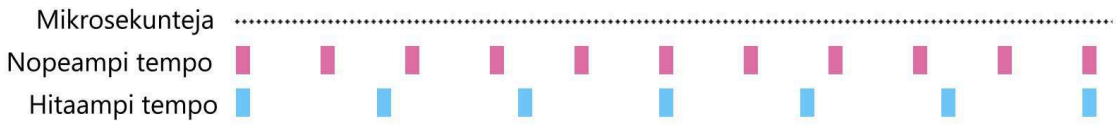
Tempon vaihtuminen

MIDIssä tempon vaihtelu on huomioitu metatapahtumalla set_tempo. Kyseiset tapahtumat ovat yleensä ensimmäisellä raidalla musiikki-instrumenttien nuottien ollessa muilla raidoilla (MIDI-luokat 1 ja 2). MIDI-luokka 0 on yksiraitainen, joten tempomuutokset ovat siinä samalla raidalla kuin musiikki-instrumenttien nuotit. Tempomuutos tallennetaan omalle listalleen aika-arvoineen, josta se haetaan oikeassa ajankohdassa. Kuvassa 23 on lokitekstiä, jossa näkyy muun muassa sykäysten väli mikrosekunteina eri tempoilla. Sykäysten määrä pysyy samana neljäsosanuottia kohden (ticks per beat) vaikka tempo muuttuu hitaammaksi tai nopeammaksi.

```
 time 4.625, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
ticks per beat: 480
updating tempo, tracktime: 5, bpm: 60, microsecs per tick: 2083.33333333333
 time 5 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 5, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
 time 5.5 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 5.5, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
 time 6 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 6, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
 time 6.5 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 6.5, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
 time 7 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 7, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
 time 7.5 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 7.5, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
 time 8 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 8, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
 time 8.5 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 8.5, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
ticks per beat: 480
updating tempo, tracktime: 9, bpm: 120, microsecs per tick: 1041.66666666667
 time 9 delta_ticks: 240 note off 8 channel 9 note 38 velocity 64
 time 9, deltaticks: 0 note on 9 channel 9 note 38 velocity 76
```

Kuva 23. Tempomuutoksia, note on ja note off -tapahtumia.

MIDI laskee aina yhtä monta sykäystä (ticks) samanpituisten nuottien välissä riippumatta temposta. Kun tempo on 120, voi kahdeksasosanuottien välissä olla 240 sykäystä ja joka sykäyksen väli on 1040 mikrosekuntia. Kun tempo on 200, sykäysten määrä on yhä 240, mutta niiden aikaväli on 620 mikrosekuntia. Tempon muuttuminen siis vaikuttaa siihen, montako mikrosekuntia odotetaan sykäysten välissä. Kuvassa 24 nopeampitempoisia sykäyksiä mahtuu aikavälille useampia kuin hitaampitempoisia sykäyksiä.



Kuva 24. Esimerkki sykäysten tiheydestä tietyllä aikavälillä hitaammalla ja nopeammalla tempolla.

Esimerkkikoodissa 12 tempomuutostapahtuma luetaan ja tallennetaan. Tempomuutoksen arvotavut `microsecs_per_quarternote` kertovat tempon mikrosekunteina neljäsosatahtia kohden. `Bpm` -muuttujaan tallennettava arvo saadaan laskutoimituksella `60 000 000 / microsecs_per_quarternote`. Tempomuutos tallennetaan tempomuutoslistaan aikaleiman kanssa.

```
METAEVENT.set_tempo:
     #Tempo changes have to be saved and implemented, on other tracks
     index+=1 #move to read tempo length
     var tempo_length = read_value() #how many bytes the tempo value is
     var microsecs_per_quarternote = 0

     #combine tempo byte values
     for i in range(tempo_length):
          microsecs_per_quarternote = (microsecs_per_quarternote << 8)
| (file[index+i] )
          index+=tempo_length #go forward indexes

     bpm = (60000000.0/microsecs_per_quarternote)
     microsecs_per_tick = microsecs_per_quarternote / ticks_per_beat

     #append tempo change to list to be used on (drum-) notetrack
     tempo_changes.append([tracktime_in_seconds,bpm])
```

Esimerkkikoodi 12. Koodi tempon asettamista ja tallennusta varten.

Mahdollinen tempomuunnos tarkastetaan aina aikamuutoksen lukemisen jälkeen (esimerkkikoodi 13). Tempomuutokset ovat aikajärjestyksessä tempomuutoslistassa (`tempo_changes[])`. Jos tempomuutoksen aikaleima on suurempi tai yhtäsuuri kuin raidan senhetkisen kohdan aika, suoritetaan tempon muuttaminen.

```
if reading_note_tracks:
     if len(tempo_changes) > 0 and tempo_index < len(tempo_changes):
          if tempo_changes[tempo_index][0] <= tracktime_in_seconds:
               update_notetrack_tempo()
               tempo_index += 1
```

Esimerkkikoodi 13. Mahdollinen tempomuutos tarkastetaan aina aikamuutoksen jälkeen.

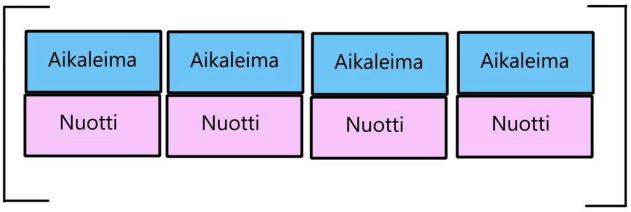
Jos tempo muuttuu, ajetaan `update_notetrack_tempo()` -metodi (esimerkkikoodi 14), jossa muuttujasta `bpm` käännetään takaisin mikrosekunteja ilmoittava muuttuja `microsecs_per_quarternote`. Jotta saadaan selville, montako mikrosekuntia yksi MIDIn aikasykäys kestää, käytetään laskutoimitusta microsecs_per_quarternote / ticks_per_beat. `Ticks_per_beat` -muuttuja on saanut arvon jo MIDIn otsikkoa luettaessa.

```
func update_notetrack_tempo():
     #update tempo, take second index (tempo value)
     bpm = tempo_changes[tempo_index][1]
     var microsecs_per_quarternote = 60000000.0 / bpm

     microsecs_per_tick = microsecs_per_quarternote / ticks_per_beat
     print("updating tempo, tracktime: ", tracktime_in_seconds, ", bpm:
",bpm, ", microsecs per tick: ", microsecs_per_tick)
```

Esimerkkikoodi 14. Metodi tempomuutoksen ajamiseksi.

Jokaisen raidan lopussa tarkastetaan, onko siitä löytynyt pisin rumpunuotteja sisältävä raita. Jos raita on pisin, siitä luotu nuottilista tallennetaan rumpunuottiraidaksi ja raidan mahdollinen nimi tallennetaan myös. Ennen kuin rumpunuottilista palautetaan pelin käyttöön, se käännetään toisinpäin, jolloin nuotti, jolla on pienin aika-arvo, on listan viimeisenä. Pelin aikana nuottilistaa luetaan loppupäästä ja nuotit poistetaan sitä mukaan, kun ne luodaan peliin. Tämän

tarkoituksena on tehostaa listan käsittelyä, koska loppupäästä poistettaessa ei tarvitse järjestää koko nuottilistaa joka kerta uudelleen. Kuvassa 25 esitetään nuottilista, joka koostuu nuoteista aikaleimoineen.



Kuva 25. Nuottilista.

Protovaiheessa MIDI-lukija tulostaa lokiin listan mahdollisista rumpunuoteista, jotka eivät kuulu testisoittimena käytetyn sähkörumpusetin soitinvalikoimaan. Koska pelin kehitysversiossa ei tällä hetkellä voinut säätää rumpusetin kokoonpanoa, osa MIDI-lukijan palauttamista nuoteista vaihdettiin rumpusetin nuoteiksi pelaamisen aikana. Jos rumpusettiin ei kuulunut rumpuinstrumenttia kyseisellä MIDI-numerolla, numero vaihdettiin toiseen rumpusetistä löytyvään numeroon. Liite 1 sisältää listan MIDI-rumpunuoteista. Vaihtaminen tapahtui `match`-lausekkeella (esimerkkikoodi 15). Koodissa on sanakirja nimellä `drumkit`, jonka avaimina toimivat MIDI-rumpunuottien numerot ja arvoina on jokaiselle nuotille määritelty olio. Olion tehtävänä on luoda kyseiselle nuotille liikkuva nuottiolio metodilla `spawn_moving_note()`. Esimerkkikoodi 15:n `match` -lauseke muuttaa ennaltamääritellyt rumpunuottien numerot numeroiksi, jotka löytyvät pelin testirumpusetistä eli `Global.INSTRUMENT_TYPE` - listasta.

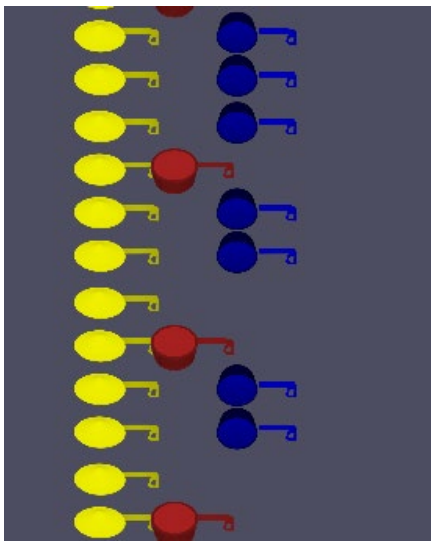
```
if note.instrument in drumkit.keys():
              drumkit[note.instrument].spawn_moving_note()
          else:
              #change numbers (at the moment) for better fit set
              match note.instrument :
                  35: note.instrument = Global.INSTRUMENT_TYPE.BASS
                  41:note.instrument = Global.INSTRU-
MENT_TYPE.HIGH_TOM
                  45:note.instrument = Global.INSTRUMENT_TYPE.LOW_TOM
                  57:note.instrument = Global.INSTRU-
MENT_TYPE.CRASH_CYMBAL
                  59: note.instrument = Global.INSTRU-
MENT_TYPE.RIDE_CYMBAL
                  53: note.instrument = Global.INSTRU-
MENT_TYPE.RIDE_CYMBAL
                  50: note.instrument = Global.INSTRU-
MENT_TYPE.HIGH_TOM
                  40: note.instrument = Global.INSTRUMENT_TYPE.SNARE
```

Esimerkkikoodi 15. Ote rumpunuottien vaihtamisesta koodissa.

5.4 Jatkokehitysideoita

Peli ei ilmaise, milloin tahti alkaa tai päättyy. Tahtien merkitseminen helpottaisi mukana pysymistä soiton ajan. Tahteja ilmaisevat viivat voisivat olla neljäsosatahtien tai kokotahtien välein.

Normaalisti nuotituksesta ja hyvästä tabulatuurista näkee myös nuottien pituuden, tämän ansiosta on helpompi tietää, montako rumpuiskua tulee lyödä, jos nuotit ovat tiheästi. Pelissä nuotit tulevat yleensä pitkinä pötköinä, kuten kuvasta 26 näkyy. Tällöin voi olla vaikea laskea, montako iskua tulisi olla kussakin tahdissa. MIDI-lukijan tulee jatkossa merkitä, milloin kyseinen rumpunuotti päättyy note off -tapahtumalla tai milloin soitetaan sama nuotti uudestaan. Tämän perusteella nuotille voitaisiin tallentaa myös pituusarvo, joka kertoisi, onko nuotti esimerkiksi neljäsosanuotti, kahdeksasosanuotti tai muu nuotti. Kun nuottien pituudet tietää, on helpompi laskea, montako kertaa kyseistä rumpunuottia soitetaan tahtia kohden.



Kuva 26. Rumpunuotteja pelissä.

Tempon vaihtumista ei ilmaista vielä pelin aikana mitenkään. Myös tempon vaihtuminen tulee lähettää jatkossa peliin, jolloin pelin aikana näkyy ilmoitus, kun tempo muuttuu. SMTPE -aikakoodin käyttäminen tulee myös olla pelissä mukana siltä varalta, että jotkut käännetyt kappaleet käyttävät sitä.

6 MIDI-tiedostonlukijan arviointi

MIDI-lukija tekee sen mitä siltä vaaditaan ja voisi käsitellä tapahtumia, joita ei tällä hetkellä vielä tarvita pelin kehitysversiossa. Pelin kehityssuunnitelma on muuttunut MIDI-lukijan luonnin aikana. Muutokset tulevat vaikuttamaan myös MIDI-lukijaan. MIDI-lukijaa testattiin erilaisilla MIDI-tiedostoilla: osa oli ilmaisia pop- ja rock-bändien kappaleita, joita voi löytää internetistä. Ilmaiset kappaleet olivat MIDI-formaateissa nolla tai yksi, eli osa oli moniraitaisia, osassa oli vain yksi raita, johon koko MIDI-kappaleen tieto oli tallennettu. Suuri osa MIDI-lukijan testaamiseen käytetyistä musiikkikappaleista oli Guitar Pro -ohjelmaan yleisesti saatavilla olevista tiedostoista MIDI-tiedostoiksi käännettyjä kappaleita. Käännetyt MIDI-tiedostot olivat formaatissa 1, eli moniraitaisia. Kehityksen aikana tempon vaihtelun huomioiminen ei aluksi toiminut, mutta projektin päättyessä myös tempon vaihtaminen toimii, eli rumpunuottien ilmaantumisväli suurenee, kun tempo hidastuu.

MIDI-lukijassa on vielä kehitettävää: sen on pystyttävä varautumaan erikoistilanteisiin. Tällä hetkellä peli kaatuu, jos valittu MIDI-tiedosto ei sisällä rumpuraitaa. Nyt MIDI-lukija vain avaa MIDI-tiedoston ja lähettää siitä listan nuotteja eteenpäin. Tulevaisuudessa voisi mahdollisesti tallentaa kappaleesta omaan formaattimuotoon tiedoston, jossa olisi enemmän tietoa, kuten paras soittotulos, soittokertojen määrä ja nuottikohtainen tarkkuus, josta näkisi, mitkä nuotit tuottavat eniten vaikeuksia kappaleen soitossa. Projektin MIDI-lukija ei tue uutta MIDI 2.0 -tyyppiä, eikä sen suunnitellusti tarvitsekaan.

7 Yhteenveto

MIDI-rajapinta ja MIDI-tiedostot ovat erinomainen tapa musiikillisen tiedon välittämiseen, ja MIDIä voisi käyttää muihinkin tarkoituksiin. MIDI-tiedoston toimintaperiaatteeseen ja rakenteeseen tutustuttiin onnistuneesti. Jokaisen MIDI-tapahtuman käyttämä tila tavuina tulee ottaa huomioon, vaikkei niitä tarvittaisi rumpupelissä.

MIDI-tiedoston lukemisessa käytettiin bittioperaatioita, jotka ovat matalan tason toimintoja ja erinomaisia yksityiskohtaisessa ohjelmoinnissa. Vaikka Godot-pelimoottorin GDScript on käytännössä korkean tason ohjelmointikieli, eli suuri osa matalamman tason ohjelmointitoiminnoista on piilotettu, voi GDScriptillä ohjelmoida myös bittioperaatioita.

MIDI-lukija avaa MIDI-tiedoston ja lukee otsikon, jonka jälkeen se alkaa lukea MIDI-tiedoston raitoja. Raidat luetaan toistosilmukalla, ja niistä tallennetaan mahdolliset tempomuutokset ja rumpunuotteja sisältävän raidan nuotit aikaleimoineen. Kun MIDI-tiedoston kaikki raidat on luettu, lähettää MIDI-lukija rumpunuotteja sisältävän listan eteenpäin pelin käyttöön.

MIDI-lukijan kehityksessä ja testaamisessa käytettiin apuna useita ohjelmia, joista mainittavimpia ovat Hxd-heksaeditori ja Guitar Pro -nuotitusohjelma, jota käytettiin MIDI-tiedostojen luomiseen sekä avaamiseen tiedostojen tarkistusta varten.

Lähteet

49 SL MK III MIDI-kontrolleri. Verkkoaineisto. F-Musiikki. <https://f-musiikki.fi/collections/midi-kontrollerit/products/49-sl-mk-iii-no49slmkiii>. Luettu 3.3.2025.

Back, David. 1999. Standard MIDI-File Format Spec. 1.1. Verkkoaineisto. <https://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html>. Luettu 25.2.2025.

Godot Engine. Pelimoottorin kotisivut. Verkkoaineisto. Godot. <https://godotengine.org/>. Luettu 27.3.2025.

Huber 1. Huber D. M. 2020. The MIDI Manual. Verkkoaineisto. <https://learning.oreilly.com/library/view/the-midi-manual/9780240807980/06_ch01.xhtml#sec1>. Luettu 7.3.2025.

Huber 2. Huber D. M. 2020. The MIDI Manual. Verkkoaineisto. <https://learning.oreilly.com/library/view/the-midi-manual/9780240807980/06_ch01.xhtml#sec2>. Luettu 7.3.2025.

InputEventMIDI. Verkkoaineisto. Godot. <https://docs.godotengine.org/en/stable/classes/class_inputeventmidi.html>. Luettu 27.3.2025.

javidx9. 2020. Programming MIDI. Verkkoaineisto. <https://www.youtube.com/watch?v=040BKtnDdg0>. Katsottu 28.3.2025.

MIDI Events. Verkkoaineisto. Mixage. <https://www.mixagesoftware.com/en/midikit/help/>. Luettu 6.3.2025.

Rock Band. Verkkoaineisto. Harmonix. <https://www.rockband4.com/>. Luettu 11.3.2025.

Rocksmith. Verkkoaineisto. Ubisoft. <https://www.ubisoft.com/en-gb/game/rocksmith/plus>. Luettu 11.3.2025.

What is Endianness?. 2024. Big-Endian & Little-Endian. Verkkoaineisto. Geeksforgeeks. <https://www.geeksforgeeks.org/little-and-big-endian-mystery/>. Päivitetty 23.5.2024. Luettu 1.3.2025.

Yamaha DTX402. Verkkoaineisto. Yamaha. <https://fi.yamaha.com/fi/products/musical_instruments/drums/el_drums/drum_kits/dtx402_series/products.html>. Luettu 11.3.2025.

Liite 1 MIDI-rumpunuotit

- 35 Acoustic Bass Drum or Low Bass Drum
- 36 Electric Bass Drum or High Bass Drum
- 37 Side Stick
- 38 Acoustic Snare
- 39 Hand Clap
- 40 Electric Snare or Rimshot
- 41 Low Floor Tom
- 42 Closed Hi-hat
- 43 High Floor Tom
- 44 Pedal Hi-hat
- 45 Low Tom
- 46 Open Hi-hat
- 47 Low-Mid Tom
- 48 High-Mid Tom
- 49 Crash Cymbal 1
- 50 High Tom
- 51 Ride Cymbal 1
- 52 Chinese Cymbal
- 53 Ride Bell
- 54 Tambourine
- 55 Splash Cymbal
- 56 Cowbell
- 57 Crash Cymbal 2
- 58 Vibraslap
- 59 Ride Cymbal 2
- 60 High Bongo
- 61 Low Bongo
- 62 Mute High Conga
- 63 Open High Conga

- 64 Low Conga
- 65 High Timbale
- 66 Low Timbale
- 67 High Agogô
- 68 Low Agogô
- 69 Cabasa
- 70 Maracas
- 71 Short Whistle
- 72 Long Whistle
- 73 Short Güiro
- 74 Long Güiro
- 75 Claves
- 76 High Woodblock
- 77 Low Woodblock
- 78 Mute Cuíca
- 79 Open Cuíca
- 80 Mute Triangle
- 81 Open Triangle