MUSIIKIN NUOTINNUS VOICE NOTER -MOBIILISOVELLUKSELLA

Juho Itä

Opinnäytetyö
Toukokuu 2015
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

ITÄ, JUHO:
Musiikin nuotinnus Voice Noter -mobiilisovelluksella

Opinnäytetyö 56 sivua, joista liitteitä 21 sivua
Toukokuu 2015

Voice Noter -mobiilisovellus on Herwood Technologies Oy:n ja Kangain Oy:n kehittämä laulamisen harjoitteluohjelma Android-käyttöjärjestelmälle. Kun laulaja laulaa melodiaa, sovellus näyttää sävelet mobiililaitteen näytöllä vaakapalkkeina. Näytön näkymä on nuottiviivaston kaltainen; ruudun alareunassa on matalat äänet, ja ylälaidassa on korkeat äänet. Esimerkiksi Playstation SingStar -pelissä on samantapainen käyttöliittymä.

Työssä kehitettiin sovellukseen ominaisuus, joka nuotintaa ohjelmalle lauletun tai soitetun melodian. Tavoitteeksi asetettiin, että uusi ominaisuus tunnistaa ja nuotintaa yksiäänisiä melodioita. Android-sovellukset ohjelmoidaan tavallisesti Java-ohjelmointikielellä. Digitaalinen signaalinkäsittely tehdään kuitenkin suoritusnopeuden parantamiseksi yleensä C- tai C++-ohjelmointikielellä; Voice Noter:ssa käytetään C++-kieltä. Tutkimuksessa pohdittiin myös nuotinnustoiminnon käyttöliittymän toteutustapaa; monimutkainen käyttöliittymä on vaikea toteuttaa mobiililaitteella.

Voice Noter tallettaa äänisignaalin datan bittikarttaan, josta ohjelma piirtää sävelet nopeasti mobiililaitteen ruudulle. Tutkimuksen aikana havaittiin, että bittikartan 49 Hz:n näytteenottotaajuus riittää neljäsosanuottien tunnistamiseen tempolla 120 iskua minuutissa. Kahdeksasosanuotin tai lyhempien nuottien tunnistamiseen vaaditaan suurempi näytteenottotaajuus. Työn tuloksena kehitetiin algoritmi, joka tunnistaa ja nuotintaa lastenlaulujen tasoisia yksiäänisiä melodioita; testaus suoritettiin soittamalla melodioita sähkökitaralla. Algoritmi muuntaa bittikartan näytteet MIDI-tiedostoksi, jota pystyy muokkaamaan tietokoneella; tietokoneella voidaan esimerkiksi lisätä kappaleeseen soinnut ja tämän jälkeen tallettaa PDF-nuotiksi. Yksinkertaisella koodin lisäyksellä ohjelma saadaan tallentamaan nuotti LilyPond-tekstitiedostona, jolloin ilmaisella LilyPond-nuotinnusohjelmalla varustetulla tietokoneella tiedosto voidaan muuntaa kahdella hiirenpainalluksella PDF-nuotiksi.

Kehittämisideana on nuottien aika-arvojen tunnistuksen parantaminen näytteenottotaajuutta lisäämällä tai hyödyntämällä Voice Noter:n toista uutta ominaisuutta - tempoharjoitustoimintoa. Toiminto etsii melodiasta rytmin ja kertoo käyttäjälle, pysyykö hän tempossa. Rytmin tunnistuksen tekniikoita pystyy todennäköisesti hyödyntämään nuotin alkamishetken määrittämisessä. Tutkimuksen nuotinnusalgoritmi tekee nuotin aina samalla sävel- ja tahtilajilla sekä tempolla. Algoritmia voidaan jalostaa nuotintamaan myös muilla asetuksilla. Lisäksi kielisoittimilla on soittotekniikoita, joista algoritmi ei pysty tunnistamaan nuotteja. Esimerkiksi liu'utettujen ja venytettyjen sävelien tunnistamiseen tarvitaan suuri näytteenottotaajuus.

Asiasanat: digitaalinen signaalinkäsittely, algoritmi, nuotinnus, ohjelmointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT
Embedded Systems and Electronics

ITÄ, JUHO:
Producing Sheet Music with Voice Noter Mobile Application

Bachelor's thesis 56 pages, appendices 21 pages
May 2015

Voice Noter is a mobile application for singing training. The application is developed by Herwood Technologies and Kangain companies. It runs on the Android operating system. The user interface is similar to the Playstation SingStar game's interface. When a singer sings a melody, notes are displayed on the mobile device's screen as horizontal beams. Higher notes are displayed at the top of the screen and lower notes are at the bottom.

In this work, a new software feature was developed for producing sheet music from sung or played melody. The goal was to recognize and make music notation from monophonic music. Usually Android applications are programmed with Java language, but in digital signal processing C or C++ programming language is used for better performance. Voice Noter's signal processing is programmed with C++. Development of the user interface was also studied, though implementation of the user interface was not part of the thesis.

Voice Noter stores audio signal data to a bitmap from which the program draws notes quickly to the screen of the device. During the research it was discovered that the bitmap's 49 Hz sampling rate is enough to recognize quarter notes at a tempo of 120 beats per minute. To recognize eight notes or shorter notes the sampling rate must be higher. As a result of this thesis an algorithm was developed. The algorithm recognizes notes and produces a MIDI file from monophonic children's songs. Testing was done by playing melodies with an electric guitar. The algorithm converts bitmap's samples into a MIDI file, which can be edited with a computer. For example one can add chords to the song with a music notation program and export the file as a PDF sheet music. A simple addition to the program code would enable exporting a text file which contains sheet music information in LilyPond syntax. With a computer one could convert the text file to PDF sheet music with two mouse clicks. LilyPond is a free text based music notation program.

In future work, the algorithm could be improved by increasing the sampling rate or by making use of Voice Noter's another new feature - the beat analyzer. The beat analyzer searches for the rhythm of the melody and gives feedback if the user is in tempo or not. The techniques used in rhythm recognition can probably be used for determining the beginning of a note. The algorithm developed in this thesis always produces sheet music with same configuration - tempo and time signature are the same. The program can be developed to produce sheet music with other configurations. With string instruments there are playing techniques like sliding and bending that currently cannot be recognized. In order to recognize these cases, the sampling rate must be increased.

Key words: digital signal processing, algorithm, sheet music, programming

ESIPUHE

Haluan kiittää työnvalmistumisesta työnohjaajaa yliopettaja Mauri Inhaa, Voice Noter -tiimiä Erkki Salosta, Reima Piililää, Ville Kankaista ja Anita Lindströmiä, oikolukijoita Jouni Harjumäkeä, Atle Kivelää, Alexandria Schultzia, Laura Kuismaa, Olli Oksaa ja Eric Malmia sekä äitiä.

Lisäksi haluan kiittää Kalmah-, Slipknot-, Medeia-, Nekrogoblikon-, Parov Stelar - ja Leevi and the Leavings -artisteja sekä -yhtyeitä hyvästä musiikista, jota kuunnellessa tutkimus ja opiskelu ovat edenneet.

SISÄLLYS

1 JOHDANTO

Ääni on aaltoliikettä, joka syntyy väliaineen värähtelystä. Korva tulkitsee äänen tärykalvon värähtelyn avulla. Samalla tavalla myös mikrofonissa on kalvo, joka havaitsee ilmanpaineen vaihtelun. Mikrofonin kalvon värähtely muunnetaan sähkösignaaliksi, ja sähkösignaali muunnetaan A/D-muuntimen eli analogia-digitaalimuuntimen avulla tietokoneella tulkattavaan digitaaliseen muotoon. Tietokoneen avulla bittisarjat voidaan nuotintaa automaattisesti, jolloin ihminen voi helposti lukea ja ymmärtää nuotin.

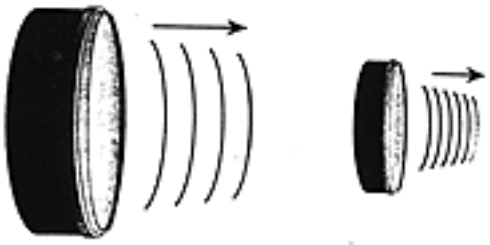
Työssä tutkittiin, miten mobiililaitteella pystytään nuotintamaan musiikkia. Tavoitteeksi asetettiin nuotintaa yksiäänisiä melodioita. Tutkimuksen nuotinnustoiminto toteutettiin Voice Noter -mobiilisovellukseen, joka on Herwood Technologies Oy:n ja Kangain Oy:n kehittämä laulamisen harjoitteluohjelma. Nuotinnus toteutettiin MIDI-tiedostoksi (Musical Instrument Digital Interface). MIDI-tiedoston voi viedä tietokoneelle, jolloin tiedostoon voidaan lisätä esimerkiksi soinnut tietokoneen nuotinnusohjelmalla. Tietokoneen nuotinnusohjelmalla MIDI-tiedosto pystytään muuntamaan PDF-nuotiksi. Tavoitteena oli myös, että nuotti saataisiin mahdollisimman helposti tulostettavaksi. Tähän tarkoitukseen hyödynnettiin tekstipohjaista LilyPond-nuotinnusohjelmaa. LilyPond:lla pystyy generoimaan PDF-nuotteja kahdella hiirenpainalluksella.

Työhön kuului signaalinkäsittelyn teoriaa, käyttöliittymäsuunnittelua, MIDI-ohjelmointia, C++-ohjelmointia ja algoritmisuunnittelua. Työvälineinä käytettiin Windows 7 -käyttöjärjestelmällä varustettua kannettavaa tietokonetta, Samsung Galaxy S1 ja S3 -matkapuhelimia, Samsung Galaxy Tab 2 7.0 -tablettia, Logitech 2.1 -kaiutinjärjestelmää, Line6 GuitarPort -laitetta, Code::Blocks integroitua ohjelmointiympäristöä, Eclipse integroitua ohjelmistoympäristöä, Git-versionhallintaohjelmistoa, Audacity-äänitysohjelmaa, GuitarPro 6, TablEdit ja LilyPond -nuotinnusohjelmistoja.

2 TEORIA

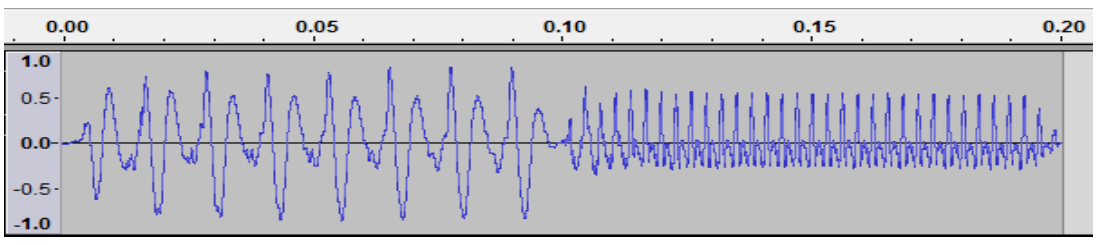
2.1 Ääni fysikaalisena ilmiönä

Ääni on aaltoliikettä, joka syntyy kappaleen tai aineen värähtelystä. Ääniaallot etenevät väliaineessa. Väliaine voi olla neste, kaasu, kiinteä aine tai plasma. Yleisin väliaine on ilma; korva havaitsee äänen ilmanpaineen vaihteluna. Ääniaalloilla on aina sama väliaineesta riippuva nopeus. Kappaleen värähtelynopeus ei vaikuta äänen etenemisnopeuteen. Kun esine värähtelee nopeasti, se saa aikaan toisiaan lähellä olevia ääniaaltoja. Nopea värähtely kuullaan korkeina ääninä. Hitaasti värähtelevä esine saa aikaan toisistaan kaukana olevia ääniaaltoja, jotka kuullaan matalina ääninä. Kuvassa 1 on esitetty, miten suuremmassa rummussa ääniaallot ovat kauempana toisistaan kuin pienemmässä rummussa. Värähtelyn nopeutta kuvataan käsitteellä *taajuus*; taajuus kertoo, kuinka monta kertaa esine värähtää sekunnissa. (Laaksonen 2006, 4-5, 7; Laakso.)



KUVA 1. Ääniaaltojen koko ja tiheys (Kuva: Laakso, muokattu)

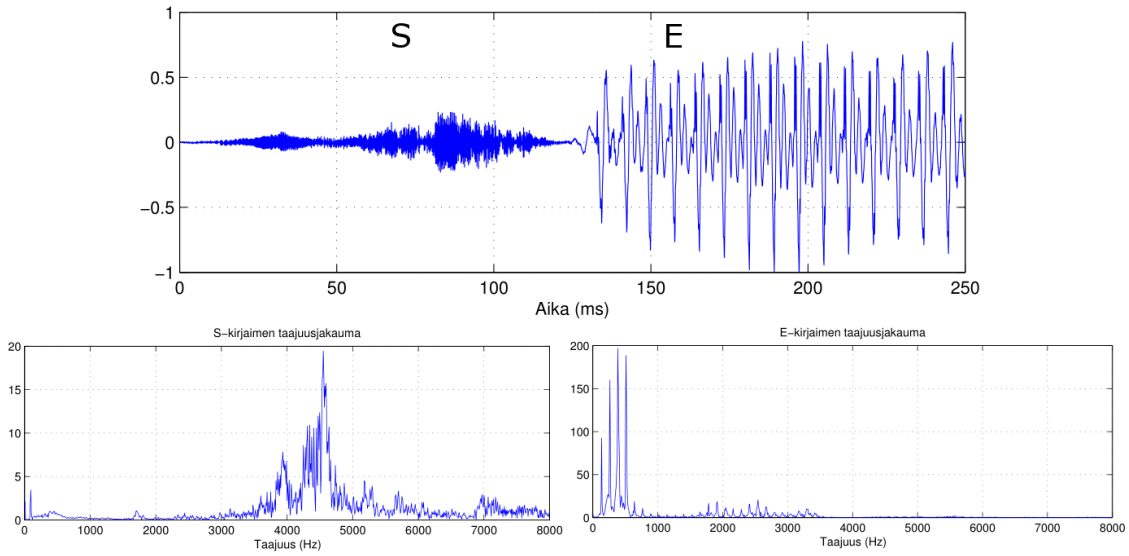
Kuvassa 2 on Audacity-ohjelmalla äänitetty äänisignaali, kun kitaralla on soitettu kaksi nuottia. Audiosignaali esitetään kuvassa aika-amplitudi-taossa. Ensimmäinen nuotti on soitettu kitaran sävelkorkeudeltaan matalimmalla kielellä ja 0,10 sekunnin kohdalla alkava toinen nuotti on soitettu kitaran sävelkorkeudeltaan korkeimmalla kielellä. Kuvaajasta näkee, kuinka matalan nuotin amplitudi on suurempi ja käyrä on harvempi kuin korkeamman nuotin.



KUVA 2. Kaksi kitaralla soitettua nuottia

Äänestä pystytään tunnistamaan paljon erilaisia piirteitä digitaalisen signaalinkäsittelyn avulla. Esimerkiksi puheentunnistus on yksi nopeasti kehittyvistä aloista. Jo nykyisin tietokoneille on kehitetty ohjelmistoja, jotka pystyvät kääntämään puheen toiselle kielelle reaaliajassa (Microsoft Research 2014).

Kuvassa 3 on havainnollistettu S- ja E-äänteiden tunnistaminen. Ylimmässä kuvassa signaalit on esitetty aikatasossa, kun alemmat kuvaajat ovat Fourier-muunnoksen tuloksia; kappaleessa 2.3 kerrotaan tarkemmin Fourier-muunnoksesta. Kun äänisignaalille tehdään Fourier-muunnos, tulokseksi saadaan tieto, kuinka paljon eri taajuuskomponentteja signaali sisältää. Taajuusjakaumista nähdään, että S-äänne sisältää eniten taajuuskomponentteja 4 ja 5 kHz:n välillä. E-äänne sisältää eniten taajuuskomponentteja 200 ja 700 Hz:n välillä. (Huttunen 2014, 2–3.)



KUVA 3. S- ja E-äänteet (Kuva: Huttunen 2014, 2–3, muokattu)

2.2 Äänen digitointi – A/D-muunnos

Analoginen ääni muunnetaan analogia-digitaalimuuntimella digitaaliseen muotoon, jotta signaalia pystytään käsittelemään digitaalisesti. Ääntä käsitellään usein digitaalisesti, koska suuren datamäärän käsittely on nopeaa ja halpaa sekä tarkkuus on varmaa. Mikropiirit ovat kehittyneet viime vuosikymmeninä hurjaa vauhtia; prosessorien suoritusnopeudet ovat kasvaneet valtavasti. Lisäksi digitaalitekniikassa suorituskyvyt eivät muutu; lämpötila, ilmanpaine, kosteus eivät vaikuta. Digitaalisessa signaalinkäsittelyssä datasta voidaan poistaa tarpeeton tieto, jolloin muistinkäyttö ja suorituskyky paranevat. Mikro-

piirien signaalinprosessointialgoritmeja voidaan muuttaa nappia painamalla, jolloin käsittelyn nopeus korostuu. (Jokinen 1998, 17–20.) Analoginen signaalinkäsittely voi olla hyvä vaihtoehto yksinkertaisissa sovelluksissa, kun sovellukseen ei tarvita kuin muutama komponentti, esimerkiksi yli- tai alipäästösuodin; tällöin toteutus on todennäköisesti myös halvempi kuin digitaalinen.

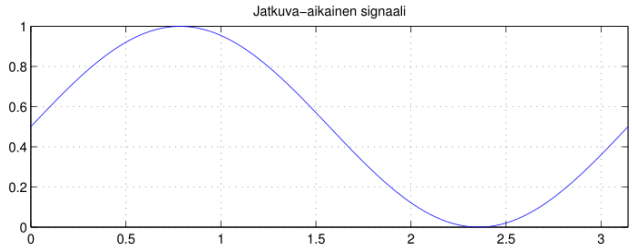
Kuvissa 4, 5 ja 6 on esitetty analogisen signaalin muunnos digitaaliseksi signaaliksi. Pystyakselilla on signaalin amplitudi, ja vaaka-akselilla on aika. Kuvassa 4 on analoginen signaali. Analogisesta signaalista otetaan Nyquist-Shannon-teoreeman mukaan näytteitä kaksinkertaisella taajuudella alkuperäisen signaalin taajuuteen verrattuna, jotta informaatiota ei katoa (Huttunen 2014, 3–4). Teoreeman yhtälö on

$$F_s = 2 * F_a\,, \qquad (1)$$

missä

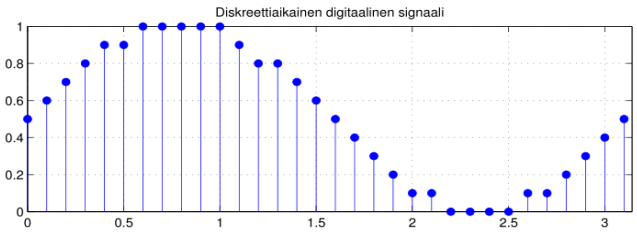
F_s = näytteenottotaajuus

F_a = alkuperäisen signaalin suurin taajuuskomponentti, Nyquistin taajuus



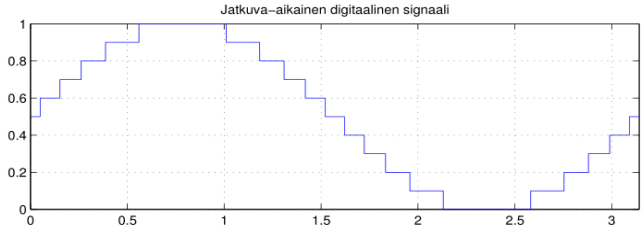
KUVA 4. Jatkuva-aikainen analoginen signaali (Kuva: Huttunen 2014, 11)

Kun analogisesta signaalista otetut näytteet laitetaan peräkkäin, saadaan kuvan 5 mukainen diskreettiaikainen signaali.



KUVA 5. Diskreettiaikainen digitaalinen signaali (Kuva: Huttunen 2014, 11)

Kun diskreettiaikaisesta signaalista rekonstruoidaan jatkuva-aikainen digitaalinen signaali, rekonstruoitu signaali on aina approksimaatio alkuperäisestä signaalista. Kuvassa 6 näkyy, kuinka rekonstruoitu signaali on sahalaitainen.

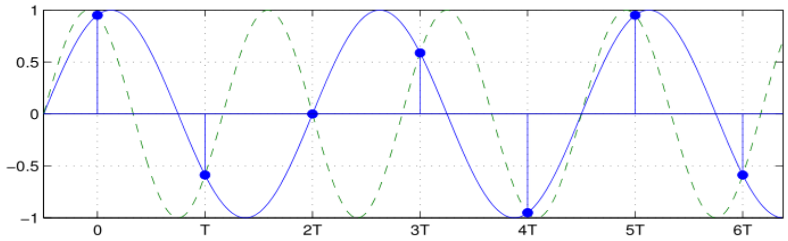


KUVA 6. Jatkuva-aikainen digitaalinen signaali (Kuva: Huttunen 2014, 11)

Ihmisen normaali kuuloalue on 20 – 22 000 Hz. CD-soittimen näytteenottotaajuus on 44.1 kHz. Tyypillinen näytteenottotaajuus nykyaikaisissa äänikorteissa on 48 kHz. Molemmat näytteenottotaajuudet ovat vähintään kaksi kertaa ihmisen kuuloalueen yläraja 22 kHz, jolloin näytteistä pystytään rekonstruoimaan ihmiskorvalle alkuperäistä analogisignaalia vastaava signaali. (Laaksonen 2006, 73; Laakso; Huttunen 2014, 123.)

Kun näytteitä otetaan analogisesta signaalista, amplitudin voimakkuus pitää tallentaa. Tallennus voi tapahtua esimerkiksi 8-bittisenä, jolloin amplitudi voidaan tallettaa välillä 0-255. 16-bittinen tallennus mahdollistaa tallennuksen lukuvälillä 0–65 535. Edellä mainituissa lukuväleissä nolla tarkoittaa suurinta negatiivista jännitettä, eli äänen suurinta alipainetta. Kuten lukuväleistä voi päätellä, 16-bittinen tallennus on paljon tarkempi kuin 8-bittinen. Kun signaalin amplitudi pyöristetään lähimpään *kvantisoituun* tasoon, tapahtuu vääristymää. *Kvantisointitaso* tarkoittaa lukuarvoa, joka riippuu äänitysresoluutiosta. 16-bitin resoluutiolla kvantisointitasoja on $2^{16} = 65\,536$. Todellisen analogisen arvon ja kvantisoindun tason eroa kutsutaan *kvantisointivirheeksi*. Nykyaikaisissa äänitykseen tarkoitetuissa äänikorteissa äänitysresoluutio on vähintään 24 bittiä. (Laakso; Klapuri & Virtanen, 5–6; Laaksonen 2006, 71–72, 84.)

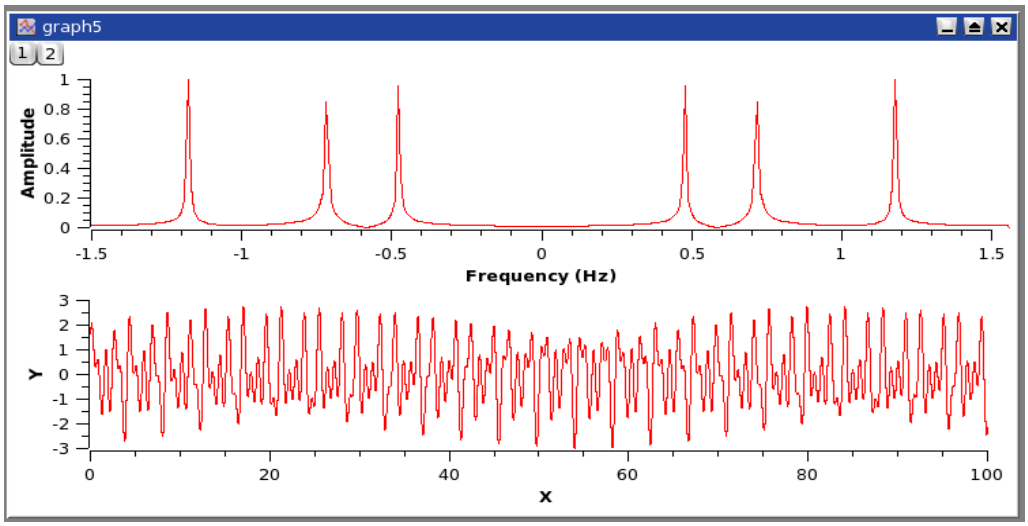
Jos äänityksessä ei käytetä yhtälön 1 mukaisesti tarpeeksi suurta näytteenottotaajuutta, tapahtuu *laskostumista* eli *alinäytteistymistä* (kuva 7). Alkuperäinen signaali on merkitty katkoviivalla ja rekonstruoitu signaali on merkitty yhtenäisellä viivalla. Kuvan molemmilla signaaleilla on samat näytearvot. Kun alkuperäinen signaali rekonstruoidaan jälleen analogiseksi signaaliksi, tulokseksi saadaan uusi pienempitaajuinen signaali. (Huttunen 2014, 4; Laaksonen 2006, 68.)



KUVA 7. Laskostuminen (Kuva: Huttunen 2014)

2.3 Fourier-muunnos

Fourier-muunnos on yksi tärkeimmistä työkaluista signaalinkäsittelyssä. Fourier-muunnoksen avulla signaalista saadaan tieto, kuinka paljon eri taajuuskomponentteja signaalissa on. FFT eli Fast Fourier Transform on Fourier-muunnoksen nopea algoritmi, joka on erittäin tärkeä sovellusten kannalta. (Huttunen 2014, 33–47.) Kuvassa 8 on esitetty signaali aika-amplitudi-tasossa ja sen Fourier-muunnos taajuus-teho-tasossa. Ylemmästä kuvaajasta nähdään, kuinka alemman kuvaajan signaalin taajuuskomponentit ovat selkeästi kolmella alueella. Ylempi kuvaaja on symmetrinen; vain toinen puoli nollasta tarvittaisiin, jotta informaatio on luettavissa.

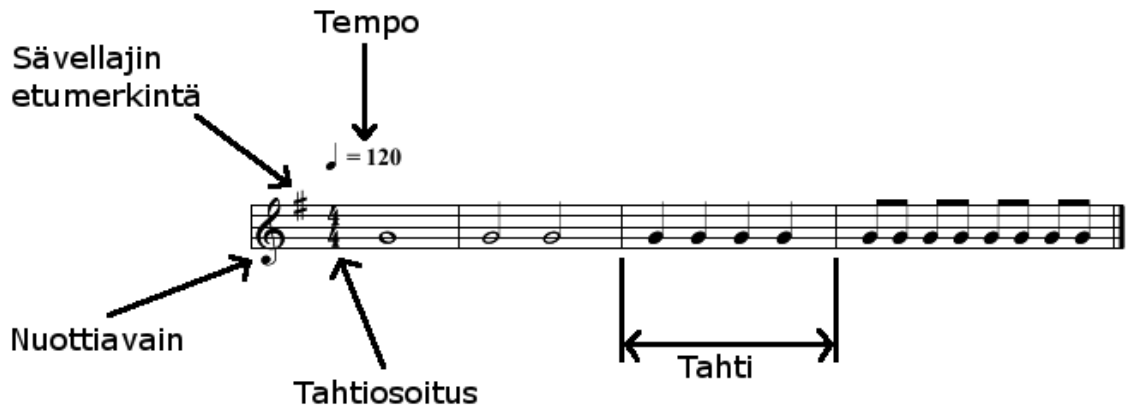


KUVA 8. Fourier-muunnos (Kuva: Gadiou, Franke & Vasilief)

Fourier-muunnoksen tuloksesta voidaan tehdä johtopäätöksiä, että tarvitaan tietynlaisia suodattimia tai joitakin taajuuksia täytyy korostaa. Musiikissa muunnosta voidaan hyödyntää soittimien äänialojen selvittämisessä. Esimerkiksi bassolla ja viululla on täysin eri äänialat. Jos signaalista halutaan esille vain viulun melodia, bassotaajuudet voidaan suodattaa pois. Jos taas bassomelodia halutaan säilyttää, viulun korkeat taajuudet voidaan suodattaa alipäästösuodatuksella pois. Myös pieniamplitudiset häiriöäänet voidaan suodattaa pois.

2.4 Musiikinteoria

Nuotit ovat musiikin kirjaimia. Nuotti voi tarkoittaa joko yksittäistä ääntä tai nuottikokonaisuutta. Kirjoitettuja nuotteja lukemalla soittaja voi toistaa musiikkikappaleita ilman, että soittaja on koskaan kuullut soitettavaa kappaletta. Nuottikirjoituksessa käytetään yhtenäistä syntaksia, jonka avulla muusikot eri puolilla voivat lukea musiikkia samalla tavalla. Kuvassa 9 on esitetty yksinkertaisen nuotin merkinnät.



KUVA 9. Nuottimerkinnät

Tempo on nopeus, jolla kappale esitetään. Kuvan merkintä $\text{♩} = 120$ tarkoittaa, että kappaleen neljäsosanuotteja mahtuu 120 yhteen minuuttiin. Tällöin yhden neljäsosanuotin pituudeksi saadaan 0,5 sekuntia. (Joutsenvirta & Perkiönmäki 2007, Rytmi.)

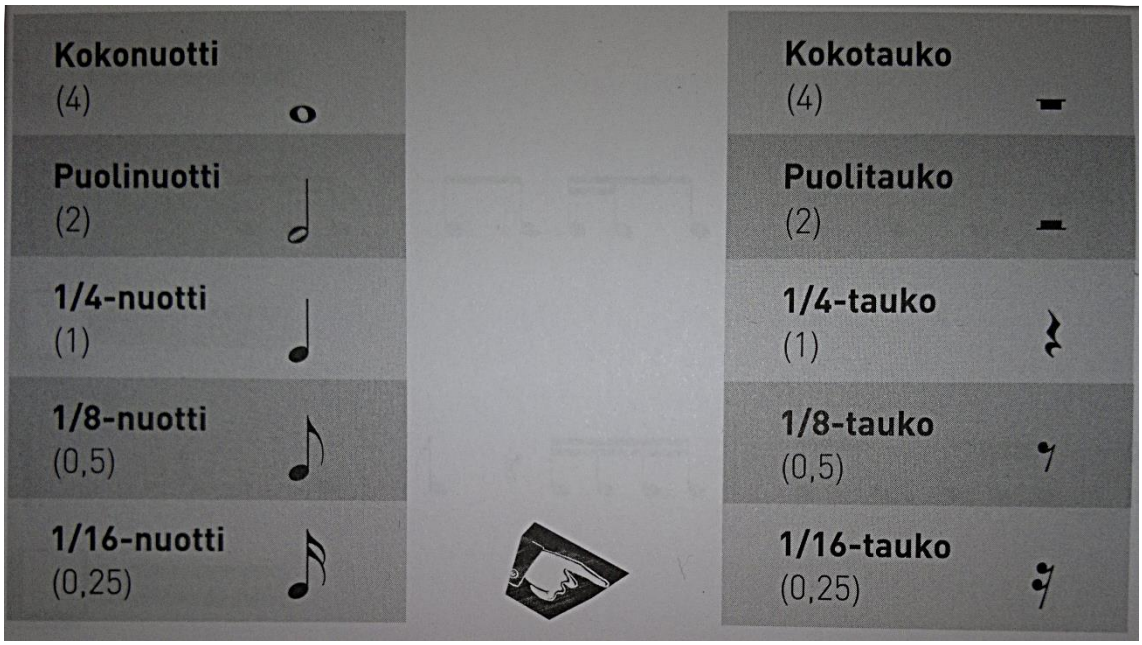
Nuottiavain kertoo, missä tietty sävel on nuottiviivastolla. Yleisin nuottiavain on diskantti- eli G-avain, joka osoittaa g-sävelen paikan viivastolla; kuvassa on G-avain. Eri soittimilla on eri äänialat, joille on eri nuottiavaimia. Esimerkiksi basson nuotit on hyvä kirjoittaa basso- eli F-avaimella, jolloin sävelet mahtuvat nuottiviivastolle paremmin kuin diskanttiavaimella. (Heikkilä & Halkosalmi 2005, 89, 92.)

Sävellaji kertoo nuotin perussävelen. Nuottiin voidaan merkitä sävellaji etumerkinnällä tai tilapäisetumerkein. Kun kappaleeseen kirjoitetaan sävellajin etumerkintä, jokaista ylennys- tai alennusmerkkiä ei tarvitse kirjoittaa nuottien eteen erikseen. (Heikkilä & Halkosalmi 2005, 98.) Sävellajin merkitseminen nuottiviivaston alkuun helpottaa nuotin kirjoitusta ja auttaa soittajaa löytämään muun muassa kappaleeseen sopivat sormitukset.

Tahtilaji kertoo, kuinka monta iskua yhdessä tahdissa on. Tahtilaji kerrotaan kappaleen alussa **tahtiosoituksella**. $\frac{4}{4}$ merkintä tarkoittaa, että yhdessä tahdissa on neljä neljäsosanuottia. Merkintä voisi olla myös esimerkiksi $\frac{6}{8}$, jolloin yhdessä tahdissa on kuusi kahdeksasosanuottia. (Heikkilä & Halkosalmi 2005, 11, 26.)

Tahti on nuottikokonaisuus, jonka pituus määritellään tahtilajilla. Kappaleen jakaminen tahteihin helpottaa nuotin lukemista. Tahti on kuin tekstin iso alkukirjain ja piste. Ilman isoa alkukirjainta ja pistettä tekstiä pystyy lukemaan, mutta se on hyvin vaikeaa.

Nuotin **aika-arvo** kertoo, kuinka pitkään nuottia soitetaan suhteessa toisiin nuotteihin. Puolinuotti on puolet kokonuotista; neljäsosanuotti on puolet puolinuotista ja niin edelleen. Kuvassa 10 on esitetty nuottiarvot, ja niitä vastaavat taukojen aika-arvot.



KUVA 10. Nuotit ja tauot (Kuva: Heikkilä & Halkosalmi 2005, 40)

2.5 MIDI

MIDI eli Musical Instrument Digital Interface on digitaalinen rajapinta musiikkisoittimille. MIDI ei sisällä analogista äänisignaalia vaan tieto kulkee bitteinä laitteelta toiselle. MIDI-teknologiaa on käytössä lähes kaikkialla, missä ollaan tekemisissä esittävän taiteen kanssa. Teknologia on käytössä soittimissa, tietokoneissa, tableteissa, puhelimissa, valoja esitystekniikassa, audiomiksereissä ja monissa muissa laitteissa. MIDI:n avulla voidaan lähettää tieto mitä ja miten pitää soittaa. (Learn About MIDI.)

Yksinkertaistettuna MIDI:n voi ajatella olevan laitteiden nuottitieto. Soittimet generoivat MIDI-datasta äänisignaalin, joka toistetaan niin hyvin kuin laitteet pystyvät. Eri laitteissa on eri kuuloisia ääninäytteitä, jolloin sama data saattaa kuulostaa erilaiselta eri instrumenteilla. Hyvin usein pätee sääntö, mitä kalliimpi laite sitä aidomman kuuloinen ääni pystytään generoimaan.

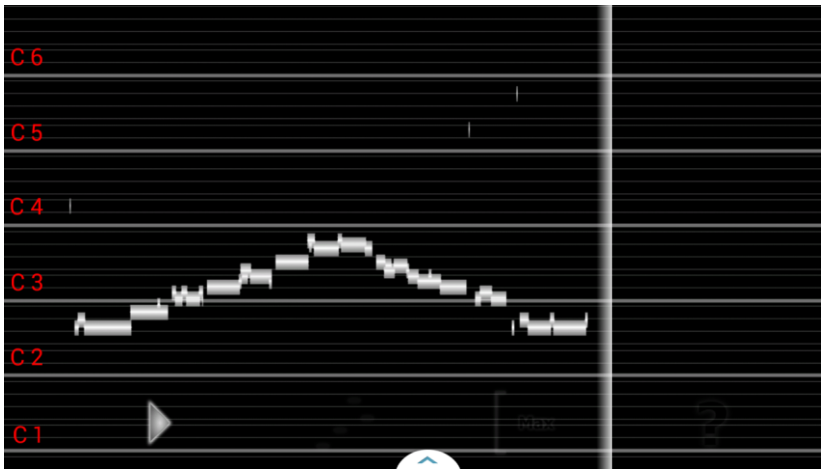
Työssä käytettiin MIDI-formaattia nuottien tallennuksessa, koska tällöin nuotit voidaan siirtää mobiililaitteelta tietokoneelle muokattavaksi. Esimerkiksi nuottiin voidaan lisätä soinnut. MIDI-formaatti on niin yleinen, että käytännössä kaikki nuotinnusohjelmat tukevat sitä.

Jos MIDI:n ohjelmointiin ei käytetä valmiita koodikirjastoja, ohjelmointi tapahtuu todella lähellä laitteistoa. MIDI on alun perin kehitetty oikeiden soitinten kommunikoinnin helpottamiseksi. Esitystilanteessa instrumenttien välinen viestien vaihto täytyy olla erittäin nopeaa, ettei kuulija havaitse viivettä musiikissa ja soittimet pysyvät samassa tahdissa. MIDI-standardi on selostettu hyvin kirjassa MIDI 1.0 Musiikkilaitteiden tiedonsiirtostandardi (Romanowski 1990).

3 VOICE NOTER -SOVELLUS

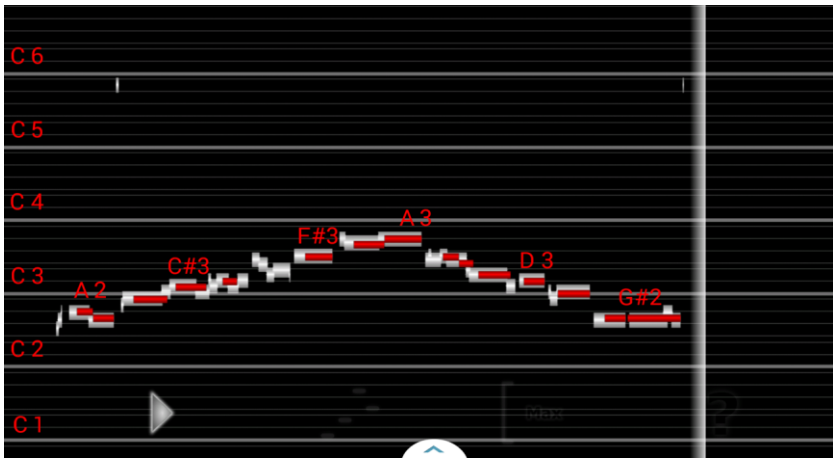
3.1 Voice Noter:n kehitys

Voice Noter -mobiilisovellus on Herwood Technologies Oy:n ja Kangain Oy:n kehittämä laulamisen harjoitteluohjelma Android-käyttöjärjestelmälle. Ohjelmisto äänittää mobiililaitteen mikrofonilla ympäristön ääntä reaaliaikaisesti. Äänisignaali suodatetaan ja käsitellään sovelluksen algoritmeilla. Signaalista tuotetaan bittikartta, joka piirretään käyttäjän näkyviin, kuten kuvassa 11 näkyy. Käyttäjän näkymä on samantapainen kuin PlayStation SingStar -pelissä.



KUVA 11. Voice Noter -bittikartta

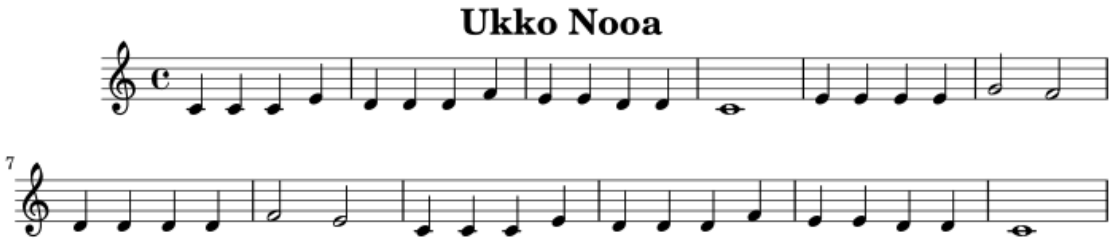
Bittikartan dataa tulkitaan ja etsitään nuotit harhaäänien seasta. Nuotit korostetaan punaisella viivalla ja nuotin nimi esitetään tekstinä. (Kuva 12.)



KUVA 12. Bittikartan tulkinta

3.2 Nuotinnusominaisuus

Työssä kehitettiin ominaisuus, joka tulkitsee bittikartan nuoteiksi ja muodostaa yksittäisistä nuoteista nuottiviivastoon sopivan kokonaisuuden. Sovituksessa täytyi ottaa huomioon kappaleen tempo, nuottien aika-arvot, tahtilaji ja sävellaji. Nuotti vastaa ulkoasultaan kuvan 13 mukaista nuotinnusta.



KUVA 13. Ukko Nooa -nuotti

Kuvan 13 nuotti on tuotettu alla olevasta tekstistä (kuva 14). Teksti on kirjoitettu LilyPond-ohjelman syntaksilla. LilyPond-tekstitiedostosta voidaan generoida PDF-nuotti kahdella hiirenpainalluksella Windows-tietokoneella; LilyPond-ohjelmisto toimii myös Linuxilla; Linuxille on tehty ohjelmistoon tekstikäyttöliittymän lisäksi graafinen käyttöliittymä. (LilyPond-ohjelmisto.)

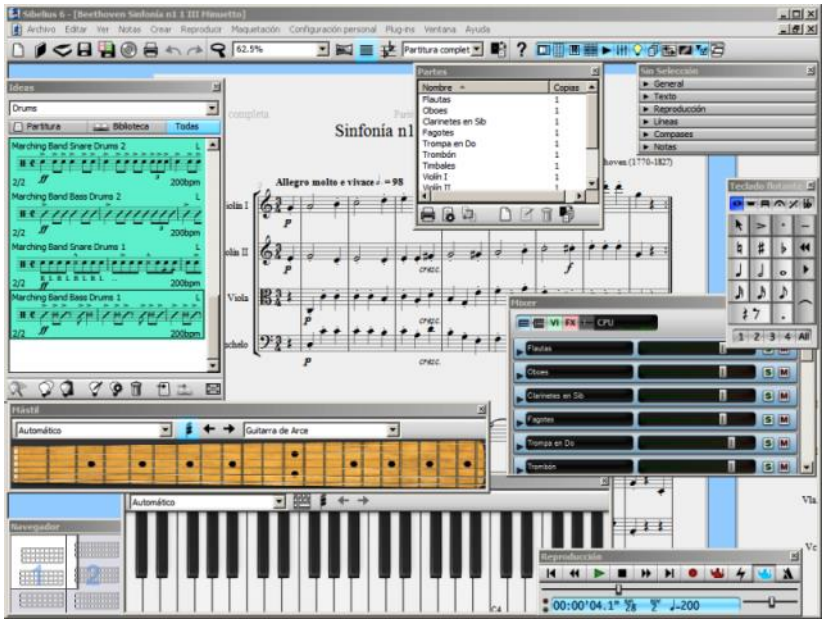
```
\header{
title = "Ukko Nooa"
                }
\tempo 4 = 120
\relative c'
{
                c c c e
                d d d f
                e e d d
                c1
                e4 e e e
                g2   f
                d4 d d d
                f2   e
                c4 c c e
                d d d f
                e e d d
                c1
}
```

KUVA 14. Ukko Nooa LilyPond-syntaksi

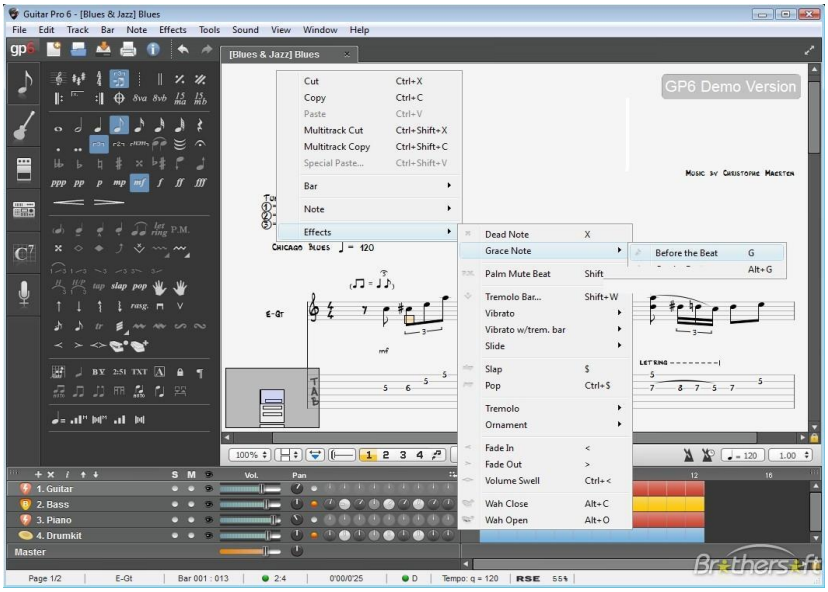
LilyPond-ohjelma havaittiin hyväksi työkaluksi, ja sen syntaksin pohjalta kehitettiin SheetMusic-luokka (liite 2), joka sisältää Note-olioita (liite 3) – Note-oliot vastaavat nuotteja. Tarkoituksena oli, että SheetMusic-oliosta on mahdollisimman helppo luoda LilyPond-syntaksinen tekstitiedosto, jonka pystyy muuntamaan helposti PDF-nuotiksi. Ohjelmistoon tehtiin funktio openFunctions (liite 4), joka tekee LilyPond-tiedostosta SheetMusic-olion. Ominaisuus tehtiin MIDI-tiedoston luonnin testausta varten. Funktion päinvastaisella toteutuksella pystyy tekemään LilyPond-tiedoston luonnin. Aikarajoituksen takia tätä ominaisuutta ei kuitenkaan ehditty toteuttamaan. Toiminnon ohjelmointi olisi kuitenkin yksinkertaisempaa kuin jo toteutetun LilyPond-SheetMusic-muunnoksen.

3.3 Nuotinnuksen käyttöliittymä

Tietokoneilla on paljon monipuolisia nuotinnusohjelmia, kuten Finale, Sibelius, GuitarPro. Mobiililaitteella on vaikea kilpailla näiden ohjelmien kanssa, koska monimutkaisten käyttöliittymien käyttäminen mobiililaitteella on vaikeaa. Kuvissa 15 ja 16 on esitetty Sibelius 6:n ja GuitarPro 6:n käyttöliittymät. Kuten kuvista huomataan, ohjelmistojen käyttöliittymiä olisi vaikea käyttää mobiililaitteen pienellä näytöllä. Tabletilla käyttö olisi ainakin jossain määrin mahdollista, ja Sibeliuksesta on versio iPad:lle. Voice Noter -ohjelmiston on kuitenkin tarkoitus toimia myös matkapuhelinten pienillä näytöillä.



KUVA 15. Sibelius 6



KUVA 16. GuitarPro 6

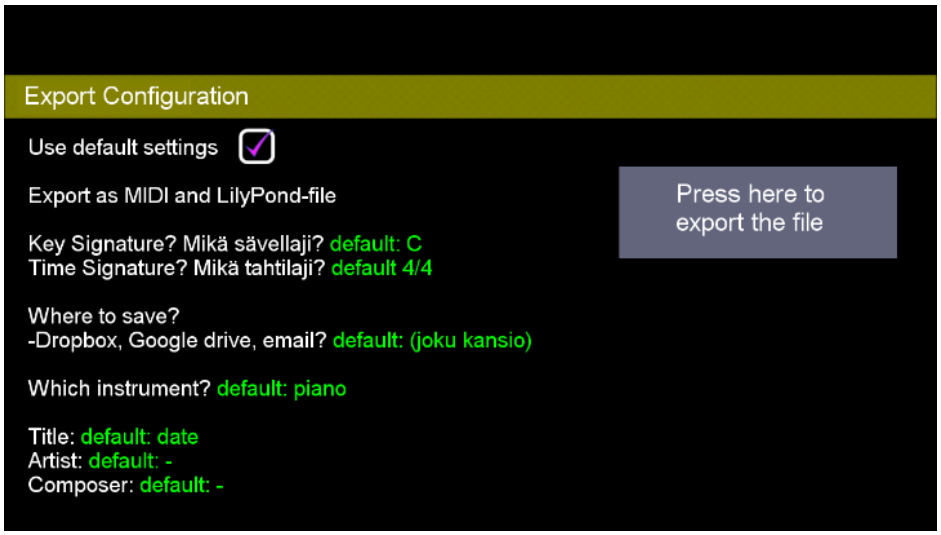
Kuvassa 17 on esitetty ensimmäinen suunnitteluversio tutkimuksessa laajennetusta käyttöliittymästä. Vihreällä ympäröidyt painikkeet ovat suunniteltu melodian nuotinnusta varten. Muut napit olivat Voice Noter:ssa jo ennen nuotinnustoiminnon kehittämisen aloittamista.

KUVA 17. Käyttöliittymä

Oikealla yläkulmassa on punainen pallo, joka on metronomia varten. Pallo vilkku tempon mukaan, joka määritellään liukukytkimen avulla. Liukukytkin on merkitty kuvassa nuottimerkillä ja numerolla 120. Luku tarkoittaa iskua minuutissa, eli kuinka monta kertaa valo vilkkuu minuutin aikana. Metronomi tarvitaan, jotta tiedetään, mihin tempoon nuotti kirjoitetaan. Valon ja tempomerkinnän vasemmalla puolella on viritystaajuuden valinta.

Yleensä pop-musiikissa käytetään taajuutta 440 Hz, mutta esimerkiksi Sibelius-Akatemialla käytetään yleisesti viritystä 442 Hz (Joutsenvirta 2005). Maailmalla käytetään myös muita viritystaajuuksia, joten virityspainike on tärkeä olla olemassa.

Record- ja Export-painikkeet ovat nuotin tallennusta varten. Record-painike aloittaa nuotin tallennuksen ja painamalla painiketta uudestaan tallennus lopetetaan. Tämän jälkeen Export-näppäintä painamalla voidaan määrittää kuvan 18 mukaiset asetukset. Ohjelma muodostaa nuotin Export-valikossa määriteltyjen asetusten mukaan.



KUVA 18. Export Configuration

3.4 Ongelmat äänisignaalin nuotinnuksessa

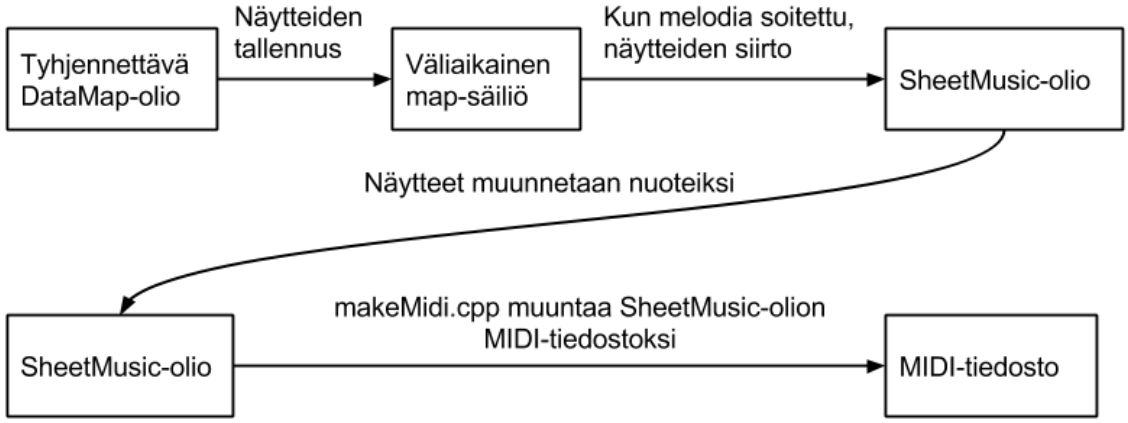
Äänenkäsittely mobiililaitteilla on haastavaa. Mobiililaitteen mikrofoni ei voi mitenkään vastata teknisesti ison studiomikrofonin laatua. Mitä parempi alkuperäinen äänityssignaali on, sitä helpompi signaalia on alkaa jatkokäsittelemään. Kuitenkin nykyaikaisissa mobiililaitteissa mikrofonit ovat erittäin hyviä kokoonsa nähden, jolloin Voice Noter:n kaltaisten sovellusten tekeminen on teknisesti mahdollista. Bassotaajuuksien ja korkeiden taajuuksien tallennus saattaa kuitenkin tuottaa ongelma.

Kuten aina käyttöliittymäsuunnittelussa on hankala ottaa huomioon kaikkien käyttäjien tarpeet. Suunnittelussa pitää ottaa huomioon, miten suunnata käyttöliittymä oikealle käyttäjäkunnalle. Jos käyttäjä ei tiedä mitään musiikinteoriasta, hänen tarpeensa täytyy ottaa erilailla huomioon kuin muusikon. Lisäksi, jos käyttäjä soittaa epävireisesti ja epätahdissa, nuotinnus on haastavaa.

4 OHJELMOINTI JA ALGORITMIKEHITYS

4.1 Luokkahierarkia lyhyesti

Työssä kehitetyn ohjelmakoodin toimintaperiaate on esitelty kuvassa 19.



KUVA 19. Luokkahierarkia

Tekstinä koodin toiminta on seuraava:

1. Näytteet tallennetaan tyhjennettävästä DataMap-oliosta uuteen map-säiliöön. DataMap-olio tallentaa näytteet vain ruudulla näyttämisen ajaksi, jolloin näytteet täytyy tallettaa toiseen säiliöön myöhempää käsittelyä varten.
2. Kun melodia on soitettu, näytteet siirretään map-säiliöstä SheetMusic-olioon:
   -SheetMusic-olio sisältää Note-olioita. Note-olion sisältämät tiedot on kuvattu kooditekstissä 1. Tässä vaiheessa Note-oliot vastaavat yksittäisiä näytteitä, eivätkä oikeita nuotteja; Note-nimitys on hieman harhaanjohtava, mutta SheetMusic-luokkaa käytetään sekä näytteiden että nuottien käsittelyyn.
3. SheetMusic-olioon tallennetut näytteet muunnetaan nuoteiksi:
   -näytteistä rakennetaan nuotteja kohdassa 4.3 esitellyn algoritmin mukaan
4. makeMidi.cpp muuntaa SheetMusic-olion MIDI-tiedostoksi
5. Lopuksi käyttäjä voi joko kuunnella MIDI-tiedoston mobiililaitteella tai siirtää tietokoneelle tarkastettavaksi, ovatko nuotit tallentuneet oikein.

```
int length_;    // Nuotin pituus aika-arvona, 4 = neljäsosanuotti
float timeSec_; // Aikaleima sekunteina
long long timeStamp_;   // Aikaleiman indeksinumero
float orgFreq_; // Näytteen tai nuotin taajuus
char pitch_;    // Sävelkorkeus, tauko merkitään r:llä
int octave_;    // Oktaaviala
char semitone_; // Puolisävelaskel, onko nuotti korotettu vai ei?
                // Korotettu = #, alennettu = b. Tyhjä ' ' tarkoittaa,
                // ettei nuotti ole alennettu eikä korotettu.
int dot_;       // Kertoo, onko nuotti pisteellinen. 0 = ei,
                // 1 = yksi piste ja 2 = kaksi pistettä.
char corrected_;// Kertoo, onko nuottia sovitettu tiettyyn sävellajiin.
                // u = nuotti on korotettu sävellajiin sopivaksi,
                // d = nuotti on laskettu sävellajiin sopivaksi ja
                // n = nuottia ei ole sovitettu.
int certainty_; // Prosenttiluku kertoo, kuinka hyvällä varmuudella nuotti on
                // tunnisettu. 0 = 0 %, tunnistuksen varmuus on nolla.
                // 100 = 100 %, nuotti on varmasti tunnistetuu oikein.
bool checked_;  // Kertoo, että näyte on muunnettu nuotiksi.
```

KOODITEKSTI 1. Note-olion data

4.2 Ohjelmakoodin toiminta yksityiskohtaisesti

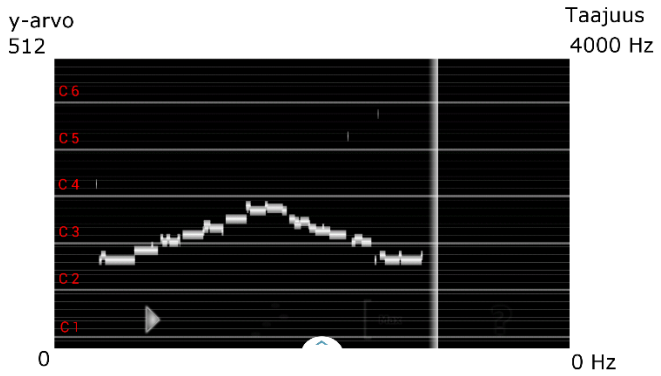
Ennen kuin Voice Noter:iin lisättiin nuotinnustoiminto, Voice Noter:n ohjelmakoodin luokka NoteDetector kävi läpi DataMap:n valkoisia pisteitä (kuva 11) ja analysoi pisteistä, pystyykö niistä tunnistamaan mielivaltaisen pituisiksi nuoteiksi, eli kuvan 12 mukaisiksi punaisiksi viivoiksi. Työssä kehitettiin NoteDetector-luokkaa eteenpäin siten, että luokka tekee näytteistä SheetMusic-olion, joka konvertoidaan midiMake.cpp:n avulla MIDI-tiedostoksi.

NoteDetector-luokka käsittelee näytteitä `map<unsigned long long,DataPoint>`-tyyppisellä säiliöllä. Näytteet ovat DataPoint-olioita, jotka sisältävät aikaleiman, y-arvon (sävelkorkeus) ja mCompareValue:n (aikaleima sekunteina). Koska näytteet tuhotaan ruudulla näyttämisen jälkeen, näytteet tallennettiin uuteen `map<unsigned long long,DataPoint> list_` -säiliöön. Säiliöstä `list_` näytteet tallennettiin SheetMusic-olioon PutDataPoints2SheetMusic-funktion avulla. PutDataPoints2SheetMusic-funktio tallensi koodiTekstin 2 mukaiset tiedot.

```
// Laske ja tallenna näytteen alkuperäinen taajuus
sample.setOrgFreq(this->CalculateNoteFreq(it_->second.y));
// Aseta näytteelle sävelkorkeus, oktaavi ja puolisävelaskel
this->NoteFreq2SheetMusic(&sample);
sample.setTimeStamp(it_->second.timeStamp); // Aseta aikaleima
sample.setTimeSec(it_->second.mCompareValue); // Aseta aikaleima sekunteina
this->melodysheet_.addNote(sample); // Lisää näytteen SheetMusic-olioon
```

KOODITEKSTI 2. Näytteen tallennus SheetMusic-olioon

Näytteen tallennuksessa käytettiin CalculateNoteFreq- ja NoteFreq2SheetMusic-funktioita. CalculateNoteFreq-funktio laskee näytteen y-arvosta taajuuden. Kuvan 20 mukaisesti DataMap:n tarkkuus on 512 näytettä pystysuunnassa. Ruudun yläreuna vastaa 4000 Hz, jolloin näytteen taajuus voidaan laskea kaavalla 4000/512 * y-arvo = näytteen taajuus.

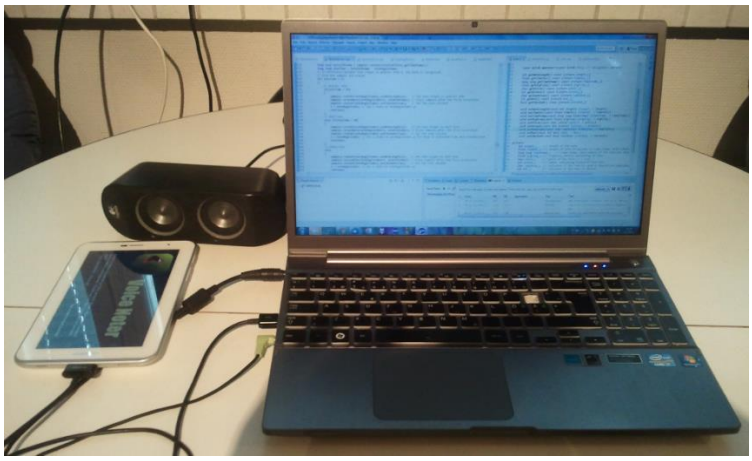


KUVA 20. Ruudun tarkkuus

Näytteen taajuuden tallennuksen jälkeen NoteFreq2SheetMusic-funktio laskee taajuudesta sävelkorkeuden, oktaavialan ja puolisävelaskeleen näytteelle. Kun näytteet on tallennettu SheetMusic-olioon, näytteet muunnetaan oikeiksi nuoteiksi ConvertDataPoints2Notes-funktiolla; kappaleessa 4.4 kerrotaan funktion toiminta. Lopuksi SheetMusic-oliosta tehdään MIDI-tiedosto MakeMIDI-funktiolla, joka sijaitsee makeMidi.cpp:ssä. Koodin makeMidi.cpp pohjana on käytetty Joel ”Bisqwit” Yliluoman tekemää midimake.cc-koodia (Yliluoma). Koodin ymmärtämiseksi luettiin Otto Romanowskin kirja MIDI 1.0 Musiikkilaitteiden tiedonsiirtostandardi sekä MIDI is the language of gods -sivuston ohjeita (Romanowski 1990; MIDI is the language of gods).

4.3 Nuotintunnistusalgoritmin kehitys

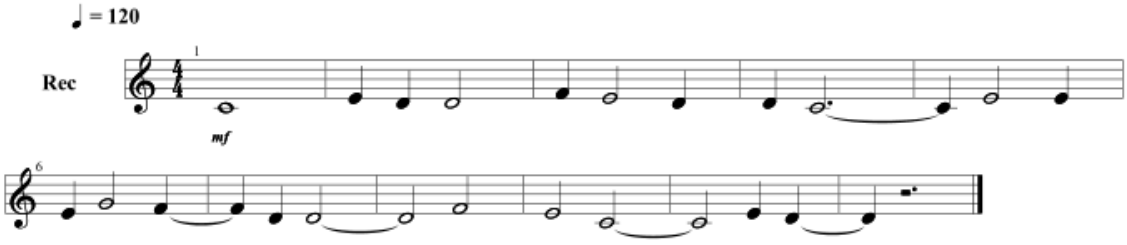
Nuotintunnistusalgoritmin kehitys aloitettiin äänittämällä tietokoneelle sähkökitaralla soitettu Ukko Nooa -kappale. Tempo oli 120 iskua minuutissa. Soitto aloitettiin yksiviivaisesta c:stä. Kitara oli viritetty taajuuteen 440 Hz. Äänitys toistettiin Samsung Tab 2 7.0 -tabletille Logitech 2.1 -kaiutinjärjestelmällä kuvan 21 mukaisella järjestelyllä.



KUVA 21. Testiäänityksen toistaminen laitteelle

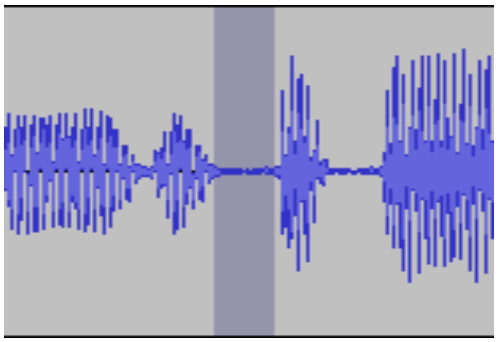
NoteDetector-luokkaan tehtiin funktio PrintSamples2File, joka tulosti SheetMusic-olioon tallennetut näytteet. Kun äänitettä soitettiin tietokoneelta tabletille, havaittiin ruudulta, että ohjelma havaitsee korkeammat taajuudet paremmin kuin matalammat taajuudet. Tällöin äänitettä transponoitiin eli nuottien sävelkorkeutta korotettiin yhden oktaavin verran. Liitteessä 5 on esitetty SheetMusic-olion näytteet, kun äänite soitettiin tabletille transponoituna yllä mainituilla järjestelyillä. Huomio, jotta liitteet 5 ja 6 eivät olisi todella pitkiä, taulukot sisältävät Ukko Nooa -nuotin neljä ensimmäistä tahtia.

Näytteistä rakennettiin MIDI-tiedosto, josta tehtiin nuotti GuitarPro 6 -ohjelmalla (kuva 22). Kuvien 22 ja 13 nuotteja vertaamalla huomattiin, että saman sävelkorkeuden peräkkäiset nuotit muunnetaan yhdeksi pitkäksi nuotiksi.



KUVA 22. Ensimmäinen Ukko Nooa -nuotti

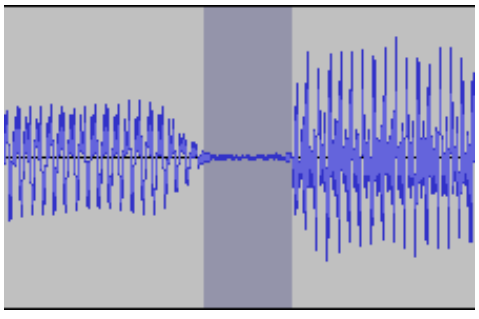
Liitteen 5 näytteitä analysoimalla huomattiin, ettei näytteissä ole lainkaan taukoja peräkkäisten nuottien välillä. Audacity-äänitysohjelmalla äänisignaalia analysoimalla havaittiin, että kahden ensimmäisen c-sävelen välinen tauko on 0,015 sekuntia. Kuvassa 23 tauko on merkitty tummalla.



KUVA 23. Transponoidun signaalin tauko normaalisti soittaen

Näytteiden aikaleimojen välejä tutkimalla nähtiin, että näytteiden välinen aika on 0,02037 tai 0,02038 sekuntia, riippuen pyöristyksestä. Tällöin tauon kohdalle voi osua vain yksi näytteenottoajankohta. DataMap:iin tallennetaan vain näytteet, joilla on sävelkorkeus, jolloin tauot pitää päätellä näytteiden aikaleimojen välisistä ajoista. DataMap:sta saatavissa näytteissä saattaa olla taukoja, vaikka nuotti olisi jatkuva; tauon täytyy siis olla tarpeeksi pitkä, että se voidaan havaita.

Transponoitua ja transponoimatonta signaalia verrattaessa havaittiin, että transponoituun signaalin ilmestyi äänipiikki tauon kohdalle – syytä ei selvitetty. Transponoimattomasta signaalista havaittiin, että kahden peräkkäisen nuotin välinen tauko on noin 0,030 sekuntia. Kuvassa 24 on merkitty tauko tummemmalla värillä; tauon pituus on 0,031 sekuntia. Tällöin teoriassa peräkkäisten nuottien välistä voitaisiin saada kaksi näytettä ilman säveltä. Kahden aikaleiman taukoa ei voida tunnistaa tauoksi, koska DataMap:sta saadussa datassa voi olla tämän pituisia taukoja, vaikka nuotti todellisuudessa jatkuu.



KUVA 24. Peräkkäisten nuottien välinen tauko normaalisti soittaen

Liitteen 5 näytteistä nähdään, että näytteiden taajuus vaihtuu, kun nuotin välissä pitäisi olla tauko. Taulukossa 5 on otettu kohta liitteen 5 taulukosta. Näytteen 19 ja 20 välillä taajuus nousee, ja näytteen 22 ja 23 välillä taajuus laskee alkuperäiseen. Näytteet ovat korostettu taulukossa sinisellä. Aikaleimoja tutkimalla voidaan olettaa, että tässä kohtaa tapahtuu nuotin vaihto. Algoritmia lähdettiin kehittämään siten, että se havaitsee taajuuden muutoksen. Äänisignaalia soitettiin sovellukselle useita kertoja; havaittiin, että taajuuden muutosta ei tapahdu aina peräkkäisten samojen sävelien välillä, vaikka testisignaali on sama joka kerta. Tämä nuottien erotustekniikka hylättiin.

TAULUKKO 1. Ote liitteestä 5

Näyte	Nuotti	Oktaavi	Aikaleima	Aikaleima (s)	Taajuus
15	c	2	206	4,19725	1046,88
16	c	2	207	4,21763	1046,88
17	c	2	208	4,238	1046,88
18	c	2	209	4,25838	1046,88
19	**c**	**2**	**210**	**4,27875**	**1046,88**
20	**c**	**2**	**211**	**4,29913**	**1054,69**
21	c	2	212	4,3195	1054,69
22	**c**	**2**	**213**	**4,33988**	**1054,69**
23	**c**	**2**	**214**	**4,36025**	**1046,88**
24	c	2	215	4,38063	1046,88
25	c	2	216	4,401	1046,88
26	c	2	217	4,42138	1046,88

Työssä tultiin siihen tulokseen, että DataMap:sta saatujen näytteiden välinen aika on liian harva. Näytteenottotaajuus saadaan yhtälöllä

$$F_s = T^{-1} = 0{,}02037^{-1} = 49{,}0918 \cong 49 \text{ Hz} \quad (2)$$

missä F_s = näytteenottotaajuus

T = näytteiden välinen aika

(Huttunen 2014, 3-4)

Kappaleessa 2.2 kerrottiin, että CD:n näytteenottotaajuus on 44.1 kHz eli yli 900-kertainen yhtälöllä 2 laskettuun DataMap:n näytteenottotaajuuteen verrattuna. Tietenkään nuottien aika-arvojen selvittämiseen ei tarvita yhtä tarkkaa näytteenottotaajuutta, mutta ero kuvastaa, missä suuruusluokassa näytteenottotaajuudet ovat. Esimerkiksi nelinkertai-

nen näytteenottotaajuus DataMap:ssa auttaisi paljon aika-arvojen erottelua. Myös kappaleen tempo ja nuottien pituudet vaikuttavat erottelukykyyn. Soittaja voisi tietysti laskea kappaleen tempoa, mutta kokemattomalle soittajalle tempon laskeminen saattaa johtaa nuottien aika-arvojen vääristymiseen. Lisäksi tempo 120 iskua minuutissa on hyvin yleinen länsimaalaisessa musiikissa, joten tempon laskeminen ei ratkaise ongelmaa.

4.4 Tutkimuksen nuotintunnistusalgoritmin viimeinen versio

Ukko Nooasta äänitettiin uusi versio, jossa kappale soitettiin staccatona eli nuotit toisistaan terävästi erottaen. Kappale soitettiin tempolla 120 iskua minuutissa, virityksellä 440 Hz ja keski-C:stä aloittaen. Tätä signaalia ei transponoitu. Tällöin saatiin liitteen 6 mukainen näytelistaus. Taulukossa 2 on esitetty kolmen ensimmäisen c-nuotin näytteet sekä e-nuotin ensimmäinen näyte. Taulukoon on laskettu Erotus- ja Pituus-sarakkeet. Erotus tarkoittaa, mikä on rivillä olevan aikaleiman ja seuraavalla rivillä olevan aikaleiman erotus. Pituus on laskettu vähentämällä nuotin viimeisen näytteen aikaleimasta nuotin ensimmäisen näytteen aikaleima; ensimmäinen pituus on laskettu $124 - 117 = 7$. Näytteen 7 erotus seuraavaan näytteeseen on 17 aikaleimaa; voidaan olettaa, että tässä välissä on tauko. Nuottien pituudet ja tauon kertovat rivit on korostettu vihreällä.

TAULUKKO 2. Ote Staccato Ukko Nooan näytteistä (liite 6)

Näyte	Nuotti	Oktaavi	Aikaleima	Erotus	Pituus	Aikaleima (s)	Taajuus
1	c	1	117	2		2,38388	531,25
2	c	1	119	1		2,42463	515,625
3	c	1	120	1		2,445	515,625
4	c	1	121	1		2,46538	515,625
5	c	1	122	1		2,48575	515,625
6	c	1	123	1		2,50613	515,625
7	**c**	**1**	**124**	**17**	**7**	**2,5265**	**515,625**
8	c	1	141	2		2,87288	531,25
9	c	1	143	2		2,91363	515,625
10	c	1	145	1		2,95438	515,625
11	c	1	146	1		2,97475	515,625
12	**c**	**1**	**147**	**16**	**6**	**2,99513**	**515,625**
13	c	1	163	2		3,32113	531,25
14	c	1	165	2		3,36188	531,25
15	c	1	167	1		3,40263	515,625
16	c	1	168	1		3,423	515,625
17	**c**	**1**	**169**	**18**	**6**	**3,44338**	**515,625**
18	e	1	187	2		3,81013	656,25

Kun liitteen 6 dataa analysoitiin lisää, havaittiin, että lyhin neljäsosanuotin pituus yllämainitulla tavalla laskien on kolme. Lyhin puolinuotin pituus on 28, ja kokonuotin pituus on 73. Tauoista lyhyin on yhdeksän näyteväliä. Näillä arvoilla algoritmista pystytään kehittämään toimiva.

Yksinkertaistettuna algoritmi toimii seuraavasti. SheetMusic-olio sisältää näytteet {x1, x2, x3, x10,x11,x14,x16,x17,x30}, jossa x on näyte ja numero x:n perässä on aikaleima. Näytteet x1, x2 ja x3 ovat peräkkäisiä ja x10 aikaleimaan erotus on 7. Selvästi näytteet x3 ja x10 kuuluvat eri nuotteihin. Näytteistä x1, x2 ja x3 muodostavat nuotin, jos näytteiden sävelkorkeus on sama. Näytteelle x1 asetetaan nuottiarvo ja näytteet x2 ja x3 tuhotaan. Näytejonon tutkiminen jatkuu näytteestä x10. Näytteiden x10, x11, x14, x16 ja x17 arvioidaan olevan yhtenäinen nuotti, jos näytteiden sävelkorkeus on sama; nuotista on vain jostain syystä jäänyt tunnistamatta välistä muutama näyte. Näytteelle x10 asetetaan aika-arvo, todennäköisesti pidempi kuin nuotille x1. Näyte x30 on ajallisesti kaukana muista näytteistä ja se on yksin, joten sen arvioidaan olevan harhaääni. Näyte x30 tuhotaan, koska yhdestä näytteestä ei voi tehdä nuottia.

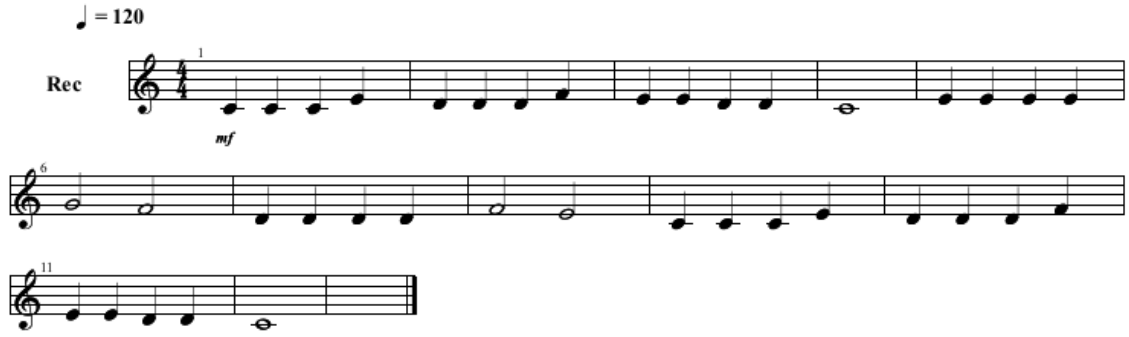
Pseudokoodina algoritmi toimii seuraavasti (kooditeksti 3). Liitteessä 1 on algoritmin oikea koodi.

```
1.  void NoteDetector::ConvertDataPoints2Notes(SheetMusic * samples) {
2.      for (Käy läpi kaikki SheetMusic-olion näytteet) {
3.          if(Viimeinen näyte = tarkistettu, eli kaikki näytteet on käyty läpi)
4.              break
5.          if(Näyte on tarkistettu, hypätään seuraavaan näytteeseen)
6.              continue
7.          noteBeginIndex = mistä indeksistä nuotti alkaa
8.          noteEndIndex = mihin indeksiin nuotti loppuu
9.          Tallenna iteraatiokierroksen x-nuotin tiedot:
10.            sävelkorkeus, puolisävelaskel, oktaavi ja aikaleima
11.
12.         prevTimeStamp = x-näytteen aikaleima
13.
14.         for(käy läpi x-nuotista seuraavat näytteet läpi) {
15.             Tallenna seuraavan nuotin tiedot (samat kuin yllä)
16.
17.             if(Jos näyteväli on yli 5, näyte ei kuulu samaan nuottiin)
18.                 break
19.
20.             else if(Jos näyteväli on alle 6 ja näytteiden tiedot ovat samat
21.             aikaleimaa lukuun ottamatta, näyte kuuluu nuottiin)
22.                 noteEndIndex = tämän iteraatiokierroksen näytteen indeksi
23.
24.             prevTimeStamp = tämän iteraatiokierroksen näytteen aikaleima
25.         }
26.
27.         noteTime = (nuotin ensimmäinen aikaleima - viimeinen aikaleima)
28.         // Aikaleimat selvitetään noteBeginIndex- ja
            // noteEndIndex-indeksien avulla
29.         // noteTime = nuotin kesto
30.         if(Jos noteTime on yli 2, näytteistä rakennetaan nuotti)
31.         {
32.             if(Jos noteTime < 25, nuotti on neljäsosanuotti)
33.             {
34.                 -Asetetaan nuotin ensimmäiselle näytteelle aika-arvoksi 4
35.                 -Poistetaan nuotin muut näytteet
36.                 -Asetetaan nuottiin tieto, että se on tarkistettu
37.                 -Asetetaan uloimman for-silmukan iteraatiokierrokseksi
                     noteBeginIndex, jolloin iteroimista jatketaan seuraavasta
                     näytteestä
39.                 -continue
40.             }
41.             else if(Jos noteTime < 50, nuotti on puolinuotti)
42.             {
43.                 -Asetetaan nuotin ensimmäiselle näytteelle aika-arvoksi 2
44.                 -Toteutetaan samat asiat kuin yllä
45.             }
46.             else // Nuotin on pakko olla kokonuotti
47.             {
48.                 -Asetetaan nuotin ensimmäiselle näytteelle aika-arvoksi 1
49.                 -Toteutetaan samat asiat kuin yllä
50.             }
51.         }
52.     }
53.     for(Käy läpi kaikki SheetMusic-olion näytteet)
54.     {
55.         if(Näytteen note_ on 0, eli näytteelle ei ole asetettu aika-arvoa)
56.             -Poista näyte
57.     }
58. }
```

KOODITEKSTI 3. Nuotintunnistusalgoritmin pseudokoodi

4.5 MIDI-datan muuttaminen nuotiksi

Työssä toteutettiin muunnos Voice Noter -sovelluksen datasta MIDI-tiedostoksi. Liitteen 6 näytteistä rakennettiin MIDI-tiedosto, joka vietiin tietokoneelle. Tietokoneella MIDI-tiedosto vietiin GuitarPro 6 -ohjelmaan, ja siitä tulostettiin kuvan 25 mukainen PDF-nuotti. Kuvassa voi vertailla LilyPond-ohjelmalla tehtyä nuottia (kuva13) ja Voice Noter -ohjelmalla tehtyä nuottia (kuva 25). Nuotit ovat samat Voice Noter:n viimeistä tyhjää tahtia lukuun ottamatta.



KUVA 25. Äänisignaalista konvertoitu Ukko Nooa

5 POHDINTA

5.1 Tutkimuksen eteneminen

Tutkimuksen alussa havaittiin yhteensopivuusongelmia työkalujen eri ohjelmaversioiden välillä. Eclipse-ohjelmaan piti asentaa Android-lisäosat sekä C++-kielen kehitystyökalut. Lisäksi Voice Noter vaatii ohjelmista määrätyt versiot, jotta kokonaisuus toimii; kaikista ohjelmien osista on monia versioita ja niihin tulee jatkuvasti päivityksiä. Näiden haasteiden ratkettua kehitystyö päästiin käynnistämään kunnolla.

MIDI-tiedostojen luomisessa olisi voinut käyttää valmiita kirjastoja. Työssä haluttiin kuitenkin tutkia tarkkaan, miten MIDI-tiedostoja valmistetaan ohjelmakooditasolla. MIDI-tiedostojen rakenteen hyvä tunteminen auttaa kehittämään Voice Noter:n ominaisuuksia ja täysin uusia ohjelmia. MIDI-ohjelmoinnin opettelun vaikeutena oli esimerkkien vähyys. MIDI:n kehitysorganisaation sivuilla on hyvät taulukot teknisestä spesifikaatiosta, mutta esimerkkejä on vähän (MIDI Manufacturers Association). Otto Romanowskin kirja MIDI:stä on edelleen erittäin hyvä perusteos, vaikka se on kirjoitettu vuonna 1989 (Romanowski 1990).

Yhtenä tavoitteena oli valmistaa PDF-nuotti. Nuotti voidaan valmistaa tietokoneella MIDI-tiedosto nuotinnusohjelmalla konvertoimalla, mutta nuotin tekemiseen löydettiin myös yksinkertaisempi tapa. Tekstipohjaisella LilyPond-nuotinnusohjelmalla pystyy konvertoimaan LilyPond-muotoillun tekstitiedoston helposti PDF:ksi. LilyPond:n syntaksin tutkiminen oli myös tärkeä osa SheetMusic-luokan kehityksessä. MIDI-tiedoston luonti aloitettiin muuntamalla LilyPond-tekstitiedosto SheetMusic-olioksi, josta rakennettiin MIDI-tiedosto. Työn aikataulu ei riittänyt, jotta SheetMusic-olion muuttaminen LilyPond-tiedostoksi olisi ehditty tehdä. Ominaisuuden tekeminen on kuitenkin yksinkertaisempaa kuin jo toteutettu LilyPond-tiedoston muuntaminen MIDI:ksi.

Tutkimuksessa kului yllättävän paljon aikaa MIDI-standardiin tutustumiseen ja LilyPond-MIDI-muunnoksen toteuttamiseen. Nuotintunnistusalgoritmin kehitystä varten syntyi kuitenkin monia ideoita MIDI-ohjelmoinnin aikana. Kun Voice Noter:n valmis koodi oli sisäistetty, algoritmin kehitys edistyi nopeasti - olihan algoritmikehitys melkein pelkkää matematiikkaa ja ohjelmointia.

Tutkimuksen edistystä seurattiin kuvaamalla Opinnäytetyön lisälehdet -videoblogi YouTube-videosivustolle (Herwood Technologies Oy). Videoblogia kuvattiin noin kerran viikossa viikoittaisen palaverin yhteydessä. Videoblogin kuvaaminen edisti työntekemistä, koska kameralle puhuminen loi lisäpainetta saada tuloksia.

5.2 Lopputulosten analysointi

Toteutettu ohjelmakoodi muuntaa Voice Noter:lta saadun bittikartan MIDI-tiedostoksi. Tempolla 120 iskua minuutissa soitetun yksiäänisen melodian nuotinnus onnistuu neljäsosanuotin tarkkuudella. Jotta neljäsosanuottia lyhemmät nuotit olisi mahdollista tunnistaa, bittikartan näytteenottotaajuutta täytyy kasvattaa. Toisaalta bittikartalla on tarkoitus visualisoida nuotit käyttäjälle. Tällöin toisena parempana vaihtoehtona olisi rakentaa nuotit Fourier-muunnoksen tuloksen näytteistä ennen kuin ne on talletettu bittikarttaan.

Kolmantena mahdollisuutena olisi tutkia Voice Noter:n toisen uuden ominaisuuden, tempoharjoitustoiminnon, algoritmeja. Toiminto etsii melodiasta rytmin ja kertoo käyttäjälle, pysyykö hän tempossa. Tempoharjoitustoiminnon algoritmien avulla olisi todennäköisesti mahdollista parantaa nuotin alkamishetken määritystä. Kun nuotti soitetaan tai lauletaan, signaalin amplitudi on suuri nuotin alkamishetkellä ja pienenee vähitellen; tosin kaikilla soittimilla signaalin amplitudi ei pienene samalla nopeudella. Esimerkiksi ilmavirralla toimivien soittimien sävelet pystyy pitämään suhteellisen tasaisina, kunhan ilmavirran voimakkuus pysyy vakiona.

Tutkimuksen MIDI-tiedosto kirjoitetaan C-duuri-sävellajissa, tempolla 120 iskua minuutissa ja 4/4-tahtilajilla. Pienillä lisäyksillä ohjelma pystyy tekemään MIDI-tiedostot myös eri asetuksilla. Algoritmi ei tunnista taukoja; työssä keskityttiin tunnistamaan nuotit. Kun nuotit ovat tunnistettu oikein, tauot voidaan määrittää laskemalla nuottien aikaleimojen erotus. On myös olemassa soittotekniikoita, joilla soitettuja nuotteja ohjelma ei tunnista. Esimerkiksi kielisoittimilla on tekniikoita kuten liu’ut ja venytykset; näillä tekniikoilla nuotti vaihtuu tasaisesti sävelestä toiseen ja vaihdos merkitään nuottiin tekniikkaa vastaavalla merkinnällä. Jotta liu’ut ja venytykset tunnistettaisiin oikein, näytteenottotaajuutta pitää kasvattaa. Ilman näytteenottotaajuuden kasvatusta tapahtuu kappaleessa 2.2 esitettyä laskostumista.

Ohjelmaa testattiin myös teräskielisellä akustisella kitaralla. Sovellukselle soitettiin kappaleita Nuotteja 4- ja 10 -reikäiselle huularille -nuottivihosta (Rosa 2013). Testikappaleet eivät sisältäneet neljäsosanuottia lyhempiä nuotteja. Akustisessa kitarassa on kaikukoppa, joka vahvistaa ääntä. Jos kieliä ei soittaessa sammuta kunnolla, ne jäävät resonoimaan ja tuottamaan hälyääniä. Sähkökitaralla hälyäänet on helppo suodattaa suodattimilla pois. Tutkimuksessa havaittiin, että akustisen kitaran soitossa täytyy olla paljon tarkempi kuin sähkökitaran. Näytteenottotaajuutta nostamalla voidaan helpottaa näytteiden analysointia, mitkä ovat hälyääniä ja mitkä oikeita nuotteja.

Jatkotutkimuksessa nuotinnuksen algoritmia voi testata tekemällä nuotinnusohjelmalla MIDI-tiedosto ja soittamalla tiedosto Voice Noter:lle. Tämän jälkeen ohjelman tekemää MIDI:ä voidaan verrata alkuperäiseen MIDI:in. Työssä testaus suoritettiin soittamalla kappaleet ohjelmalle uudelleen joka testikierroksella. Testauksen nopeuttamiseksi muutaman eri melodian ja soittokerran näytteet voisi tallettaa tiedostoon, jolloin algoritmia voisi testata nopeasti eri näytelistoilla. Lisäksi testidatan analysointia voidaan nopeuttaa ohjelmoimalla automaattinen analysointifunktio, joka kertoo nuottien aikaleimojen erotukset – esimerkiksi pienimmän ja suurimman tauon nuottien välissä. Todennäköisesti samaa analysointifunktiota, voitaisiin käyttää myös itse nuotinnuksessa.

Note-luokkaan tehtiin valmiiksi yksityisiä muuttujia, joita ei hyödynnetä tässä työssä. Tutkimuksessa suunniteltiin, että jokaiselle nuotille voisi asettaa tiedon certaintity_ eli varmuuden. Varmuus kertoo, kuinka hyvällä todennäköisyydellä nuotti on tunnistettu oikein. Kun suurta määrää yksittäisiä nuotteja aletaan sovittamaan yhtenäiseksi nuottikirjoitukseksi, on kokonaisuuden hahmottaminen vaikeaa. Jos nuotit laitetaan peräkkäin vain tunnistettujen aika-arvojen perusteella, kokonaisuus on todennäköisesti väärin. Jos ensimmäinen nuotti kirjoitetaan liian pitkäksi, kaikki nuotit ovat sen jälkeen väärällä paikalla; nuotti saattaa jakaantua kahteen osaan tahtien kesken, jolloin sen lukeminen vaikeutuu. Kun nuoteilla on varmuus, jokainen tahti voidaan analysoida erikseen. Analysoinnilla selvitetään, ovatko nuotit todennäköisesti oikeilla paikoilla. Jos kaikkien tahtien nuottien arvioidaan olevan väärillä paikoilla, todennäköisesti virhe on heti ensimmäisessä tahdissa. Tällöin ensimmäisen tahdin vähiten varma nuotti, muunnetaan kokonaisuudelle sopivaksi.

Voice Noter:sta julkaistaan uusi versio, kun uudet ominaisuudet on yhdistetty toimivaksi kokonaisuudeksi. Myös sovelluksen käyttöliittymä uudistetaan täysin. Vaikka tutkimuksen vastuulla ei ollut toteuttaa uutta käyttöliittymää, on tärkeää välittää sovelluksen kehitysryhmälle työn käyttöliittymäsuunnittelun tulokset.

LÄHTEET

Gadiou R.; Franke K.; Vasilief I. Chapter 6. Analysis of data and curves. Luettu 23.1.2015 http://scidavis.sourceforge.net/manual/c4166.html

Heikkilä P.; Halkosalmi V. 2005. Tohtori Toonika. Helsinki: Otavan Kirjapaino Oy.

Herwood Technologies Oy. Opinnäytetyön lisälehdet. YouTube 2015. Julkaistu 17.1.2015. Päivitetty 6.5.2015. Katsottu 17.1.2015. https://www.youtube.com/channel/UCs966frVzxqQ_brPpyvD5-g

Huttunen H. 2014. Signaalinkäsittelyn perusteet -opetusmoniste. Tampereen teknillinen yliopisto. Signaalinkäsittelyn laitos.

Jokinen R. 1998. Digitaalinen signaalinkäsittely, lineaariset järjestelmät. Turun ammatti-korkeakoulu. Päivitetty 2003. Luettu 12.3.2015 http://www.hovirinta.fi/audio/suunnittelu/teoriaa/DSP/Digitaalinen%20signaa-link%E4sittely%2003.pdf

Joutsenvirta A. 2005. Säveltasot hertsiarvoina. Julkaistu lokakuussa 2005. Päivitetty 24.2.2009. Luettu 21.4.2015. http://www2.siba.fi/akustiikka/index.php?id=18

Joutsenvirta A.; Perkiönmäki J. 2007. Musiikinteoria 1. Julkaistu 5.11.2007. Päivitetty 9.12.2008. Luettu 21.4.2015 http://www2.siba.fi/muste1/index.php?id=1&la=fi

Klapuri A.; Virtanen. AD/DA-muunnos. Luettu 23.1.2015 http://www.cs.tut.fi/~digaudio/adda.pdf

Laakso S. Ääni. Tietotekniikan alkeet. Luentomoniste. Päivitetty 30.12.1998. Luettu 22.1.2015. http://www.cs.helsinki.fi/u/salaakso/alkeet/luentomoniste/aani.html

Laaksonen J. 2006. Äänityön kivijalka. Porvoo: Painoyhtymä.

Learn About MIDI. Luettu 23.1.2015 http://midi.org/aboutmidi/index.php

LilyPond-ohjelmisto. Luettu 20.2.2015. http://lilypond.org/index.html

Microsoft Research. 2014. Enabling Cross-Lingual Conversations in Real Time. Julkaistu 27.5.2014. Luettu 20.3.2015. http://research.microsoft.com/en-us/news/features/translator-052714.aspx

MIDI is the language of gods. Luettu 3.1.2015.
http://www.blitter.com/~russtopia/MIDI/~jglatt/

MIDI Manufacturers Association. Luettu 12.1.2015
http://midi.org/

Romanowski O. 1990. MIDI 1.0 Musiikkilaitteiden tiedonsiirtostandardi. Helsinki: Valtion painatuskeskus.

Rosa Y. 2013. Nuotteja 4- ja 10 -reikäiselle huularille. Luettu 1.5.2015.
http://yellingrosa.com/Opetus/Nuottivihko.pdf

Yliluoma J. midimake.cc. Luettu 23.1.2015.
http://bisqwit.iki.fi/jutut/kuvat/programming_examples/midimake.cc

LIITTEET

Liite 1. NoteDetector-koodi

1 (4)

```
1.  // Author Juho Itä. There's more code in NoteDetector.cpp,
2.  // but it's not for public eyes.
3.
4.  void NoteDetector::OutToFile(DataMap * datalist) {
5.      this->PutDataPoints2SheetMusic();
6.      this->PrintSamples2File(&melodysheet_,
7.                                          "/storage/sdcard0/test_samples.txt");
8.      this->ConvertDataPoints2Notes(&melodysheet_);
9.      this->PrintSamples2File(&melodysheet_,
10.                                           "/storage/sdcard0/test_sheet.txt");
11.     MakeMidi(&melodysheet_, "/storage/sdcard0/test_melody.mid");
12. }
13.
14. float NoteDetector::CalculateNoteFreq(int yValue) {
15.     // 4000 Hz / 512 = 7.8125 Hz, Screens height is 512 pix-
    els and pixel 512 equals 4000 Hz
16.     return 7.8125 * (float)yValue;
17. }
18.
19. void NoteDetector::PutDataPoints2SheetMusic() {
20.     for (it_=list_.begin(); it_!=list_.end(); ++it_) {
21.         Note sample;
22.         // Set frequency to the sample
23.         sample.setOrgFreq(this->CalculateNoteFreq(it_->second.y));
24.         // Set pitch, octave and semitone to the sample
25.         this->NoteFreq2SheetMusic(&sample);
26.         // Set time stamp to the sample
27.         sample.setTimeStamp(it_->second.timeStamp);
28.         // Set the real time when sample occurred
29.         sample.setTimeSec(it_->second.mCompareValue);
30.         // Add note to SheetMusic object
31.         this->melodysheet_.addNote(sample);
32.     }
33. }
34.
35. void NoteDetector::NoteFreq2SheetMusic(Note * dataPointSample) {
36.     // Original frequency must be set before calling this function!
37.
38.     float NoteFreq = dataPointSample->getOrgFreq();
39.     const float minNoteFreqHz = grid4Functions_.GetScaleRefFreqHz();
40.     if (NoteFreq>=minNoteFreqHz) {
41.
42.         float keyNumber = grid4Functions_.NoteKeyNumber(NoteFreq);
43.
44.         const int referenceOffsetForOctave = 3;
45.         const int halfStepsInOctave=grid4Functions_.GetHalfStepsInOctave();
46.         float b =  (keyNumber-referenceOffsetForOctave)/halfStepsInOctave;
47.
48.         dataPointSample->setOctave((int)(b+1)); // round
49.         int stepInOctave = (int)keyNumber % halfStepsInOctave;
50.
51.         switch (stepInOctave) {
52.         case 0:
53.             dataPointSample->setPitch('a');
54.             break;
```

(jatkuu)

```
55.         case 1:
56.             dataPointSample->setPitch('a');
57.             dataPointSample->setSemitone('#');
58.             break;
59.         case 2:
60.             dataPointSample->setPitch('b');
61.             break;
62.         case 3:
63.             dataPointSample->setPitch('c');
64.             break;
65.         case 4:
66.             dataPointSample->setPitch('c');
67.             dataPointSample->setSemitone('#');
68.             break;
69.         case 5:
70.             dataPointSample->setPitch('d');
71.             break;
72.         case 6:
73.             dataPointSample->setPitch('d');
74.             dataPointSample->setSemitone('#');
75.             break;
76.         case 7:
77.             dataPointSample->setPitch('e');
78.             break;
79.         case 8:
80.             dataPointSample->setPitch('f');
81.             break;
82.         case 9:
83.             dataPointSample->setPitch('f');
84.             dataPointSample->setSemitone('#');
85.             break;
86.         case 10:
87.             dataPointSample->setPitch('g');
88.             break;
89.         case 11:
90.             dataPointSample->setPitch('g');
91.             dataPointSample->setSemitone('#');
92.             break;
93.         default:
94.             assert(0);
95.             break;
96.         }
97.     }
98.     else {
99.               // Shouldn't do this, there's some kind of error
100.              dataPointSample->setPitch('x');
101.         }
102.     }
103.
104.     // Test function for printing the samples into a file
105.     void NoteDetector::PrintSamples2File(SheetMusic * samples,
106.                                                          string path) {
107.         FILE* fp = std::fopen(path.c_str(), "wb");
108.         stringstream tmp;
109.         tmp << samples->notesSize() << endl;
110.         string s = tmp.str();
111.         std::fwrite(s.c_str(), 1, s.length(), fp);
```

```
112.          for(int i = 0; i < samples->notesSize(); i++)
113.          {
114.              stringstream ss;
115.              ss << samples->noteAt(i).getPitch();
116.              ss << samples->noteAt(i).getSemitone();
117.              ss << samples->noteAt(i).getOctave() << " ";
118.              ss << samples->noteAt(i).getNoteLength() << " ";
119.              ss << samples->noteAt(i).getTimeStamp() << " ";
120.              ss << samples->noteAt(i).getTimeSec() << " ";
121.              ss << samples->noteAt(i).getOrgFreq() << " ";
122.              ss << endl;
123.              string values = ss.str();
124.              std::fwrite(values.c_str(), 1, values.length(), fp);
125.          }
126.          std::fclose(fp);
127.      }
128.
129.      void NoteDetector::ConvertDataPoints2Notes(SheetMusic * samples) {
130.          // this->PutDataPoints2SheetMusic();
131.          // must be done before samples can be converted to actual notes!
132.
133.          for(int i = 0; i < samples->notesSize(); i++) {
134.           // If the last note is converted, all notes are converted, break
135.              if(samples->noteAt(samples->notesSize()-1).getChecked()==true)
136.                  break;
137.              if(samples->noteAt(i).getChecked() == true)
138.                  continue;
139.
140.              int noteBeginIndex = i;
141.              int noteEndIndex = i;
142.              char pitch1 = samples->noteAt(i).getPitch();
143.              char semitone1 = samples->noteAt(i).getSemitone();
144.              int octave1 = samples->noteAt(i).getOctave();
145.              long long timeStamp1 = samples->noteAt(i).getTimeStamp();
146.              long long prevTimeStamp = timeStamp1;
147.
148.              for(int j = i+1; j < samples->notesSize(); j++) {
149.                  char pitch2 = samples->noteAt(j).getPitch();
150.                  char semitone2 = samples->noteAt(j).getSemitone();
151.                  int octave2 = samples->noteAt(j).getOctave();
152.                  long long timeStamp2 = samples->noteAt(j).getTimeStamp();
153.
154.                  // If the difference in timeStamps
155.                  // is greater than 2, the notes are separate
156.                  if(timeStamp2 - prevTimeStamp > 5)
157.                  {
158.                      break;
159.                  }
160.                  else if(timeStamp2 - prevTimeStamp < 6 &&
161.                          pitch1 == pitch2 &&
162.                          semitone1 == semitone2 && octave1 == octave2)
163.                  {
164.                      noteEndIndex = j;
165.                      prevTimeStamp = timeStamp2;
166.                      continue;
167.                  }
168.                  prevTimeStamp = timeStamp2;
169.              }
```

```
170.                long long noteBeginStamp =
171.                                samples->noteAt(noteBeginIndex).getTimeStamp();
172.                long long noteEndStamp =
173.                                 samples->noteAt(noteEndIndex).getTimeStamp();
174.                long long noteTime = noteEndStamp - noteBeginStamp;
175.
176.                // If difference between time stamps is greater than 4,
177.                // the note is recognized
178.                // Else the samples are erased
179.                if( noteTime > 2)
180.                {
181.                    // Quarter note
182.                    if(noteTime < 25)
183.                    {   // Set note length to quarter note
184.                        samples->noteAt(noteBeginIndex).setNoteLength(4);
185.                        // Erase samples after the first occurrence
186.                        samples->eraseNote(noteBeginIndex+1, noteEndIndex);
187.                        // Set the note checked
188.                        samples->noteAt(noteBeginIndex).setChecked(true);
189.                        // Set i index to noteBeginIndex so for-loop
190.                        // is continued from last checked note
191.                        i = noteBeginIndex;
192.                        continue;
193.                    }
194.                    // Half note, works same as above except
195.                    // note length is set to half note
196.                    else if(noteTime < 50)
197.                    {   // Set note length to half note
198.                        samples->noteAt(noteBeginIndex).setNoteLength(2);
199.                        samples->eraseNote(noteBeginIndex+1, noteEndIndex);
200.                        samples->noteAt(noteBeginIndex).setChecked(true);
201.                        i = noteBeginIndex;
202.                        continue;
203.                    }
204.                    // Whole note, works same as above except
205.                    // note length is set to whole note
206.                    else
207.                    {
208.                        samples->noteAt(noteBeginIndex).setNoteLength(1);
209.                        samples->eraseNote(noteBeginIndex+1, noteEndIndex);
210.                        samples->noteAt(noteBeginIndex).setChecked(true);
211.                        i = noteBeginIndex;
212.                        continue;
213.                    }
214.                }
215.            }
216.            // Erase samples whose NoteLength is not set
217.            // This erases too short notes
218.            for(int i = 0; i < samples->notesSize(); i++) {
219.                if(samples->noteAt(i).getNoteLength() == 0)
220.                {
221.                    samples->eraseNote(i);
222.                    --i;
223.                }
224.            }
225.        }
```

Liite 2. SheetMusic-luokka

```
1.  // Author Juho Itä
2.
3.  #ifndef SHEETMUSIC_H
4.  #define SHEETMUSIC_H
5.
6.  #include "note.h"
7.  #include <string>
8.  #include <vector>
9.
10. using std::string;
11. using std::vector;
12.
13. class SheetMusic
14. {
15. public:
16.     SheetMusic();
17.     SheetMusic(string title, string keysignature,
18.     int * timesignature, int tempo);
19.     ~SheetMusic();
20.
21.     string getTitle() const {return title_;}
22.     string getKeysignature() const {return keysignature_;}
23.     vector<int> getTimesignature() const {return timesignature_;}
24.     int getTempo () const {return tempo_;}
25.
26.     void setTitle(const string title) {title_ = title;}
27.     void setKeysignature(const string keysignature)
28.                             {keysignature_ = keysignature;}
29.     void setTimesignature(vector<int> timesignature)
30.                             {timesignature_ = timesignature;}
31.     void setTempo(const int tempo) {tempo_ = tempo;}
32.
33.     // Adds note to end of the notes_ vector
34.     void addNote(const Note ¬e);
35.     // Erase note at given index
36.     void eraseNote(int index);
37.     // Erases range of notes ([first, last])
38.     void eraseNote(int first, int last);
39.
40.     // Returns note at param index
41.     Note& noteAt(const int index);
42.     // Returns size of notes_ vector
43.     size_t notesSize() const;
44.
45. private:
46.     // title of the song
47.     string title_;
48.     // key signature of the song
49.     string keysignature_;
50.     // 2/4 signature is as the first element is 2 and the second is 4
51.     vector<int> timesignature_;
52.     // Tempo as beats per minute
53.     int tempo_;
54.     // notes and rests in one vector in order which
55.     // they appear in sheet music
56.     vector<Note> notes_;
57. };
58.
59. #endif // SHEETMUSIC_H
```

Liite 3. Note-luokka

```
1.  // Author Juho Itä
2.  #ifndef NOTE_H
3.  #define NOTE_H
4.  class Note
5.  {
6.  public:
7.      Note(); // default constructor
8.      Note(int length, float timeSec, long long timeStamp, float orgFreq,
9.          char pitch, int octave, char semitone, int dot, char corrected,
10.         bool checked); // parameter constructor
11.     ~Note(); // destructor
12.     Note(const Note& orgiginal); // copy constructor
13.
14.     const Note& operator=(const Note& rhs); // assignment operator
15.
16.     int getNoteLength() const {return length_;}
17.     float getTimeSec() const {return timeSec_;}
18.     long long getTimeStamp() const {return timeStamp_;}
19.     float getOrgFreq() const {return orgFreq_;}
20.     char getPitch() const {return pitch_;}
21.     int getOctave() const {return octave_;}
22.     char getSemitone() const {return semitone_;}
23.     int getDot() const {return dot_;}
24.     bool getChecked() const {return checked_;}
25.
26.     void setNoteLength(const int length) {length_ = length;}
27.     void setTimeSec(const float timeSec) {timeSec_ = timeSec;}
28.     void setTimeStamp(const long long timeStamp) {timeStamp_ = timeStamp;}
29.     void setOrgFreq(const float orgFreq) {orgFreq_ = orgFreq;}
30.     void setPitch(const char pitch) {pitch_ = pitch;}
31.     void setOctave(const int octave) {octave_ = octave;}
32.     void setSemitone(const char semitone) {semitone_ = semitone;}
33.     void setDot(const int dot) {dot_ = dot;}
34.     void setChecked(const bool checked) {checked_ = checked;}
35.
36. private:
37.     // length of the note
38.     int length_;
39.     // length of note in seconds or time stamp,
40.     // when sample of the note has been detected
41.     float timeSec_;
42.     // time stamp, when sample of the note has been detected
43.     long long timeStamp_;
44.     // Frequency before converting to note
45.     float orgFreq_;
46.     //  pitch of the note, r = rest
47.     char pitch_;
48.     // number indicating octave of the note
49.     int octave_;
50.     // indicates if the note is up=# or down=b one semitone,
51.     // empty=' ' means not up or down
52.     char semitone_;
53.     // indicates it the note is dotted
54.     int dot_;
55.     // indicates if the note has been corrected to certain key,
56.     // u=up, d=down, n=no
57.     char corrected_;
58.     // indicates how certain is it that this note is correctly detected
59.     // "not sure at all" = 0=0% and "definitely correct" = 100 = 100%
60.     int certainty_;
61.     // indicates that sample has been converted to note
62.     bool checked_;
63. };
64.
65. #endif // NOTE_H
```

Liite 4. openFunctions.cpp


```
1.  // Author Juho Itä
2.  #include "openFunctions.h"
3.  #include <fstream>    //ifstream
4.  #include <iostream> //cout, endl, cin
5.  #include <string.h> //strcpy
6.  #include <sstream>
7.  #include <cctype>
8.
9.  using std::fstream;
10. using std::ifstream;
11. using std::cin;
12. using std::cout;
13. using std::endl;
14. using std::istringstream;
15.
16. SheetMusic OpenLilyFileAndroid(string file_name)
17. {
18.     fstream infile;
19.     infile.open(file_name.c_str(), fstream::in);
20.
21.     SheetMusic melodyline;
22.     if(!infile)
23.     {
24.         cout << "Couldn't open " << file_name << endl;
25.         return melodyline;
26.     }
27.     else
28.     {
29.         string cdefgabr = "cdefgabr";
30.         string line;
31.         int note_len = 4; // Default length of note
32.         bool title_found = false;
33.         bool tempo_found = false;
34.         bool key_found = false;
35.         while(getline(infile, line)) // loop to get lines from file
36.         {
37.             istringstream iss(line);
38.
39.             // Find if there's title marked in the line,
40.             // set the title and change title_found to true if found
41.             if(!title_found && line.find("title") != string::npos)
42.             {
43.                 size_t first = line.find_first_of("\"") + 1;
44.                 size_t last = line.find_last_of("\"");
45.                 string title = line.substr(first, last-first);
46.                 melodyline.setTitle(title);
47.                 title_found = true;
48.             }
49.             // Find if the tempo is in the file
50.             else if(!tempo_found && line.find("\\tempo 4 = ")!=string::npos)
51.             {
52.                 size_t first = line.find_first_of("=") + 1;
53.                 // If there's space between = and 120, like "= 120"
54.                 if(line[first] == ' ')
55.                     first += 1;
56.                 string tempo = line.substr(first, line.length()-1);
```

(jatkuu)

```
57.                 int tempo_int;
58.                 // convert string tempo to int tempo_int
59.                 istringstream (tempo) >> tempo_int;
60.                 melodyline.setTempo(tempo_int);
61.                 tempo_found = true;
62.             }
63.             else if(!key_found && line.find("\\relative c'") !=string::npos)
64.             {
65.                 // Do nothing because key is set to relative c as default
66.                 key_found = true;
67.             }
68.             else if(line[0] == '{')
69.             {
70.                 while(getline(infile, line))
71.                 {
72.                     for(unsigned i = 0; i < line.length(); i++)
73.                     {
74.                         if(line[i] == '\t' || line[i] == ' ')
75.                             continue;
76.                         for(unsigned j = 0; j < cdefgabr.length(); j++)
77.                         {
78.                             if(isdigit(line[i+1]))
79.                             {
80.                                 // Converts char to ASCII value "1 = 49"
81.                                 // Reduce 48 to get right value
82.                                 note_len = line[i+1] - 48;
83.                             }
84.                             if(line[i]==cdefgabr[j])
85.                             {
86.                                 Note tmp_note;
87.                                 tmp_note.setNoteLength(note_len);
88.                                 tmp_note.setPitch(line[i]);
89.                                 melodyline.addNote(tmp_note);
90.                             }
91.
92.                         }
93.                     }
94.                     if(line[0] == '}')
95.                         break;
96.                 }
97.             }
98.         }
99.     }
100.         return melodyline;
101.     }
```

Liite 5. Normaali Ukko Nooa

Näyte	Nuotti	Oktaavi	Aikaleima	Erotus	Aikaleima (s)	Taajuus
1	c	2	192	1	3,912	1046,88
2	c	2	193	1	3,93238	1046,88
3	c	2	194	1	3,95275	1046,88
4	c	2	195	1	3,97313	1046,88
5	c	2	196	1	3,9935	1046,88
6	c	2	197	1	4,01388	1046,88
7	c	2	198	1	4,03425	1046,88
8	c	2	199	1	4,05463	1046,88
9	c	2	200	1	4,075	1046,88
10	c	2	201	1	4,09538	1046,88
11	c	2	202	1	4,11575	1046,88
12	c	2	203	1	4,13613	1046,88
13	c	2	204	1	4,1565	1046,88
14	c	2	205	1	4,17688	1046,88
15	c	2	206	1	4,19725	1046,88
16	c	2	207	1	4,21763	1046,88
17	c	2	208	1	4,238	1046,88
18	c	2	209	1	4,25838	1046,88
19	c	2	210	1	4,27875	1046,88
20	c	2	211	1	4,29913	1054,69
21	c	2	212	1	4,3195	1054,69
22	c	2	213	1	4,33988	1054,69
23	c	2	214	1	4,36025	1046,88
24	c	2	215	1	4,38063	1046,88
25	c	2	216	1	4,401	1046,88
26	c	2	217	1	4,42138	1046,88
27	c	2	218	1	4,44175	1046,88
28	c	2	219	1	4,46213	1046,88
29	c	2	220	1	4,4825	1046,88
30	c	2	221	1	4,50288	1046,88
31	c	2	222	1	4,52325	1046,88
32	c	2	223	1	4,54363	1046,88
33	c	2	224	1	4,564	1046,88
34	c	2	225	1	4,58438	1046,88
35	c	2	226	1	4,60475	1046,88
36	c	2	227	1	4,62513	1046,88
37	c	2	228	1	4,6455	1046,88
38	c	2	229	1	4,66588	1046,88
39	c	2	230	1	4,68625	1046,88
40	c	2	231	1	4,70663	1046,88
41	c	2	232	1	4,727	1046,88
42	c	2	233	1	4,74738	1046,88
43	c	2	234	1	4,76775	1046,88
44	c	2	235	1	4,78813	1046,88
45	c	2	236	1	4,8085	1039,06
46	c	2	237	1	4,82888	1039,06
47	c	2	238	1	4,84925	1046,88
48	c	2	239	1	4,86963	1046,88
49	c	2	240	1	4,89	1046,88
50	c	2	241	1	4,91038	1046,88
51	c	2	242	1	4,93075	1046,88
52	c	2	243	1	4,95113	1046,88
53	c	2	244	1	4,9715	1046,88
54	c	2	245	1	4,99188	1046,88
55	c	2	246	1	5,01225	1046,88
56	c	2	247	1	5,03263	1046,88
57	c	2	248	1	5,053	1046,88

(jatkuu)

58	c	2	249	1	5,07338	1046,88
59	c	2	250	1	5,09375	1046,88
60	c	2	251	1	5,11413	1046,88
61	c	2	252	1	5,1345	1046,88
62	c	2	253	1	5,15488	1062,5
63	c	2	254	7	5,17525	1062,5
64	e	2	261	1	5,31788	1328,12
65	e	2	262	1	5,33825	1328,12
66	e	2	263	1	5,35863	1328,12
67	e	2	264	1	5,379	1328,12
68	e	2	265	1	5,39938	1328,12
69	e	2	266	1	5,41975	1328,12
70	e	2	267	1	5,44013	1328,12
71	e	2	268	1	5,4605	1328,12
72	e	2	269	1	5,48088	1328,12
73	e	2	270	1	5,50125	1328,12
74	e	2	271	1	5,52163	1328,12
75	e	2	272	1	5,542	1328,12
76	e	2	273	1	5,56238	1328,12
77	e	2	274	1	5,58275	1320,31
78	e	2	275	1	5,60313	1320,31
79	e	2	276	1	5,6235	1320,31
80	e	2	277	1	5,64388	1328,12
81	d#	2	278	1	5,66425	1234,38
82	d#	2	279	5	5,68463	1242,19
83	d	2	284	2	5,7865	1179,69
84	d	2	286	1	5,82725	1179,69
85	d	2	287	1	5,84763	1179,69
86	d	2	288	1	5,868	1179,69
87	d	2	289	1	5,88838	1179,69
88	d	2	290	1	5,90875	1179,69
89	d	2	291	1	5,92913	1179,69
90	d	2	292	1	5,9495	1179,69
91	d	2	293	1	5,96988	1179,69
92	d	2	294	1	5,99025	1179,69
93	d	2	295	1	6,01063	1179,69
94	d	2	296	1	6,031	1179,69
95	d	2	297	1	6,05138	1171,88
96	d	2	298	1	6,07175	1171,88
97	d	2	299	1	6,09213	1171,88
98	d	2	300	1	6,1125	1171,88
99	d	2	301	1	6,13288	1171,88
100	d	2	302	1	6,15325	1171,88
101	d	2	303	1	6,17363	1171,88
102	d	2	304	1	6,194	1179,69
103	d	2	305	1	6,21438	1179,69
104	d	2	306	5	6,23475	1179,69
105	d	2	311	1	6,33663	1179,69
106	d	2	312	1	6,357	1179,69
107	d	2	313	1	6,37738	1179,69
108	d	2	314	1	6,39775	1179,69
109	d	2	315	1	6,41813	1179,69
110	d	2	316	1	6,4385	1179,69
111	d	2	317	1	6,45888	1179,69
112	d	2	318	1	6,47925	1179,69
113	d	2	319	1	6,49963	1179,69
114	d	2	320	1	6,52	1179,69
115	d	2	321	1	6,54038	1179,69
116	d	2	322	1	6,56075	1179,69
117	d	2	323	1	6,58113	1179,69
118	d	2	324	1	6,6015	1179,69

119	d	2	325	1	6,62188	1179,69
120	d	2	326	1	6,64225	1179,69
121	d	2	327	1	6,66263	1179,69
122	d	2	328	1	6,683	1179,69
123	d	2	329	1	6,70338	1179,69
124	d	2	330	1	6,72375	1187,5
125	d	2	331	1	6,74413	1187,5
126	d	2	332	1	6,7645	1187,5
127	d	2	333	1	6,78488	1187,5
128	d	2	334	1	6,80525	1187,5
129	d	2	335	1	6,82563	1187,5
130	d	2	336	1	6,846	1187,5
131	d	2	337	1	6,86638	1179,69
132	d	2	338	1	6,88675	1179,69
133	d	2	339	1	6,90713	1179,69
134	d	2	340	1	6,9275	1179,69
135	d	2	341	1	6,94788	1179,69
136	d	2	342	1	6,96825	1187,5
137	d	2	343	1	6,98863	1187,5
138	d	2	344	1	7,009	1187,5
139	c#	2	345	2	7,02938	1140,62
140	c#	3	347	14	7,07013	2234,38
141	f	2	361	1	7,35538	1406,25
142	f	2	362	1	7,37575	1406,25
143	f	2	363	1	7,39613	1406,25
144	f	2	364	1	7,4165	1406,25
145	f	2	365	1	7,43688	1398,44
146	f	2	366	1	7,45725	1406,25
147	f	2	367	1	7,47763	1406,25
148	f	2	368	1	7,498	1406,25
149	f	2	369	1	7,51838	1406,25
150	f	2	370	1	7,53875	1398,44
151	f	2	371	1	7,55913	1398,44
152	f	2	372	1	7,5795	1398,44
153	f	2	373	1	7,59988	1406,25
154	f	2	374	1	7,62025	1398,44
155	f	2	375	1	7,64063	1406,25
156	e	2	376	1	7,661	1343,75
157	e	2	377	1	7,68138	1343,75
158	e	2	378	6	7,70175	1343,75
159	e	2	384	1	7,824	1328,12
160	e	2	385	1	7,84438	1328,12
161	e	2	386	1	7,86475	1328,12
162	e	2	387	1	7,88513	1328,12
163	e	2	388	1	7,9055	1328,12
164	e	2	389	1	7,92588	1320,31
165	e	2	390	1	7,94625	1320,31
166	e	2	391	1	7,96663	1320,31
167	e	2	392	1	7,987	1320,31
168	e	2	393	1	8,00738	1320,31
169	e	2	394	1	8,02775	1320,31
170	e	2	395	1	8,04813	1320,31
171	e	2	396	1	8,0685	1320,31
172	e	2	397	1	8,08888	1320,31
173	e	2	398	1	8,10925	1320,31
174	e	2	399	1	8,12963	1320,31
175	e	2	400	1	8,15	1320,31
176	e	2	401	1	8,17038	1328,12
177	e	2	402	1	8,19075	1328,12
178	e	2	403	1	8,21113	1328,12
179	e	2	404	1	8,2315	1328,12

180	e	2	405	1	8,25187	1328,12
181	e	2	406	1	8,27225	1328,12
182	e	2	407	1	8,29263	1328,12
183	d#	2	408	1	8,313	1273,44
184	e	2	409	1	8,33337	1328,12
185	e	2	410	1	8,35375	1328,12
186	e	2	411	1	8,37413	1328,12
187	e	2	412	1	8,3945	1320,31
188	e	2	413	1	8,41488	1328,12
189	e	2	414	1	8,43525	1328,12
190	e	2	415	1	8,45563	1328,12
191	e	2	416	1	8,476	1328,12
192	e	2	417	1	8,49638	1328,12
193	e	2	418	1	8,51675	1328,12
194	e	2	419	1	8,53713	1328,12
195	e	2	420	1	8,5575	1328,12
196	e	2	421	1	8,57788	1328,12
197	e	2	422	1	8,59825	1328,12
198	e	2	423	1	8,61863	1328,12
199	e	2	424	1	8,639	1320,31
200	e	2	425	1	8,65938	1320,31
201	e	2	426	1	8,67975	1320,31
202	e	2	427	1	8,70013	1320,31
203	e	2	428	1	8,7205	1320,31
204	d#	2	429	1	8,74088	1218,75
205	d#	2	430	1	8,76125	1234,38
206	d#	2	431	5	8,78163	1218,75
207	d	2	436	1	8,8835	1179,69
208	d	2	437	1	8,90388	1171,88
209	d	2	438	1	8,92425	1171,88
210	d	2	439	1	8,94463	1179,69
211	d	2	440	1	8,965	1179,69
212	d	2	441	1	8,98538	1179,69
213	d	2	442	1	9,00575	1171,88
214	d	2	443	1	9,02612	1171,88
215	d	2	444	1	9,0465	1171,88
216	d	2	445	1	9,06688	1171,88
217	d	2	446	1	9,08725	1171,88
218	d	2	447	1	9,10763	1171,88
219	d	2	448	1	9,128	1171,88
220	d	2	449	1	9,14838	1171,88
221	d	2	450	1	9,16875	1171,88
222	d	2	451	1	9,18913	1171,88
223	d	2	452	1	9,2095	1171,88
224	d	2	453	1	9,22988	1171,88
225	d	2	454	1	9,25025	1179,69
226	d	2	455	3	9,27063	1179,69
227	d	2	458	1	9,33175	1187,5
228	d	2	459	1	9,35213	1187,5
229	d	2	460	1	9,3725	1187,5
230	d	2	461	1	9,39288	1179,69
231	d	2	462	1	9,41325	1187,5
232	d	2	463	1	9,43363	1179,69
233	d	2	464	1	9,454	1179,69
234	d	2	465	1	9,47438	1179,69
235	d	2	466	1	9,49475	1179,69
236	d	2	467	1	9,51513	1179,69
237	d	2	468	1	9,5355	1179,69
238	d	2	469	1	9,55588	1179,69
239	d	2	470	1	9,57625	1179,69
240	d	2	471	1	9,59663	1179,69

241	d	2	472	1	9,617	1171,88
242	d	2	473	1	9,63738	1171,88
243	d	2	474	10	9,65775	1171,88
244	c	2	484	1	9,8615	1054,69
245	c	2	485	1	9,88188	1046,88
246	c	2	486	1	9,90225	1046,88
247	c	2	487	1	9,92263	1046,88
248	c	2	488	1	9,943	1046,88
249	c	2	489	1	9,96338	1046,88
250	c	2	490	1	9,98375	1046,88
251	c	2	491	1	10,0041	1046,88
252	c	2	492	1	10,0245	1046,88
253	c	2	493	1	10,0449	1046,88
254	c	2	494	1	10,0653	1046,88
255	c	2	495	1	10,0856	1046,88
256	c	2	496	1	10,106	1046,88
257	c	2	497	1	10,1264	1046,88
258	c	2	498	1	10,1468	1046,88
259	c	2	499	1	10,1671	1046,88
260	c	2	500	1	10,1875	1046,88
261	c	2	501	1	10,2079	1046,88
262	c	2	502	1	10,2283	1046,88
263	c	2	503	1	10,2486	1046,88
264	c	2	504	1	10,269	1046,88
265	c	2	505	1	10,2894	1046,88
266	c	2	506	1	10,3098	1046,88
267	c	2	507	1	10,3301	1046,88
268	c	2	508	1	10,3505	1046,88
269	c	2	509	1	10,3709	1046,88
270	c	2	510	1	10,3913	1046,88
271	c	2	511	1	10,4116	1046,88
272	c	2	512	1	10,4116	1046,88
273	c	2	513	1	10,432	1046,88
274	c	2	514	1	10,4524	1046,88
275	c	2	515	1	10,4728	1046,88
276	c	2	516	1	10,4931	1046,88
277	c	2	517	1	10,5135	1046,88
278	c	2	518	1	10,5339	1046,88
279	c	2	519	1	10,5543	1046,88
280	c	2	520	1	10,5746	1046,88
281	c	2	521	1	10,595	1046,88
282	c	2	522	1	10,6154	1046,88
283	c	2	523	1	10,6358	1046,88
284	c	2	524	1	10,6561	1046,88
285	c	2	525	1	10,6765	1046,88
286	c	2	526	1	10,6969	1046,88
287	c	2	527	1	10,7173	1046,88
288	c	2	528	1	10,7376	1046,88
289	c	2	529	1	10,758	1046,88
290	c	2	530	1	10,7784	1046,88
291	c	2	531	1	10,7988	1046,88
292	c	2	532	1	10,8191	1046,88
293	c	2	533	1	10,8395	1046,88
294	c	2	534	1	10,8599	1046,88
295	c	2	535	1	10,8803	1046,88
296	c	2	536	1	10,9006	1046,88
297	c	2	537	1	10,921	1046,88
298	c	2	538	1	10,9414	1046,88
299	c	2	539	1	10,9618	1046,88
300	c	2	540	1	10,9821	1046,88
301	c	2	541	1	11,0025	1046,88

302	c	2	542	1	11,0229	1046,88
303	c	2	543	1	11,0433	1046,88
304	c	2	544	1	11,0636	1046,88
305	c	2	545	1	11,084	1046,88
306	c	2	546	1	11,1044	1046,88
307	c	2	547	1	11,1248	1046,88
308	c	2	548	1	11,1451	1046,88
309	c	2	549	1	11,1655	1046,88
310	c	2	550	1	11,1859	1046,88
311	c	2	551	1	11,2063	1046,88
312	c	2	552	1	11,2266	1046,88
313	c	2	553	1	11,247	1046,88
314	c	2	554	1	11,2674	1046,88
315	c	2	555	1	11,2878	1046,88
316	c	2	556	1	11,3081	1046,88
317	c	2	557	1	11,3285	1046,88
318	c	2	558	1	11,3489	1046,88
319	c	2	559	1	11,3693	1046,88
320	c	2	560	1	11,3896	1046,88
321	c	2	561	1	11,41	1046,88
322	c	2	562	21	11,4304	1046,88
323	e	2	583	1	11,8786	1312,5
324	e	2	584	2	11,8786	1312,5
325	e	2	586	1	11,9398	1312,5
326	e	2	587	1	11,9398	1312,5
327	e	2	588	1	11,9601	1312,5
328	e	2	589	1	11,9805	1312,5
329	e	2	590	1	12,0009	1312,5
330	e	2	591	1	12,0213	1312,5
331	e	2	592	1	12,0416	1312,5
332	e	2	593	1	12,062	1312,5
333	e	2	594	1	12,0824	1312,5
334	e	2	595	1	12,1028	1312,5
335	e	2	596	1	12,1231	1312,5
336	e	2	597	1	12,1435	1312,5
337	e	2	598	1	12,1639	1312,5
338	e	2	599	1	12,1843	1312,5
339	e	2	600	1	12,2046	1312,5
340	e	2	601	1	12,225	1312,5
341	e	2	602	1	12,2454	1312,5
342	e	2	603	1	12,2658	1312,5
343	e	2	604	1	12,2861	1312,5
344	e	2	605	1	12,3065	1312,5
345	e	2	606	1	12,3269	1312,5
346	e	2	607	1	12,3473	1320,31
347	e	2	608	1	12,3676	1320,31
348	e	2	609	1	12,388	1320,31
349	e	2	610	1	12,4084	1320,31
350	e	2	611	1	12,4288	1320,31
351	e	2	612	1	12,4491	1320,31
352	e	2	613	1	12,4695	1320,31
353	e	2	614	1	12,4899	1320,31
354	e	2	615	1	12,5103	1320,31
355	e	2	616	1	12,5306	1312,5
356	e	2	617	1	12,551	1312,5
357	e	2	618	1	12,5714	1312,5
358	e	2	619	1	12,5918	1312,5
359	e	2	620	1	12,6121	1312,5
360	e	2	621	1	12,6325	1312,5
361	e	2	622	1	12,6529	1312,5
362	e	2	623	1	12,6733	1312,5

363	e	2	624	1	12,6936	1312,5
364	e	2	625	1	12,714	1312,5
365	e	2	626	1	12,7344	1320,31
366	e	2	627	2	12,7548	1328,12
367	e	2	629	4	12,8159	1328,12
368	e	2	633	1	12,8974	1328,12
369	e	2	634	1	12,8974	1320,31
370	e	2	635	1	12,9178	1320,31
371	e	2	636	1	12,9381	1320,31
372	e	2	637	1	12,9585	1320,31
373	e	2	638	1	12,9789	1320,31
374	e	2	639	1	12,9993	1320,31
375	e	2	640	1	13,0196	1320,31
376	e	2	641	1	13,04	1320,31
377	e	2	642	1	13,0604	1312,5
378	e	2	643	1	13,0808	1312,5
379	e	2	644	1	13,1011	1312,5
380	e	2	645	1	13,1215	1312,5
381	e	2	646	1	13,1419	1312,5
382	e	2	647	1	13,1623	1312,5
383	e	2	648	1	13,1826	1312,5
384	e	2	649	1	13,203	1312,5
385	e	2	650	1	13,2234	1312,5
386	e	2	651	1	13,2438	1320,31
387	e	2	652	1	13,2641	1320,31
388	e	2	653	3	13,2845	1320,31
389	e	2	656	1	13,366	1328,12
390	e	2	657	1	13,366	1320,31
391	e	2	658	1	13,3864	1320,31
392	e	2	659	1	13,4068	1312,5
393	e	2	660	1	13,4271	1312,5
394	e	2	661	1	13,4475	1312,5
395	e	2	662	1	13,4679	1320,31
396	e	2	663	1	13,4883	1312,5
397	e	2	664	1	13,5086	1312,5
398	e	2	665	1	13,529	1320,31
399	e	2	666	1	13,5494	1328,12
400	e	2	667	8	13,5698	1320,31
401	e	2	675	2	13,7531	1328,12
402	e	2	677	6	13,7939	1320,31
403	g	2	683	1	13,9161	1570,31
404	g	2	684	1	13,9161	1570,31
405	g	2	685	1	13,9365	1570,31
406	g	2	686	1	13,9569	1570,31
407	g	2	687	1	13,9773	1578,12
408	g	2	688	1	13,9976	1578,12
409	g	2	689	1	14,018	1570,31
410	g	2	690	1	14,0384	1570,31
411	g	2	691	1	14,0588	1570,31
412	g	2	692	1	14,0791	1570,31
413	g	2	693	1	14,0995	1570,31
414	g	2	694	1	14,1199	1570,31
415	g	2	695	1	14,1403	1570,31
416	g	2	696	1	14,1606	1570,31
417	g	2	697	1	14,181	1570,31
418	g	2	698	1	14,2014	1570,31
419	g	2	699	1	14,2218	1570,31
420	g	2	700	1	14,2421	1570,31
421	g	2	701	1	14,2625	1570,31
422	g	2	702	1	14,2829	1570,31
423	g	2	703	1	14,3033	1570,31

424	g	2	704	1	14,3236	1570,31
425	g	2	705	1	14,344	1570,31
426	g	2	706	1	14,3644	1570,31
427	g	2	707	1	14,3848	1562,5
428	g	2	708	1	14,4051	1562,5
429	g	2	709	1	14,4255	1562,5
430	g	2	710	1	14,4459	1562,5
431	g	2	711	1	14,4663	1562,5
432	g	2	712	1	14,4866	1562,5
433	g	2	713	1	14,507	1562,5
434	g	2	714	1	14,5274	1562,5
435	g	2	715	1	14,5478	1562,5
436	g	2	716	1	14,5681	1562,5
437	g	2	717	1	14,5885	1562,5
438	g	2	718	1	14,6089	1562,5
439	g	2	719	1	14,6293	1562,5
440	g	2	720	1	14,6496	1562,5
441	g	2	721	1	14,67	1562,5
442	g	2	722	1	14,6904	1562,5
443	g	2	723	1	14,7108	1562,5
444	g	2	724	1	14,7311	1562,5
445	f	2	725	5	14,7719	1437,5
446	f	2	730	1	14,8738	1398,44
447	f	2	731	1	14,8738	1398,44
448	f	2	732	1	14,8941	1390,62
449	f	2	733	1	14,9145	1390,62
450	f	2	734	1	14,9349	1390,62
451	f	2	735	1	14,9553	1390,62
452	f	2	736	1	14,9756	1390,62
453	f	2	737	1	14,996	1390,62
454	f	2	738	1	15,0164	1390,62
455	f	2	739	1	15,0368	1390,62
456	f	2	740	1	15,0571	1390,62
457	f	2	741	1	15,0775	1390,62
458	f	2	742	1	15,0979	1390,62
459	f	2	743	1	15,1183	1390,62
460	f	2	744	1	15,1386	1390,62
461	f	2	745	1	15,159	1390,62
462	f	2	746	1	15,1794	1390,62
463	f	2	747	1	15,1998	1390,62
464	f	2	748	1	15,2201	1390,62
465	f	2	749	1	15,2405	1390,62
466	f	2	750	1	15,2609	1390,62
467	f	2	751	1	15,2813	1390,62
468	f	2	752	1	15,3016	1390,62
469	f	2	753	1	15,322	1390,62
470	f	2	754	1	15,3424	1390,62
471	f	2	755	1	15,3628	1390,62
472	f	2	756	1	15,3831	1390,62
473	f	2	757	1	15,4035	1390,62
474	f	2	758	1	15,4239	1390,62
475	f	2	759	1	15,4443	1390,62
476	f	2	760	1	15,4646	1390,62
477	f	2	761	1	15,485	1390,62
478	f	2	762	1	15,5054	1390,62
479	f	2	763	10	15,5258	1390,62
480	e	2	773	7	15,7499	1320,31
481	d	2	780	1	15,8925	1171,88

Liite 6. Staccato Ukko Nooa

Näyte	Nuotti	Oktaavi	Aikaleima	Erotus	Pituus	Aikaleima (s)	Taajuus
1	c	1	117	2		2,38388	531,25
2	c	1	119	1		2,42463	515,625
3	c	1	120	1		2,445	515,625
4	c	1	121	1		2,46538	515,625
5	c	1	122	1		2,48575	515,625
6	c	1	123	1		2,50613	515,625
7	**c**	**1**	**124**	**17**	**7**	**2,5265**	**515,625**
8	c	1	141	2		2,87288	531,25
9	c	1	143	2		2,91363	515,625
10	c	1	145	1		2,95438	515,625
11	c	1	146	1		2,97475	515,625
12	**c**	**1**	**147**	**16**	**6**	**2,99513**	**515,625**
13	c	1	163	2		3,32113	531,25
14	c	1	165	2		3,36188	531,25
15	c	1	167	1		3,40263	515,625
16	c	1	168	1		3,423	515,625
17	**c**	**1**	**169**	**18**	**6**	**3,44338**	**515,625**
18	e	1	187	2		3,81013	656,25
19	e	1	189	1		3,85088	656,25
20	e	1	190	1		3,87125	656,25
21	e	1	191	1		3,89163	656,25
22	**e**	**1**	**192**	**20**	**5**	**3,912**	**656,25**
23	d	1	212	1		4,3195	593,75
24	d	1	213	1		4,33988	585,938
25	d	1	214	1		4,36025	585,938
26	d	1	215	1		4,38063	585,938
27	**d**	**1**	**216**	**22**	**4**	**4,401**	**585,938**
28	d	1	238	1		4,84925	585,938
29	d	1	239	1		4,86963	585,938
30	d	1	240	1		4,89	585,938
31	**d**	**1**	**241**	**21**	**3**	**4,91038**	**578,125**
32	d	1	262	1		5,33825	593,75
33	d	1	263	2		5,35863	585,938
34	d	1	265	1		5,39938	585,938
35	d	1	266	1		5,41975	585,938
36	d	1	267	1		5,44013	585,938
37	d	1	268	1		5,4605	585,938
38	d	1	269	1		5,48088	585,938
39	d	1	270	1		5,50125	585,938
40	**d**	**1**	**271**	**18**	**9**	**5,52163**	**585,938**
41	f	1	289	1		5,88838	695,312
42	f	1	290	1		5,90875	703,125
43	f	1	291	1		5,92913	703,125
44	f	1	292	1		5,9495	703,125
45	f	1	293	1		5,96988	703,125
46	f	1	294	1		5,99025	703,125
47	f	1	295	1		6,01063	703,125
48	f	1	296	1		6,031	703,125
49	f	1	297	1		6,05138	710,938
50	**f**	**1**	**298**	**13**	**9**	**6,07175**	**710,938**
51	e	1	311	1		6,33663	656,25
52	e	1	312	1		6,357	656,25
53	e	1	313	1		6,37738	656,25
54	e	1	314	1		6,39775	656,25
55	e	1	315	1		6,41813	656,25
56	e	1	316	1		6,4385	656,25
57	e	1	317	1		6,45888	656,25

58	e	1	318	1		6,47925	656,25
59	e	1	319	1		6,49963	656,25
60	e	1	320	1		6,52	656,25
61	**e**	**1**	**321**	**15**	**10**	**6,54038**	**664,062**
62	e	1	336	1		6,846	656,25
63	e	1	337	2		6,86638	656,25
64	e	1	339	1		6,90713	656,25
65	e	1	340	1		6,9275	656,25
66	e	1	341	1		6,94788	656,25
67	e	1	342	1		6,96825	656,25
68	e	1	343	3		6,98863	648,438
69	e	1	346	3		7,04975	656,25
70	e	1	349	2		7,11088	656,25
71	**e**	**1**	**351**	**10**	**15**	**7,15163**	**656,25**
72	d	1	361	1		7,35538	593,75
73	d	1	362	2		7,37575	585,938
74	**d**	**1**	**364**	**20**	**3**	**7,4165**	**585,938**
75	d	1	384	1		7,824	593,75
76	d	1	385	2		7,84438	585,938
77	**d**	**1**	**387**	**23**	**3**	**7,88513**	**585,938**
78	c	1	410	1		8,35375	531,25
79	c	1	411	1		8,37413	531,25
80	c	1	412	1		8,3945	531,25
81	c	1	413	1		8,41488	531,25
82	c	1	414	1		8,43525	531,25
83	c	1	415	1		8,45563	523,438
84	c	1	416	1		8,476	523,438
85	c	1	417	1		8,49638	523,438
86	c	1	418	1		8,51675	523,438
87	c	1	419	1		8,53713	523,438
88	c	1	420	1		8,5575	531,25
89	c	1	421	1		8,57788	523,438
90	c	1	422	1		8,59825	531,25
91	c	1	423	1		8,61863	531,25
92	c	1	424	1		8,639	523,438
93	c	1	425	1		8,65938	531,25
94	c	1	426	1		8,67975	531,25
95	c	1	427	1		8,70013	523,438
96	c	1	428	1		8,7205	531,25
97	c	1	429	1		8,74088	531,25
98	c	1	430	1		8,76125	531,25
99	c	1	431	1		8,78163	531,25
100	c	1	432	1		8,802	531,25
101	c	1	433	1		8,82238	531,25
102	c	1	434	1		8,84275	531,25
103	c	1	435	1		8,86313	531,25
104	c	1	436	1		8,8835	523,438
105	c	1	437	1		8,90388	523,438
106	c	1	438	1		8,92425	523,438
107	c	1	439	1		8,94463	531,25
108	c	1	440	1		8,965	531,25
109	c	1	441	1		8,98538	523,438
110	c	1	442	1		9,00575	531,25
111	c	1	443	1		9,02612	523,438
112	c	1	444	1		9,0465	531,25
113	c	1	445	1		9,06688	531,25
114	c	1	446	1		9,08725	531,25
115	c	1	447	1		9,10763	531,25
116	c	1	448	1		9,128	531,25
117	c	1	449	1		9,14838	523,438
118	c	1	450	1		9,16875	531,25

119	c	1	451	1		9,18913	523,438
120	c	1	452	1		9,2095	531,25
121	c	1	453	1		9,22988	531,25
122	c	1	454	1		9,25025	531,25
123	c	1	455	1		9,27063	531,25
124	c	1	456	1		9,291	531,25
125	c	1	457	1		9,31138	523,438
126	c	1	458	1		9,33175	531,25
127	c	1	459	1		9,35213	531,25
128	c	1	460	1		9,3725	523,438
129	c	1	461	1		9,39288	531,25
130	c	1	462	1		9,41325	523,438
131	c	1	463	1		9,43363	531,25
132	c	1	464	1		9,454	531,25
133	c	1	465	1		9,47438	523,438
134	c	1	466	1		9,49475	531,25
135	c	1	467	1		9,51513	523,438
136	c	1	468	1		9,5355	523,438
137	c	1	469	1		9,55588	531,25
138	c	1	470	1		9,57625	523,438
139	c	1	471	1		9,59663	531,25
140	c	1	472	1		9,617	531,25
141	c	1	473	1		9,63738	523,438
142	c	1	474	1		9,65775	531,25
143	c	1	475	1		9,67813	531,25
144	c	1	476	1		9,6985	531,25
145	c	1	477	1		9,71888	531,25
146	c	1	478	1		9,73925	523,438
147	c	1	479	1		9,75963	531,25
148	c	1	480	1		9,78	531,25
149	c	1	481	1		9,80038	531,25
150	c	1	482	1		9,82075	531,25
151	**c**	**1**	**483**	**27**	**73**	**9,84113**	**523,438**
152	e	1	510	1		10,3913	656,25
153	e	1	511	1		10,4116	664,062
154	e	1	512	1		10,4116	656,25
155	e	1	513	1		10,432	656,25
156	e	1	514	1		10,4524	656,25
157	e	1	515	1		10,4728	656,25
158	e	1	516	1		10,4931	656,25
159	e	1	517	1		10,5135	656,25
160	**e**	**1**	**518**	**16**	**8**	**10,5339**	**656,25**
161	e	1	534	1		10,8803	656,25
162	e	1	535	1		10,8803	656,25
163	e	1	536	1		10,9006	656,25
164	e	1	537	1		10,921	656,25
165	e	1	538	1		10,9414	656,25
166	e	1	539	1		10,9618	656,25
167	e	1	540	1		10,9821	656,25
168	e	1	541	1		11,0025	656,25
169	**e**	**1**	**542**	**16**	**8**	**11,0229**	**664,062**
170	e	1	558	1		11,3693	656,25
171	e	1	559	1		11,3693	656,25
172	e	1	560	1		11,3896	656,25
173	e	1	561	1		11,41	656,25
174	e	1	562	1		11,4304	656,25
175	e	1	563	1		11,4508	656,25
176	e	1	564	1		11,4711	656,25
177	**e**	**1**	**565**	**16**	**7**	**11,4915**	**656,25**
178	e	1	581	1		11,8379	656,25
179	e	1	582	1		11,8379	656,25

180	e	1	583	1		11,8583	656,25
181	e	1	584	1		11,8786	656,25
182	e	1	585	1		11,899	656,25
183	e	1	586	1		11,9194	656,25
184	e	1	587	1		11,9398	656,25
185	e	1	588	1		11,9601	656,25
186	**e**	**1**	**589**	**19**	**8**	**11,9805**	**648,438**
187	g	1	608	1		12,388	781,25
188	g	1	609	1		12,388	781,25
189	g	1	610	1		12,4084	781,25
190	g	1	611	1		12,4288	781,25
191	g	1	612	1		12,4491	781,25
192	g	1	613	1		12,4695	781,25
193	g	1	614	1		12,4899	781,25
194	g	1	615	1		12,5103	781,25
195	g	1	616	1		12,5306	781,25
196	g	1	617	1		12,551	781,25
197	g	1	618	1		12,5714	781,25
198	g	1	619	1		12,5918	781,25
199	g	1	620	1		12,6121	781,25
200	g	1	621	1		12,6325	781,25
201	g	1	622	1		12,6529	781,25
202	g	1	623	1		12,6733	781,25
203	g	1	624	1		12,6936	781,25
204	g	1	625	1		12,714	781,25
205	g	1	626	1		12,7344	781,25
206	g	1	627	1		12,7548	781,25
207	g	1	628	1		12,7751	781,25
208	g	1	629	1		12,7955	781,25
209	g	1	630	1		12,8159	781,25
210	g	1	631	1		12,8363	781,25
211	g	1	632	1		12,8566	781,25
212	g	1	633	1		12,877	781,25
213	g	1	634	1		12,8974	781,25
214	g	1	635	1		12,9178	781,25
215	g	1	636	1		12,9381	781,25
216	g	1	637	1		12,9585	781,25
217	g	1	638	1		12,9789	781,25
218	g	1	639	1		12,9993	781,25
219	g	1	640	1		13,0196	781,25
220	g	1	641	1		13,04	781,25
221	g	1	642	1		13,0604	781,25
222	g	1	643	1		13,0808	781,25
223	g	1	644	1		13,1011	781,25
224	g	1	645	1		13,1215	781,25
225	g	1	646	1		13,1419	781,25
226	**g**	**1**	**647**	**9**	**39**	**13,1623**	**781,25**
227	f	1	656	1		13,366	703,125
228	f	1	657	1		13,366	703,125
229	f	1	658	1		13,3864	703,125
230	f	1	659	1		13,4068	703,125
231	f	1	660	1		13,4271	703,125
232	f	1	661	1		13,4475	703,125
233	f	1	662	1		13,4679	703,125
234	f	1	663	1		13,4883	703,125
235	f	1	664	1		13,5086	703,125
236	f	1	665	1		13,529	703,125
237	f	1	666	1		13,5494	703,125
238	f	1	667	1		13,5698	695,312
239	f	1	668	1		13,5901	695,312
240	f	1	669	1		13,6105	695,312

241	f	1	670	1		13,6309	695,312
242	f	1	671	1		13,6513	695,312
243	f	1	672	1		13,6716	703,125
244	f	1	673	1		13,692	703,125
245	f	1	674	1		13,7124	703,125
246	f	1	675	1		13,7328	703,125
247	f	1	676	1		13,7531	703,125
248	f	1	677	1		13,7735	703,125
249	f	1	678	1		13,7939	703,125
250	f	1	679	1		13,8143	703,125
251	f	1	680	1		13,8346	703,125
252	f	1	681	1		13,855	703,125
253	f	1	682	1		13,8754	703,125
254	f	1	683	1		13,8958	703,125
255	**f**	**1**	**684**	**19**	**28**	**13,9161**	**703,125**
256	d	1	703	1		14,3236	593,75