Tallinna Ülikool

Digitehnoloogiate instituut

MIDI FAILE VÕRDLEVA TARKVARA ARENDAMINE JA TESTIMINE

Bakalaureusetöö

Autor: Are Kangus

Juhendaja: Jaagup Kippar

Autor:..... ,,..... ,, 2017

Juhendaja:..... ,,..... ,, 2017

Instituudi direktor:..... ,,..... ,, 2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et kaesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (allkiri)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Are Kangus (sünnikuupäev: 11. juuni 1994)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „MIDI faile võrdleva tarkvara arendamine ja testimine”, mille juhendaja on Jaagup Kippar säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

....

(kuupäev) (allkiri

Sisukord

Sissejuhatus

Tänapäeval on muusikatööstus suur majandusharu, mille globaalne turuväärtus ulatub miljarditesse eurodesse ning on põhjust arvata, et aja kulgedes see suureneb (Forde, 2016). Kuna muusikat peetakse kultuuriliselt tähtsaks ning selles äris tippu jõudnud artistid on hästi tasustatud, siis esineb aeg-ajalt süüdistusi muusikutele, kelle loodud teoseid on kahtlustatud loomevarguses. Samuti võiks selline rakendus olla ka kasulik heliloojatele, kes soovivad enda loodud teoste originaalsust kontrollida juhul, kui tekib kahtlus, et valminud loometöö sarnaneb liialt mõnele varasemalt loodud meloodiale.

Autor on motiveeritud looma rakendust, mis võrdleks sarnasusi erinevate meloodiate ning teoste vahel seoses isikliku huviga nii muusika kui ka programmeerimise vastu. Järgnevas töös seletab autor enda loodud algoritmi ning selle realiseerimist, kasutades selle jaoks erinevaid vabavaralisi Java teeke ja raamistikke. Programmi eesmärk on koguda infot mõlema meloodia kohta ning seejärel võrrelda saadud tulemusi ja graafilisel kasutajaliidesel kuvada välja analüüsimise tulemused. Lisaks testib autor enda loodud rakendust näiteandmetega, mis on valitud vastavalt artiklitele, kus on välja toodud plagieeritud muusikateoseid.

Töö eesmärk on arendada eelnevalt kirjeldatud rakendust ning testida erinevaid MIDI faile ja anda hinnang programmi funktsionaalsusele ning selle poolt loodud väljundi kohta. Samuti keskendub autor testimise käigus vigade leidmisele ning nende ilmnemisel lahenduste välja pakkumisele.

1. Teoreetiline taust

Selles peatükis seletab autor rakenduse töötamise põhimõtteid skeemide ja jooniste abil ning toob välja failide analüüsimisel arvesse võetud tegureid. Kuna tarkvara arendamine ei koosne ainult programmeerimisest, siis on autori hinnangul tähtis kirjeldada ka mõttetööd ja planeerimist, mis selle rakenduse arendamisse investeeriti.

1.1 Failiformaadi valik

MIDI ehk muusikalise instrumendi digitaalne liides (inglisekeelne akronüüm *Musical Instrument Digital Interface*) tutvustati esimest korda 1982 aasta augustis. Idee selle taga seisneb digitaalsele muusikainstrumendile info saatmises läbi konkreetsete parameetrite. Muusika loomisel kasutaja salvestab näiteks MIDI kontrolleril meloodia ning digitaalsesse salvestisse talletatakse andmed nootide kõrguste, pikkuste, tugevuste jpt näitajate kohta. MIDI on väga laialt kasutatud formaat muusika loomiseks ning tagasiesitamiseks (Swift, 1997).

Kuna meloodia kohta salvestatakse eelnevalt nimetatud näitajad väga täpselt, teeb see andmetega töötlemise märksa lihtsamaks, kui näiteks analoog helisalvestuse analüüsimine. Viimase puhul sõltuks heli analüüsimine suuresti salvestuse helikvaliteedist ning sellised faktorid nagu staatiline müra, diskreetimissagedus, bitisügavus võivad muuta heli analüüsi tulemusi.

1.2 Analüüsi tegurid

Selleks, et teha järeldusi kahe meloodia sarnasuste kohta, tuleks autori hinnangul arvesse võtta järgmisi tegureid:

- Meloodiate tempod - hinnates kahe teose sarnasust, peaks autori hinnangul kindlasti arvesse võtma selle tempo, kuna teose esitamise kiirusest võib sõltuda meloodia üldine kõlapilt
- Kõrgeimad ja madalaimad noodid - nende näitajate järgi saab näiteks eristada, millisesse muusikalisse intervalli teosed jäävad
- Madalduste ja kõrgenduste arvud - võttes arvesse nimetatud tegureid, saab hinnata paremini meloodia kulgemist. Näiteks kui ühes toeses järgnevad toonid vastavalt A C D C ning teises D C A A, siis on tegemist siiski erinevate meloodiatega, sõltumata asjaolust, et toonid on samad. Antud näites esimese puhul oleks kaks kõrgendust ja üks madaldus ning teise puhul kaks madaldust. Jättes see asjaolu arvestamata, peaks selle

näite puhul programm tagastama, et tegemist on väga sarnaste meloodiatega, ehkki kuulaja jaoks on need väga erinevad.

- Unikaalsete nootide arv - rakendus arvestab mõlema meloodia korral eristatavate nootide summat selleks, et leida kahe meloodia ühiste nootide arvu
- Nootide osakaal – tagastatakse protsentuaalselt samade helikõrguste intervallide osakaal kahe faili kohta võttes arvesse ka toonide esinemise sagedusi
- Ühiste nootide arv - kuvades kasutaja jaoks välja, kui palju kõikidest unikaalsetest nootides on täpselt samad kahe teose vahel, aitab autori hinnangul hästi võrrelda kahe meloodia sarnasust. Näiteks kui üks viis sisaldab endast 12 erinevat tooni ning teine viis 15 erinevat tooni ja 10 nendest on täpselt samad, siis sõltumata nende paigutusest või pikkusest kõlavad need kuulaja jaoks sarnaselt.

1.3 Faili analüüsimise algoritm

Lisa 1 skeem aitab seletada, kuidas käsitletakse programmis andmeid ning kuidas on moodustatud tulemused. Meetodi sisendina saadakse kogu faili sisu objektina ning selleks, et erineva ülesehitusega meloodiaid edukalt analüüsida, peab ükshaaval vaatlema kõiki failis olevad partituurid, partituurides olevad fraasid ning fraasides olevad noodid. Parema tulemuse saamiseks hinnatakse ka, kas leitud noot on madalam või kõrgem eelnevast noodist. Nootide toonid ning pikkused ise salvestatakse globaalsesse massiivi, et nendega vajadusel ka teistes meetodites hiljem töötada. Kui andmete töötlemine on lõpetatud, talletatakse saadud tulemused objekti Meloodia, et vajadusel neid hiljem välja kuvada või kasutada teistes klassides ja meetodites.

2. Arendus

Järgnev peatükk kirjeldab rakenduse arendamise protsessi ning programmi enda funktsionaalsust ja tööpõhimõtteid. Lisaks annab peatükk ülevaate ka kasutatud raamistikest.

Rakenduse programmeerimiseks kasutati Eclipse tarkvara, kuna autor on puutunud sellega kokku Java koodi arendamisel seniste ülikooli õpingute käigus kõige enam. Programmi koodi parema haldamise jaoks on funktsionaalsus on jaotatud nelja klassi: Analyys.java, Meloodia.java, Vaade.java ning Graaf.java. Klass Analyys.java sisaldab meetodeid failide avamiseks ning failide sisuga toimetamiseks. Sellesse klassi on kokku kogutud kogu rakenduse loogika, mis puudutab programmi otstarvet. Meloodia.java klassis on defineeritud objekt Meloodia, mida kasutatakse andmete talletamiseks ühe faili kohta. Vaade.java eesmärk on kasutaja jaoks luua rakenduse interaktiivne graafiline liides soosides kasutajamugavust. Klassis Graaf.java on defineeritud meetodid meloodia kulgemise lineaarse graafiku koostamiseks.

2.1 Loodud funktsioonid ja nende selgitused

Selles alapeatükis selgitab autor klassi kaupa erinevate meetodite tööpõhimõtteid ning põhjendab oma valikut algoritmi realiseerimise lähenemisele.

2.1.1 Analyys.java

Selles klassis kirjeldatakse globaalseid muutujaid meloodiate kohta, mida kasutatakse ka teistes klassides andmete välja kuvamiseks.

2.1.1.1 Andmete analüüsimine

Failist andmete sisselugemiseks on loodud meetod nimega Analyys1, mille sisendiks on JMusic teegi objekt *Score*, massiiv *noodid* ning objekt *Meloodia*. Rakenduse otstarbel on vaja kätte saada failist nootide toonid ning nende pikkused ja kuna MIDI failis võib olla jaotatud meloodia eraldi osadeks ning fraasideks, siis tuleb need kõik ükshaaval läbi käia. Objekti *skoor* andmetega töötamiseks on loodud eraldi ajutine *Enumeration* (lühidalt enum) objekt, mis erineb mõnevõrra tavalisest massiivist.

Kui massiivi võib pidada ühte tüüpi andmete hulgaks, siis enum on kasutaja poolt defineeritud andmetüübi väärtuste hulk (Oracle Corporation, kuupäev puudub). Selleks, et salvestada failist kõik noodid ühekaupa massiivi, on loodud *while loop*, mis töötab seni kuni puudub rohkem fraasi, mida töödelda. Samuti tuvastatakse, kas noodi puhul on tegu kõrgendusega või madaldusega. Pärast nootide salvestamist, talletataks failist saadud andmed objekti *meloodia*. Lisa 3 on välja toodud ka kirjeldatud funktsioonide programmikood.

2.1.1.2 Failide laadimine ning kontrollimine

Klassis Analyys.java on ka loodud funktsioonid failide avamiseks ning nende olemasolu kontrollimiseks. Rakenduse kasutajamugavuse eesmärgil on loodud faili avamiseks dialoogiaken, kus kasutaja saab sirvida kõvaketta kataloogides, et leida sobiv sisend.

2.1.1.3 Helistike ja tempode samastamine

Selleks, et samastada kahe meloodia helistikud, on arvutatud välja mõlema faili kõige madalamate nootide toonide vahe ning vastavalt sellele transponeeritakse nootide kollektsioone. Vaatamata asjaolule, et sellist lähenemist võib pidada primitiivseks, parandab ta oluliselt graafikute ning astmikdiagrammide loetavust. Kahe meloodia tempode samastamiseks kasutatakse Jmusic teegi meetodit setTempo().

2.1.1.4 Unikaalsete ning ühiste nootide loendamine

Arendatud rakenduse funktsionaalsuse alla kuulub ka failist massiivi talletatud nootide kõrguste eristamine ning individuaalsete väärtuste loendamine. Selle jaoks on loodud kolm *HashSet* objekti, millest esimesse kahte talletatakse individuaalselt esimese ning teise faili nootide toonide väärtused ning kolmandasse esimese kahe *HashSet*i sisu. *HashSet*i andmeid talletades moodustatakse nendest paisktabel, mille abil saab lihtsasti loendada nende individuaalseid väärtusi (Oracle Corporation, kuupäev puudub).

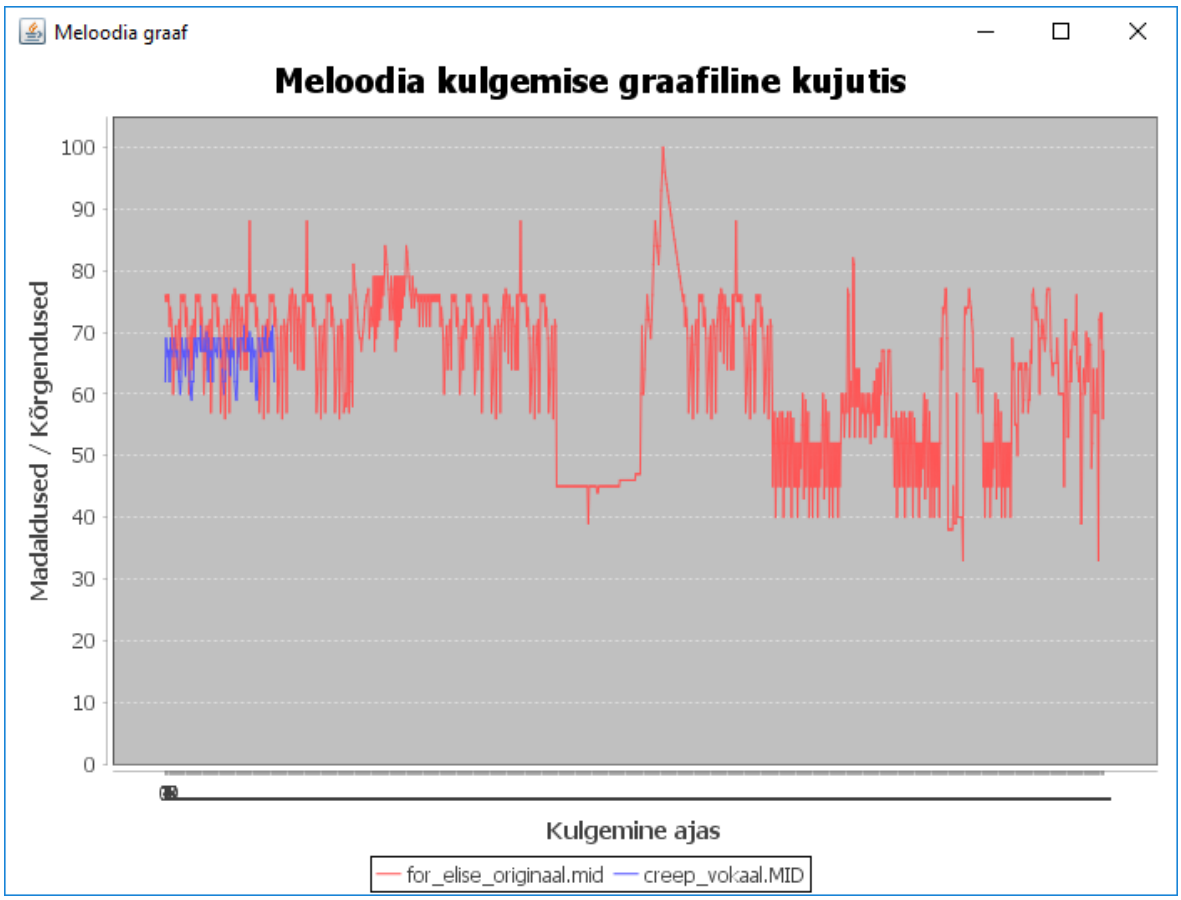
Meetodis arvutaProtsent() tagastatakse protsentuaalselt nootide osakaal võttes arvesse ka nende esinemise sagedusi. Selle jaoks on loodud kaks listi mõlema failide nootide talletamiseks ning ühe kirje kaupa kontrollitakse, kas ühe listi element on olemas ka teises listis. Juhul kui esineb täpselt sama helikõrgus mõlemas listis, siis eemaldatakse see kirje teisest listist ning suurendatakse muutujat loendur, mille otstarve on arvuliselt järge pidada samade toonide esinemisest. Kui listi kõik kirjed on kontrollitud, arvutatakse samade toonide esinemiste arvu osakaal esimese listi elementide arvust sõltuvalt.

2.1.2 Meloodia.java

Selles klassis on loodud objekt meloodia, mida kasutatakse töödeldud andmete talletamiseks ning arendaja jaoks mugavamaks refereerimiseks teistes klassides. Objekti salvestatakse erinevad väärtused nagu kõrgeim ja madalaim noot, lühim ja pikim noot, madalduste ja kõrgenduse arv jms.

2.1.3 Graaf.java

Klassis Graaf.java on loodud meetodid eelnevalt töödeldud andmete kandmiseks lineaarse graafiku peale, mille loomiseks on kasutatud raamistiku JFreeChart abi. Arendamise käigus testiti erinevaid mooduseid, kuidas kujutada meloodia kulgu graafi peal. Näiteks prooviti kujutada Y telje peal nootide tooni ning X telje peal tooni indeksi väärtuse ja noodi pikkuse korrutist. Selle eesmärk oli võtta arvesse meloodia graafilisel kujutamisel ka rütmi, kuid kahjuks selline lahendus osutus kehvasti loetavaks ning visuaalse graafiku parendamine ajakulukaks. Autori hinnangul tagas meloodiate võrdlemise suhtes parima tulemuse Y teljel noodi kõrguse kujutamine ning mööda X telge nootide massiivi vastava indeksi kujutamine.



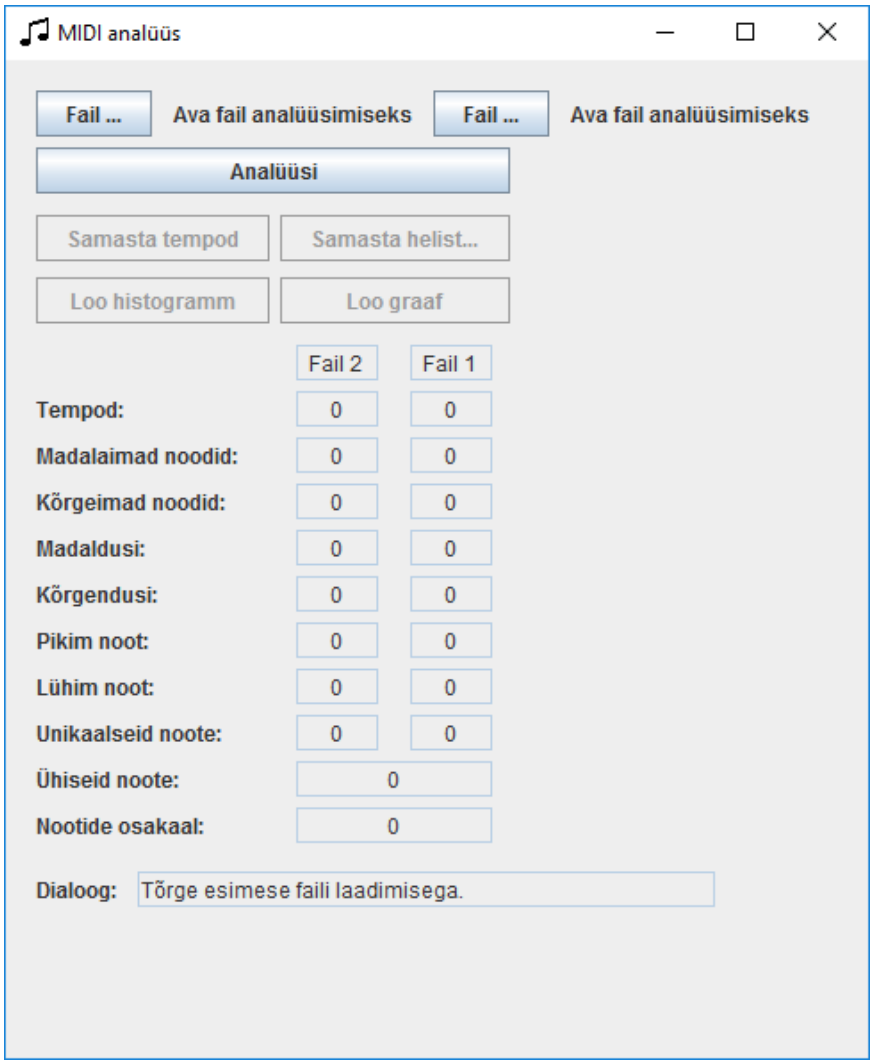
Pilt 1 Graafi näidis

2.1.4 Vaade.java

Klass Vaade.java on loodud eesmärgiga moodustada kasutaja jaoks graafiline interaktiivne kasutajaliides, mille kasutamisega saaks hakkama tavapärane arvutikasutaja. Kasutajaliidese loomiseks on kasutatud raamistikku WindowBuilder, mis vähendas rakenduse arendamise ajakulukust oluliselt.

2.1.4.1 Graafilised objektid

Kasutajamugavuse parendamiseks on loodud nupud, mis käivitavad funktsioonid andmetega töötlemiseks või nende kuvamiseks.

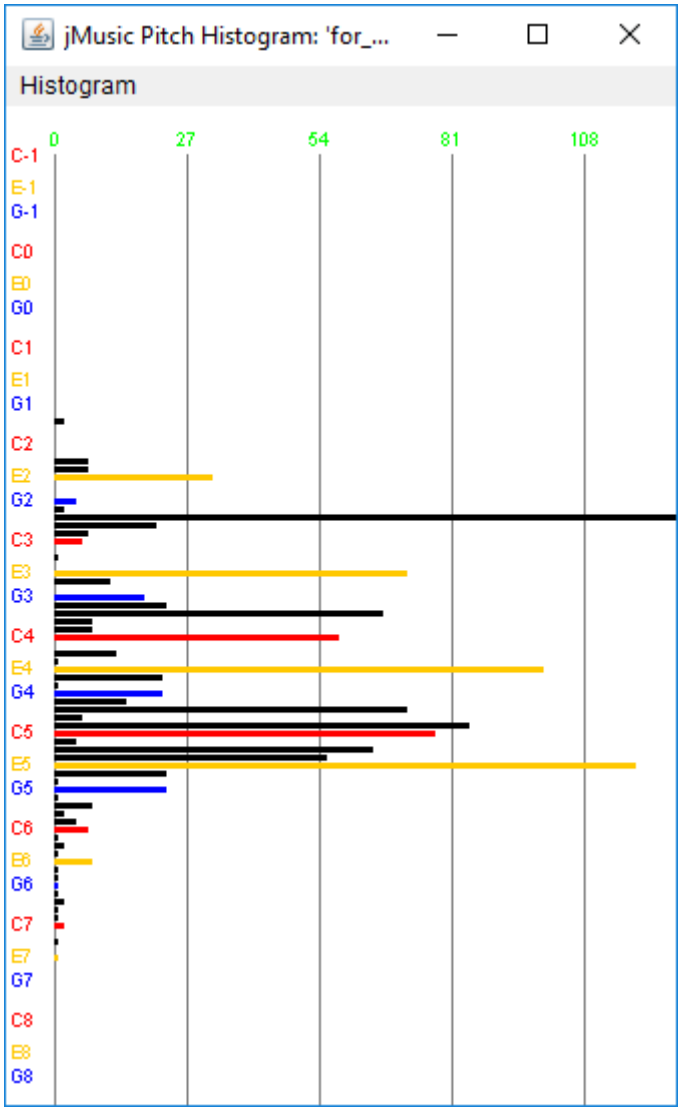


Pilt 2 Rakenduse graafiline kasutajaliides

Kasutajaliidesel on loodud järgmised nupud:

- Fail … - käivitab faililaadimise funktsiooni ning tekitab failidialoogi akna kasutajale

- Analüüsi - käivitab Analyys klassi peamise funktsiooni Analyys() juhul, kui failid on programmi laetud. Nupp muutub passiivseks pärast failide edukat analüüsimist ning aktiveerib nupub “Samasta tempo”, “Samasta helistikud”, “Loo histogramm” ning “Loo graaf”.
- Samasta tempod - käivitab Analyys klassis vastava funktsiooni.
- Samasta helistikud - käivitab Analyys klassis vastava funktsiooni.
- Loo histogramm - käivitab Analyys klassis vastava funktsiooni ning loob juurde kaks lisaakent, kus kujutatakse graafiliselt nootide koguselist jaotust vastavalt toonile.
- Loo graaf - käivitab Graaf klassis vastava funktsiooni ning tekitab ühe lisaakna, kus kujutatakse meloodia kulgemist graafiliselt.



Pilt 3 Astmikdiagrammi näidis

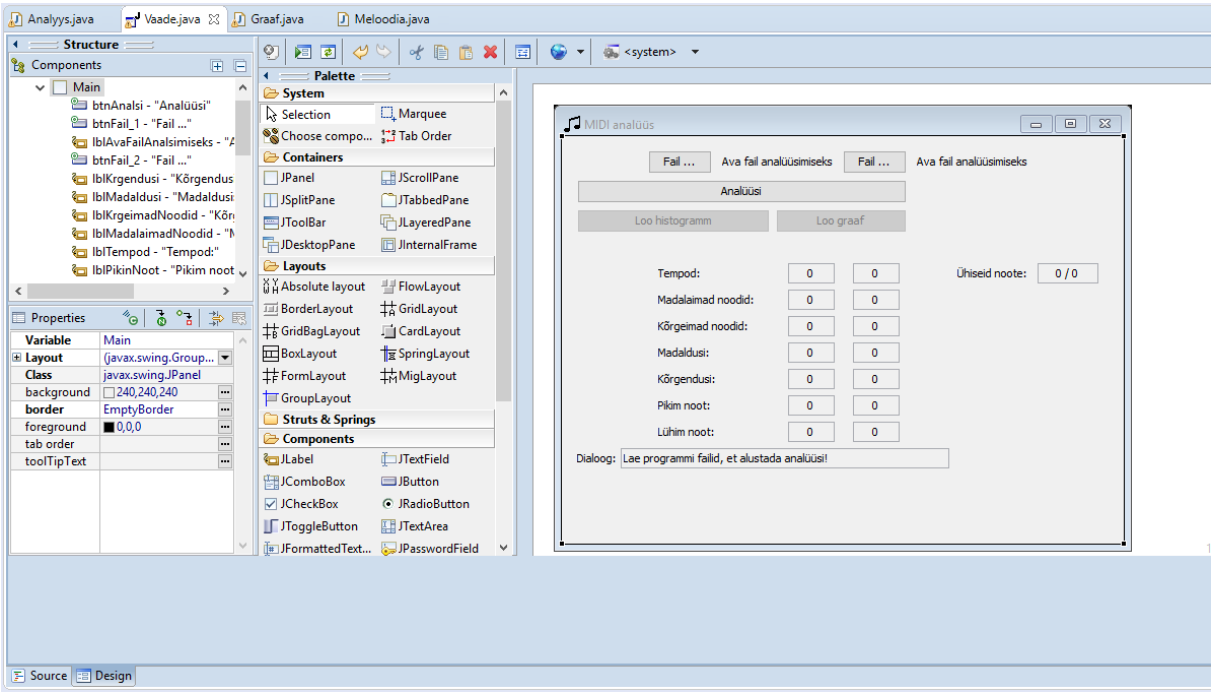
Graafiliselt kujutatakse kasutaja jaoks välja ka erinevate arvutuste tulemused. Algupäraselt on kõik väärtused nullitud ning pärast analüüsi funktsiooni käivitamist asendatakse lahtrid reaalsete tulemustega. Lahtris "Ühised noodid" kujutatakse kahes meloodias korduvate nootide osakaalu samade meloodiate unikaalsete toonide summa suhtes. Lisaks on ka loodud dialoogiaken, kuhu kuvatakse erinevaid veateateid ning teavitatakse kasutajat käivitatud protsessidest.

2.2 Kasutatud raamistikud ja teegid

Ehkki raamistikud ei ole tarkvara arendamise jaoks hädavajalikud, teevad nad töötamise protsessi programmeerija jaoks mugavamaks ja kiiremaks. Kasutades raamistikke saab arendaja keskenduda peamiselt rakenduse põhiliste eesmärkideni jõudmisele, kulutades minimaalselt aega triviaalsete probleemidele. Peamiselt kasutati programmeerimisel raamistikke graafiliste objektide loomiseks ning kasutaja jaoks mugavuse parendamiseks (Clifton, 2003).

2.2.1 WindowBuilder

Graafilise interaktiivse kasutajaliidese loomiseks kasutati Eclipse lisamoodulit WindowBuilder, mis lubab *drag and drop* meetodil luua nii lihtsaid kui ka keerulisi graafilisi liideseid ilma suurt hulka koodi kirjutamata. See moodul loob abstraktse süntaksipuu selleks, et navigeerida lähtekoodi ning visuaalse esituse loomiseks kasutatakse Eclipse enda raamistikku GEF (*Graphical Editing Framework*) (Eclipse Technology, 2013). Kasutaja saab valida, millise objekti ta soovib paneelile lisada (näiteks raadionupp, tekstiväli vms) ning lohistamise meetodil asetatakse objekt soovitud lõuendi peale. Objektide lisamisel kirjeldatud meetodil genereeritakse Java kood, mis ei vaja käivitamiseks lisamooduleid ega raamistikke. Selle raamistiku kasutuselevõtt vähendas oluliselt rakenduse arendamise peale kulunud aega.



Pilt 4 Windowbuilder kasutajaliides

2.2.2 JFreeChart

JFreeChart on avaliku lähtekoodiga ja tasuta (LGPL litsentsi alusel) Java raamistik, mille abil saab luua kerge vaevaga professionaalse väljanägemisega graafikuid ja tabeleid andmete visualiseerimiseks. JFreeCharti on olnud arenduses üle kümne aasta ning see on üks populaarsemaid vabavaralisi Java raamistikke. JFreeCharti on alla laaditud üle 2 miljoni korra 2014 aasta seisuga (jfree.org, 2014).

2.2.3 JMusic

Jmusic on Java programmeerimiskeele jaoks tehtud teek, mis on eelkõige arendatud eesmärgiga olla abiks heliloojatele ning muusikutele loometöö tegemisel. Samuti leiab JMusic ka kasutust programmeerijate seas, kes soovivad enda muusikaga seotud rakendusi luua. JMusic arendamist alustati 1990 aastate lõpul Queenslandi Tehnikaülikoolis (Queensland University of Technology), esimene avalik väljalase nägi ilmavalgust aastal 2001 ning see on endiselt aktiivselt arenduses ja uusi versioone avaldatakse iga mõne aasta tagant.

JMusic on tasuta (GNU litsentsi alusel) ning selle kasutamine on suunatud vähese programmeerimiskogemusega inimestele. Kuna JMusic kasutab täielikult Java keelt ilma lisamoodulite või -raamistiketa, siis ei valmista selle kasutamine raskusi algajale programmeerijale (Brown & Sorensen, kuupäev puudub). Antud rakenduse arendamisel omas Jmusic teek suurt tähtsust, kuna valdav osa programmi loogikast kasutab just selle teegi meetodeid ning objekte.

2.3 Arendusel esinenud probleemid

Peamised tõkked, mis esinesid rakenduse arendamisel olid seotud eelkõige algoritmi realiseerimisega ning graafi loomisega. Arendaja seisukohast lähtudes ei olnud keeruline kirjutada koodi, mille tulemusel luuakse graafiline kujutis olemasolevatest andmetest, milleks olid nootide kõrgused ning pikkused. Kahjuks aga osutus arendamisprotsessi käigus väga ajakulukaks nende andmete põhjal luua visuaalne meloodia kulg, mille põhjal saaks selgelt tuvastada sarnasusi ning erisusi.

3. Rakenduse testimine

Peamiselt testiti rakenduse funktsionaalsust eesmärgiga välja selgitada, kas vajalikke tegevusi on võimalik edukalt läbi viia. Selle jaoks testitakse rakendust teostega, mida peetakse plagiaatides ning vastukaaluks ka lauludega, mis kõlavad väga erinevalt. Selle alusel antakse hinnang tulemuste pädevusele ning sellele toetudes tehakse järeldused, mida võiks rakenduses parendada.

3.1 Testandmed

Rakenduse testimise aluseks võeti 2016 aastal avaldatud veebiartikkel portaalist sheknows.com, milles on loetletud 31 erinevat pop ning rokkmuusika teost, mida on peetud varasemate laulude plagiaatideks (Sprankles, 2016). Autori hinnangul on tähtis siinkohal välja selgitada, kas programmi väljundist sõltuvalt saab hinnata, kui sarnased teosed ning kas meloodiate ligilähedane kõla väljendub ka arvutustes või graafiliselt. Plagiaatidena võrdlemiseks on valitud sellest nimekirjast teosed „Creep“ esitajalt Radiohead vastukaalus laulule „The Air That I Breathe“ esitajalt Albert Hammond ning tuntud poppmuusika pala „Viva La Vida“ esitajalt Coldplay koos sooloartisti Joe Satriani teosega „If I Could Fly“. Testimiseks valitud MIDI failid on alla laetud portaalist www.midiworld.com ning vajadusel kohandatud kasutades programmi Reaper (versioon 5.40) MIDI redaktorit.

Testandmeid oli tarvis kohandada programmi jaoks, kuna rakenduse eesmärk oli võrrelda meloodiaid, mis oleks loodud ühest MIDI rajast. Kuna enamikel kasutatud testandmetest oli ühes failis koos kõik instrumentaalosad ning vokaalipartiid, siis eri radade eraldamiseks oli tarvis kasutada MIDI redaktorit.

Lisaks plagiaadikahtlusega teoste võrdlemisele testiti programmi ka juhuslikult valitud lauludega, et paremini välja selgitada, kas teoste sarnasus väljendub rakenduse väljundandmetes. Juhuslikult valitud muusikapalad on samuti võetud samast veebiartiklist, kuid testitud omavahel juhuslikult.

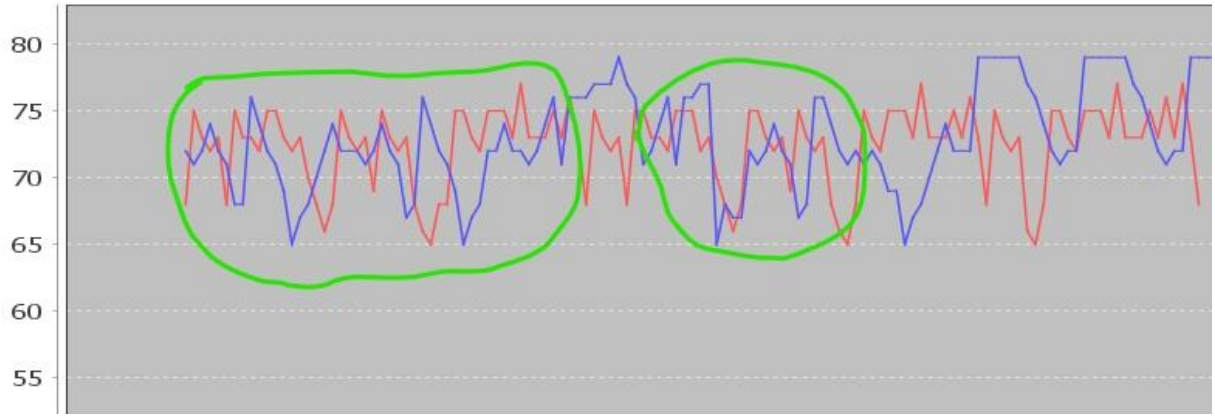
3.2 Testimise tulemused

Plagiaadikahtlusega teostel võrreldi eraldi vokaalipartiisid ning vajadusel ka eraldi instrumentide MIDI radasid. Arendatud programmi funktsionaalsuse hindamiseks on aluseks võetud Lisa 2 tabel, milles on välja toodud võrdluse tulemusel tekkinud andmed ning programmi loodud erinevad graafid.

Nende andmete abil selgitatakse välja, kas rakenduse arvutuste läbi väljendub teoste sarnasus või erisus. Selle puudumise korral arutletakse, millest tingituna võisid ebausutavad tulemused tekkida.

3.2.1 Plagieeritud laulude testimine

Esimesena võrreldi laule „Creep“ esitajalt Radiohead ja „The Air That I Breathe“ esitajalt Albert Hammond. Lisa 2 tablis on välja toodud analüüsi tulemused eraldi elektrikitarri, basskitarri ning vokaalipartiide analüüsi tulemused. Andmetest võib järeldada, et eri partiide meloodiate intervallid on väga ligilähedased, erinedes teineteisest vaid mõned pooltoonid. Ehkki ühiste unikaalsete nootide osakaal on igal kolmel võrdlusel sarnaselt suur, ei väljendu sarnasus nootide protsentuaalses osakaalus. Märkimisväärne sarnasus on vaid basskitarri partiidel, kus samade helikõrguste esinemise protsent on 53.



Graafik 1 Radiohead (punane) ja Albert Hammond (sinine) vokaalipartiid

Graafikul 1 on kujutatud nimetatud teoste vokaalipartiide meloodia kulgemist ajas graafiliselt pärast helistike samastamist. Joonistatud graafikul on näha sarnasusi meloodia kulgemise esimeses pooles ning keskel, kus sarnasused on esile toodud rohelise ringiga.

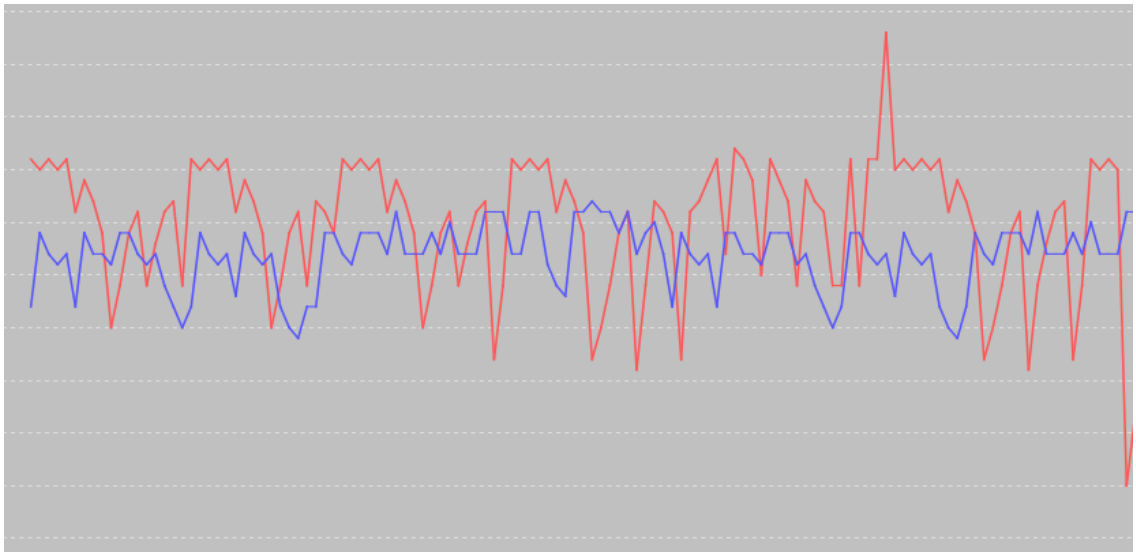
Teine võrdlus teostati lauludel „Viva La Vida“ ning „If I Could Fly“ esitajatelt Coldplay ning Joe Satriani. Võrreldavad meloodiad valiti vastavalt testandmete allikale, milleks olid Coldplay teose refrään ning Satriani laulu juures kitarrimeloodia üks partiidest (Sprankles 2016). Ehkki tulemuste tabelis test nr 4 arvutatud sarnasus on väga väikene, tuleb kahe meloodia sarnasus selgemini välja jällegi kasutades graafiku joonistamise funktsiooni.

Graafik 2 Coldplay (punane) ja Joe Satriani (sinine) sarnased meloodiad

Graafikul on näha, et selge sarnasus on meloodia alguses ning keskel. Sellest sõltumata langeb kokku vaid alla 5% meloodiast täpselt samadel intervallidel. Autori hinnangul tuleneb ebatõenäoliselt väikene arvutatud sarnasus vähestest nootide arvust failides ning laulude helistike erinevustest.

3.2.2 Mitte plagieeritud laulude testimine

Selleks, et veenduda rakenduse funktsionaalsuse töökindluses, teostati test kahe väga erineva laulu puhul. Sel korral valiti testandmeteks eelnevalt kasutusel olnud ansambli Radiohead teose „Creep“ ning tuntud klassiku Ludwig van Beethoveni „Für Elise“. Kuna testitavate teoste pikkused olid väga erinevad, siis täpsemate tulemuste saamiseks lühendati faili Beethoveni laulu faili. Lisa 2 tabelis on toodud välja testi number 5 arvutuste tulemused. Arvutustest võib järeldada, et meloodiate intervallid erinevad teineteisest pea 20 pooltooni ning vähem kui veerand nootidest langeb kokku. Lisaks on ka näha visuaalselt, et valitud kaks meloodiat kulgevad erinevalt.



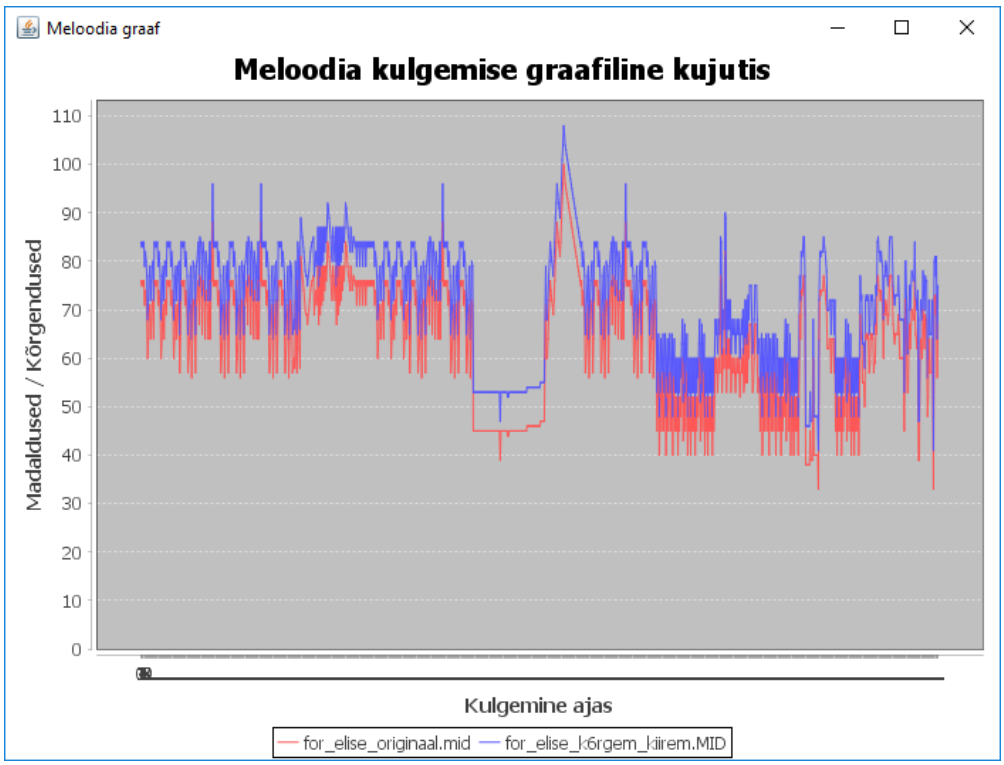
Graafik 3 Beethoveni "Für Elise" (punane) ning Radioheadi "Creep" meloodia kulgemine

3.2.3 Sama failiga testimine

Selleks, et testida rakenduse matemaatilisi arvutuste täpsust ning korrektsust, on autor valinud ühe teose, mille MIDI faili on kopeeritud ning duplikaadi helikõrgust ja tempot muudetud. Autor peab sellist testi tähtsaks, kuna kirjeldatud programm peaks olema suuteline tuvastama, kas selliste muutuste puhul on siiski tegu sama lauluga.

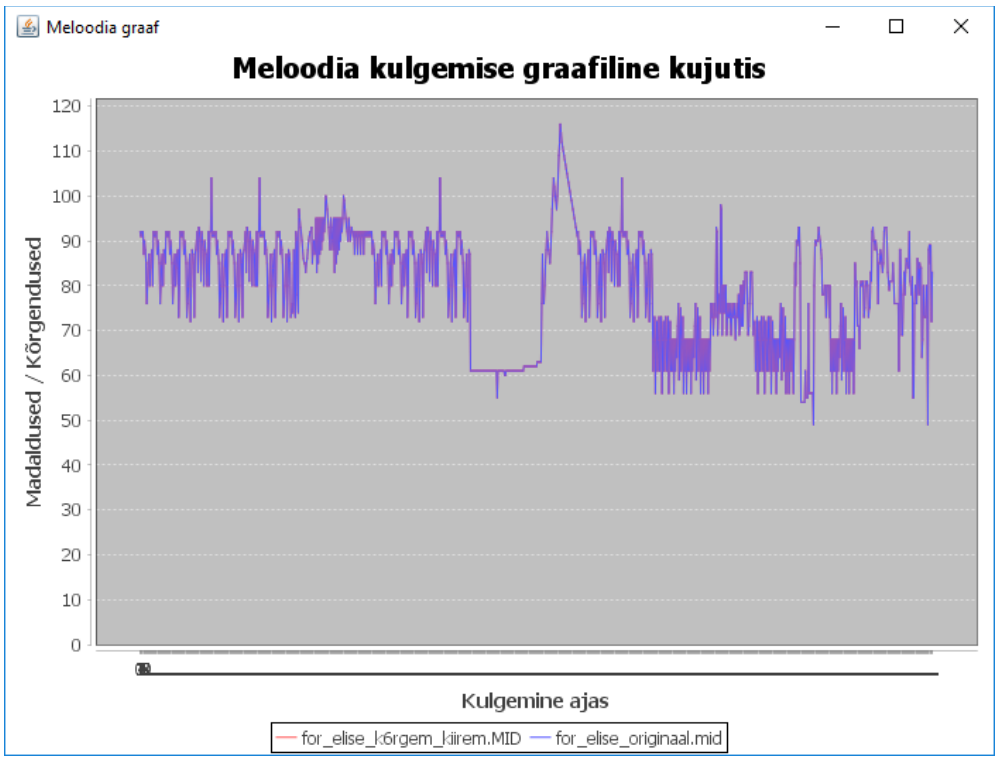
Katsetamiseks valiti klassiku Ludwig van Beethoveni teos „Für Elise“. Duplikaatfailil on helistikku tõstetud 8 pooltooni ning tempot muudetud kiiremaks. Lisa 2 tabelis on välja toodud testide tulemused. Esimesel juhul (Test nr 6) on jäetud tempo ning helistik muutmata ning Test nr 7 puhul on samastatud helistik ning tempo.

Tabeli andmete järgi võib esimese testi puhul järeldada, et tegu oleks justkui erinevate, kuid väga sarnaste teostega, kuid kui uurida lähemalt programmi loodud graafi, tuleb failide samasus paremini esile.



Graafik 4 Für Elise graafik Test 6

Graafi järgi on selgesti aru saada, et tegemist on samade meloodiatega, mille helistikud erinevates teineteisest. Kui kasutada rakenduse helistike samastamise funktsiooni, tuleb samasus veel paremini esile.



Graafik 5 Für Elise graafid pärast helistike samastamist

Samuti tagastab programm eelduste kohaselt ka korrektsed arvutuste andmed. Pärast helistike samastamist on ühiste unikaalsete nootide arv 56 (mõlema loo puhul on 56 unikaalset tooni , mis on testitud kahe faili puhul kõik täpselt samad) ning kogu nootide osakaal 100%.

3.2.4 Testide järeldused

Testitud andmete põhjal võib väita, et hetkel on rakenduse tugevamaks küljeks graafide joonistamine, kuna nende abil tuleb kõige selgemini välja meloodiate sarnasused ning erisused. Rakenduse arvutused aga kahjuks ei ole nii veenvad, et nende põhjal saaks otsustada, kas tegemist on plagieeritud teostega või mitte. Autori hinnangul tuleneb nimetatud asjaolu sellest, et ehkki inimkõrvale võivad tunduda meloodiad sarnase kõlaga, ei pruugi see samasus väljenduda matemaatilistes arvutustes. Selleks, et adekvaatsemaid arvutustulemusi saada, peaks leidma parema mooduse sarnasuste arvutamiseks, ehkki sama faili testimise põhjal võib järeldada, et rakendus tehtud arvutused on korrektsed.

3.3 Rakenduse võimalikud täiustused

Selles alapeatükis tuuakse välja autori hinnangul detailid ja võimalikud juurdearendused, mis parendaksid rakenduse funktsionaalsust ning kasutusmugavust programmi tarbija jaoks. Samuti tuuakse välja programmi kasutamist häirivad tegurid ja nende potentsiaalsed lahendused.

3.3.1 Funktsionaalsus

Autori hinnangul võiks olla paremini teostatud helistike samastamine ning meloodia kulgemise graafikute visuaalne esitlemine. Hetkel leitakse helistike erinevus vaid kõige madalama noodi abil ning selline primitiivne lahendus töötab paljude testandmetega hästi, kuid graafikul parema joondiagrammi loomise jaoks oleks tarvis paremat lahendust selleks, et kasutaja saaks tulemusi paremini tõlgendada.

Mis puudutab meloodia kulgemise visuaalset kujutamist, siis graafiku vaade võiks omada paremat viisi pilti suurendada. Vaadatavust parendaks suurendamise võimaldamine nii X kui ka Y telje suhtes näiteks klikkamise või suurendatava ala märgistamise teel hiire abil.

Astmikdiagrammi loomisel esineb hetkel tõrkeid mõningate failide puhul väljaselgitamata põhjustel. Autori hinnangul on probleem seotud analüüsitavate failide radade arvuga, kuna sisendfailidel, mis sisaldavad ainult ühte meloodia rida, esineb tõrkeid märksa vähem.

Lisaks tuleks kasuks võimalus salvestada analüüsi tulemused faili. Selle jaoks oleks hea lahendus tekitada üks tekstifail, mis sisaldaks erinevaid arvutamise tulemusi ning eraldi salvestada pildiformaadis astmikdiagramm ning graaf.

3.3.2 Graafiline liides

Arendatud rakenduse graafiline kasutajaliides on lihtne ja arusaadav, kuid autori hinnangul parandaks rakenduse üldist ilmet suurema kirjasuuruse kasutamine ning andmete parem liigendamine. Kasuks tuleks ka kogu info kuvamine ühele paneelile selle asemel, et astmikdiagramm ja graaf kujutada eraldi akendes.

Kokkuvõte

Käesoleva bakalaureusetöö „MIDI faile võrdleva tarkvara arendamine ja testimine“ eesmärkideks olid arendada rakendus, mis arvutaks kahe MIDI faili sarnasust, et järeldada kahe meloodia kohta, kas tegemist on plagiaadiga ning testida seda rakendust näiteandmetega. Lisaks on autor töös välja toonud võimalikud kasulikud juurdearendused ning võimalused praeguse rakenduse parendamiseks.

Eesmärkide saavutamiseks arendas autor Java programmeerimiskeeles rakenduse ajavahemikus jaanuar 2017 – aprill 2017 ning programmi testiti jooksvalt selles ajavahemikus. Rakenduse arendamisel kasutati erinevaid Java raamistikke, mille vajalikkus ning kasu on põhjendatud autori poolt. Töös on detailselt kirjeldatud arendatud tarkvara tööpõhimõtteid ning peamisi andmete analüüsimise funktsioone. Peale selle on autor andnud ka ülevaate rakenduse teoreetilisest poolest, põhjendades failiformaadi valikud ning analüüsimiseks valitud tegureid.

Programmi testimise näiteandmete aluseks võeti veebiartikkel muusikateostest, mida peetakse plagiaatideks. Samuti pandi proovile rakenduse funktsionaalsus kasutades sisendina sama muusikateose duplikaati muudetud tempo ning helikõrgusega, et teha selgeks, kas programmi abil saab tuvastada täpselt sama muusikateose teistkordset esitamist.

Testide tulemustest võib järeldada, et rakendus ei ole suuteline arvutuste teel väljendama kahe meloodia sarnasust täpselt. Küll aga graafide joonistamise abil on laulude samasus märksa paremini arusaadav. Ehkki arendatud programmi arvutuste tulemused ei ole veenvad, on autori hinnangul loodud rakendus siiski heaks aluseks, et luua selle juurdearendusi ning uusi funktsioone, parendamaks programmi funktsionaalsust. Kõik testandmed ning kogu programmi lähtekood on kaasa pandud selle töö lisana CD-kettale.

Summary

The main purpose of this bachelor's thesis tilted „The Development And Testing Of A MIDI File Comparing Software“ was to develop a program, which will be able to compare the musical similarities of two melodies in order to acknowledge, whether one of those songs is plagiarized or not. In addition to this, the author proposes possible additional functionalities and ideas for bettering the current program.

In order to achieve these purposes, the author developed a program in the Java programming language from January 2017 to April 2017 and tested it in this time period as well. During the development, different Java frameworks were used and the author has explained the necessity and benefit of using them. The author has also explained the ideas behind the functionality of MIDI data comparing used in the program.

Test data for this program was chosen from a web article, which contains a list of plagiarized songs. In addition to this, the program was also tested with one song and its duplicate file, where the pitch and tempo was changed in order to find out if it is possible to identify the same song in an altered version.

The results after testing conclude that the developed program is not able to calculate the similarities of songs precisely, although the functionality of drawing graphs based on the data proves to be much more convincing. Regardless of this, the author considers the program to be a good basis for additional developing in order to improve its functionality. All test data and the whole source code of the thesis is included with the paper on a CD.

Kasutatud kirjandus

Brown, A & Sorensen, A (kuupäev puudub). An introduction to jMusic. Loetud aadressil: http://explodingart.com/jmusic/jmtutorial/t2.html

Clifton, M (2003). What Is A Framework? Loetud aadressil: https://www.codeproject.com/Articles/5381/What-Is-A-Framework

Eclipse Technology (2013). WindowBuilder User Guide. Loetud aadressil: http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.wb.doc.user%2Fhtml%2Findex.html

Forde, E (2016). GLOBAL MUSIC MARKET UP 3.2% BUT IFPI TRAINS GUNS ON 'VALUE GAP'. Loetud aadressil: http://musically.com/2016/04/12/global-music-market-up-3-2-but-ifpi-trains-guns-on-value-gap/

JFreeChart (2014). The Project. Loetud aadressil: http://www.jfree.org/jfreechart/

JFreeChart (kuupäev puudub). JFreeChart Class Library (versioon 1.0.19-fx). Loetud aadressil: http://www.jfree.org/jfreechart/api/javadoc/

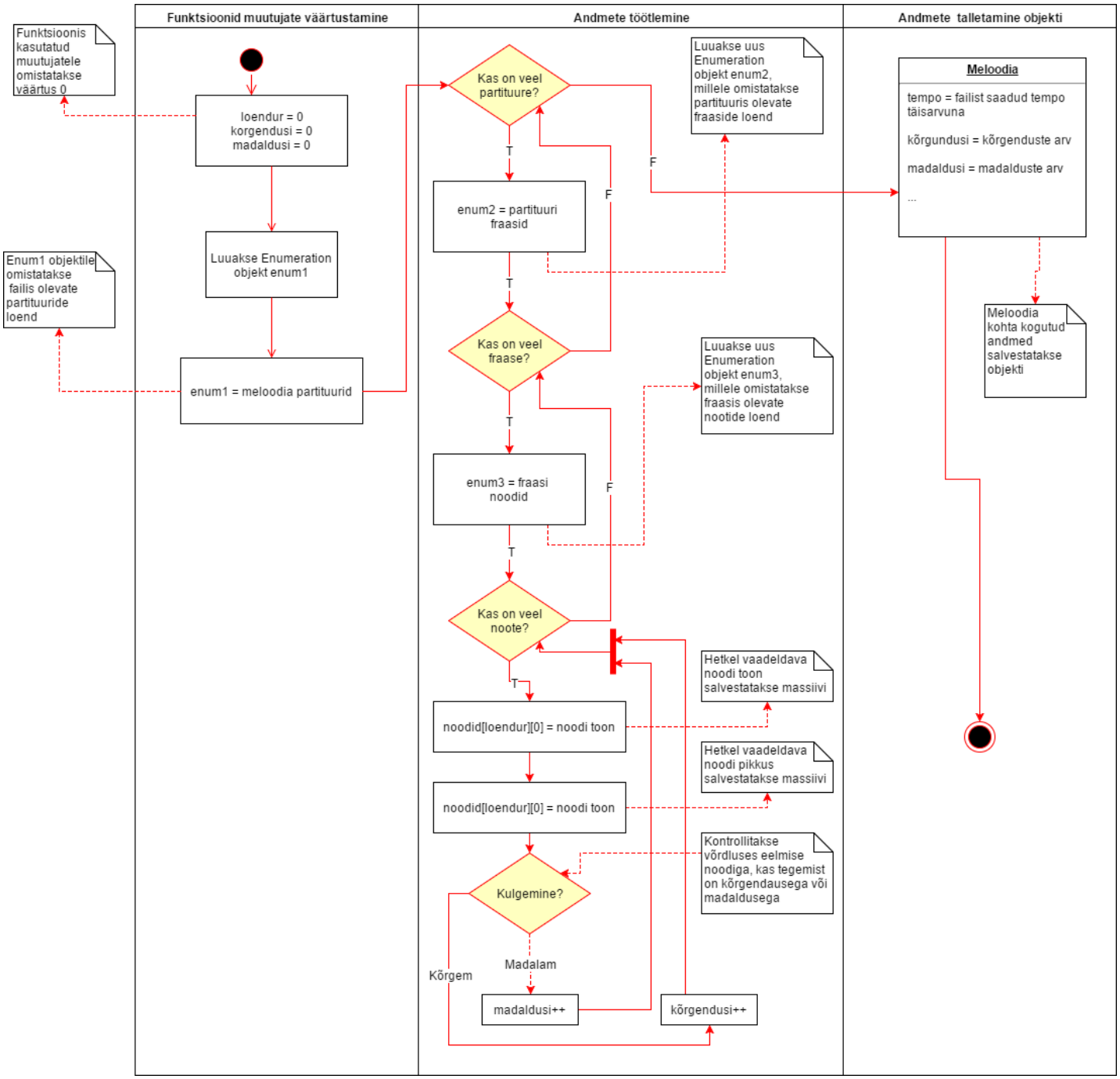
Oracle Corporation (kuupäev puudub). Java™ Platform, Standard Edition 7 API Specification. Loetud aadressil: https://docs.oracle.com/javase/7/docs/api/

Sprankles, J (2016). 31 songs you didn't know were (allegedly) plagiarized. Loetud aadressil: http://www.sheknows.com/entertainment/articles/1049435/songs-you-didnt-know-were-allegedly-plagiarized

Swift, A (1997). Introduction to MIDI. Loetud aadressil: http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/

Lisad

Lisa 1 Meloodia analüüsimise algoritmi skeem



Lisa 2 Testandmete tulemuste tabel

Testi nr	Fail 1	Fail 2	Madalaimad noodid	Kõrgeimad noo-did	Ühiseid noote	Nootide osakaal (%)
1	Hammond_bass.MID	Radiohead_bass.MID	30 / 28	41 / 43	5	53
2	Hammond_kitarr.MID	Radiohead_kitarr.MID	41 / 43	67 / 65	8	14,44
3	Hammond_vokaal.MID	Radiohead_vokaal.MID	53 / 59	67 / 71	5	27,06
4	Coldplay_ref.MID	Satriani_meloodia.MID	65 / 61	79 / 88	2	4,545
5	Radiohead_creep_meloodia.MID	Fur_Elise_lyhem.MID	40 / 59	88 / 74	8	23,30
6	Fur_Elise_originaal.MID	Fur_Elise_muudetud.MID	33 / 41	100 / 108	44	31,37
7	Fur_Elise_originaal.MID	Fur_Elise_muudetud.MID	41 / 41	108 / 108	56	100

Lisa 3 Rakenduse andmete analüüsimise lähtekood

```
import jm.JMC;
import jm.gui.histogram.Histogram;
import jm.music.data.*;
import jm.music.tools.*;
import jm.util.*;
import java.util.Vector;
import java.awt.event.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public final class Analyys extends Frame implements JMC {

      static int unikaalseid_1 = 0;
      static int unikaalseid_2 = 0;
      static int korgendusi = 0;
      static int madaldusi = 0;
      static int eelnev = 60;
      static Score s = new Score();
      static Score s2 = new Score();
      static String dialoog = "";

      //massiivid nootide ja nende koguste hoidmiseks
      static double noodid1[][] = new double[50000][2];
      static double noodid2[][] = new double[50000][2];

      //objektid meloodiate kohta info talletamiseks
      static Meloodia meloodia1 = new Meloodia();
      static Meloodia meloodia2 = new Meloodia();

    public void Analyys1(Score skoor, Meloodia meloodia, double[][] noo-
did){

      int loendur = 0;

      //nullitakse globaalsed muutujad
      korgendusi = 0;
      madaldusi = 0;

      Enumeration enum1 = skoor.getPartList().elements();
            while(enum1.hasMoreElements()){
                  Part nextPt = (Part)enum1.nextElement();
                  Enumeration enum2 = nextPt.getPhraseList().elements();
                  while(enum2.hasMoreElements()){
                        Phrase nextPhr = (Phrase)enum2.nextElement();
                        Enumeration enum3 =
nextPhr.getNoteList().elements();
                        while(enum3.hasMoreElements()){
                              Note nextNote = (Note)enum3.nextElement();

                              int toon = nextNote.getPitch();
                              noodid[loendur][0] = toon;
                              loendur++;

                              double rv = nextNote.getRhythmValue();

                              noodid[loendur][1] = rv;
                              upOrDown(toon);
                        }
                  }
            }
```

```
      meloodia.tempo = Math.round(skoor.getTempo());
      meloodia.madalaimNoot = skoor.getLowestPitch();
      meloodia.korgeimNoot = skoor.getHighestPitch();
      meloodia.pikimNoot = skoor.getLongestRhythmValue();
      meloodia.luhimNoot = skoor.getShortestRhythmValue();
      meloodia.korgendusi = korgendusi;
      meloodia.madaldusi = madaldusi;
    }


      public void Analyys() {

                  //kontrolli, kas on mõlemad failid laetud
            if(kontrolliFaile()){

                  Analyys1(s, meloodia1, noodid1);
                  Analyys1(s2, meloodia2, noodid2);

                  dialoog = "Analüüsiti edukalt faili "+s.getTitle()+" ja
"+s2.getTitle()+".";

            }else{
                  dialoog = "Analüüsimine lõpetatud. Viga failide laadimi-
sel.";
            }
      }


      public void LaeFail1(){
            FileDialog fd;
            Frame f = new Frame();

            fd = new FileDialog(f,
                "Ava MIDI fail",
                                    FileDialog.LOAD);
                  fd.show();

                  if(fd.getFile() ==null) {
                        dialoog = "Tõrge esimese faili laadimisega.";
                  }else{
                        s.setTitle(fd.getFile());
                        meloodia1.failinimi = s.getTitle();
                        Read.midi(s, fd.getDirectory()+fd.getFile());
                        System.out.println(fd.getFile());
                        dialoog = "Esimene fail laeti edukalt.";
                  }
      }

      public void LaeFail2(){
            FileDialog fd;
            Frame f2 = new Frame();

            fd = new FileDialog(f2,
                "Ava MIDI fail",
                                    FileDialog.LOAD);
                  fd.show();

                  if(fd.getFile() ==null) {
                        dialoog = "Tõrge teise faili laadimisega.";
                  }else{
                        s2.setTitle(fd.getFile());
                        meloodia2.failinimi = s2.getTitle();
                        Read.midi(s2, fd.getDirectory()+fd.getFile());
                        dialoog = "Teine fail laeti edukalt.";
                  }
      }
```

```
	public float arvutaProtsent(){
		//moodustan listi massiivist
		ArrayList listA = new ArrayList();
		ArrayList listB = new ArrayList();
		float protsent = 0;
		int loendur = 0;

		for(int i=0; i<noodid1.length; i++){
			if(noodid1[i][0]>0) listA.add(noodid1[i][0]);
		}
		for(int i=0; i<noodid2.length;i++){
			if(noodid2[i][0]>0) listB.add(noodid2[i][0]);
		}
		//arvutan nootide osakaalu
		if(listA.size() >= listB.size()){
			for(int i=0; i<listA.size(); i++){
				if(listB.contains(listA.get(i))){
					listB.remove(listA.get(i));
					loendur ++;
					}
			}
			System.out.println("=================");
				System.out.println(listA.size());
				System.out.println(loendur);
				protsent = (float) loendur / (float) listA.size();
			}else{
					for(int j=0; j<listB.size(); j++){
					  if(listA.contains(listB.get(j))){
						listA.remove(listB.get(j));
						loendur ++;
					  }
					}
					System.out.println("=================");
					System.out.println(listB.size());
					System.out.println(loendur);
					protsent = ((float) loendur / (float)
listB.size());
				}

		return protsent;
	}

	public void samastaTempod(){
		s2.setTempo(s.getTempo());
	}

	public void samastaHelistikud(){
		//leiab erinevuse kasutades kõige madalamat nooti
		int erinevus = Math.abs(s.getLowestPitch() -
s2.getLowestPitch());

		Mod.transpose(s, erinevus);
		System.out.println(s.getLowestPitch()+",
"+s2.getLowestPitch());

		System.out.println(erinevus);

		if(s.getLowestPitch() != s2.getLowestPitch()){
			Mod.transpose(s2, 2*erinevus);
		}
		System.out.println(s.getLowestPitch()+",
"+s2.getLowestPitch());
	}

	public void looHistogramm(){
		if(kontrolliFaile()){
			View.histogram(s);
			View.histogram(s2);
		}else{
```

```
                dialoog = "Failide avamisel tekkis tõrge.";
            }
        }

        public boolean kontrolliFaile(){
            if(s.getSize() == 0 || s2.getSize() == 0){
                dialoog = "Failide avamisel tekkis tõrge.";
                return false;
            }else{
                return true;
            }
        }

        public static String loeNoodid(){

            Set<Integer> set1 = new HashSet<Integer>();
            Set<Integer> set2 = new HashSet<Integer>();


            Set<Integer> yhised = new HashSet<Integer>();

            for(int i = 0; i < noodid1.length; i++){
                if(noodid1[i][0] > 0)set1.add((int)noodid1[i][0]);
            }

            for(int i = 0; i < noodid2.length; i++){
                if(noodid2[i][0] > 0)set2.add((int)noodid2[i][0]);
            }

            System.out.println("===ESIMESE MELOODIA UNIKAALSED NOODID===");

            Iterator it = set1.iterator();
            Iterator it2 = set2.iterator();
            Iterator it3 = yhised.iterator();

            while(it.hasNext()) {
              System.out.println(it.next());
            }

            System.out.println("===TEISE MELOODIA UNIKAALSED NOODID===");

            while(it2.hasNext()) {
                System.out.println(it2.next());
            }
            yhised.addAll(set1);
            yhised.addAll(set2);


            unikaalseid_1 = set1.size();
            unikaalseid_2 = set2.size();

            int yhised_noodid = (unikaalseid_1 + unikaalseid_2)-
yhised.size();

            //return yhised.size()+" / "+(unikaalseid_1 + unikaalseid_2);
            return ""+yhised_noodid;

        }

        public void upOrDown(int pitch) {
            if(pitch < eelnev) madaldusi++;
            if(pitch > eelnev) korgendusi++;
            eelnev = pitch;
        }
}
```