Ohjelmoinnin peruskurssi Y2
Projektityö: Nuottipiirturi

1. Henkilötiedot

Otsikko: Nuottipiirturi (Sheet Music Maker)

Tekijän nimi: Timo Vehviläinen, opiskelijanumero 225474

Koulutusohjelma: BIO, 3. vuosikurssi

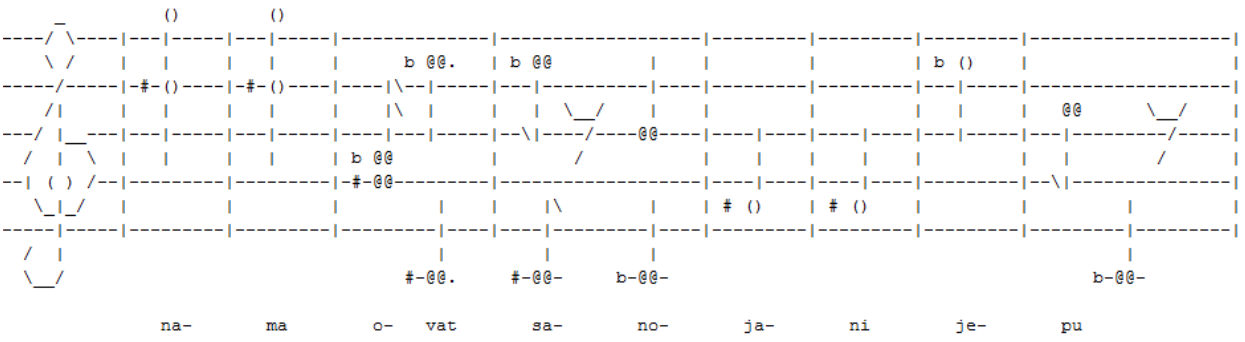
Päiväys: 24.4.2014

2.Yleiskuvaus

Työssä toteutettiin ohjelma, joka sekä tiedostosta lukien, että käyttäjän syöttämänä lukee tietoa musiikkillisista nuoteista, ja piirtää ne ASCII-kirjaimilla muodostetulle nuotistolle. Ohjelma osaa automaattisesti täyttää taukoja, hallitsee ylennykset ja alennukset, sekä tiedostoon tallentamisen. Se hallitsee eri pituisia nuotteja, taukoja, tahteja ja viivastoja, sekä päällekkäisiä nuotteja ja sanoituksia.

Tahtiviivan ylittävät nuotit rikotaan automaattisesti oikein tahtien oikeille puolille. Nuotteja voi lisätä ja poistaa (muuttaa tauoiksi), sekä kaikkia muitakin kappaleen tietoja voi muuttaa tekstipohjaisen käyttöjärjestelmän kautta

Työn pystyi suorittamaan kolmella eri vaikeusasteella, joista tähän toteutukseen suoritettiin ”vaikea”



Esimerkkikuva tulosteesta

3.Käyttöohje

Alustaminen tiedostosta

Ohjelma on tarkoitus käyttää main.py-tiedoston kautta. Komentoriviltä ajettaessa tälle ohjelmalle voi antaa yhden argumentin, jonka tulee sisältää oikeassa formaatissa oleva nuotinnos (.txt formaatissa). Mikäli argumenttia ei anneta, tai se on viallinen, käytetään oletusarvoisesti valmista empty.txt-tiedostoa. Formaatti on seuraavanlainen

1.tiedoston tulee alkaa rivillä ”#SHEETMUSIC”

2. Loppu tiedostosta on aseteltu lohkoihin, joita edeltää lohkon tunnus. Nämä lohkot, ja niiden tunnukset ovat seuraavat:

#SONG INFO

Tähän lohkoon tulee kappaleen nimi ja tekijä. Nämä annetaan seuraavanlaisesti:

title : [nimi, esim ”Nyan Cat”]

author : [tekijä, esim ”Mr. Nyan”]

#TIME

Tässä lohkossa ilmoitetaan kappaleen tahtilaji, sekä tahtien määrä seuraavanlaisesti:

bars : [tahtien määrä, esim ”5”]

signature : [tahtilaji, esim ”2/4”]

#NOTES

Tämä lohko sisältää kaikki nuotit. Jokainen nuotti sisältää 3 osatietoa: korkeuden, pituuden ja mahdollisen harmonian. Korkeus ilmoitetaan järjestyksessä nuotin nimi/etumerkki/oktaavi. Etumerkki ja oktaavi ovat vapaaehtoisia. Korkein mahdollinen nuotti on g#2 ja alin mahdollinen nuotti on cb1. Jos korkeuteen kirjoitetaan jotain muuta, se tulkitaan tauoksi.
Nuotin kesto ilmaistaan murtolukuna (tai desimaalilukuna) kokonuotista. Esimerkiksi neljäsosanuotin pituus on ”1/4”. Pisin mahdollinen nuotti on ”3/2” (pisteellinen kokonuotti) ja lyhin mahdollinen on ”1/16”.
Harmonianuotin korkeus ilmaistaan samalla tavalla kuin normaalin nuotin korkeus.
Nuotteja voi olla niin monta kuin halutaan, mutta ei enempää kuin tahteihin mahtuu.
Ohjelmaa ei myöskään ole suunniteltu tulostamaan pidempiä kuin yhden rivin viivastoja.
Eli nuotit asetetaan näin:

pitch : [nuotin korkeus, esim. a1]

duration : [nuotin pituus, esim. 3/8]

harmony : [harmonianuotin korkeus, esim c#2]

#LYRICS

Tämä on vapaaehtoinen lohko, johon voi lisätä sanoituksia kappaleeseen. Sanoitusten tulee mahtua yhdelle riville. Sanat erotetaan toisistaan välilyönnillä, ja tavut tavuviivalla. Ohjelma ei tue viittä kirjainta pidempiä tavuja, vaan tällöin sanat menevät pois paikoiltaan. Asetus tapahtuu siis esim. seuraavanalaisesti:

Bad-ger bad-ger bad-ger mush-room

Kaikissa näissä lohkoissa voi olla mielivaltainen määrä whitespacea, kunhan lohkojen tunnukset ja osatiedot kuten ”pitch” alkavat rivin alusta. Osatiedot erotetaan arvoistaan kaksoispisteellä (paitsi lyriikoissa, joissa ei ole etuliitettä osatiedoilla.)

Formaatin mukaan tiedosto lopetetaan #END – tunnuksella, mutta tämä ei ole pakollinen. Jos mitään tiedostoa ei anneta argumenttina, käytetään alustukseen oletusarvoisesti empty.txt-tiedostoa (4 tyhjää tahtia, tahtilaji 4/4)

Muokkaaminen komentoriviltä

Valmista tiedostoa tai tyhjää tiedostoa voidaan muokata tekstikomennoilla. Valintavaihtoehdot ovat seuraavat:

1. Modify a note

    Tämän avulla voi muokata valmiita nuotteja tai taukoja. Niiden korkeutta ja pituutta voi muuttaa. Nuotti valitaan antamalla sen tahtiluku, ja monesko nuotti se on sillä hetkellä siinä tahdissa.

2. Harmonize a note

    Tämän avulla voi valmiina olevaan nuottiin lisätä toisen yhtäaikaisen nuotin harmoniaan. Lisäys tapahtuu samalla tavalla kuin valmiin nuotin muokkaaminen, paitsi että harmonianuotin pituus on sama kuin sen rinnakkaisen nuotin

3. Edit song info

    Tällä vaihtoehdolla voidaan muuttaa kappaleen nimeä, tekijää, tahtilajia ja pituutta

4. Modify lyrics

    Tällä vaihtoehdolla voi muuttaa/lisätä lyriikoita. Ne lisätään samalla tavoin kuin tiedostossa.

5. Save to file

    Tämä vaihtoehto tallentaa nuotiston nykyisessä tilassaan SheetMusicMaker_Output.txt-nimiseen tiedostoon, joka tallennetaan data-kansioon.

6.Exit Sheet Music Maker

    Lopettaa ohjelman toiminnan

4.Ohjelman rakenne

Ohjelma on rakennettu neljään eri tiedostoon: main, parse, staff ja note. Näiden tehtävät ovat seuraavanlaiset:

main.py:

Main lataa mahdollisen valmiin tiedoston, ja vastaanottaa käyttäjän komentoja. Komentoja varten kysytään mahdolliset tarvittavat lisätiedöt, ja ne luovutetaan eteenpäin parse-luokalle.

parse.py

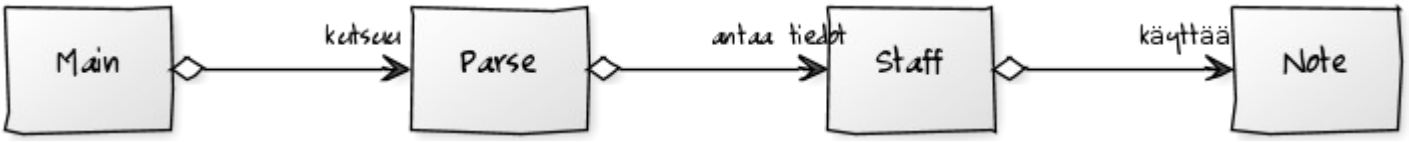
Parsen tarkoitus on parsia eli prosessoida käyttäjän antama tiedosto ja komennot numeeriseen muotoon, jotta ne voidaan tallentaa staff-objektiin pyöriteltäväksi dataksi. Muutamia keskeisiä metodeja ovat handleNotes(), modifyNotes(), convertTime()

staff.py

Staff (eli nuottiviivasto) on se kokonaisuus, joka pitää kaikki tauot, nuotit, tahdit, tahtilajin, ja viivaston printtaamisen hyppysissään. Täällä tapahtuu kaikki musiikillisesti merkityksellinen työ kuten tahtien pituuksien laskeminen, nuottien piirtäminen ja tahtien suoristaminen. Muutamia keskeisiä funktiota ovat esim. addNote(), printStaff() ja StraightenStaff()

note.py

Note-luokka on se peruspalikka, jota staff-luokassa pyöritellään. Note pitää sisällään tiedon yksittäisen nuotin korkeudesta, pituudesta, harmoniasta ja etumerkistä. Näillä ei ole omaa toiminnallisuutta, vaan niitä käsitellään vain kasana dataa.



5.Algoritmit

Ohjelman keskeisin algoritmi on varmaankin se, jolla nuotit muunnetaan niiden tekstimuotoisista nimistä numeerisiksi arvoiksi. Tässä käytetään hyödyksi sitä, että kirjaimilla a-g (jotka ovat nuottien nimiä) on myös vierekkäiset ASCII-arvot. Tästä syystä valittiin g myös korkeimmaksi mahdolliseksi nuotiksi, sillä se tekee kvantisoimisesta helpompaa. Kirjain g muunnetaan luvuksi 0, f luvuksi 1, e luvuksi 2, d luvuksi 3 jne. Nämä luvut käyvät sopivasti Staff-luokassa käytettävään tulostamista varten luotavaan merkkimatriisiin rivikoordinaateiksi.

Yhtenä monista muista vaihtoehdoista olisi ollut asettaa esim. alimmaksi mahdolliseksi nuotiksi a, jolloin nuottien numeeriset arvot olisivat nousseet loogisesti nuottiviivastolla alhaalta ylös noustessa. A = 1, b = 2 jne. Valitsemaani matriisitulostusmenetelmää varten käännös olisi kuitenkin jouduttu tekemään tulostusvaiheessa.

Toinen keskeinen algoritmi on juurikin tulostuksessa käytettävä algoritmi. Tätä ongelmaa lähestyttiin enemmän tuhlaavaisella otteella, muistitilaa säästelemättä. Viivaston piirtämistä varten luotiin jokaiselle tarvittavalle merkille (välilyönnit mukaanlukien) oma paikka kaksiulotteiseen merkkimatriisiin, josta sitten nuotteja lisäillessä muutettiin merkkejä yksi kerrallaan.

Kolmanneksi ja neljänneksi olennaisimmat algoritmit käsittelevät saman kolikon hieman eri puolia. Nämä ongelmat ovat vajaiden tahtien täyttö ja ylitäysien tahtien tasaus.

Vajaat tahdit täytetään toteutetussa algoritmissa siten, että tahdin loppuun jääneeseen tilaan sovitetaan iteratiivisesti aina suurinta saatavilla olevaa taukoa, kunnes tahti on täysi.

Ylitäydet tahdit tasataan siten, että tahtiviivan ensinnä ylittävä nuotti katkaistaan kahtia, ja kaikki tahdissa olevat nuotit (ja tauot) sen jälkeen siirretään seuraavaan tahtiin. Tämä suoritetaan jokaiselle tahdille, ja lopussa luodaan lisää tahteja, mikäli on tarpeen. Yksi vaihtoehto tälle menetelmälle olisi ollut vain poistaa kaikki ylimääräiset nuotit tahdeista, mutta tämä ei olisi mukaillut vaativalle vaatimustasolle vaadittuja ominaisuuksia.

6. Tietorakenteet

Työssä käytettiin paljon dynaamisia tietorakenteita, lähinnä listoja ja 2D-listoja. Nämä osoittautuivat käytännölliksi, koska viivastolla voi olla vaihteleva määrä tahteja, tahdissa vaihteleva määrä nuotteja, sanoissa vaihteleva määrä tavuja jne.
Yksi vaihtoehtoinen lähestymistapa olisi ollut tehdä tahditkin omana luokkanaan, ja antanut niiden puhua nuoteista keskenään, sen sijaan että Staff yksinään hallitsee kaikkea. Katsoin kuitenkin tahtien välillä olevan niin paljon liikettä, että sitä olisi helpompi hallita ylhäältäpäin.

Kaikki tieto minkä pystyi varhaisessa vaiheessa puristamaan puhtaasti numeeriseen muotoon, kvantisoitiin liki suoraan. Tähän sisältyi niin nuottien korkeudet kuin etumerkit, pituudet ja tahtilaji.

Esim tahtilajin olisi voinut säilyttää erikseen ns. ”osoittajana” ja ”nimittäjänä”, mutta lopulta päädyin kuitenkin vain ilmaisemaan sen yhtenä lukuna. Tästä mm. seurasi se, että tätä lukua oli hyvin haastavaa kääntää uudestaan murtolukumuotoon tulostettavaksi (eikä tätä tulostusominaisuutta siis lopulta toteutettukaan).'

Yksi ehkä epäonnistuneemmista tietorakennevalinnoista oli harmonianuottien rakenne. Toteutuksessa jokaisella ”varsinaisella nuotilla” voi olla omistuksessa yksi ”harmonianuotti”, jota muut nuotit eivät nää. Tähän ratkaisuu päädyttiin, koska esim. varsien suunta piti harmonianuoteissa vain päättää jommasta kummasta nuotista. Tämä tuotti kuitenkin ongelmia ylennysten ja alennusten voimassapysymisess', kun kaikki viivastolla olevat nuotit eivät nähneet toisiaan helposti.

7. Tiedostot

Ylimmässä ”SheetMusicMaker”-kansiossa on kaikki ajamiseen tarvittavat kooditiedostot, sekä yksikkötestausohjelma:

- main.py
- parse.py
- staff.py
- note.py
- corruptedFileError.py
- unitTests.py

Data-nimisessä kansiossa on monia esimerkkejä erinäisistä nuotinnos tiedostoista, joita käytetään yksikkötestauksessa (.txt muodossa). Yksi näistä on myös main-ohjelman alustamiseen tarvittava empty.txt tiedosto. Siellä on myös vapaavalintainen G-cleff.txt -tiedosto nuottiavaimen tulostusta varten. Tällä tavoin tähän voisi lisätä vaikka muitakin nuottiavaimia. Lisäksi, tallennettaessa nuottikuvia ohjelman sisältä, tallentuvat myös ne data-kansioon.

8.Testaus

Ohjelmaa testattiin rakennusvaiheessa yksikkötestauksilla, jotka muuttuivat vaativimmiksi joka kerta kun lisää toiminnallisuuksi lisättiin. Ohjelma läpäisee kaikki tämänhetkiset yksikkötestit. Kun ohjelma saatiin käytettävään muotoon (eli main.py oli kutakuinkin valmis), testattiin sitä lukuisia kertoja erilaisia komentoja käsin syöttämällä. Joillakin komennoilla testattiin toimivia ominaisuuksia, mutta useimmilla yritettiin erityisesti löytää puutteita, ja kohtia, joissa ohjelma saattaisi yllättäen kaatua. Aivan kaikista tälläisistä testeistä ei ohjelma selvinnyt, ja näitä paikattiin ajan salliessa.

En näe testauksen suunnittelussa olennaisia aukkoja. Vaikka testit olisivat periaatteessa voinut olla etukäteen pidemmälle suunniteltuja, koko ohjelman rakenne eli sen verran kirjoitusprosessin aikana, että siitä olisi saattanut olla loppujenlopuksi enemmän haittaa kuin hyötyä.

9. Ohjelman tunnetut puutteet ja viat

Haluttuja ominaisuuksia, joita ei keretty/pystytty saada mukaan:

- Tahtilajin esitys viivastolle, sekä murtolukumuodossa infoteksteihin
- Sävelajien toteutus
- palkit vierekkäisten 8-osien ja 16-osien välille
- palautusmerkit

Nämä olisin lähes kaikki toteuttanut lisäyksillä ja muokkauksilla Staff-luokan metodeihin

Tunnettuja bugeja/epäkohtia valmiissa koodissa:

- Ohjelma ei siedä, jos tiedostossa on enemmän nuotteja kuin mitä tahteihin mahtuu
- Jos harmonianuotissa on etumerkki, se ei vaikuta muihin samassa tahdissa oleviin nuotteihin
- Jostain syystä joskus myös ¼-osaa pidemmillekin harmonianuoteille tulee kiinteä pää.
- Joskus ala-c:n apuviiva peittää pisteellisen nuotin pisteen.
- Jostain syystä, kun siirsin tiedostot Windows-koneelle, tallennetut tekstitiedostot näyttivät vääriltä (sain lopulta toimimaan notepad++ ohjelmalla. Kaikki testaus on kuitenkin suoritettu Ubuntulla)

Osaa näistä en osaa selittää, sillä niitä tapahtuu vaikka luulin korjanneeni tilanteen (lähinnä kaksi viimeistä kohtaa). Ensimmäisen bugin korjaamiseen tarvitaan parempaa virheenhallintaa, ja toiseen Nuotti-luokan uudelleentyöstämistä.

10. 3 parasta ja 3 heikointa kohtaa

3 omasta mielestäni parasta kohtaa:

- nuottikorkeuksien, pituuksien ja tahtilajin kvantisointialgoritmit
- Tahtien suoristamisalgoritmi
- Metodijako luokkien sisällä, kommentointi

3 omasta mielestä heikointa kohtaa:

- Harmonioiden käsittely
- Etumerkkien puutteet
-Tulostusalgoritmin tilatehokkuus ja koodin tehokkuus

11. Poikkeamat suunnitelmasta

Suuria poikkeamia suunnitelmaan ei tullut. Aikaa tuli käytettyä vähemmän kuin ehkä oli aikomus, osittan pakosta ja kiireestä. Toteutusjärjestys pysyi samana.

12.Toteutunut työjärjestys ja aikataulu

Kaikenkaikkiaan työ tuli tehtyä yhteensä noin viikon sisällä, josta ensimmäinen puolikas oli yhtenä viikonloppuna ennen välitapaamisen aikaa ja toinen juuri ennen palautusta (Dokumentti kirjoitettu 3:ssa tunnissa). Äkillisen ulkomaamatkan sijoittuminen dediksen päälle haittasi lopputyöskentelyä. Tekemistä olisi kieltämättä voinut koittaa sijoittaa enemmän sinne periodin alkupäähän. Välitapaamista en sitten kuitenkaan varannut, koska suunnittelutapaamisessa tämän sanottiin olevan vapaaehtoista.

13. Arvio lopputuloksesta

Arvioisin kuitenkin onnistuneeni tavoitteissani ja suunnitelmassani aika hyvin, vaikka kaikki ei ihan nappiin mennytkään. Olen työn tulokseen tyytyväinen, ottaen huomioon siihen panostetun työajan, ja olen varma, että jos olisin voinut käyttää siihen lisää aikaa, olisi siihen tullut helposti lisää laatua. Antaisin itselleni projektista 4/5.

14.Viitteet

Apuna tuli käytettyä lähinnä paljon Googlea, jossa ehdottomasti suosituin hakutulos ole stackoverflow.com.

15.Liitteet

Liitteenä on kaikki vaadittavat tiedostot zip:ssä. Valitettavasti Ubuntu-koneestani loppui kriittisillä hetkillä akku, eikä Windows-koneessani ole tarvittavaa ohjelmistoa esimerkkiajon liittämiseen (ei Python 3:a, puoli tuntia ulkomaanlaivan lähtöön enkä siis kerkiä asentelemaan)