*The Benefits of Computer Systems in Composing Music*

By: James Loui

Faculty Mentor: Professor Sarah Zelikovitz

INTRODUCTION

This paper will discuss how data used by computer systems can benefit the composition process of music. There is previous work that relates to our objective and illustrates and explains how music can be modified using computer systems with several types of data. One objective of our work is to design a program that composes, plays or modifies music. Music theory concepts will be discussed as these concepts have an influence towards the design of the program. The program is written in Python, a programming language, and uses pyo, a library related to Python and music.

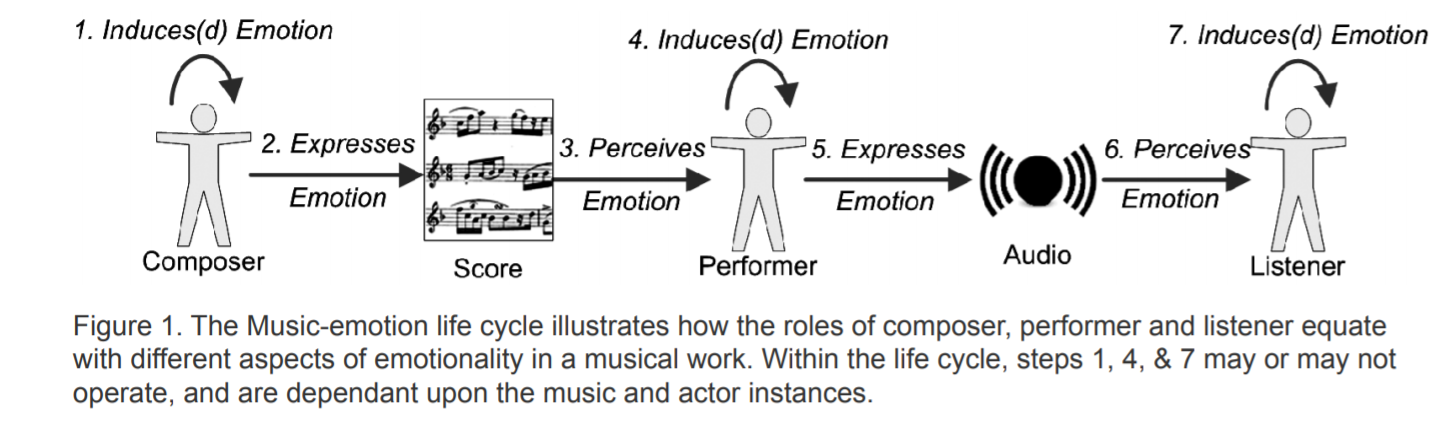
LITERATURE REVIEW

This section mentions several literatures that had an influence towards the design of the program and will start with briefly describing the composition of music as other subsections of this section will mention details in regards to the composition of music.

COMPOSITION OF MUSIC

The composition of music includes the written structure of a song and sounds and lyrics played by musical instruments and sung by a singer. The written structure, sounds, and lyrics are represented by the score or performance of a song. In addition, the composition of the written structure, sounds, and lyrics are influenced from contextual details which can vary. The types of data used by computer systems can supply those contextual details and those types of data will be described shortly.

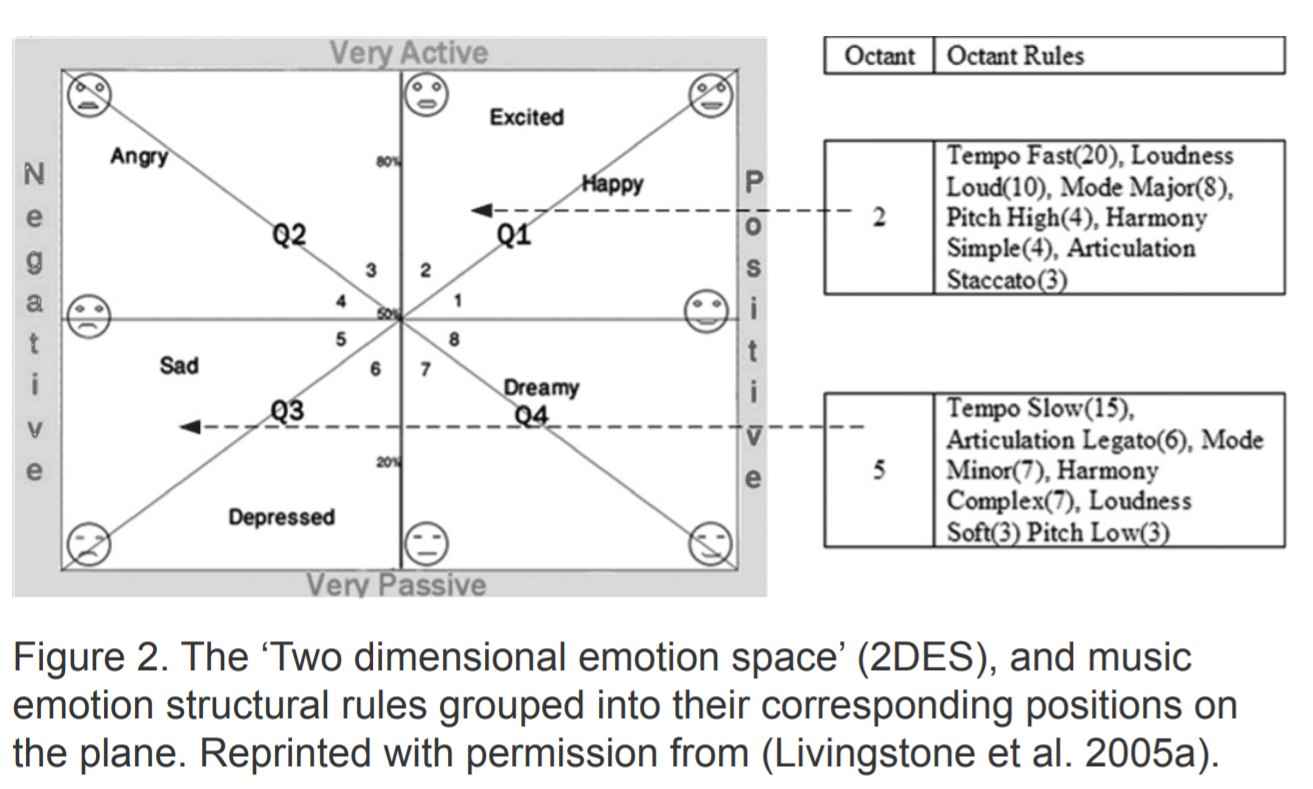
MUSICAL EMOTIONALITY

According to Livingstone, Mühlberger, Brown, and Loch, musical emotion is stated to be looked at as a “transmissible entity in a multi-stage process; or life cycle” and is represented as “an expressive representation”, “an affective response”, and “perceived expressive intent” (Livingstone, Mühlberger, Brown, and Loch 44). The figure below showcases musical emotion as a life cycle.

Source: Livingstone et al. 2007, p. 44

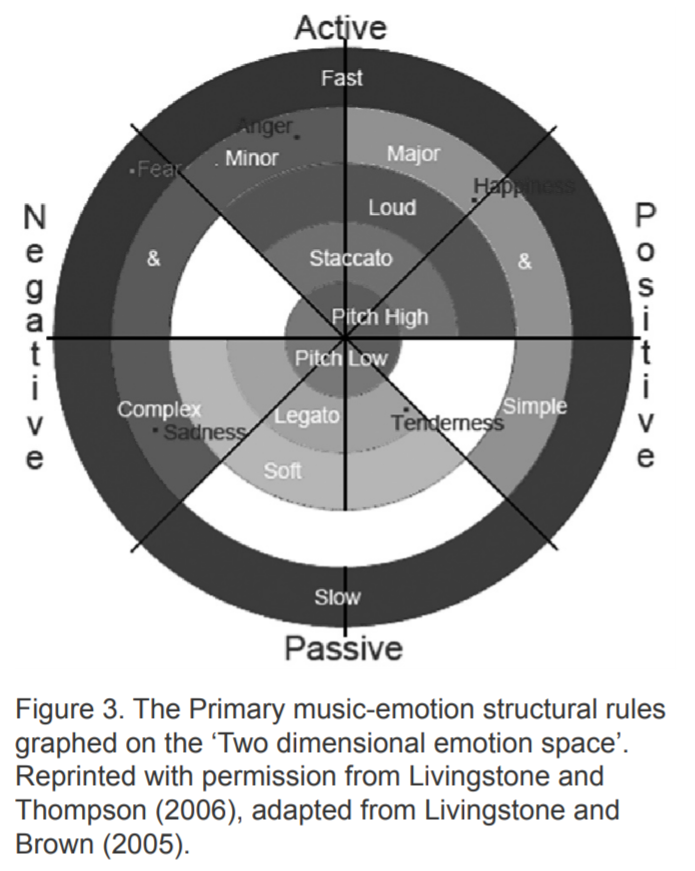
The life cycle is intended to showcase the difference between induced, expressed, and perceived emotions (44). Expressed emotion is the type of emotion found from audio, visual, or other forms of stimulus and is most likely the expressive representation (45). Induced emotion is given to a person by at least one of the stimuli and is most likely the affective response (45). Perceived emotion is the type of emotion formed from observing expressed emotions and is most likely the perceived expressive intent (45).

Rules for music emotions exist and are divided into several categories. A couple of the categories are structural rules and performative rules (46). Structure rules are utilized by composers and associated with music score while performative rules are utilized by performers who interpret the music score (46). Structural rules are also used with the figure below.



Source: Livingstone et al. 2007, p. 46

The above figure has eight “emotional octants” and each two octants are divided into a “quadrant subdivision” (46). A part of the structural rules for Octant 2 and Octant 5 are displayed from the figure which are known as “octant rules” (46). For instance, tempo fast, tempo slow, and harmony complex are all structural rules, see Figure 2. The value inside the parenthesis next to each rule is “the number of independent studies that reported” that musical works with their corresponding rule were described as either sad or excited depending on the octant associated with them (46). The figure below represents a more detailed illustration of the structure rules based on Figure 2 including the octant rules not displayed from the figure.



Source: Livingstone et al. 2007, p. 46

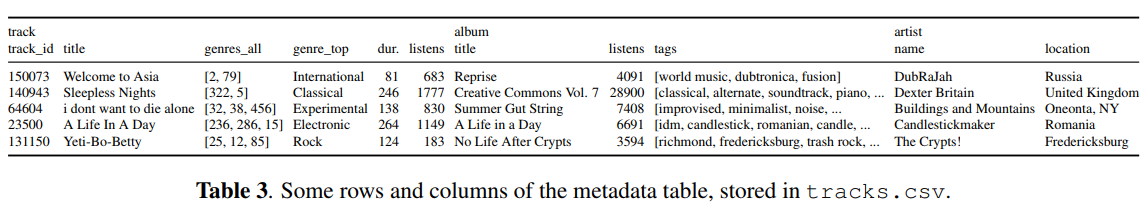
The structural rules from Figure 3 are displayed as “concentric rings” (46). The octant rules associated with the musical works that have the highest number of studies are represented close to the outer edge of the circular graph while those associated with musical works that have the lowest number of studies are represented close to the middle of the graph (46). Each description of each emotional octant associated with the rules, such as happy or sadness, can also be found on the graph (46). The “use of opposing rules reflected around” the vertical axis, separating positive from negative emotions, and horizontal axis, separating active from passive emotions, can be treated as a “communication mechanism” (46-47). The communication mechanism is the most likely way to receive data for those types of music emotions.

TYPES OF DATA

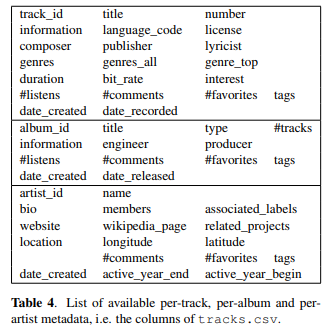
The types of data that can be used for testing purposes include environmental contextual data, audio data, user data, meta-data, and free-form text. Environmental contextual data can include measured environmental temperature, humidity, and light and GPS data that can identify a location as an urban or rural one. (Weinel, Cunningham, Grifiths, Roberts, and Picking 17-18). That type of data can be used to influence the contextual details of lyrics and the arrangement of sounds to match that data. Audio data includes tempo data which describes the speed of a composition of a song and pitch data which can describe a scale of a music (18). Audio data can be used to improve the composition process of music. User data is data specific to the listeners of the music and can include play counts, favorites, and comments (Defferrard, Benzi, Vandergheynst, and Bresson 1). In addition, user emotion or desire can be part of user data. Meta-data is made up of song titles, albums, artists, and genres (1). Free-form text is composed of description, biography, and tags (1). User data, meta-data and free-form text can be used to modify or create the lyrics, sounds, and structure of a song. Collecting these types of data can be done with the reported tools or methods.

METHOD TO COLLECT THE DATA

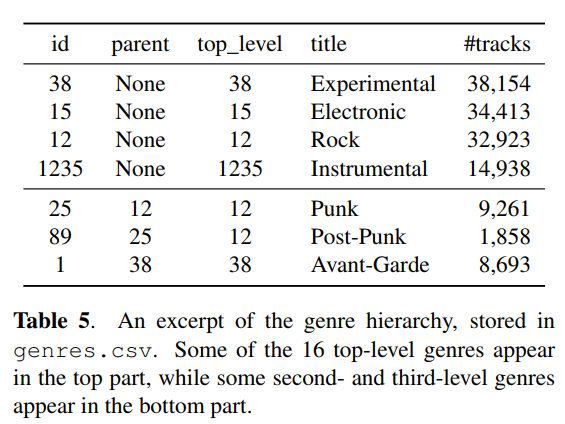
The methods or tools that can be used to collect data are sensors, audio analysis, interpretive algorithms, datasets, cross-references, and APIs. Sensors can be used to record environmental contextual data since GPS sensors can record the GPS data that can identify urban or rural locations and an Arduino-based “Sensor Belt” can be used to record environmental temperature, humidity, and light (Weinel, Cunningham, Grifiths, Roberts, and Picking 17). Audio analysis can record and identify tempo data and pitch data with tempo analysis and pitch analysis, respectively (18). Interpretive algorithms can be used to record user emotion or desire (18). Datasets can be used to archive meta-data, user data made up of play counts, favorites, and comments, free-form text, and audio data available from the API (Defferrard, Benzi, Vandergheynst, and Bresson 1-3).



Source: Defferrard et al. 2017, p. 3.



Source: Defferrard et al. 2017, p. 3.

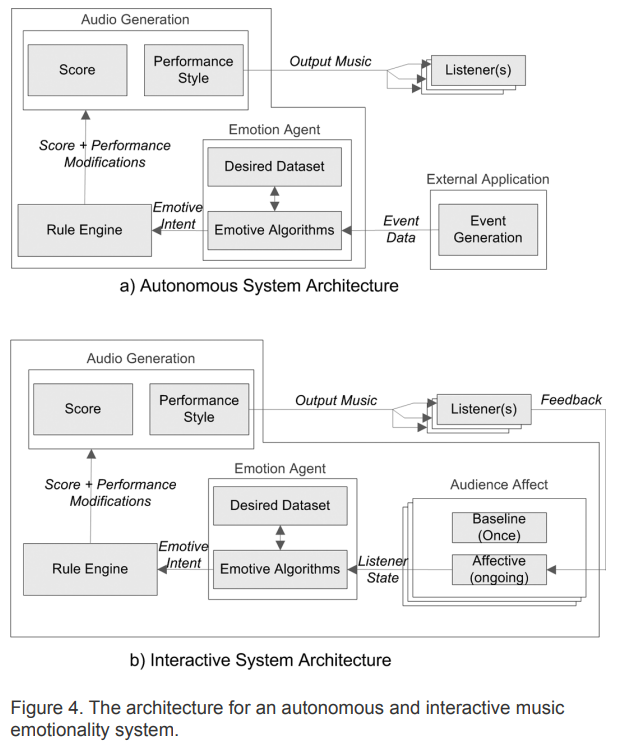


Source: Defferrard et al. 2017, p. 4.

The tables above are representative of some of the datasets used to archive meta-data and the user data. Furthermore, cross-referencing can be used to include more user data, meta-data, and free form text as the method can be used to receive free form text. Specifically, social tags in online music services, as represented in computer systems can be added through analysis (Saari and Eerola 2548-49). All of the types of data described can be obtained by the above methods or tools and examples of how computer systems utilize the data to modify music will be explained shortly.

METHODS TO TEST THESIS

Computer systems can utilize the data to modify music through a few methods. One of them would be using algorithms to modify music based on the data. Another one would be to use a computational architecture to modify music based on the data.



Source: Livingstone 2007, p. 50

The illustration above uses both an algorithm and computational architecture and illustrates how music can be modified using computer systems and several types of data. The source that includes the illustration has provided an example on how data used by computer systems can influence music based on the “Interactive System Architecture” part of the illustration. The “Interactive System Architecture” part of the illustration had an influence on how the program was designed.

EXPERIMENTATION

MUSIC THEORY

Several concepts of music theory has had an influence towards the design of the program. The concepts involve notes, rest notes, octaves, and scales. Notes in music theory can be represented as whole notes, half notes, quarter notes, eighth notes, and sixteenth notes (Duckworth 7). These notes represent the duration of sound and are proportionately related with one another (7-8). For example, one half note is equivalent to two quarter notes in duration and one whole note is equivalent to four quarter notes in duration (7). The concepts of these type of notes relate to a code from the program involved with the concept of the duration of a beat. Rest notes exist in music theory and represent the duration of silence (8). Rest notes can be represented as whole rest, half rests, quarter rests, eight rests, sixteenth rests and these rests are proportionately related with one another (8). Rest notes are implemented in the program which will be explained in this paper. The concept of octave identification involves determining notes or pitches in a specific octave (77). Notes are partially represented as one of the first seven letters of the alphabet (11). An octave is a way to measure the distance between a note and the next note of the same name (76). On an illustrated piano, locating a key twelve keys to the right from C3, a key that can be represented as a note, results in C4, another key that can be represented as a note, to be located (77). This calculation involved with locating C4 is related to how the octave concepts are implemented in the program. How the songs from the program are composed is involved with the concept of scales. Scales represent a basic pattern of notes or pitches that make up the melodies of songs (95). Two types of these scales exist and one of them is major scale with the other being minor scale (95). The songs composed from the program use minor scale in which one of them uses B-flat minor scale and the other uses E minor scale. The notes from the songs are represented as MIDI note numbers in the program in which several notes found in songs can be represented as MIDI note numbers. For instance, C4 is represented as 60 in MIDI note number form (“MIDI Note Numbers and Center Frequencies”).

PROGRAM DESIGN

As this project covers multiple parts of the field of computer science, an explanation of how programs are designed will be explained in this section. Programs are designed through the use of files and written code. Programming languages provide specific sets of code that can be written on the files associated with their respective programming languages to design programs. How programs function depends on the content of the code from the files that store the code. Programs can also access other external files to read data from them in which the program that will be covered in a later section accesses an Excel Spreadsheet file to read music related data from the file. That is done through the use of programming languages’ libraries which consist of combinations of code in which each combination makes up what is called a function that can be used by programs to perform tasks associated with their respective libraries.

PYTHON

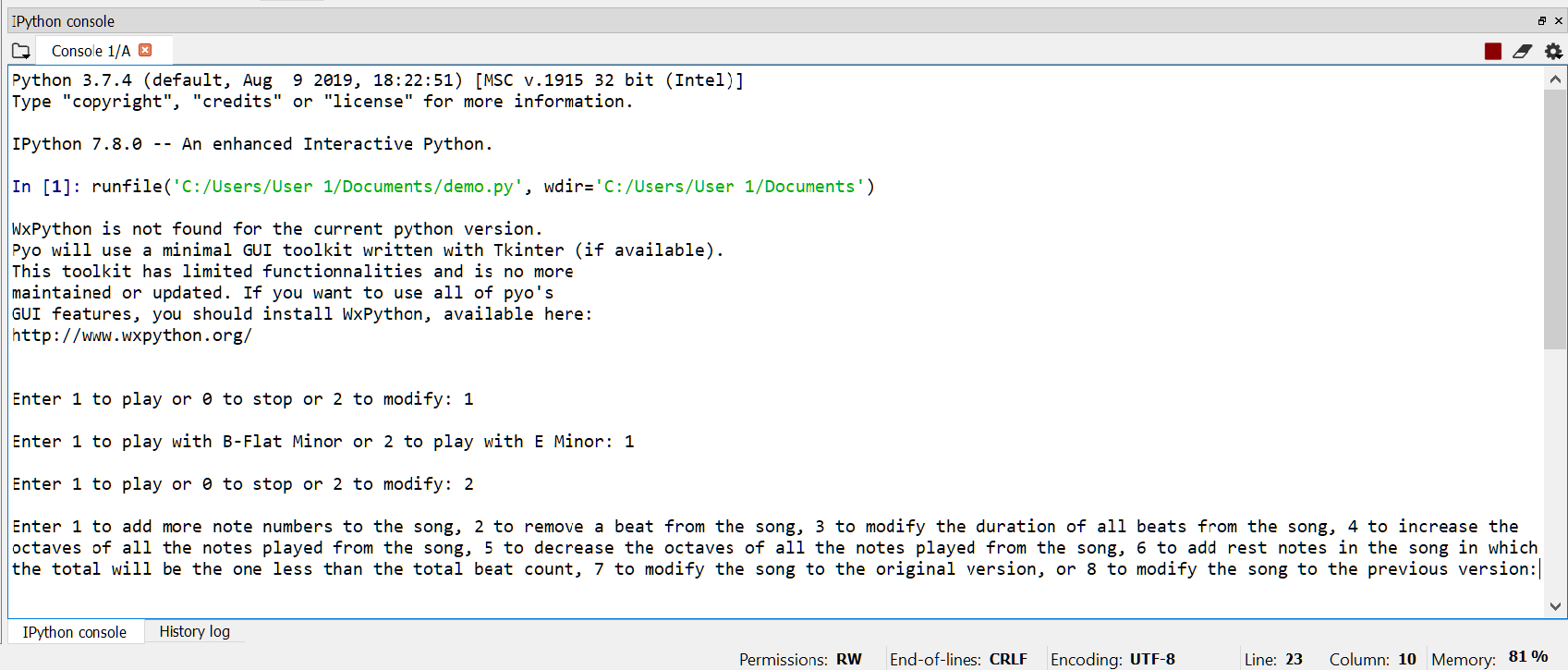
This section will cover Python and its libraries that are used for the design of the program except for the library, pyo, as that will be covered in another section. Python is a programming language and the libraries excluding pyo that are imported to the program so they can be used are openpyxl, time, and random (Python Software Foundation). The openpyxl library is imported to allow for the program to read data from an Excel Spreadsheet file (“Openpyxl”). The time library offers time-related functions in which the sleep function from the time library contributes to the rest note implementation of the program (Python Software Foundation). While the random library is imported to the program, it is not used by the program currently, but is considered for possible implementations which is the reason it is kept in the program (Python Software Foundation). Python version 3.7 was chosen for the design of the program as it can work with the libraries imported to the program and is still supported.

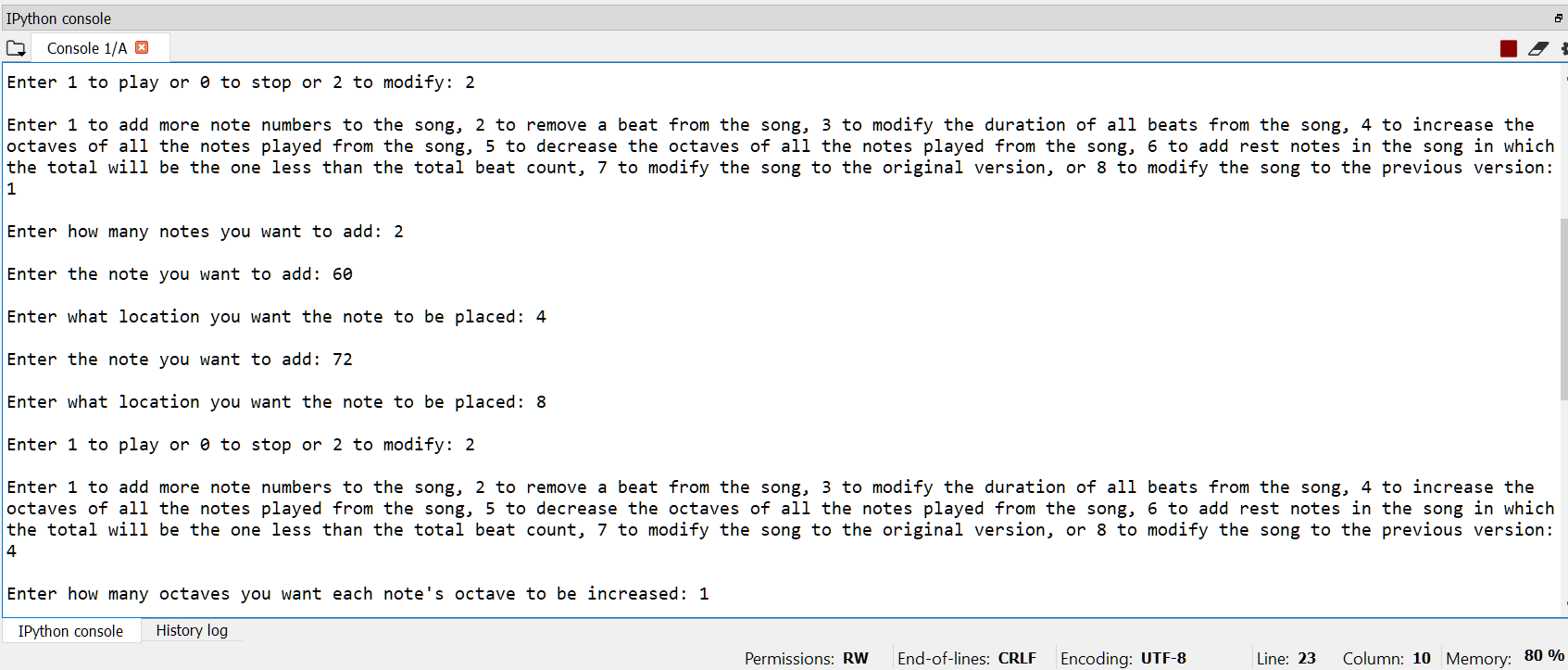
PYO

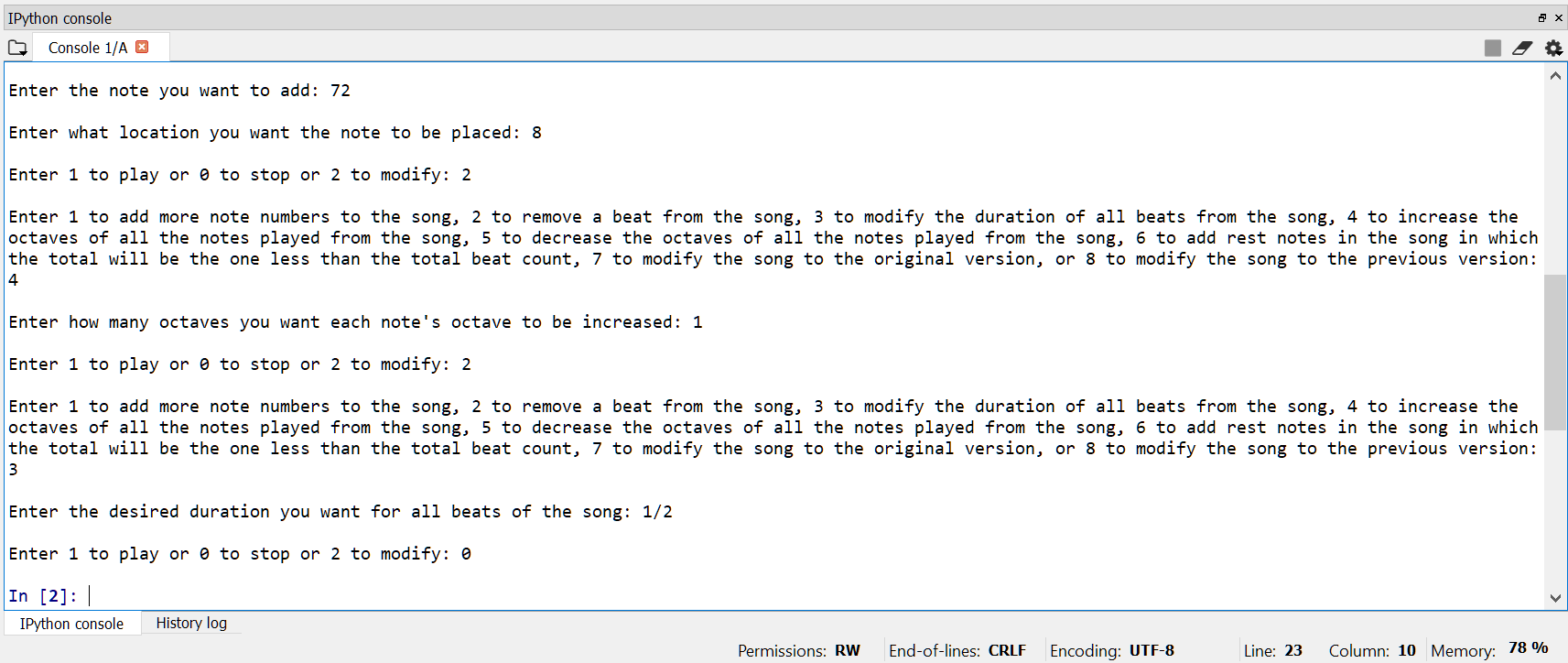
This section will cover the pyo library in greater detail including most of its functions that are used in the program. The pyo library is stated to help in digital signal processing script creation and provide ways to compose algorithmic music (*About pyo*). The library also provides ways to compose music in a direct manner with the functions that will be mentioned. Several of those functions are Server().boot function, Events function, EventSeq function, and midiToHz function (*About pyo*). The Server().boot function is used to be able to use the other functions (*About pyo*). The midiToHz function converts a MIDI note number to Hertz frequency (*About pyo*). The EventSeq function takes in the converted MDI note number or a converted list of them in order for music based on the MIDI note number arrangements to be played (*About pyo*). The music can be played infinitely or not depending on what value the occurrences part of the function is set in and this will be touched on more in the next section (*About pyo*). The Events function allow for the music to be played or stopped and its components that are used in the program are its freq component and its beat component (*About pyo*). Its freq component’s value is dependent on the EventSeq function’s values and requires for the values to be in a form relating to frequency which is the reason the midiToHz function is used in the program (*About pyo*). The beat component represents the duration of each of the converted MIDI note numbers in beat value in which 1 beat value is equivalent to a quarter note (*About pyo*).

PROGRAM FUNCTIONALITY

This section will cover the program’s functionalities and refer to several details mentioned in previous sections. The program executes its code from top to bottom and left to right. Initially, the program will give the user the option to play music, modify music, or terminate the program. It is preferred for the user to pick the option to play music first to better understand how to modify the music that will be playing. Once the option is picked, the user will be given the option to choose which type of music to play in which two options exist in the program. One option is to play a music in B-flat minor and the other option is to play a music in E minor. Depending on the option that is chosen, the music associated with the option will be played. The user is given a choice to play the music again, modify the music, or terminate the program. The option to play the music again has not been implemented in the program, but is considered to be used for future versions of the program which is why it is kept there. Picking the option to modifying the music would be the preferred next step in which a list of options is then given on how the music can be modified. First option allows for the user to add more MIDI note numbers that are represented as beats to the arrangement of the music in specific positions of the arrangement. The second option allows for the user to remove a MIDI note number from the arrangement of the music in a specific position of the arrangement. The third option grants the user permission to modify the beat value of all of the notes from the arrangement of the music. Option four grants the user the permission to increase the octaves of all of the notes from the arrangement. Option five allows for the user to decrease the octaves of all of the notes from the arrangement. The sixth option allows for the user to add rest notes to the arrangement in which the total the user will have to enter in will be one less than the total number of notes in the arrangement. Option seven allows for the user to modify the song to the original version. Option eight grants the user the option to modify the song to the previous version before a modification was made. After either of the options are chosen and then the right user inputs are entered in, the user will be granted the option to modify the music again or terminate the program.







These screenshots illustrate how the program’s functionalities are executed and the program’s code can be found in Appendix A.

CONCLUSION

This research paper explains how data used by computer systems can benefit the composition process of music. The functionalities of the program enable one to understand specifically how music can be modified to benefit the composition process of music. Aspects that influenced the design process of the program were music theory concepts and previous works that relate to our objective. Specifically, the illustration of the algorithm and computational architecture had a major influence on the design process. One conclusion that can be made is that the program can serve as a starting point in understanding how to modify music through the use of computer systems.

Works Cited

Defferrard, Michaël, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. “FMA: A Dataset

For Music Analysis.” *eprint arXiv:1612.01840v2*, 25 Apr. 2017, pp. 1-8.

Duckworth, William. *A Creative Approach to Music Fundamentals*. Schirmer Cengage Learning,

2010.

Livingstone, Steven R., Ralf Mühlberger, Andrew R. Brown, and Andrew Loch. “Controlling

Musical Emotionality: An Affective Computational Architecture for Influencing Musical

Emotions.” *Digital Creativity*, vol. 18, no. 1, Mar. 2007, pp. 43–53. *EBSCOhost*,

doi:10.1080/14626260701253606.

Saari, Pasi, and Tuomas Eerola. “Semantic Computing of Moods Based on Tags in Social Media

of Music.” *IEEE Transactions on Knowledge & Data Engineering*, vol. 26, no. 10, Oct.

2014, pp. 2548–2560. *EBSCOhost*, doi:10.1109/TKDE.2013.128.  
Weinel, Jonathan, Stuart Cunningham, Darryl Grifiths, Shaun Roberts, and Richard Picking .

“Affective Audio.” *Leonardo Music Journal*, vol. 24, no. 1, Dec. 2014, pp. 17–20.

*EBSCOhost*, doi:10.1162/LMJpass:[\_]a\_00189.

“MIDI Note Numbers and Center Frequencies.” RSS,

[www.inspiredacoustics.com/en/MIDI\_note\_numbers\_and\_center\_frequencies](http://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies).

Python Software Foundation. Python Language Reference, version 3.7. Available at

<http://www.python.org>

*About pyo*. Ajax Sound Studio, 2016, ajaxsoundstudio.com/pyodoc/about.html. Accessed 2 Sep.

2019.

“Openpyxl.” PyPI, pypi.org/project/openpyxl/.

Appendix A

# -\*- coding: utf-8 -\*-

"""

Created on Thu Feb 13 17:19:37 2020

@author: User 1

"""

import openpyxl as xl

from random import randrange

from pyo import \*

from time import \*

##testing on opening Excel Spreadsheet

wb = xl.load\_workbook('pyoMIDISongDataTest.xlsx')

sheet = wb["Sheet1"]

cell = sheet["a1"]

s = Server().boot()

s.start()

i = eval(input("Enter 1 to play or 0 to stop or 2 to modify: "))

if i == 1:

bFlatMinor = [70, 72, 73, 75, 77, 78, 80, 82]

#B-Flat Minor = [A#4/Bb4, C5, C#5/Db5, D#5/Eb5, F5, F#5/Gb5, G#5/Ab5, A#5/Bb5]

midiSongB = []

for row in range(2, sheet.max\_row + 1):

cell = sheet.cell(row, 1)

midiSongB.append(cell.value)

print(cell.value)

#midiSongB = [70, 72, 75, 73, 78, 73, 77, 75, 78, 75, 80, 78, 80, 78, 77, 75, 72, 70, 73, 77, 78, 75]

#midiSongB += [72, 75, 82, 80, 82, 80, 78, 77, 72, 73, 72, 68, 66, 60, 63, 60, 61, 58, 60, 65, 66, 68]

#midiSongB += [70, 72, 73, 80, 77, 82, 84, 87, 89, 92, 94, 92, 89, 84, 80, 77, 73, 75, 72, 73, 72, 70]

midiSongB += [77, 73, 75, 72, 73, 72, 70]

eMinor = [64, 66, 67, 69, 71, 72, 74, 76]

#E Minor = [E4, F#4, G4, A4, B4, C5, D5, E5]

midiSongE = [64, 66, 69, 66, 71, 69, 72, 76, 74, 72, 69, 67, 66, 64, 66, 69, 71, 74, 78, 79]

midiSongE += [83, 86, 88, 84, 79, 74, 71, 69, 66, 64, 66, 71, 72, 74, 76, 78, 76, 72, 69, 67]

midiSongE += [69, 72, 76, 78, 83, 84, 83, 81, 79, 78, 76, 72, 74, 71, 69, 71, 67, 69, 66, 64]

choice = eval(input("Enter 1 to play with B-Flat Minor or 2 to play with E Minor: "))

if choice == 1:

e = Events(freq=EventSeq(midiToHz(midiSongB), occurrences = 0), beat=1.).play()

midiSong = midiSongB[:]

if choice == 2:

e = Events(freq=EventSeq(midiToHz(midiSongE), occurrences = 0), beat=1.).play()

midiSong = midiSongE[:]

#midiSong to be used with this list, [70, 72, 75, 73, 78, 73, 77, 75, 78, 75, 80, 78, 80, 78, 77, 75, 72, 70, 73, 77, 78, 75, 77, 73, 75, 72, 73, 72, 70]

#############################################midiSong = midiSong2[:]########

midiSongHolder = midiSong[:]

duration = 1

durationHolder = duration

restNoteDuration = []

restNoteDurationHolder = restNoteDuration[:]

durationHolder2 = duration

while i == 1 or i == 2:

i = eval(input("Enter 1 to play or 0 to stop or 2 to modify: "))

if i == 2:

#midiSong = midiSongHolder

choice2 = eval(input("Enter 1 to add more note numbers to the song, "

"2 to remove a beat from the song, "

"3 to modify the duration of all beats from the song, "

"4 to increase the octaves of all the notes played from the song, "

"5 to decrease the octaves of all the notes played from the song, "

"6 to add rest notes in the song in which the total will be the one less than the total beat count, "

"7 to modify the song to the original version, "

"or 8 to modify the song to the previous version: "))

e.stop()

if choice2 == 1:

midiSongHolder2 = midiSong[:]

durationHolder2 = duration

restNoteDurationHolder = restNoteDuration[:]

numNotesAdded = eval(input("Enter how many notes you want to add: "))

numNotesAddedHolder = numNotesAdded

while numNotesAdded > 0:

noteAdded = eval(input("Enter the note you want to add: "))

location = eval(input("Enter what location you want the note to be placed: "))

midiSong.insert(location - 1, noteAdded)

numNotesAdded -= 1

if len(restNoteDurationHolder) == 0:

e = Events(freq=EventSeq(midiToHz(midiSong), occurrences = 0), beat=duration).play()

else:

while numNotesAddedHolder > 0:

restNoteDuration.append(0)

numNotesAddedHolder -= 1

counter = 0

while counter < len(midiSong):

e = Events(freq=EventSeq(midiToHz([midiSong[counter]]), occurrences = 0), beat=duration).play()

sleep(restNoteDuration[counter])

counter += 1

if choice2 == 2:

midiSongHolder2 = midiSong[:]

durationHolder2 = duration

restNoteDurationHolder = restNoteDuration[:]

location = eval(input("Enter what location you want a specfic note to be removed: "))

del midiSong[location - 1]

if len(restNoteDurationHolder) == 0:

e = Events(freq=EventSeq(midiToHz(midiSong), occurrences = 0), beat=duration).play()

else:

restNoteDuration.pop()

counter = 0

while counter < len(midiSong):

e = Events(freq=EventSeq(midiToHz([midiSong[counter]]), occurrences = 0), beat=duration).play()

sleep(restNoteDuration[counter])

counter += 1

if choice2 == 3:

#midiDuration = []

#midiDurationAdded = len(midiSong)

midiSongHolder2 = midiSong[:]

durationHolder2 = duration

restNoteDurationHolder = restNoteDuration[:]

duration = eval(input("Enter the desired duration you want for all beats of the song: "))

if len(restNoteDurationHolder) == 0:

e = Events(freq=EventSeq(midiToHz(midiSong), occurrences = 0), beat=duration).play()

else:

counter = 0

while counter < len(midiSong):

e = Events(freq=EventSeq(midiToHz([midiSong[counter]]), occurrences = 0), beat=duration).play()

sleep(restNoteDuration[counter])

counter += 1

if choice2 == 4:

midiSongHolder2 = midiSong[:]

durationHolder2 = duration

restNoteDurationHolder = restNoteDuration[:]

while True:

try:

octaveNum = int(input("Enter how many octaves you want each note's octave to be increased: "))

except ValueError:

print("Please enter the correct input again!")

continue

else:

break

counter = 0

while counter < len(midiSong):

midiSong[counter] += (octaveNum \* 12)

counter += 1

if len(restNoteDurationHolder) == 0:

e = Events(freq=EventSeq(midiToHz(midiSong), occurrences = 0), beat=duration).play()

else:

counter = 0

while counter < len(midiSong):

e = Events(freq=EventSeq(midiToHz([midiSong[counter]]), occurrences = 0), beat=duration).play()

sleep(restNoteDuration[counter])

counter += 1

if choice2 == 5:

midiSongHolder2 = midiSong[:]

durationHolder2 = duration

restNoteDurationHolder = restNoteDuration[:]

while True:

try:

octaveNum = int(input("Enter how many octaves you want each note's octave to be decreased: "))

except ValueError:

print("Please enter the correct input again!")

continue

else:

break

counter = 0

while counter < len(midiSong):

midiSong[counter] -= (octaveNum \* 12)

counter += 1

if len(restNoteDurationHolder) == 0:

e = Events(freq=EventSeq(midiToHz(midiSong), occurrences = 0), beat=duration).play()

else:

counter = 0

while counter < len(midiSong):

e = Events(freq=EventSeq(midiToHz([midiSong[counter]]), occurrences = 0), beat=duration).play()

sleep(restNoteDuration[counter])

counter += 1

if choice2 == 6:

midiSongHolder2 = midiSong[:]

durationHolder2 = duration

restNoteDurationHolder = restNoteDuration[:]

restDurationAdded = len(midiSong) - 1

counter = 0

#while restDurationAdded > 0:

while counter < restDurationAdded:

durationAdded = eval(input("Enter the desired duration one at a time you want for each rest note for the song (Iteration " + str(counter + 1) + "): "))

restNoteDuration.append(durationAdded)

counter += 1

#restDurationAdded -= 1

restNoteDuration.append(0)

counter = 0

while counter < len(midiSong):

e = Events(freq=EventSeq(midiToHz([midiSong[counter]]), occurrences = 0), beat=duration).play()

sleep(restNoteDuration[counter])

counter += 1

if choice2 == 7:

midiSong = midiSongHolder[:]

duration = durationHolder

e = Events(freq=EventSeq(midiToHz(midiSong), occurrences = 0), beat=duration).play()

if choice2 == 8:

midiSong = midiSongHolder2[:]

duration = durationHolder2

restNoteDuration = restNoteDurationHolder[:]

if len(restNoteDurationHolder) == 0:

e = Events(freq=EventSeq(midiToHz(midiSong), occurrences = 0), beat=duration).play()

else:

counter = 0

while counter < len(midiSong):

e = Events(freq=EventSeq(midiToHz([midiSong[counter]]), occurrences = 0), beat=duration).play()

sleep(restNoteDuration[counter])

counter += 1

#midiSong = midiSongHolder[:]

if i == 0:

e.stop()

s.stop()