2018/19

Student Name: Ziwei.Lin

Student ID: 201298343

Project Title: COMP390

Computer Creativity III - traditional "automated" music composition methods (Serialism)

Supervisor: Paul.E.Dunne

DEPARTMENT OF COMPUTER SCIENCE

The University of Liverpool, Liverpool L69 3BX

Abstract

The history of music has seen significant development in people’s life, notions, imagination, and creativity. Until the 1990s, Arnold Schoenberg, with the form of the Second Viennese School, began to banish the tonality of music and devised the 12-tone technique (also called serialism) for atonal music composition, which is followed by Milton Babbitt who continued the research of serialism and devoted to the serialism of rhythm and volume. However, atonal music is generally not harmonious at all and cannot be accepted by the public. Therefore, this project aims to introduce this serialism notion and produce a system to implement the serialism of melody, rhythm and volume to provide a platform for the user to appreciate the effects of serial music.

This dissertation will first give the general introduction of the whole project, the background information for the emergence of atonal music, some rudimentary music theories, research about the details of algorithm design, MIDI, Java Sound API and the music sheet engraving software Lilypond, as well as the functional and non-functional requirements will be provided next. While the revised design documentation which contains the detailed design for GUI, Java class, algorithms (using pseudocode), followed by the evaluation methodology. The realisation of each component is the central part that describes the process and the encountered problems with their current solutions of the project design. Since the success of this profession-oriented software depends on the correctness of the algorithms instead of the harmony of the music composed, the evaluation section will only cover the evaluation for the non-functional requirement such as the reliability. A conclusion for learning points, professional issues, as well as the bibliography list will be included at the end.

Contents

Abstract 1
Contents 2
1. Introduction 4
1.1 Problem, Objectives and Scope 4
1.2 Challenges 4
1.3 Solutions and Effectiveness 4
2. Background 5
2.1 The Birth of Modern Music 5
2.2 Existing Solutions and Comparison 6
2.3 Music Theory 6
2.4 Reading and Research 7
2.5 Project Requirements 12
3. Data Required 14
4. Design 15
4.1 Summary of Proposal 15
4.1.1 Backgrounds, Aims and Objectives 15
4.1.2 Recall the Aim 15
4.1.3 Objectives 15
4.1.4 Changes to the Design Documentation 15
4.2 Project Design 16
4.2.1 Description of Anticipated Components 16
4.2.2 Description of the Algorithms 16
4.2.3 Design of the Java Classes 28
4.2.4 Design of the User Interfaces 29
4.2.5 Description of the Evaluation of the System 31
4.2.6 Ethical Use of Data 31
4.3 Review against Plan 32
5. Realisation 33
5.1.1 Screenshots and Descriptions 33
5.1.2 Problems Encountered 38

5.1.3 Testing for GUI 38
5.2 Implementation of Algorithms 41
5.2.1 Melody Serialism 41
5.2.2 Rhythm Serialism 41
5.2.3 Volume Serialism 41
5.2.4 Testing of Algorithms 43
5.3 Implementation of MIDI 46
5.3.1 Problems Encountered 46
5.3.2 Testing for MIDI 47
5.4 Implementation of Music Sheet Production 48
5.4.1 Problems Encountered 48
5.4.2 Testing for Sheet Generation 48
5.5 Implementation of MIDI to MP3 Conversion 49
6. Evaluation 50
6.1 Criteria 50
6.2 Evaluation of Results 50
6.3 Evaluation of Project Strength and Weakness 53
7. Learning Points 54
8. Professional Issues 56
8.1 Issues Related to Code of Practice 56
8.2 Issues Related to Code of Conduct 56
9. Bibliography 57
Appendix I – Detail Screenshots 59
Appendix II - Full Code Listing 69
Appendix III - The Original Design Documentation 69

1. Introduction

1.1 Problem, Objectives and Scope

Automated music composition is a significant field within AI concern. The automated composition is being on stage gradually since the various applications of music have occupied people’s life from the pure sound effects of video games to the large-scaled live symphony. Some AI approaches for music composing may require the providing of a mass of music samples for the computer to analyse and generate algorithms for automated composing. Others may offer existing algorithms for the computer to compose music directly. This project aims to devise and implement algorithms for serial music composition by analysing existing concepts.

Most accessible music pieces are chord-based which differ from serial music as the latter has no tone. Developed in the early 20th-century, serial music is best known as Schoenberg’s 12-tone music but is far more than that. In order to simplify the exploration and research of serial music, a visualised way is necessary to introduce the composing mechanism and for users to appreciate the effects. Therefore, the purpose of this project is to develop a PC-based composition application software which generates a small piece of Schoenberg’s 12-tone music by performing some algorithms on the original user-entered 12 tones’ series, generating new tone-series with Babbitt’s serialism of rhythm and volume, output new pieces of music as well as their music sheets.

The target audience should have ordinary visual and auditory sense since the software intends to introduce serialism to the public by performing serial music and PDF music sheet with log data for professionals to analyse.

1.2 Challenges

The main challenges for this project lie in:

- ✧ The understanding of Babbitt’s Duration Row System and Time Point System for rhythm serialism and volume serialism by just reading papers (tutorial material could barely be found).
- ✧ MIDI programming and Java Sound Package are brand-new technologies.
- ✧ There was no idea for sheet music generation at first.
- ✧ There was no idea for MIDI file to audio file conversion.

1.3 Solutions and Effectiveness

- ◆ No effective solution for concept understanding, only searching and carefully reading more papers online.
- ◆ Search for online tutorials, read Standard MIDI-File Format Specification and test the functions by creating simple projects are effective and safe.

- Thanks to the recommendation from the supervisor who suggested a free text-based sheet-generating software Lilypond which has built-in functions to convert midi file to lilypond file and then to pdf sheet. The solution is quite effective, and the produced sheet is of high quality.
- No feasible solution has been found for converting sample file to an audio file.

2. Background

Following the introduction section, this chapter will provide a background related to the main tasks of the project. This will include a brief history of modern music, especially the concepts of serialism together with the description of existing solutions for each serialism step and some fundamental music theories. Apart from these core intentions of this project (to implement serialism algorithms), introductions for other essential components: MIDI System, Java Sound Package and the sheet music producer Lilypond as well as further reading about understanding Schoenberg’s and Babbitt’s theories will be wrapped into the “Reading and Research” section, followed by the statements of project requirement.

2.1 The Birth of Modern Music

✧ Brief History of Western Music [1]

From around the year 500, the history of western music has experienced three periods: The Early period, the Common-practice period and the 20th century and early 21st century period. The Early period is a broad musical era, and as Michael Kennedy pointed out, the most initial music period “times up to and including” the Renaissance eras. While the Common-practice period is regarded as the era for the formation and declination of the tonal system, roughly from 1650 to 1900, it covers Baroque, Classical, Romantic and Impressionist periods. After that, the 20th century period has witnessed a high diversity of music with the emergence of various styles such as modernism, impressionism, post-romanticism and expressionism. The 21st century retained some traits of the previous centuries, and bears new music genres such as Jazz, pop and rock and dives into the practice of combining classical music with multimedia.

✧ Background about Serialism

As one of the modernist movements in the 20th century, expressionism was first used in the field of poetry and painting and was introduced to describe Schoenberg’s music in 1918. The essence of this term of art is to distort the physical reality and evoke spirits and ideas by expressing emotional effects. The representative composers Arnold Schoenberg, Anton Webern and Alban Berg who formed the Second Viennese School carried out the expressionist tenet and aimed to banish the harmony and affirmative of traditional music and endow their music pieces full of dissonance with an unconscious trait [2]. The typical expressionist method is serialism.

In contrast to the techniques used to create tonal music that has a primary pitch and follows harmonic melodic lines, the general idea of serialism is to abandon this fundamental pitch and repeat the serial pattern [3]. While Milton Babbitt intended to provide the composers with a mechanism to implement all possible combinations of tones which can be applied to different music components other than melody to enable “a maximal diversity” as Mead pointed out [4], such as rhythm, timbres and dynamics (volume) serialism.

2.2 Existing Solutions and Comparison

✧ For Melody Serialism:

The 12-Tone Music which is developed by Arnold Schoenberg is the most typical implementation of melody serialism. The main concept of 12-tone music is to make all 12 notes equal (no tone in a chromatic scale is more important than any other). Schoenberg’s approach is to govern linear ordering of a given 12 pitches of the chromatic scale (by constructing a 12*12 matrix), use each 12 notes ordering pattern serially to make a musical piece, which means the 12 notes should be used non-repeatedly before the next 12-tone series, and notes in one 12-tone row/column can be grouped and played at one time [5].

✧ For Rhythm Serialism: [6]

Milton Babbitt, an American composer, expanded Schoenberg’s concept of 12-tone music and developed Duration Row System and Time Point System for rhythm serialism. As an analogue of the construction of 12-tone matrix, these two system takes an original duration row of three or six duration values and develop it by transmitting each value at an interval of three or six (and conducting a subtraction by 12 if the value exceeds 12) to generate the next duration row and form a list at last.

Comparison: In Duration Row System, the above list is called duration list which stores the actual durations for each note, while in Time Point System, the list becomes a time point list which stores the time-points (or attack points [7]) - the starting points for each tone, say, the duration between each note and its previous bar line[6].

2.3 Music Theory

✧ **The Pitch Class**

A pitch is a tone with its frequencies, and each pitch is given an alphabetical name to distinguish, which does not imply octave equivalence (means C1 is different from C2). While a pitch class is octave equivalent (means C1 is regard equally to C2, C3…). [8]

✧ **The Chromatic Scale**

The chromatic scale consists of twelve notes and each note is semitone away from its adjacent notes [9]. Using the pitch class names to list the twelve tones: C C# D D# E F F# G G# A A# B C C# D D# E …

✧ **Tonality**

Tonal Space: Tonality is an organisation of pitches which is thought to have an internal logic derived from nature. The audible pitch spectrum is divided into octaves with twelve semitones each chromatic scale in western music culture. [10]

Tone relationships: In tonal music, the twelve semitones are related to each other in a hierarchical way, which means some notes are made more important than the others, recalling that most music pieces are named with a scale name like "Wolfgang Amadeus Mozart: Rondo in ***D major***", "Frédéric Chopin: Waltz No. 7 in ***C sharp minor***, Op.64 No.2". [10]

Here is an example of the C Major scale and how its corresponded melody looks like:

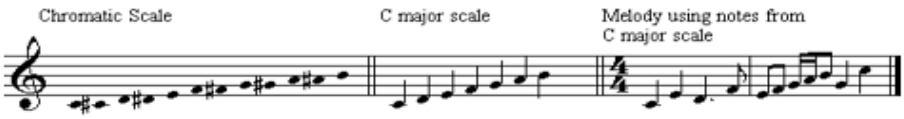


Figure 1: THE CHORDS C MAJOR [10]

2.4 Reading and Research

✧ **For how to construct the 12-tone matrix:**

CREATING A 12 TONE MATRIX **[11]**

This material clearly describes the steps and mechanism to construct a 12-tone matrix, which is the core part of the 12-tone theory and helped me to generate an algorithm for 12-tone matrix construction.

✧ **For how to generate melody patterns using the 12-tone matrix:**

THE SERIALISM OF MILTON BABBITT **[6]**

This material describes many aspects of Milton Babbitt's research and contribution in the field of serialism, including how to apply the series from 12-tone matrix to a real piece of music. The picture below is an example from Schoenberg's Violin Concerto, Op36 which represent that *as long as the notes have appeared in the order of the given series (no octave difference), the duration can be random, and one can even combine two or more notes as a chord and play*

them simultaneously (See the blue part, the series is D C# G# C G F, while C# G# C [D♭ A♭ C] are played together).

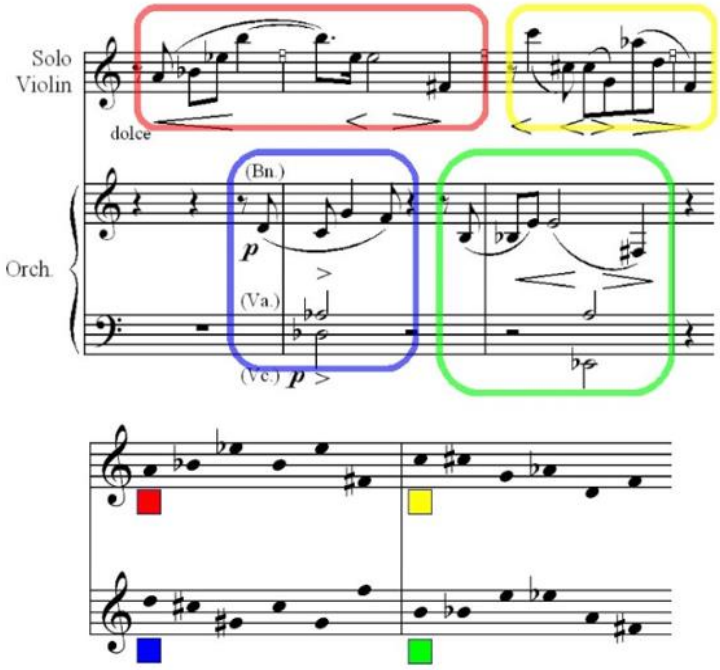


Figure 2 Opening of Schoenberg’s Violin Concerto, Op. 36 – underlying [6]

✧ **About Rhythms**

The World’s Most Popular Rhythm (Video) **[12]**

One of the world’s popular rhythms are called Son Clave is described in this video.

Son Clave:

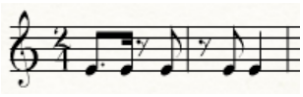


Figure 3 Son Clave Rhythm

In order to make the 12-tone melody change more rhythmic instead of rigid beeps, Son Clave will be applied to the original 12-tone melody piece.

✧ **For the algorithm of rhythm alternation:**

Twelve-Tone Rhythmic Structure and the Electronic Medium 1962 [13]

Time-Point Sets and Meter [14]

THE SERIALISM OF MILTON BABBITT [6]

These materials introduce the way to construct a rhythmic serial system technically and help me with my rhythm alternation algorithm. Babbitt has developed two ways to build the rhythm: The Duration Row System and The Time-Point System.

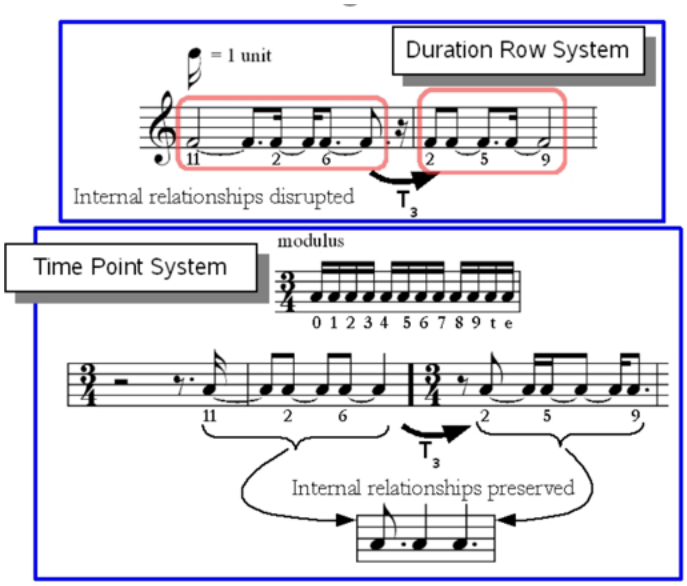


Figure 4 Comparison for the Duration Row System and the Time Point System [6]

Babbitt’s rhythm system first uses the three-note rhythm to construct the duration rows (there are three duration values in each row with an interval of three between the same position of the adjacent rows). Using a 16th note as one duration unit, in the duration row system, the numbers in these duration rows stand for the actual duration of each note. While in the time point system, these numbers are the time points of the notes, which means the duration between the previous bar line and the current note. For further description, please see 4.2.2 Description of the Algorithms.

✧ **Musical Instrument Digital Interface (MIDI)**

Standard MIDI-File Format Spec.1.1 Updated **[15]**

An Introduction to MIDI **[16]**

MIDI Tutorial **[17]**

According to “An Introduction to MIDI” [16], Musical Instrument Digital Interface, better known by the acronym MIDI, is a system providing a communication approach between electronic musical instruments and computer so that the instruction messages can be sent. Through MIDI messages which deliver time sequences of instructions defined by the status bytes and data bytes, MIDI languages tell the synthesizer when and what key to press, instead of setting the actual sound itself.

Since the MIDI files use the form “VARIABLE-LENGTH QUANTITY” for the representation of some numbers as 7 bits per byte, these one-byte numbers, including the controller number, note number, velocity number, instruments number, will range from 0 to 127 [15]. The MIDI message could contain the above information as well as the channel number (a maximum of 16 channels could be handled by the MIDI protocol, which means up to 16 notes or different instruments can be played simultaneously) [17].

Some core numbers with their corresponding features are listed below [15]:

MIDI Controllers:

NOTE ON – 144; NOTE OFF – 128; ALL NOTES OFF – 123;

CHANGE DEFAULT INSTRUMENT – 192.

MIDI Notes:

Octave #	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Figure 5 Table of MIDI Note Numbers [15]

MIDI Instruments:

Acoustic Grand Piano – 1;

Violin – 41;

Flute – 74.

✧ **Java Sound API [18]**

As a low-level API for sound support on Java platform, Java Sound API provides a significant amount of control over sound operations for applications, and supports both midi data and digital audio using “javax.sound.midi” and “javax.sound.sampled”.

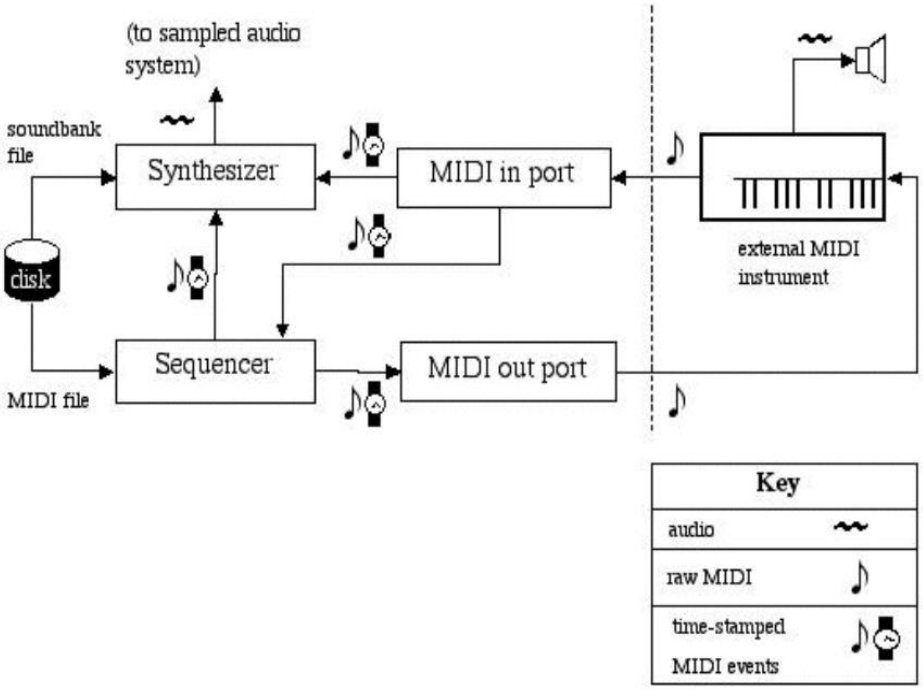


Figure 6 A Possible MIDI Configuration [18]

The above picture gives a possible MIDI configuration scenario of the Java Sound API: By loading a standard MIDI file from a disk, a software sequencer performs the MIDI file by sending the message to a synthesizer to play the notes received from the MIDI message. (The synthesizer reads a soundbank file for instrument sound emulation or uses an existing instrument to perform.) Additionally, time-stamped MIDI events should be transformed into row MIDI events through the MIDI out port to an external MIDI instrument, and the row MIDI events played by the external device should be transformed back to time-stamped MIDI events via the MIDI in port and performed by a sequencer or a synthesizer.

Different from MIDI format, javax.sound.sampled package operates digital audio signal which is a sound wave, a direct sound representation format as figure 6 illustrates, instead of the digital signal created by the MIDI system.

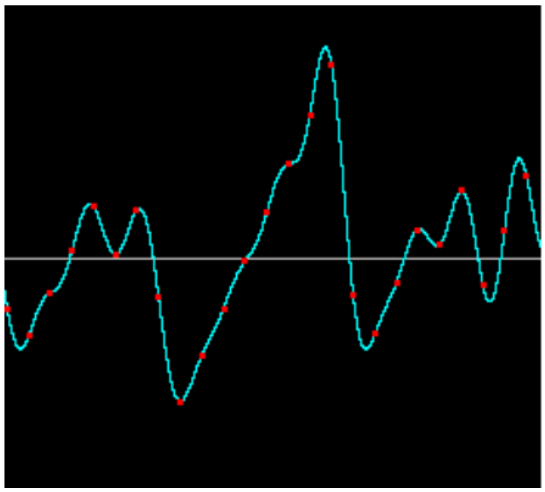


Figure 7 A Sampled Sound Wave [18]

✧ **Lilypond**

As an essential auxiliary software for the development of the “Serial Music Maker”, Lilypond is a program that aims to produce high-quality sheet music based on the best traditional style of classical music. [19] Different from most graphical music score producing programs that generate sheet music by dragging music notations directly, Lilypond requires the composer to type text – similar to programming code – and the software will compile the text to create high-quality sheet music. [20] Besides, by adding the path of Lilypond “bin” directory to the “Path” parameter of environment variables, several built-in functions can be called using command.exe, which is one of the key implementation methods utilized in “Serial Music Maker”.

2.5 Project Requirements

✧ **Functional Requirements**

Essential Features

- The User Interface should enable the user to enter the original 12-tone series using 12 buttons containing pitch names [C, C#, D, D#, E, F, F#, G, G#, A, A#, B] for the user to select.
- The User Interface should have four buttons to control the visibility of 4 views a “Welcome” view for greeting and user instructions;

  a “Melody” view for the original 12 pitches input and button for “Melody Change” and “Clear” for reset;

a “Rhythm” view for original three-note rhythm values input and buttons for “Duration Row Change”, “Time Point Change” as well as “Melody-Time Point Change”;

a “Volume” view for original three-volume values input and buttons for “Volume Change” and “Melody-Time Point-Volume Change”.

- The algorithm should

  enable the alteration of 12-tone series order (as the serial ordering of tones is the primary intention of Schoenberg) and implement melody generation;

  allow the rhythm alternation based on Babbitt’s Duration Row System and Time Point System using existing melody (for instance, Bach’s Minuet), and apply Time Point System on the 12-tone song generated previously;

  enable the volume alternation using Bach’s Minuet, and apply volume serialism on Time Point 12-tone melody.

- The output music piece should at least be a standard MIDI file (*.mid).
- Output the music score sheet of the new music piece.

Desirable Features

✧ Convert *.mid files to standard sound output files (*.mp3).

✧ **Non-functional Requirements**

- Organizational requirements: the software shall use Java as the programming language and be developed under Windows 10 system which has preinstalled Java 8 JDK, software NetBeans 8.2, as well as Lilypond 2.19.83 with configurated environment variables (See 2.4 Reading and Research – Lilypond for detailed instructions).
- Space requirements: under the consideration of the size of the *.jar file, instruction documentation and the *.mid/*.ly/*.pdf files that will be generated during program running, the system should have at least 2 Megabytes space to store the software apart from the size of the preinstalled Lilypond.
- Reliability requirements: the rate of failure occurrence should not exceed 10%.
- Performance: The Speed and the average CPU occupation rate should not exceed
- Ease of use: the software should catch almost all possible exceptions and gives appropriate instructions for the user to correct their operations.

3. Data Required

The data required in this project is the user input which includes the original 12-tone series (by pressing buttons on the graphical interface), the original three-note duration values and the initial three-note volume values (both by entering numbers in the text fields of the interface). All data comes from the software developer’s or tester’s input.

✧ **Ethical Use of Data**

This project does not involve any personal data or human participant.

4. Design

4.1 Summary of Proposal

4.1.1 Backgrounds, Aims and Objectives

The purpose of this design documentation is to introduce the project proposal, document the compositions of this project, including the design of the intended software and the evaluation, and review against the plan.

4.1.2 Recall the Aim

This project aims to develop a composition application software named “12 Tone Music Maker” for musician or researcher to generate a small piece of serial music based on Schoenberg’s 12-tone theory.

4.1.3 Objectives

- To create a computer composition application software using Java.
- To create a graphical user interface for the software.
- To implement algorithms for melody, rhythm and volume alternation to compose new music pieces based on Schoenberg’s 12-tone theory.
- To realise MIDI output of the new music pieces.
- To realise Music Sheet creation for the new music pieces.
- To convert MIDI file to audio (WAV or MP3) file.

4.1.4 Changes to the Design Documentation

1. *Add music sheet creation and MIDI file format transform to “Objectives” as efforts have been made to implement them which enrich the contents of the project.*
2. *Add four views’ control in essential features part to make the description more detail as this addresses the structure of the software which helps understand the design.*
3. *Put rhythm serialism, volume serialism and music sheet generation to essential features since these features have been fully implemented which are proved to be feasible.*
4. *The descriptions of essential features and desirable features are included in the background information (See 2.5 Project Requirements), so this part is removed from the design section.*
5. *The “Summary of Current Research and Analysis” is merged into the background (See 2.5 Reading and Research) so no restatement will be presented here.*

4.2 Project Design

4.2.1 Description of Anticipated Components

- ✧ User Interface – Theoretically, the user interface is everything which enables the interaction between the user and the computer system. In this project, the user interface contains “12-Tone Music Maker” software front-end view, keyboard to enter, mouse to click buttons and audio output devices.
- ✧ Algorithm – The algorithms implemented by the back-end process of the software is a crucial point in this project. Using Java programming, methods for generating music patterns, combining patterns, distributing time-points values to each sound, distributing volume values to each pattern or single sound will be developed.
- ✧ MIDI Output – The new music pieces is intended to output as .midi files or .mp3 files, which needs the help of Java MIDI programming technology using Java Sound API.
- ✧ Music Sheet Output – *(With the successful installation and configuration of the Lilypond, this additional component can finally light up the project.)* In the command.exe run lilypond operation(midi2ly) to convert *.midi to *.ly lilypond file, then edit and convert the lilypond file to PDF music sheet.

4.2.2 Description of the Algorithms

- ➢ Melody Generation Algorithm:
  - Purpose:
    To generate music patterns and combine them into a small music piece.
  - Problems:
    Generate 12-tone matrix from the original user-entered series;

    Use patterns in the 12-tone matrix to make a music segment;

    Set durations for each sound (in Son Clave pattern for example).
  - Existing Algorithms and Evaluation:
    The existing algorithm for serial melody generation is to construct the 12-tone matrix which was developed by Schoenberg.

    As has been described in CREATING A 12-TONE MATRIX [11], the algorithm is: for a given original 12-tone series, we first generate its Clock Map that matches the first tone of the original series with number 0, and matches the rest of the tones in the other of chromatic scale (C C# D D# E F F# G G# A A# B) with number 1 2 3 4 5 6 7 8 9 10 11. Then,

we place the original series in row 0 of the 12*12 matrix. Next, we can fill out the first column of the matrix by the formula:

$$\text{Matrix}[i][0] = 12 - \text{Matrix}[0][i]$$

Now we can fill out the rest cells of the matrix by the formula:

$$\text{Matrix}[x][y] = \begin{cases} \text{Matrix}[x][0] + \text{Matrix}[0][y] & (\text{Matrix}[x][0]+\text{Matrix}[0][y]<12) \\ \text{Matrix}[x][0] + \text{Matrix}[0][y] - 12 & (\text{Otherwise}) \end{cases}$$

Here is the test program:

```
public class Matrix {

    public int[][] matrix;

    public Matrix(int[] list) {
        if(list.length!=12) {
            System.out.println("The original pitch row must contain 12 tones!");
        }
        else {
            this.matrix = new int[12][12];
            matrix[0]=list;
            for(int i=1; i<12; i++) {
                this.matrix[i][0]=12-this.matrix[0][i];
            }
            for(int y=1; y<12; y++) {
                for(int x=1; x<12; x++) {
                    if(this.matrix[y][0]+this.matrix[0][x]<12) {
                        this.matrix[y][x]=this.matrix[y][0]+this.matrix[0][x];
                    }
                    if(this.matrix[y][0]+this.matrix[0][x]>=12) {
                        this.matrix[y][x]=this.matrix[y][0]+this.matrix[0][x]-12;
                    }
                }
            }

        }
    }
```

Figure 8 Screenshot of 12-tone matrix building testing

Enter the original series: `D C F# F C# D# A B G# A# G E`

The output is:

```
The Clock Map:
Key = C#, Value = 11
Key = A, Value = 7
Key = A#, Value = 8
Key = B, Value = 9
Key = C, Value = 10
Key = D, Value = 0
Key = E, Value = 2
Key = F, Value = 3
Key = G, Value = 5
Key = G#, Value = 6
Key = F#, Value = 4
Key = D#, Value = 1
```

Figure 9 Screenshot of Pitch Clock Map

This Clock map each tone with an integer number, which can be transferred to:

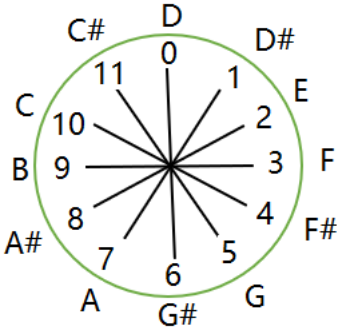


Figure 10 Screenshot of Pitch Clock

The integer representation of the original pitch class and the whole 12-tone matrix is:

```
The original row pitch class:
0.10.4.3.11.1.7.9.6.8.5.2.
The Matrix:
0,10,4,3,11,1,7,9,6,8,5,2,
2,0,6,5,1,3,9,11,8,10,7,4,
8,6,0,11,7,9,3,5,2,4,1,10,
9,7,1,0,8,10,4,6,3,5,2,11,
1,11,5,4,0,2,8,10,7,9,6,3,
11,9,3,2,10,0,6,8,5,7,4,1,
5,3,9,8,4,6,0,2,11,1,10,7,
3,1,7,6,2,4,10,0,9,11,8,5,
6,4,10,9,5,7,1,3,0,2,11,8,
4,2,8,7,3,5,11,1,10,0,9,6,
7,5,11,10,6,8,2,4,1,3,0,9,
10,8,2,1,9,11,5,7,4,6,3,0,
```

Figure 11 Screenshot of the Matrix

This matrix generation follows Schoenberg’s approach, and the test program has correctly implemented this approach.

- Approach to be Used to Solve the Problem

  Algorithm to generate a melody from the 12-tone matrix:

  Assume we want to make a small piece using only one row and one column with their Prime (left to right), Retrograde (right to left), Inversion (up to down) and Inversion Retrograde (down to up) patterns plus one duplication.

  Now the series is P, R, I, IR, P, R, I, IR with 12*8 = 96 notes in total. We can divide each of the 12 tones into segments of 1 to 4 notes (as any random chord with 5 or above notes will be a “mess” and hard to distinguish, so we limit it to 4), each segment will make a chord with notes being played simultaneously. We can do this by generating random integer numbers.

  Then, the Son Clave rhythm can be applied on this 96 notes’ piece. Take the sixteenth note as a durational unit, and the Son Clave can be represented as 3 1 2 2 2 2 4 durations (- is the duration for a pause).

  Here is the pseudocode:

Melody_Generation (Matrix m)

```
//Randomly choose a row and a column.
r ← random (0 to 11)
c ← random (0 to 11)
Let new array rArr ← m[r]
Let cArr[12] be a new array
For i from 0 to 11
    cArr[i] ← m[i][c]
End Loop
//Generate array for Prime, Retrograde, Inversion and Retrograde Inversion
Let new array prime ← rArr
Let new array inver ← cArr
Let retro[12] be a new array
For i from 0 to 11
    For j from 11 to 0
        retro[i] ← rArr[j]
    End Loop
End Loop
Let reIn[12] be a new array
For i from 0 to 11
    For j from 11 to 0
        reIn[i] ← cArr[j]
    End Loop
End Loop
//Generate the whole list of tune series
```

```
Let new List wholeList ← (prime + inver + retro + reIn)*2
/*details are ignored here*/
//Divide prime, inver, retro, reIn into segments
Let segments be a list of integers
//List of the number of notes in each segment
Let n ← 12 //The rest note number in an array
For i from 1 to 8 //There are 8 twelve-tone series patterns, so loop 8 times
    While (n > 0)
        Let x ← random (1 to 4)
        n ← n – x;
        If (n >= 0)
            segments.add(x)
        Else
            n ← n + x
    End Loop
End Loop
//Construct a list of sounds each of which is a list of notes
Let allList be a list of ArrayList
For i from 0 to segments.size() - 1
    Let ArrayList list ← []
    For j from 0 to segments.get(i) - 1
        list.add(wholeList.get(j))
    End Loop
    wholeList.removeRange(0, segments.get(i))
    allList.add(list)
```

```
End Loop

//Apply Son Clave (3122224) to the series in allList

/*Add pause as empty ArrayList [] to allList. The pause in Son Clave is in the 3rd, 5th, 10th, 12th, 17th,… position, they divide the sound as segments with 2, 1, 4, 1, 4, 1, 4,…sounds.*/

allList.add(2, [])

Let x ← 3

Let count ← 0 /* If count is odd, the next pause position will jump 2 indexes, otherwise jump 5 indexes*/

While (x < allList.size())

    count ← count + 1

    If (count%2==1) then

        x ← x + 2

    Else if (count%2==0) then

        x ← x + 5

    End If

    allList.add(x, [])

End Loop
```

Now we can match the durations in the order of 3 1 2 2 2 2 4 3 1 2 2 2 4… to the sound (ArrayList) in the allList when implementing the MIDI output.

- Methods for Analysing the Algorithm

  By outputting all the middle results generated in the above Melody_Generation Algorithm, such as the rArr, cArr, prime, retro, inver, reIn, wholeList, segments and allList (the note numbers together with their pitch class names), the correctness of the algorithm can be easily analysed.

➢ Serialised Rhythm Algorithm:

- Purpose:

  To generate a serialised rhythm using Babbitt’s method.

- Problems:

  Use the user entered “three-note rhythm” to conduct rhythm serialisation using “Duration Row System” or “Time Point System”.

- Existing Algorithms and Evaluation:

  Only some descriptions of Babbitt’s rhythm theories can be found in THE SERIALISM OF MILTON BABBITT [6].

- Approach to be Used to Solve the Problem:

  Similar to the durations list in the Melody_Generation Algorithm, here, a durations list can also be constructed by taking in the original duration row first and do T3 operation (add each number by 3, and extract by 12 if it is bigger than 12) recursively to construct the whole durations list and assign them to each sound. Again, we set the 16^{th} note as the durational unit.

  Pseudocode: *(Changes made in this part modify some errors and transform some vague concepts into detailed realisable pseudocode.)*

  Duration_Row (Array tridur, int len)

  /*tridur: tri-rhythm, the user input original row, integer len is the size of the Minuet Note List

  //Construct durations list

  Let durations be a new List of Arrays of three integers

  For i from 0 to len/3

  Let tri1 be an array of 3 integer

```
        Copy tridur to tri1
        If (tridur[0] + tridur[1] + tridur[2])%12 not equal to 0
            Let tri2 be an array of 4 integer
            /* tri2[3] is the duration for a pause to fill the 12 duration in a bar
            segment */
            Assign tridur[0] to tridur[2] to tri2[0] to tri2[2]
            Assign 12 - ((tridur[0] + tridur[1] + tridur[2]) % 12) to tri2[3]
            Add tri2 to durations
        Else Add tri1 to durations
        For j from 0 to 2
            tridur[j] = tridur[j] + 3;
            If tridur[j] > 12
                tridur[j] = tridur[j] - 12;
        End Loop
    End Loop
    //Construct minuetDurationRow – the melody list for minuet and pauses
    Let minuetDurationRow be the List of Lists of Minuet notes and pauses
    Let pause be an empty List
    Let size be the length of durations
    Add Minuet notes into minuetDurationRow
    Let index ← 0
    For i from 0 to size
        If the length of durations.get(i) is 4
        /*Add a pause at the fourth position of the duration array if the length of
        the current duration array is four*/
            Add 3 to index
```

If index <= size +1

Add pause to minuetDurationRow

Add 1 to index

Else Add 3 to index

End Loop

For “Duration Row System” approach, every number in the rhythm row stands for the exact duration of each sound. Each duration row has 3 durations, while after each row if there are still some durations left before the next bar-line, the left part will be a pause. We can map each sound duration with the number in each Array in the durations list and fill the rest of the bar-line with a pause.

For “Time Point System” approach, what is different is that the numbers in the duration row do not stand for the duration of each sound but the duration between the notes and its previous bar-line. As a result, the actual length of that sound is the modular 12 of the duration rows, which preserves the internal relationship of the original row (the time signature could be stable 3/4 or 6/8).

Duration_For_Time_Point(List<Array> durations)

//Construct the time point duration from “durations”

Let tpDuration be the List of Time Point duration arrays

Let tp be a new duration Array

Copy durations values into tp

//Convert List to Array for operation convenience

Let minuetTP be the new List of Minuet Note and pause

Let tempList be the List of Minuet note

Let index ← 0

While tempList is not empty

```
If tpDuration[index] == 12
//If the first duration is 12, add pause with length 12
    Add 12 to tpDuration
    Add pause to minuetTP
Else if index==0
//If first duration is not 12, just add the first note with first duration
    Add tp[index] to tpDuration
    Add pause to minuetTP
Else if tp[index + 1] <= tp[index]
    If (tp[index] <= tp[index - 1]
    //If previous duration > current duration > next duration, add pause with current duration and add current note with 12 - current duration
        Add tp[index] to tpDuration
        Add pause to minuetTP
        Let integer temp ← 12-tp[index]
        Add temp to tpDuration
        Add the first item of tempList to minuetTP
        Remove the first item of tempList
    Else
    //If the current duration is larger than both previous duration and next duration, Add the current note with current duration – previous duration
        Let integer temp1 ← tp[index] - tp[index-1]
        Add temp1 to tpDuration
        Add the first item of tempList to minuetTP
        Remove the first item of tempList
```

```
                If tempList is not Empty
                //If next note exist, Add a pause with 12 - tp[index] +
                tp[index+1] duration
                    Let temp2 ← 12 - tp[index] + tp[index+1]
                    Add temp2 to tpDuration
                    Add pause to minuetTP
                End If
            End If
        Else If tp[index + 1] > tp[index]
            If tp[index] <= tp[index - 1]
            //If current duration is smaller than both previous duration and next
            duration, add a pause with current duration, then add current note with
            tp[index + 1] - tp[index] duration.
                Add tp[index] to tpDuration
                Add pause to minuetTP
                Let temp ← tp[index + 1] - tp[index]
                Add temp to tpDuration
                Add the first item of tempList to minuetTP
                Remove the first item of tempList
                Else
                //If previous duration < current duration< next duration, add
                current note with tp[index] - tp[index - 1] duration
                    Let integer temp1 ← tp[index] - tp[index - 1]
                   Add temp1 to tpDuration
                    Add the first item of tempList to minuetTP
                    Remove the first item of tempList
```

End If

End If

Increase index by 1

End Loop

- Methods for Analysing the Algorithm

  Output all the results in the algorithm including all elements in durations in **Duration_Row Algorithm** and the tpDuration in **Duration_For_Time_Point Algorithm** and analyse the correctness.

➢ Serialised Volume Algorithm: (*With further research in the MIDI program and the theories of Milton Babbitt’s, this part is modified to a more convincing and concrete version.*)

Inspired by Mead’s “maximal diversity”, Milton Babbitt intended to develop a structural process for composers to avail oneself of all possible aggregation of music components, which includes the serialisation of the velocity of each note. The basic approach is to assign numbers to volume notations (fffff to ppppp), then, through serialising the numbers and apply the accordant volume to realise volume serialism. Consider that the “Standard MIDI-File Format” [11] specifies the velocity number is between 0 to 127 where 0 means the softest volume and 127 means the loudest, we can then assign 11 to 0 to the volume mark “fffff”, “ffff”, “fff”, “ff”, “f”, “mf”, “mp”, “p”, “pp”, “ppp”, “pppp”, “ppppp”, apply the duration row method to calculate the volume series and expend the number to 127 – 0 to promote the volume variation. (Unfortunately, the duration row approach is adopted instead of Babbitt’s volume serialism method as the pedagogical and understandable materials for the latter one could not be found.)

4.2.3 Design of the Java Classes

(The feedback for the original Specification and Design Document suggests that the design for the Java Class should be added to cover the design aspects fully.)

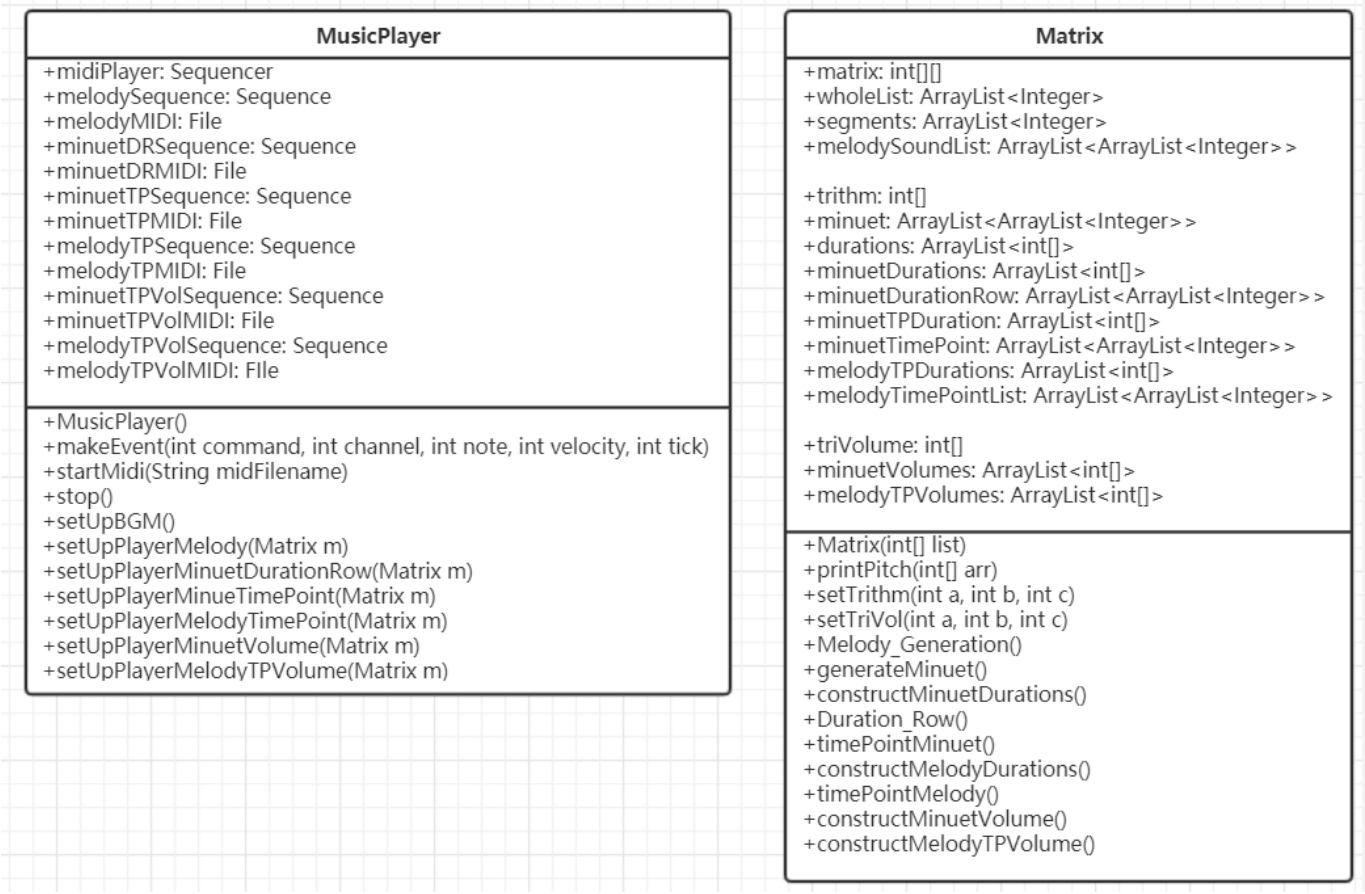


Figure 12 The Class Diagram

- ✧ The Matrix Class handles all the algorithm implementations
- ✧ The MusicPlayer Class manages all the MIDI programming operations
- ✧ Another class called Home_User which contains all Java Swing components and handles their actions is used to create the graphical user interface. (As too many elements and methods are included in this class, so this is omitted in the class diagram; however, the full code can be accessed in Appendix II.)

4.2.4 Design of the User Interfaces

Apart from all the hardware user interfaces, the software views are shown by the following storyboards:

Welcome

Figure 13 Design of the Welcome View

Melody

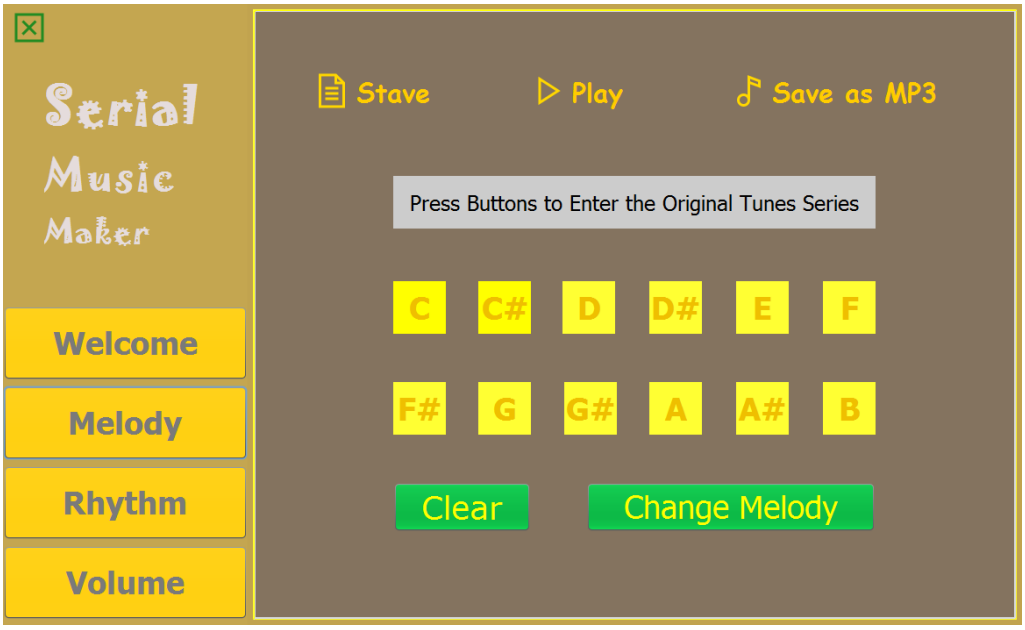


Figure 14 Design of the Melody View

Rhythm

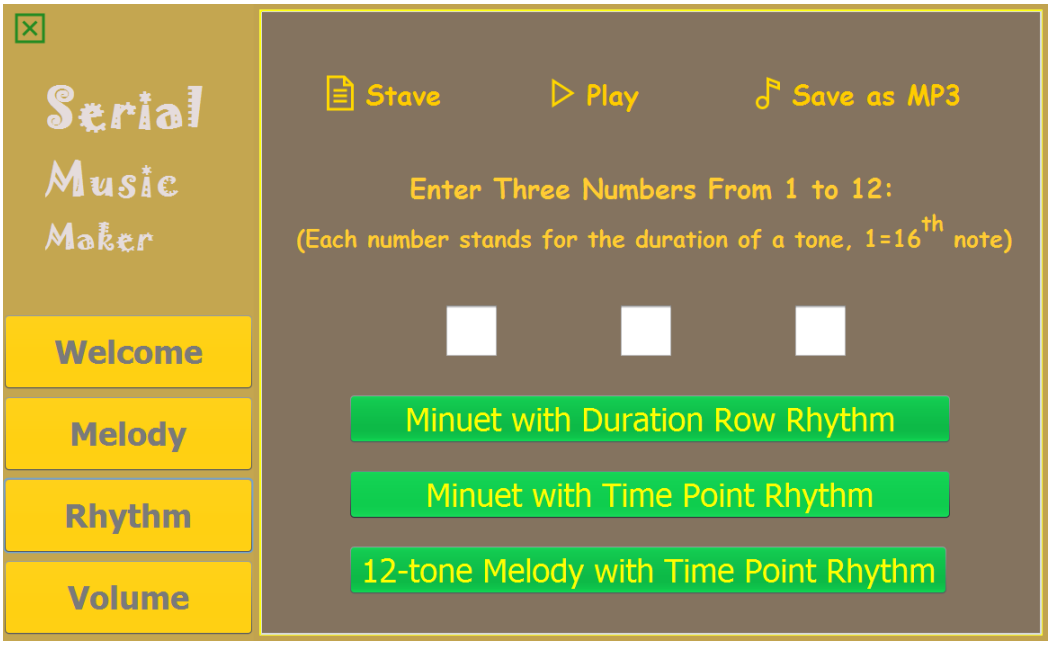


Figure 15 Design of the Rhythm View

Volume

Figure 16 Design of the Volume View

(These views which have been built successfully and are more feature-covered than the original designs are the final design for the graphical user interface.)

4.2.5 Description of the Evaluation of the System

(The evaluation will focus on non-functional features have been added as the initial design document did not consider this aspect. With respect to the high-technique for MIDI to audio format conversion, the MIDI to MP3 conversion may fail. Instead, the developer can focus on the reliability and compatibility of the software.)

- ✧ The Criteria Used to Evaluate

  Because the software is intended for academic or technical research, the most essential criterium is the correctness of the algorithms but not for public appreciation, and evidence has shown the dissonance character of this kind of music. Therefore, the evaluation criteria will only contain how many functions are satisfied and how well each function is satisfied, which relates to the correctness and efficiency of the algorithm, as well as the non-functional features like the software reliability and compatibility.

- ✧ The Approaches to Assess the Criteria

  Check the correctness of the algorithms by analysing both console outputs and the generated music staves. Check the number of functions that have been accomplished. (These will be done in the testing part of section 5 Realisation.)

- ✧ The Testing to be Done

  Check If: The components on the UI all work well

  The algorithms all work well

  The MIDI output can be implemented well

  The music sheets can be made

  The MIDI file can be converted to MP3

  The failure rate can be less than 10%

- ✧ The Expected Conclusion

  Most of the functions can be achieved, and improvement can be made if the MIDI file can be converted to MP3.

4.2.6 Ethical Use of Data

This project does not involve any human participant’s data.

4.3 Review against Plan

New Gantt Chart

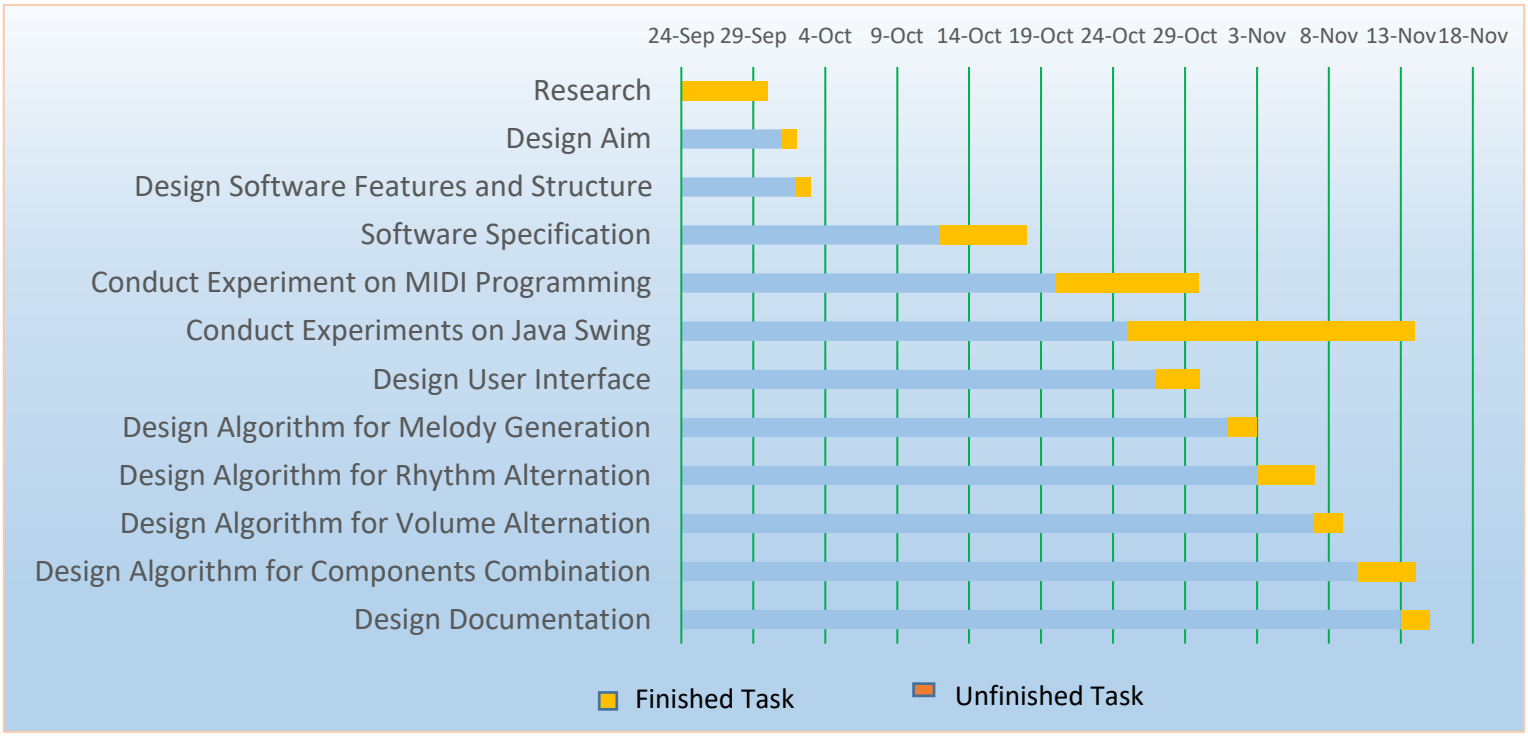


Figure 17 Gantt Chart for 24 Sep. 2018 to 18 Nov. 2018

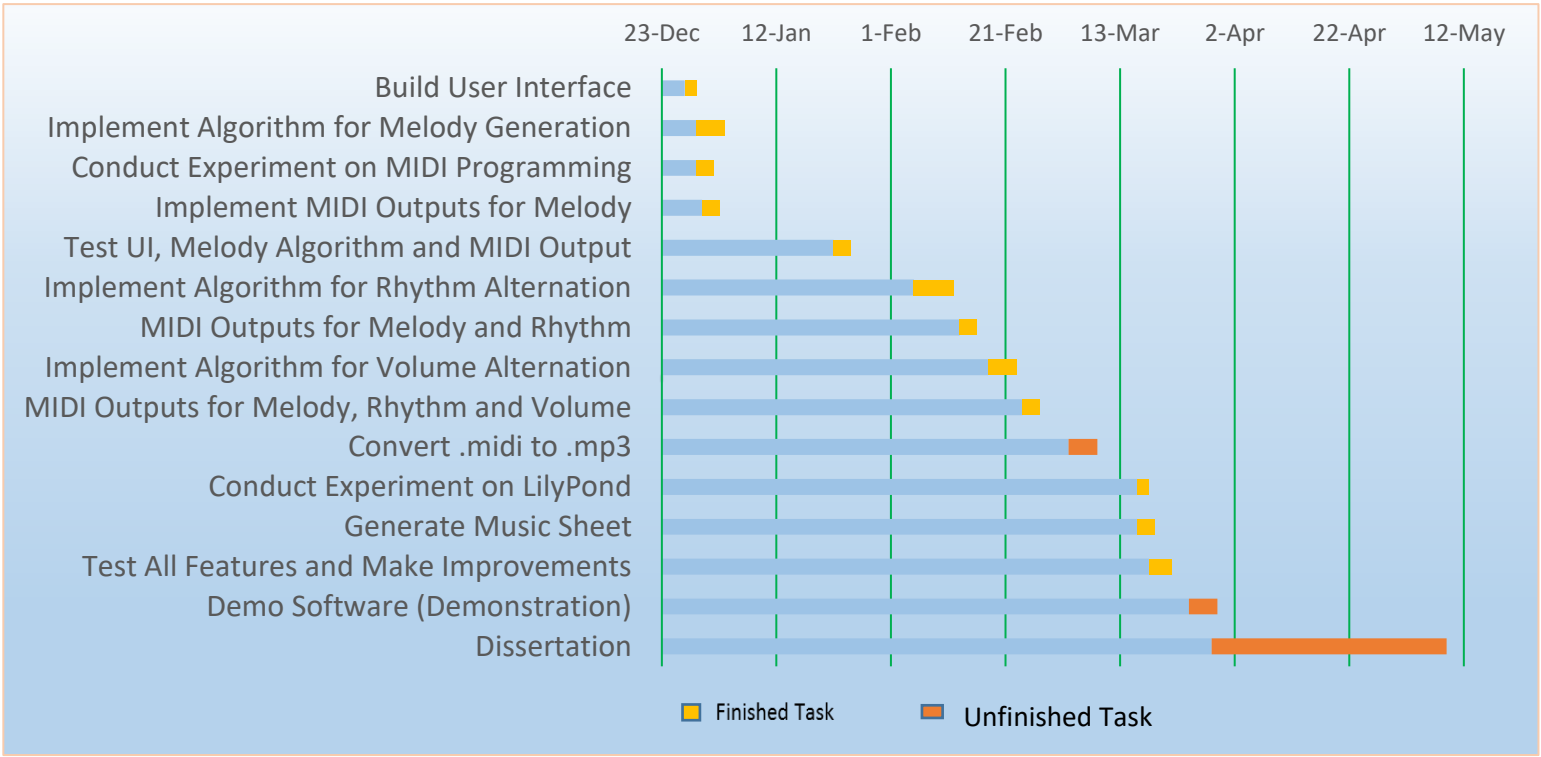


Figure 18 Gantt Chart for 26 Dec. 2018 to 9 May 2019

The comparison with the original design document:

Almost all expected tasks have been accomplished, except midi to mp3 conversion. As I directly use the melody generated by the Melody Generation Algorithm to conduct rhythm conversion, so no separate feature for melody-rhythm combination is needed.

5. Realisation

5.1 Implementation of User Interface

NetBeans IDE supports Swing GUI Building feature that enables GUI design by dragging Swing components from a palette and positioning them onto a canvas with automatic spacing and alignment – introduced by Apache NetBeans [21]. Based on this convenience, the “Serial Music Maker” is developed using NetBeans IDE 8.2 and use GUI Building tool for user interface building.

5.1.1 Screenshots and Descriptions

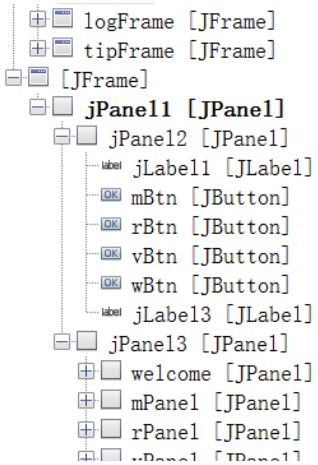


Figure 19 A screenshot of Swing GUI design structure

This screenshot shows the structure of the GUI of the software. The main JFrame is covered by a JPanel which contains a small JPanel, jPanel2 for the left sidebar, and another JPanel, jPanel3 for the four views. The buttons in the left sidebar take charge of the visibility of the four views in jPanel3. While logFrame is used for all data output to test the correctness of the algorithms and tipFrame is intended to give operating instructions for users.

Figure 20 A screenshot of the Welcome View

This picture demonstrates the welcome view of the “Serial Music Maker” which is the starting view of the software that gives the introduction of what the software is used for and the general instruction of how to manipulate the software. Additionally, background music (Bach’s Minuet) is played when the program stays in this view, stopped when switching to other views and started again when clicking on “Welcome” button to return to this view.

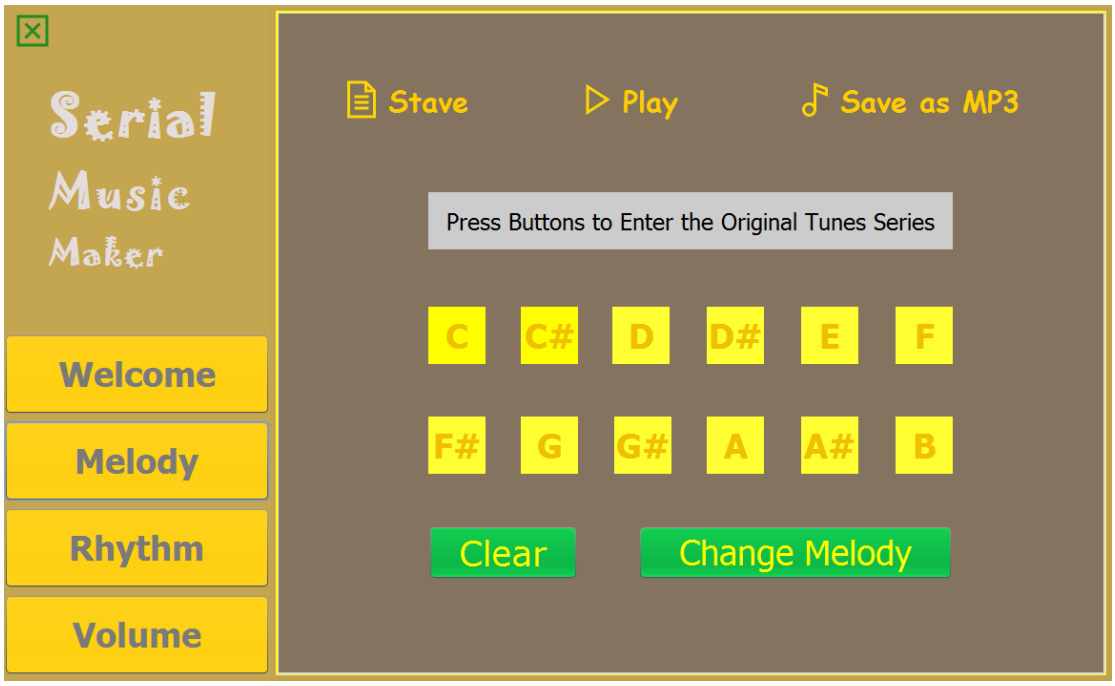


Figure 21 A screenshot of the Melody View

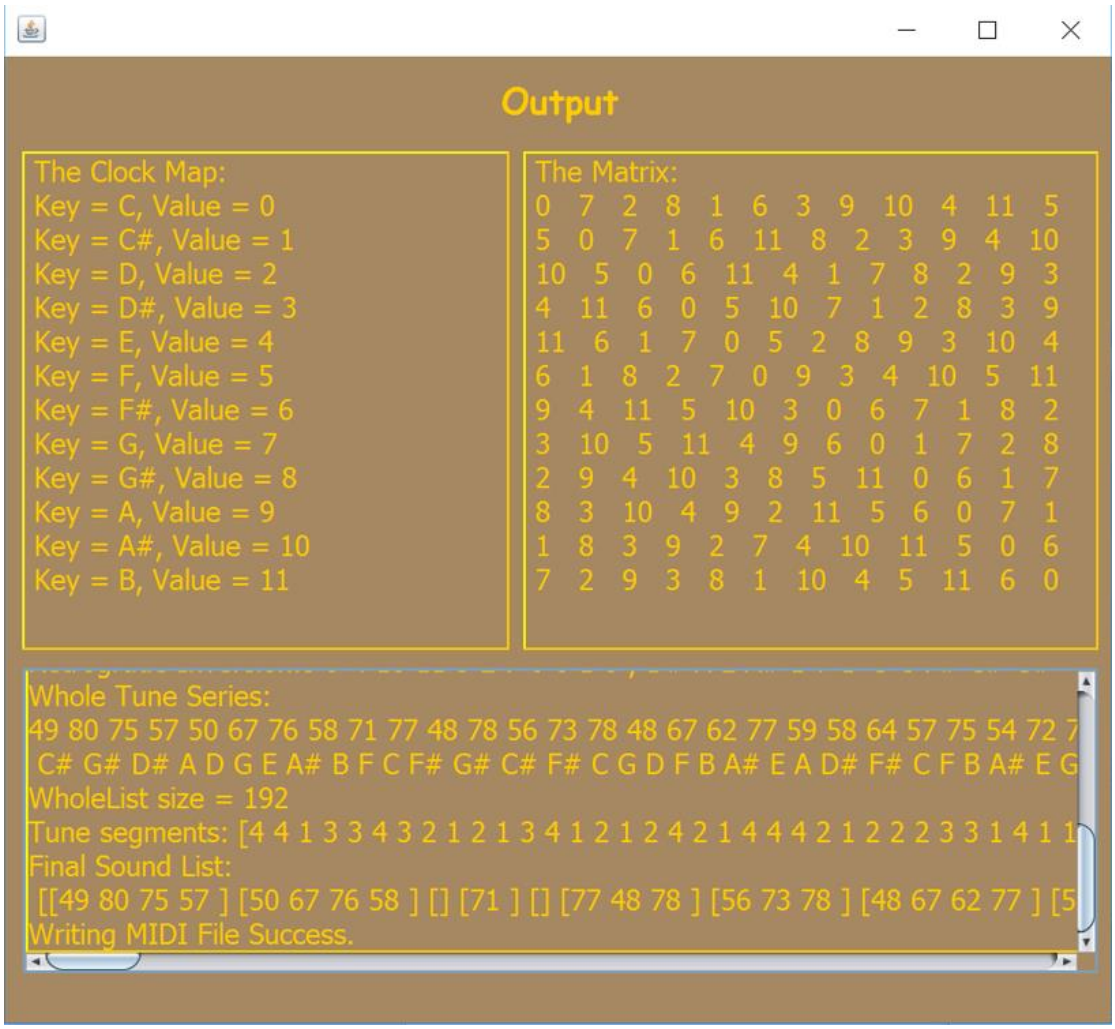


Figure 22 Screenshot of the logFrame

This screenshot shows the melody view which is visible when the “Melody” button is clicked. Instead of asking the user to enter the pitch names manually which may cause various errors and exceptions, the music maker offers buttons for users to click to enter. Each button will change colour to light up when the mouse hovers (See Appendix II figure 3) on and change another colour (See Appendix II figure 4) and disable it to be clicked again when the button has been clicked to inform the user and avoid duplication. “Clear” is used to reset all previous settings while “Change Melody” is used to execute the Melody_Generation algorithm, and show the logFrame with the generated pitch clock, matrix, the final melody tone list and other middle outputs (See Appendix II figure 5). After the “Change Melody” is clicked, a new MIDI file would be written, and by clicking on “Play”, the midi file would be played with “Play” button change to “Stop”(See Appendix II figure 6). The “Stop” button will change to “Play” if the button is clicked and the melody stop playing or the midi file is finished playing once. The fonts of “Stave” “Play”, “Save as MP3” change if the mouse hovers on them (See Appendix II figure 7).

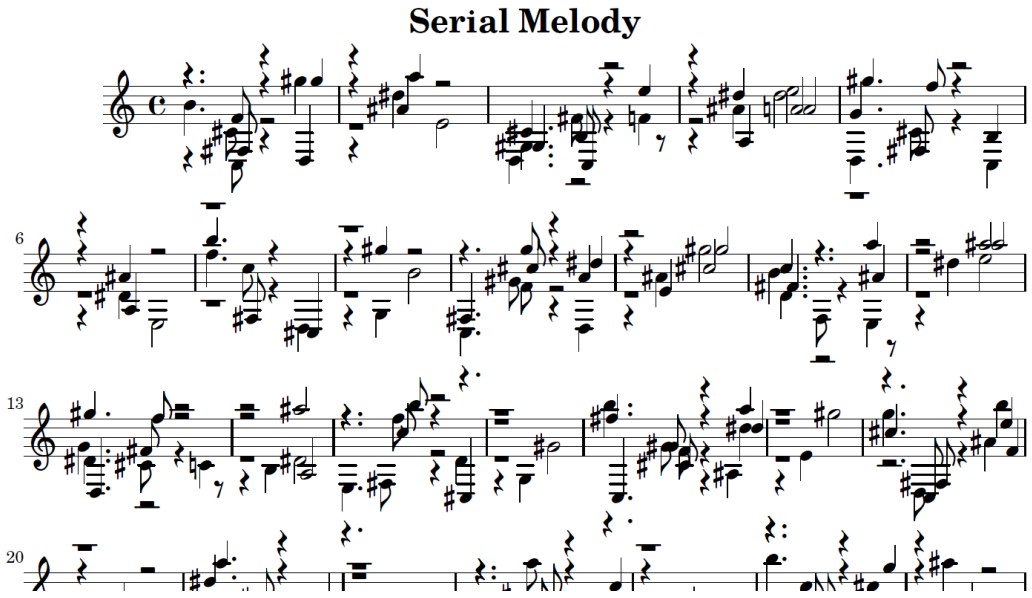


Figure 23 Screenshot of the Serial Melody sheet

This figure shows parts of the PDF music sheet of the Melody generated by clicking on “Change Melody” button and “Stave” button. The tipFrame with “Stave successfully generated” will pop up simultaneously (See Appendix II figure 9).

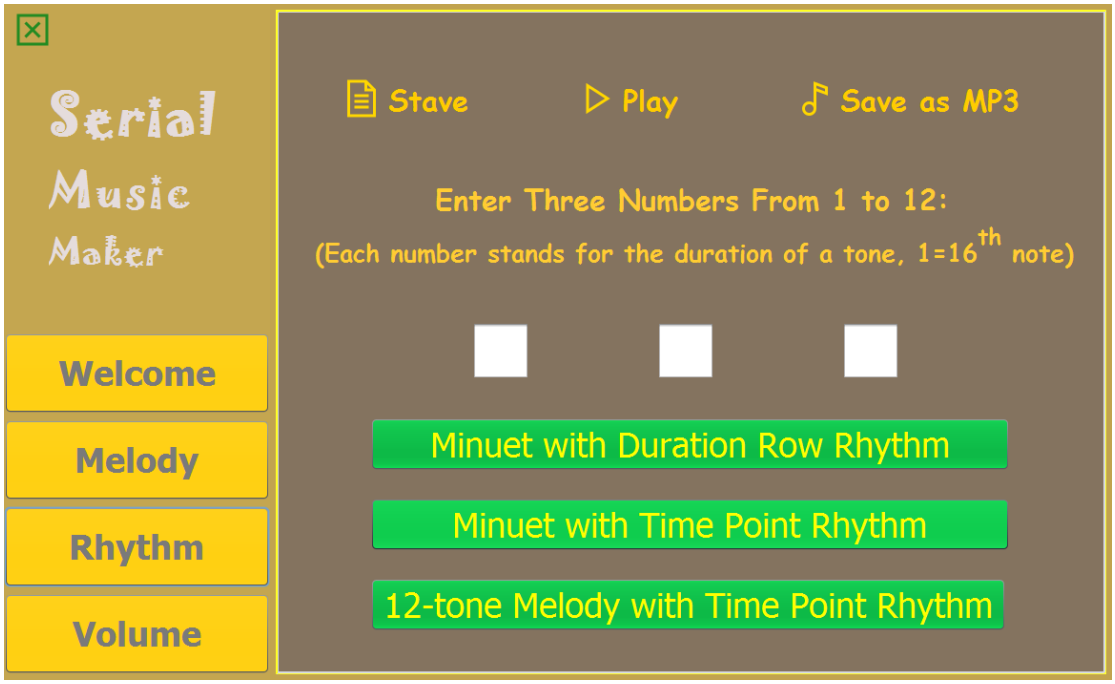


Figure 24 Screenshot of the Rhythm View

This figure demonstrates the rhythm view which is visible when clicking on “Rhythm” button. This view asks the user to enter numbers 1 to 12 into the three boxes as the original three-note durations which would be used to generate durations using Duration Row System and Time Point System. The midi file played when “Play” is clicked

depends on which rhythm generating button is clicked directly before the “Play” is clicked and so does the case when “Stave” is clicked. If “Minuet with Duration Row Rhythm” button is clicked, the duration row durations will be applied to Minuet and minuetDR.mid will be played when “Play” is clicked, minuetDR_sheet.pdf will be generated and opened when “Stave” is clicked (See Appendix II figure 9); if “Minuet with Time Point Rhythm” is clicked, time point durations will be applied to Minuet and the minuetTP.mid will be played, minuetTP_sheet.pdf is produced and opened (See Appendix II figure 9); if “12-Tone Melody with Time Point Rhythm” is clicked, time point durations will be applied to the 12-tone melody generated in the melody view and the melodyTP.mid will be played, melodyTP_sheet.pdf is created and shown (See Appendix II figure 9).

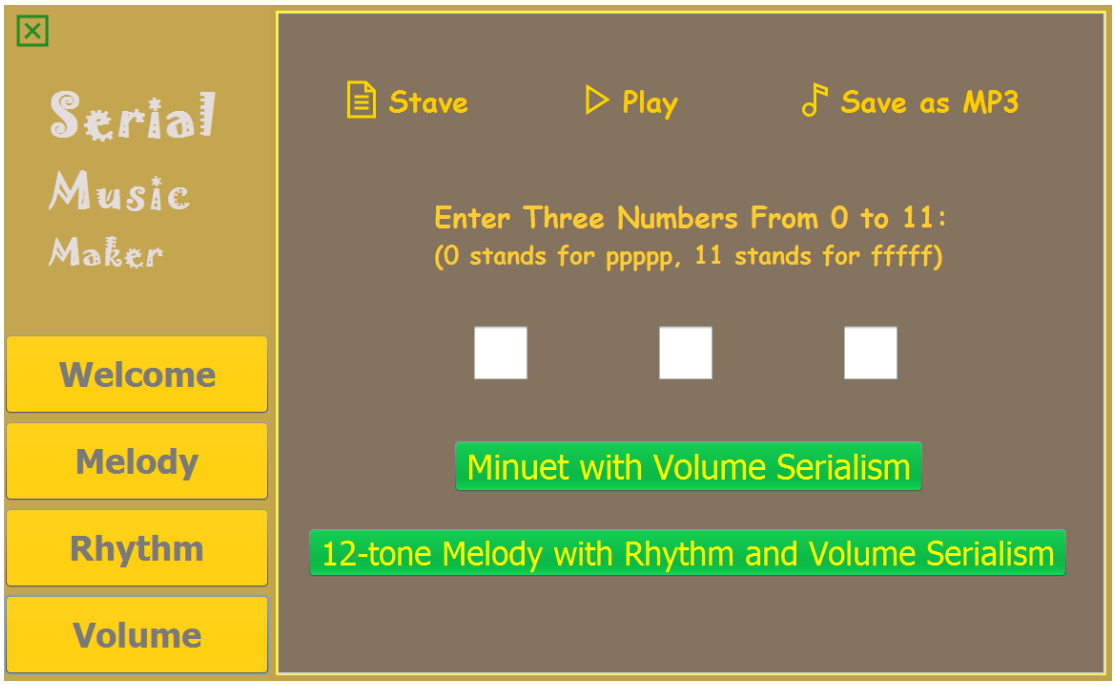


Figure 25 Screenshot of the Volume View

This figure illustrates the volume view that is visible when the “Volume” button is clicked. The user is required to input 0 to 11 into the three boxes as the original three-note volume values for the bearing of a volume series to accomplish the dynamic serialism. By clicking on “Minuet with Volume Serialism” the serial volume will be applied to Bach’s Minuet (original piece) and minuetVolume.mid is produced while pressing “12-tone Melody with Rhythm and Volume Serialism” will combine the volume series with the previously made time point melody with the creation of melodyTPV.mid. The “Play” and “Stave” features are the same as the former descriptions for the rhythm view. It’s a pity that the volume signatures (ppppp to fffff) could not be attached to the music sheet from MIDI file conversion, so the sheet results for melodyTPV_sheet.pdf looks the same as melodyTP_sheet.pdf by now (See Appendix II figure 9 for minuetVolume_sheet.pdf and melodyTPV_sheet.pdf).

5.1.2 Problems Encountered

- Cannot enable multiple-line and dynamic fonts in JLabel – use HTML scripts and CSS style to edit.
- Attach icons beside “Stave” “Play” “Stop” and “Save as MP3” – get the image resource and set icon.
- When clear and reset, the new generated matrix and pitch clock will be attached to the end of the old one – the whole logFrame must be rebuilt to completely clear the former contents.
- Program will crash if the user did not operate in the expected order (for example, generating time point 12-tone music before the 12-tone melody is created.) – catch exceptions and give proper instructions in tipFrame (See Appendix II figure 9).

5.1.3 Testing for GUI

Table 5.1.3.1 GUI Test Cases I

Case Name	Description	Input	Action	Expected Output	Actual Output
Start Executing	Start executing the Java built .jar file to run the program.	No Input	Double-click SerialMusicMaker.jar	Show the Welcome View with background music playing.	Same as the expected output.
Click on Left Side Buttons	Click on “Melody” “Rhythm” “Volume” and “Welcome” to see if the right view is shown and the background melody is played or stopped.	No Input	Click on “Melody” “Rhythm”“Volume” buttons	The background music stops and melody view, rhythm view, volume view is shown respectively.	Same as the expected output.
			Click on “Welcome” button	The background music is played, and the welcome view is shown.	Same as the expected output.
Click Play	After clicking “Play” the button will switch to “Stop” with its icon.	No Input	Click on “Play”	The play icon will turn to stop icon and “Play” text will change to “Stop”.	Same as the expected output.

Table 5.1.3.2 GUI Test Cases II

Case Name	Description	Input	Action	Expected Output	Actual Output
Mouse Hover	Hovers on each pitch buttons and “Stave” “Play” “Save as MP3” buttons in each view to see if the colour changes or the fonts change.	No Input	Hovers on each button.	Colour will change to light up the button by hovering on. The font will change from “Comic Sans MS” to “Jokerman” when hovering on the three buttons on the top.	Same as the expected output.
Click on Pitch Buttons	Click the same button twice to see if the colour changes and the button cannot be double clicked.	No Input	Click the same button twice.	Colours change after clicking on the pitch button. Same button cannot be clicked twice.	Same as the expected output.
Click on Buttons in the Wrong Orders	Click on buttons in the wrong orders to see if exceptions are all caught correctly without program crashing.	No Input	Click “Stave” “Play” before melody generating and midi file creation; Click “Change Melody” before all pitch buttons are pressed; Click “12-Tone Melody with Time Point Rhythm” before “Change Melody”; Click “12-tone Melody with Rhythm and Volume Serialism” before “12-Tone Melody with Time Point Rhythm”.	TipFrame with correct instructions will pop up.	Same as the expected output.

Table 5.1.3.3 GUI Test Cases III

Case Name	Description	Input	Action	Expected Output	Actual Output
Leave Empty Box	When the user does not enter all the three duration boxes before clicking on the three buttons on rhythm view, or not enter all the three volume boxes	Leave any of the three boxes blank	Click “Minuet with Duration Row Rhythm”/ “Minuet with Time Point Rhythm” / “12-tone Melody with Time Point Rhythm” with one or more box/boxes blank in rhythm view; Click “Minuet with Volume Serialism”/ “12-tone Melody with Rhythm and Volume Serialism” with one or more box/boxes blank in volume view.	TipFrame with instruction “Please, do not leave any box blank.” will pop up.	Same as the expected output.
Input Numbers Exceed the Range	When the user enters a number exceeds the range specified, a warning will pop up after clicking on the three buttons on rhythm view, or the two buttons in the volume view.	Input number >12 or <1 in box of the rhythm view; Input number >11 or <0 in the box of volume view.	Click “Minuet with Duration Row Rhythm”/ “Minuet with Time Point Rhythm” / “12-tone Melody with Time Point Rhythm” with number(s) in one or more box/boxes exceed [1,12] in rhythm view; Click “Minuet with Volume Serialism”/ “12-tone Melody with Rhythm and Volume Serialism” with number(s) in one or more box/boxes exceed [0,11] in volume view.	tipFrame with the instruction “The duration is 1 to 12, please re-enter” or “The volume is 0 to 11, please re-enter” will pop up.	Same as the expected output.

5.2 Implementation of Algorithms

According to 4.2.2 Description of the Algorithms and 4.2.3 Design of the Java Classes, all algorithms are implemented in the Matrix class by constructing a 12-tone matrix and generate melodySoundList, minuetDurationRow, minuetTimePoint, melodyTimePoint, minuetVolume, melodyTPVolume step by step based on the matrix. For the reason that the creation of *mid file requires the definitions of each note together with its duration, duration list should also be generated for each note list. In this case, it should be pointed out that the durations for the pauses and the empty ArrayLists of notes represent the pauses are essential.

5.2.1 Melody Serialism

After the construction of the 12-tone matrix, by randomly choosing a prime row, retrograde row, inversion column, retrograde inversion column the original note list is constructed. In as much as the music should not be merely a single tone melody piece, while chords above four tones are basically inconsistent and the tones are even indistinguishable, the original notes will be divided into segments of one to four notes and become an ArrayList of ArrayLists of notes. The last step is to insert the empty list in the position of the 3^{rd}, 5^{th}, 10^{th}, 12^{th}, 17^{th}... and apply Son Clave rhythm to avoid monotonous rhythm.

✧ **Problems Encountered**

With the tests conducting on another Java project before the design settled, no significant problem encountered in this part.

5.2.2 Rhythm Serialism

Deeply into more aspects for serialism, referring to the pseudocode derived from Babbitt’s Duration Row System and Time Point System, the software takes Bach’s Minuet as the note list for the sake of comparing these two rhythm systems explicitly. The implementation of method constructMinuetDurations() -> Dration_Row(), timePointMinuet() and timePointTwelveTone() first takes in the user input duration row and strictly following the pseudocode in the design part.

✧ **Problems Encountered**

If the length of the note list (minuetDurationRow, minuetTimePoint, timePointList) is different from the duration list (minuetDurations, minuetTPDurations, tpDurations), then the melody will not completely be played by MIDI sequencer or the program may even crash. – loop for Math.ceil() times to generate more durations and loop for note list size times to play the melody to avoid IndexOutOfBound Exception.

5.2.3 Volume Serialism

Milton Babbitt also devised approaches for volume serialism, however, the acceptable material for the actual serialism method could not be found. Therefore, it is replaced by the same method used for constructing the duration list under the duration row

system. As MIDI velocity supports number 0 to 127, for the sake of differentiating and promoting the volume contrast, the mapping of the input number, volume notation and the corresponding velocity value are: 1 – ppppp – 1; 2 – pppp – 8; 3 – ppp – 20; 4 – pp – 31; 5 – p – 42; 6 – mp – 53; 7 – mf – 64; 8 – f – 80; 9 – ff – 96; 10 – fff – 112; 11 – ffff – 120; 12 – fffff – 127.
The Minuet and the serial melody with time point rhythm is integrated with the serial volume respectively to enrich the feature of the software and provide the user with opportunities for effect observation.

✧ **Problems Encountered**

◆ Accessing data in an ArrayList of ArrayLists and arranging the consistency of the indexes among note list, duration list and volume list causes chaos – construct another array of volumes and convert the number to the corresponding velocity value simultaneously.

5.2.4 Testing of Algorithms

Table 5.2.4.1 Test Case Melody Generation

Case Name	Description	Input	Action	Expected Output	Actual Output
Melody Generation	After entering the original row, the program will show the pitch clock that maps each pitch to number 0 to 11, the matrix, the chosen prime/retrograde/inversion /retrograde inversion lines and the whole note list with pitch names, the segment and the final sound list in the logFrame. The test is to see if everything above works and the pitch names in the whole list is right as what they are matched in the clock.	GCF#D C# G#A D#A#EB F	Click “Melody Change”.	The Clock Map: G – 0; G# – 1; A – 2; A# – 3; B – 4; C – 5; C# – 6; D – 7; D# – 8; E – 9; F – 10; F# – 11; The Matrix: 0 5 11 7 6 1 2 8 3 9 4 10 7 0 6 2 1 8 9 3 10 4 11 5 1 6 0 8 7 2 3 9 4 10 5 11 5 10 4 0 11 6 7 1 8 2 9 3 6 11 5 1 0 7 8 2 9 3 10 4 11 4 10 6 5 0 1 7 2 8 3 9 10 3 9 5 4 11 0 6 1 7 2 8 4 9 3 11 10 5 6 0 7 1 8 2 9 2 8 4 3 10 11 5 0 6 1 7 3 8 2 10 9 4 5 11 6 0 7 1 8 1 7 3 2 9 10 4 11 5 0 6 2 7 1 9 8 3 4 10 5 11 6 0	Same as the expected output with the chosen: Prime:10 3 9 5 4 11 0 6 1 7 2 8; F A# E C B F# G C# G# D A D# Retrograde:8 2 7 1 6 0 11 4 5 9 3 10; D# A D G# C# G F# B C E A# F Inversion: 9 4 10 2 3 8 7 1 6 0 5 11; E B F A A# D# D G# C# G C F# Retrograde Inversion:11 5 0 6 1 7 8 3 2 10 4 9; F# C G C# G# D D# A# A F B E The chosen row and column have been high-lightened, and the pitch names are consistent with the clock map.

Table 5.2.4.2 Test Cases for Rhythm Algorithms

Case Name	Description	Input	Action	Expected Output	Actual Output
Minuet Duration	After entering the original duration row, click “Minuet with Duration Row Rhythm” to check if the Minuet duration row list - the note list - is consistent with the Minuet Durations ArrayList – the full duration list - (the fourth position of the current duration list corresponds with an empty list in the note list if the duration list has four elements) and check if the interval of adjacent durations is 3.	4,6,9	Click “Minuet with Duration Row Rhythm”	Minuet Duration ArrayList: [4,6,9,5] [7,9,12,8] [10,12,3,11] [1,3,6,2] [4,6,9,5]… Minuet Duration Row List: [79][72][74][][76] [77][79][][72][72] …	Same as the expected output.
Minuet Time Point	After entering the original duration row, click “Minuet with Time Point Rhythm” and “Stave”. Then read the sheet to calculate the duration between each note and its previous bar line and analyse the correctness by comparing them with the duration values in the original duration row duration list.	4,6,9	Click “Minuet with Time Point Rhythm”	The durations of each note should be: 4, 6, 9 7, 9, 12 10, 12, 3 1, 3, 6 4, 6, 9…	Same as the expected output.
12-tone Melody Time Point	After entering the original duration row, click “12-tone Melody with Time Point Rhythm” and “Stave”. Then read the sheet to calculate the duration between each tone and its previous bar line and analyse the correctness by comparing them with the duration values in the original duration row duration list.	4,6,9	Click “12-tone Melody with Time Point Rhythm”	The durations of each tone should be: 4, 6, 9 7, 9, 12 10, 12, 3 1, 3, 6 4, 6, 9…	Same as the expected output.

Table 5.2.4.3 Test Cases for Volume Algorithms

Case Name	Description	Input	Action	Expected Output	Actual Output
Minuet Volume	After entering the original volume row, click “Minuet with Volume Serialism” to check if the intervals of volume values between adjacent volume ArrayList are three.	4,7,10	Click “Minuet with Volume Serialism ”	The volume list should be: [4, 7, 10] [7, 10, 1] [10, 1, 4] [1, 4, 7] [4,7,10]…	Same as the expected output.
Time Point Serial Melody Volume	After entering the original volume row, click “12-tone Melody with Rhythm and Volume Serialism” to check if the intervals of volume values between adjacent volume ArrayLists are three.	2, 5, 9	Click “12-tone Melody with Rhythm and Volume Serialism ”	The volume list should be: [2, 5, 9] [5, 8, 12] [8, 11, 7] [11, 2, 10] [2, 5, 9]…	Same as the expected output.

5.3 Implementation of MIDI

The implementation of these four MIDI components is covered in the MusicPlayer class:

- ✧ MidiEvent: apart from the raw MIDI message which contains the controller number, channel index, note number and velocity value, makeEvent() method takes in an additional tick value (the time point of command execution) to construct a MidiEvent to be added into the track.
- ✧ Track: a sequence of MidiEvents that defined by the makeEvent() method.
- ✧ Sequence: a collection of multiple tracks and timing information and will be played by the Sequencer.
- ✧ Sequencer: handle the MIDI data and command the instruments to play the notes.

The MusicPlayer class defines setUpPlayerMelody(), setUpMinuetDurationRow(), setUpPlayerMinuetTimePoint(), setUpPlayerTimePoint(), setUpPlayerMinuetVolume() and setUpPlayerVolume() methods for the implementations of creating serial melody, serial rhythm minuet, serial rhythm melody and their serial volume pieces. Each method first creates a Sequence using PPQ (Pulse Per Quarter Note or Ticks Per Quarter Note) to set the relative length (the number of ticks) of one quarter note [22]. (The speed of a piece depends on the PPQ and tempo which is described as “the amount of quarter notes in every minute”) Then, it construct a track in the Sequence to add MidiEvents returned by makeEvent(), where the tick between a Note-On and a Note-Off events is the duration of the note, and the channel numbers vary from 1 to 4 to enable up to four notes to play at the same time. The current time-stamp should be defined and increased by the duration of the current note for each channel. Moreover, the velocity of the first channel is set to be louder than that of the other channels to form a main texture of melody. Finally, the method writes the Sequence into a new MIDI file.

The StartMidi() method reads the Sequence stored in a midi file, sets the tempo and asks the Sequencer midiPlayer to start playing the Sequence once, while the stop() method aims to stop the Sequencer.

5.3.1 Problems Encountered

- ◆ Music collapses tended to happen when trying to play the sequence directly after the track is constructed – Use the MidiSystem built-in method to write new MIDI File for each music pieces and playing them by the StartMidi() method.
- ◆ IndexOutOfBound Exception would happen when the number of notes is different from the number of durations – compare the numbers first and iterate for the less number times.
- ◆ Intended to stop the playing and close the Sequencer after the MIDI file is played once with the “Stop” button change to “Play” automatically – Java provides a method called addMetaEventListener() for the Sequence which contains an inner method that detects a Meta message and reconstruct the JLabel icon and text if the Meta message has type 47 (reaches the end of the sequence).

5.3.2 Testing for MIDI

Table 5.3.2 Test Cases for MIDI

Case Name	Description	Input	Action	Expected Output	Actual Output
MIDI File Created	Giving executable inputs that can generate the note list and duration list successfully and see if the MIDI file can be generated.	Any acceptable input.	Click on “Change Melody”/ “Minuet with Duration Row Rhythm”/ “Minuet with Time Point Rhythm”/ “12-tone Melody with Time Point Rhythm”/ “Minuet with Volume Serialism”/ “12-tone Melody with Rhythm and Volume Serialism”	“Writing MIDI File Success” will appear at the end of the logFrame.	Same as the expected output.
MIDI File Playing	After the successful generating of MIDI file, click on “Play” to see if the MIDI file can be played and the button will change to “Play” if it finishes playing once.	No input.	Click on “Play” in each view after Case “MIDI File Created”	The melody would play with the button changing to “Stop” and the button will return to “Play” if it finishes playing.	Same as the expected output.

5.4 Implementation of Music Sheet Production

The implementation of the music sheet generation contains:

- ✧ The installation and configuration of Lilypond: download the latest version of Lilypond on the official website http://lilypond.org/development.html, install and configure the environment variables by adding the bin direction to the Path. After that, run “lilypond” in command line to test if the configuration is successful.
- ✧ Implement a ProcessBuilder to execute command to run “midi2ly” to convert *.mid file to *.ly file (the format for lilypond file which provides predefined syntax and gives a text-based musical notation.)
- ✧ Call a self-defined function lyFileChange(filename, new_filename, title) to read the *.ly file and write a modified **.ly copy by omitting the line with “midi”, and add a title for the sheet by inserting “\header { title = \markup {“title”}}” from the third line.
- ✧ Implement ProcessBuilder again to run command “lilypond **.ly” and “**.pdf” to create the PDF sheet for **.ly and open it.

5.4.1 Problems Encountered

- ◆ There are two versions of Lilypond on the website, and after install the stable version 2.18.2, the menu of the software displayed garbage characters – although the latest version 2.19.83 is marked as “unstable” on the website, this problem is solved by installing this “unstable” version.

5.4.2 Testing for Sheet Generation

Table 5.4.2 Test Case for Sheet Generation

Case Name	Description	Input	Action	Expected Output	Actual Output
Open Sheet	After the midi file is generated, by clicking on “Stave” the program will call CMD to execute “midi2ly” built-in function of Lilypond, write a modified copy of the lilypond file and run lilypond again to convert the *.ly file to pdf sheet and open.	The generated *.mid file.	Click “Stave” after MIDI files have been created.	The pdf sheet file will open. And a successful sheet music generation information will show in the tipFrame.	If the sheet music file has not been created, then it may not open but just show the tipFrame. Second click of “Stave” is needed to open the pdf file.

5.5 Implementation of MIDI to MP3 Conversion

Converting the MIDI file to MP3 or WAV format is a desired feature for the system, and because the MP3 format belongs to audio type which is produced by the real-time recording of the waveform which is totally different from the digital signal of keyboard typing instructions of MIDI-type. The possible solution could be getting the Sequence of the MIDI file, converting the raw MIDI data and sending the time-stamped MIDI event to the Synthesizer through the Sequencer to produce the sample audio. However, this implementation of format conversion fails using Java programming because details for connecting a Sequencer and Synthesizer and recording the sample sound to write a WAV/MP3 file have not been figured out. Therefore, this system use java.net.* package and told the software to open an existing format converting website in the browser to make a compliment.

6. Evaluation

6.1 Criteria

Arnold Schoenberg himself has pointed out: “My music is not modern, it is merely badly played.” [11] Consider that the atonal music pieces that the software composed is generally not harmonic and anti-popular, users’ opinions to the music pieces will not be listed as an aspect for evaluation. The core criteria for the success of the software lies in the implementation of algorithms which could be evaluated by analyse the data in the logFrame during the testing stage, while the GUI design that gives proper instructions for users in the tipFrame has also been tested before. Therefore, only the non-functional requirement will be evaluated.

- To evaluate the reliability of the software, the failure rate for the software could not exceed 10%, this will be evaluated by complete the whole composing process of the software and click on buttons in all possible orders to see if the software would crash for any cases.
- To evaluate the performance of the software, open the CPU monitor to track the utilisation and speed of the CPU when running and operating the program. The desirable data is no more than 35% for the utilisation, and the average CPU consumption is less than 15%.

6.2 Evaluation of Results

After 20 times’ testing for each operation, although one failure of playing the “12-tone melody with rhythm and volume serialism” happens, it succeeds again after regenerating the MIDI sequence. While no system crashing has happened (for instance, generating Minuet Duration Row or 12-tone Time Point before generating 12-tone melody), which has also indicated that the crashing rate of the software is smaller than 10%, in another word, the reliability of the system is proved to be high.

As for the performance, the Task Manager of Windows system is used to monitor the current utilisation of CPU:

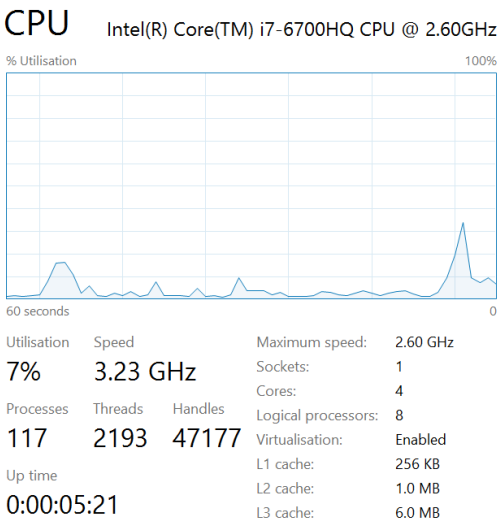


Figure 26 When stay in the Welcome view with minuet.mid playing

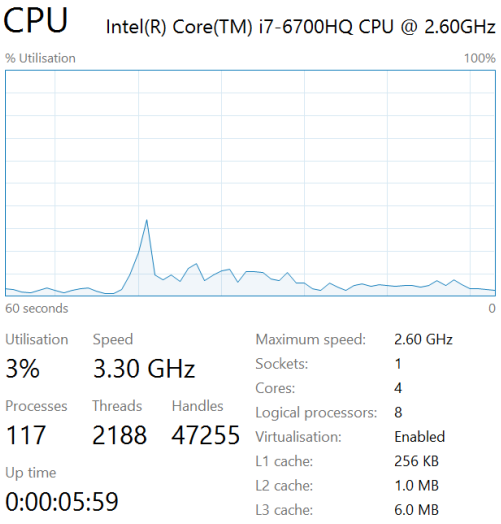


Figure 27 When stay in other view except for Welcome view.

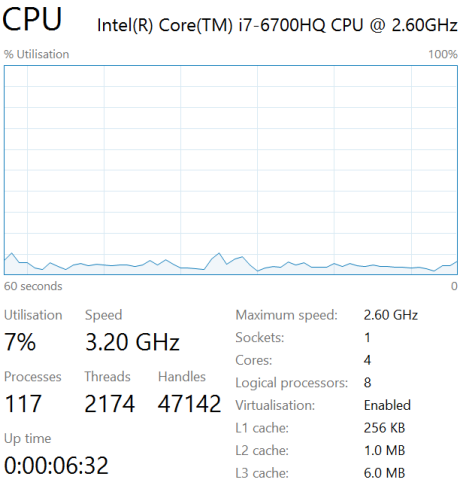


Figure 28 When “Change Melody” is calculating.

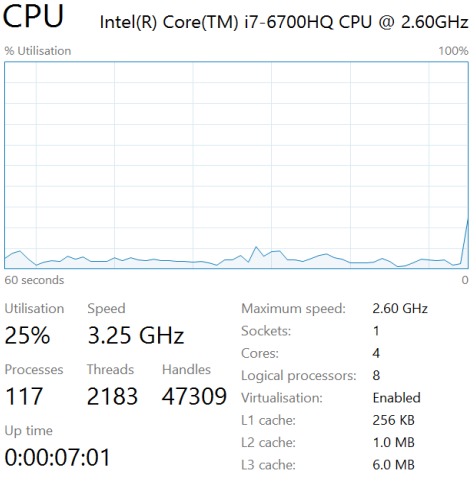


Figure 29 At the start point of playing a MIDI file

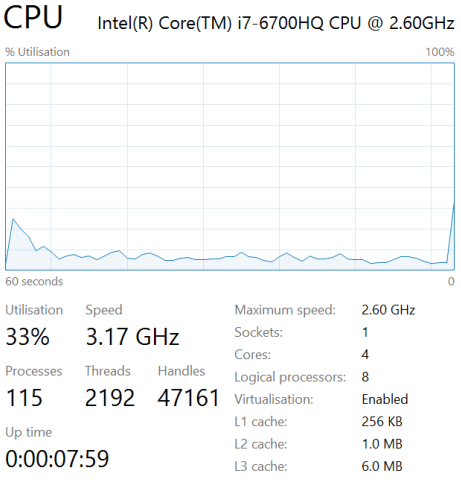


Figure 30 When generating the sheet music by clicking on the “Stave”.

The above diagrams show that apart from the CPU utilisation caused by other processes which is about 3%, the calculation and performance of music pieces take about 7% – 3% = 4% CPU space, while cases may happen at the start point of playing the MIDI file (with 25% - 3% = 22% utilisation at a moment). The maximum occurred when generating the sheet (with 33% - 3% = 30% utilisation once) since the process needs the running of command.exe and Lilypond.

The average CPU consumption is also test by the Resource Monitor:

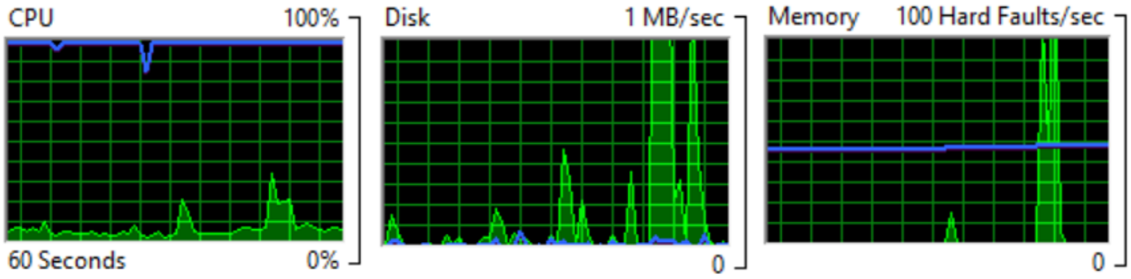


Figure 31 The diagrams for CPU, Disk and Memory monitor provided by the Resource Monitor.

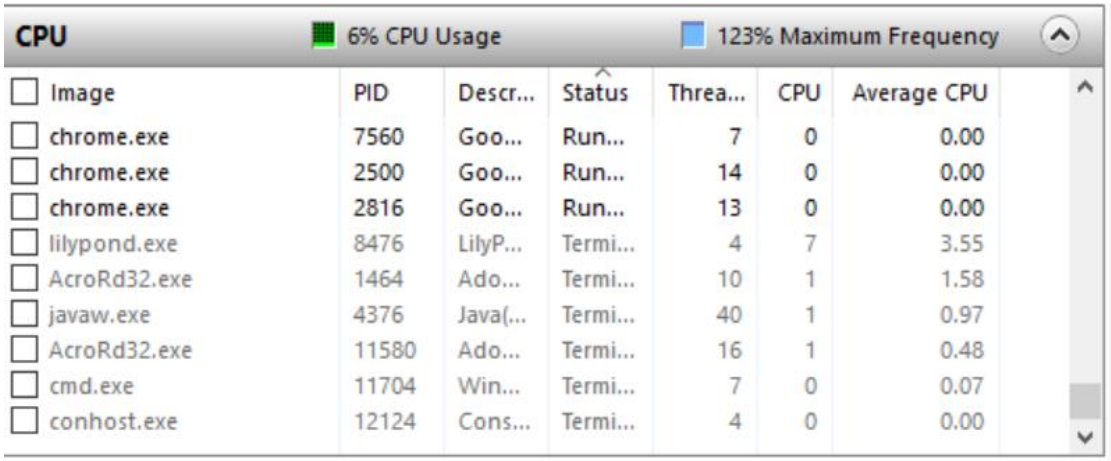


Image	PID	Descr...	Status	Threa...	CPU	Average CPU
chrome.exe	7560	Goo...	Run...	7	0	0.00
chrome.exe	2500	Goo...	Run...	14	0	0.00
chrome.exe	2816	Goo...	Run...	13	0	0.00
lilypond.exe	8476	LilyP...	Termi...	4	7	3.55
AcroRd32.exe	1464	Ado...	Termi...	10	1	1.58
javaw.exe	4376	Java(...	Termi...	40	1	0.97
AcroRd32.exe	11580	Ado...	Termi...	16	1	0.48
cmd.exe	11704	Win...	Termi...	7	0	0.07
conhost.exe	12124	Cons...	Termi...	4	0	0.00

Figure 32 The diagrams provided by the Resource Monitor where the last column is the average percent of CPU consumption by the process.

Figure 32 shows that the average CPU consumption for lilypond.exe, javaw.exe and cmd.exe is 3.55% + 0.97% + 0.07% = 4.59% with the utilisation taken by other software components (about 4% ~ 5%), the total consumption is smaller than 15%.

The only problem of functional feature is that when generating the music sheet using Lilypond, the PDF file of the sheet cannot open at the first time and a second clicking of the “Stave” button need to be done to open and see the sheet.

6.3 Evaluation of Project Strength and Weakness

✧ Strength

From the developer’s perspective: this project contains various sorts of technologies, from GUI building using Java Swing, generating and playing new pieces using MIDI programming, to creating music sheet using Lilypond. Moreover, the project also maintains code with precise comments and Javadoc webpages that helps the developers with future software improvement.

From users’ perspective: firstly, the project developed an atonal music composing system which provides users with a quick overview of how serial music is produced and felt like by conducting melody serialism, rhythm serialism and volume serialism step by step and playing the music pieces directly. Secondly, the project offers an aesthetic graphical user interface and gives instructions for users to help them operate correctly. Thirdly, the software can create the music sheets PDF of the newly generated pieces which gives a direct perspective for the user to analyse the music piece. Finally, the reliability of the software is high with a very low crashing rate.

✧ Weakness

The software is suitable for professional research but not for public promotion, since the system only provide text data and complex stave for user to analyse the serialism notion, and the atonal music itself aims to break the harmonious property which causes scarce audiences. From a technical view, the format conversion – from MIDI file to MP3 file – has not been accomplished which can be improved in the future.

7. Learning Points

This section will give a summary for the key learning points in this project, which include using NetBeans GUI builder to build the graphical user interface, the extraction of the idea described in the paper and the implementation by coding algorithm, the MIDI programming with Java Sound API, the music sheet generation with Lilypond.

- For GUI Building

  The GUI is built with Java Swing components. It would be an exhausting task if the frame design can only be seen after the initialisation of each component including the setting of the size, position, background colour and font. In contrast, NetBeans IDE offers a GUI Builder feature which allows the developer to click and drag the Swing components in the right positions to the canvas and generates the corresponding Java code automatically. The developer could test the effect directly and the code can be added to the particular method block for mouse event or action to implement the functions of the system. What should be point out are that to enable multi-line JLabel and set the font and style for each part of the text, HTML and CSS code should be applied as the actual content in the JLabel, while clearing the existing Matrix, Pitch Clock and all log in the logFrame utterly requires the rebuild of the whole JFrame.

- For Algorithm Extraction

  Although some music knowledge has been acquired previously, the idea of Schoenberg’s 12-tone atonal music is a brand-new field, and only a minority of didactical materials can be found. These have proved to extract the abstract and intricate concepts of serial music from merely professional researchers’ papers and to rearrang the methods by representing them via pseudocode a huge challenge. However, with the careful analysis of 12-tone matrix construction from “12tone_CREATE_MATRIX” as well as Milton Babbitt’s rhythm (using Duration Row System where the generated durations series are the actual durations for each note, and Time Point System where the generated values are the time point for each note which means they are the time between the current note and the previous bar line) and diversity serialism using Paul Riker’s “THE SERIALISM OF MILTON BABBITT” with an continuous interest in the development of music, the algorithms for serial music composing can be finally devised.

- For MIDI and Java Sound API

  Java MIDI programming is another brand-new knowledge: MIDI files store the instruction signal for the keyboard to operate instead of the actual waveform sound stored in formatted audio files. Java Sound API provides MidiMessage object for the program to send instructions to form a MIDI track in a MIDI sequence to write in a midi file and use a sequencer to play the midi file using the default instrument.

- For Lilypond

  Lilypond is a music engraving software that produces traditional aesthetic high-quality music sheet. Instead of dragging music notations from a toolbar to the blank staff, Lilypond complies (interprets) a piece of text-formatted music to create the sheet. It has some built-in functions like “midi2ly” to convert midi file to a lilypond file (*.ly) and “lilypond” to convert lilypond file (*.ly) to PDF sheet or midi file (depends on whether there is a “\midi{}” line). After the configuration of environment variables – add the “bin” directory to “Path”, one can run the above functions in command.exe.

8. Professional Issues

8.1 Issues Related to Code of Practice

Generally, the project conducts an excellent performance in key IT practices and business functions:

- When planning the project, the scope, deliverables and timescales are estimated in advance. The pitfalls caused by the utilisation of new software tools (ScoreCloud, Lilypond and NetBeans 8.2) are also estimated.
- When tracking progress, although the progress did not follow the planning in the Gantt Chart strictly, all milestone points is accomplished on time. Also, due to further research of the feasibility of each component, the plan was modified halfway to better suit the practice.
- When designing software, both experienced and inexperienced users are suitable to use the system, since the system gives proper instructions for users to conduct correct operations.
- When programming, to benefit the future maintenance and help other programmers understand the software, the project is aware of maintaining a well-structured source code of Java, with detailed comments as well as elaborate Javadoc description which is available at “https://student.csc.liv.ac.uk/~sgzlin9/Javadoc/Home/package-summary.html”.
- When writing the technical documentation, the design documentation provides necessary background and detailed design for each stage, including precise pseudocode for the primary algorithms.

8.2 Issues Related to Code of Conduct

- The project protects the public interest. All professional activities including the preparing research, the software utility right and the writing of all documentations are conducted without discrimination of sex, marital status, colour, ethnic region or any other condition.
- As for professional competence and integrity, all professional knowledge used in this project is within the developer’s abilities acquired from the beginning or during the project process. For instance, the basic music theory is originally acquired; while the MIDI programming skill is obtained by searching and studying the MIDI specification and Java Sound API tutorial; the sheet music generation method is developed after a careful reading of Lilypond official documentation. The project rejects any bribery or unethical inducement and does not undertake any work that is not possessed by the developer

9. Bibliography

[1] Wikimedia Foundation, Inc., (2019, May 4). *Classical music* [Online]. Available: https://en.wikipedia.org/wiki/Classical_music. [Accessed: 6 - May - 2019]

[2] Wikimedia Foundation, Inc., (2018, Oct. 24). *Expressionism* [Online]. Available: https://en.wikipedia.org/wiki/Expressionist_music. [Accessed: 6 - May - 2019]

[3] John Harbison (1992) Symmetries and the “New Tonality”, Contemporary Music Review, 6:2, 71-79, DOI: 10.1080/07494469200640141.

[4] Andrew Mead, *An Introduction to the Music of Milton Babbitt.* Princeton University Press, 1994.

[5] George Perle. (1915-). *Serial Composition and Atonality: An Introduction to the Music of Schoenberg, Berg, and Webern* (1991) [Online]. Available: https://books.google.co.uk/books?id=4C8RjEaBRf4C&printsec=frontcover&dq=0520052943&hl=en&sa=X&ved=0ahUKEwj7v9e1-s7eAhXCDcAKHd5FDpYQ6AEIUzAH#v=onepage&q&f=false. [Accessed: 6 - May - 2019]

[6] Paul Riker. (2018, May 31). *THE SERIALISM OF MILTON BABBITT* [Online]. Available: http://paulriker.com/words/The_Serialism_of_Milton_Babbitt.pdf. [Accessed: 6 - May - 2019]

[7] Lejaren Hiller and Ramon Fuller, “Structure and Information in Webern’s Symphonie, Op. 21”, Journal of Music Theory 11, no. 1 (Spring 1967): 60–115. Citation on p. 94.

[8] Hybrid Pedagogy Inc., (2012 - 2018). *Pitch (class)* [Online]. Available: http://openmusictheory.com/pitch(Class).html. [Accessed: 6 - May - 2019]

[9] PIANOSCALES.ORG. (2012 - 2019). *Chromatic Scale* [Online]. Available: https://www.pianoscales.org/chromatic.html. [Accessed: 6 - May - 2019]

[10] Thomas Pankhurst. (2019). *Introduction to Tonality* [Online]. Available: http://www.tonalityguide.com/introoverview.php. [Accessed: 6 - May - 2019]

[11] James Gholson. (2019). *CREATING A 12 TONE MATRIX* [Online]. Available: https://unitus.org/FULL/12tone.pdf. [Accessed: 6 - May - 2019]

[12] Patreon, *The World’s Most Popular Rhythm*. [Online]. Available: https://www.youtube.com/watch?v=Ye7d5mPNfYY. [Accessed: 6 - May - 2019]

[13] Stephen Peles, Stephen Dembski, Andrew Mead and Joseph N. Straus. (1962). *The Collected Essays of Milton Babbitt. Twelve-Tone Rhythmic Structure and the Electronic Medium* [Online]. Available: http://www.jstor.org/stable/j.ctt7rfx5.17. [Accessed: 6 - May - 2019]

[14] William Marvin Johnson, Jr: Perspectives of New Music, Vol. 23, No. 1. (Autumn - Winter, 1984). pp. 278-293. *Time-Point Sets and Meter* [Online]. Available: http://www.jstor.org/stable/832917. [Accessed: 6 - May - 2019]

[15] David Back. (1999). *Standard MIDI-File Format Spec. 1.1, updated* [Online]. Available: http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html. [Accessed: 6 - May - 2019]

[16] MIDI Manufacturers Association. (2019). *An Introduction to MIDI* [Online]. Available: https://www.midi.org/images/easyblog_articles/43/intromidi.pdf. [Accessed: 6 - May - 2019]

[17] Arpege Music - Dominique Vandenneucker. (2012). *MIDI Tutorial* [Online]. Available: http://www.music-software-development.com/midi-tutorial.html. [Accessed: 6 - May - 2019]

[18] Oracle and/or its affiliates. (1994 - 2019). *The Java Tutorials - Trail: Sound* [Online]. Available: https://docs.oracle.com/javase/tutorial/sound/index.html. [Accessed: 6 - May - 2019]

[19] LILYPOND.ORG. (1996). *Introduction* [Online]. Available: http://lilypond.org/introduction.html. [Accessed: 6 - May - 2019]

[20] LILYPOND.ORG. (1996). *“Compiling” Music* [Online]. Available: http://lilypond.org/text-input.html. [Accessed: 6 - May - 2019]

[21] The Apache Software Foundation. (2017 - 2019). *Swing GUI Builder (formerly Project Matisse)* [Online]. Available: https://netbeans.org/features/java/swing.html. [Accessed: 6 - May - 2019]

[22] Christopher Dobrian. (2014, May 19). *Timing in MIDI Files* [Online]. Available: https://sites.uci.edu/camp2014/2014/05/19/timing-in-midi-files/. [Accessed: 6 - May - 2019]

Appendix I – Detail Screenshots

Figure 1 Welcome View

Figure 2 Melody View

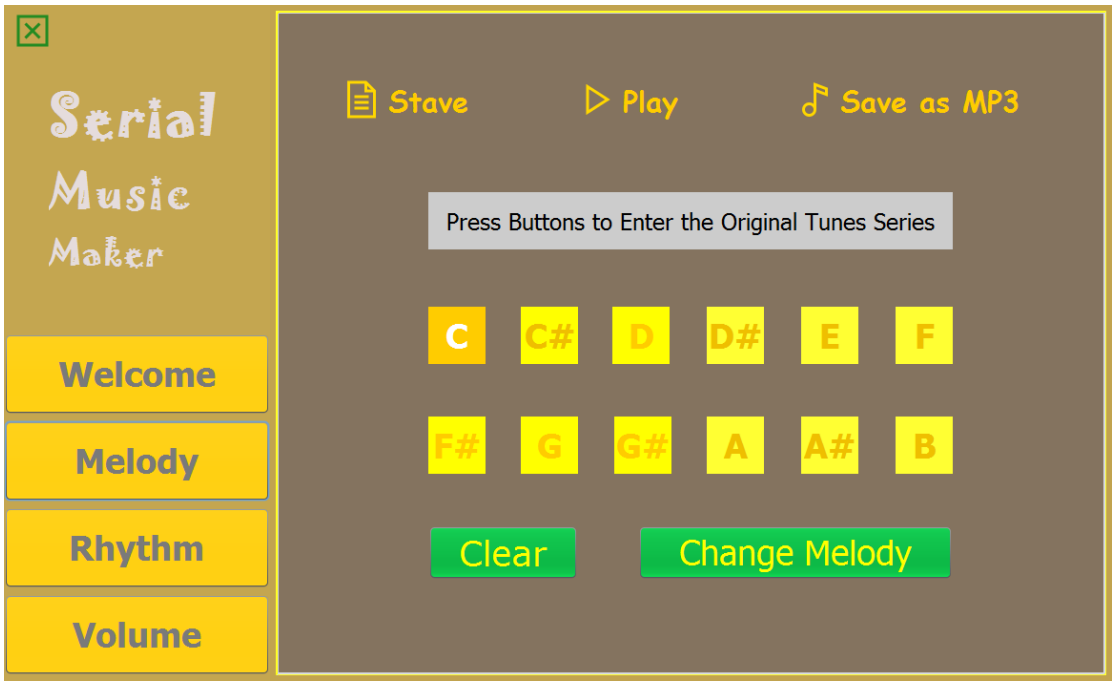


Figure 3 Melody View – mouse hovers on the pitch button

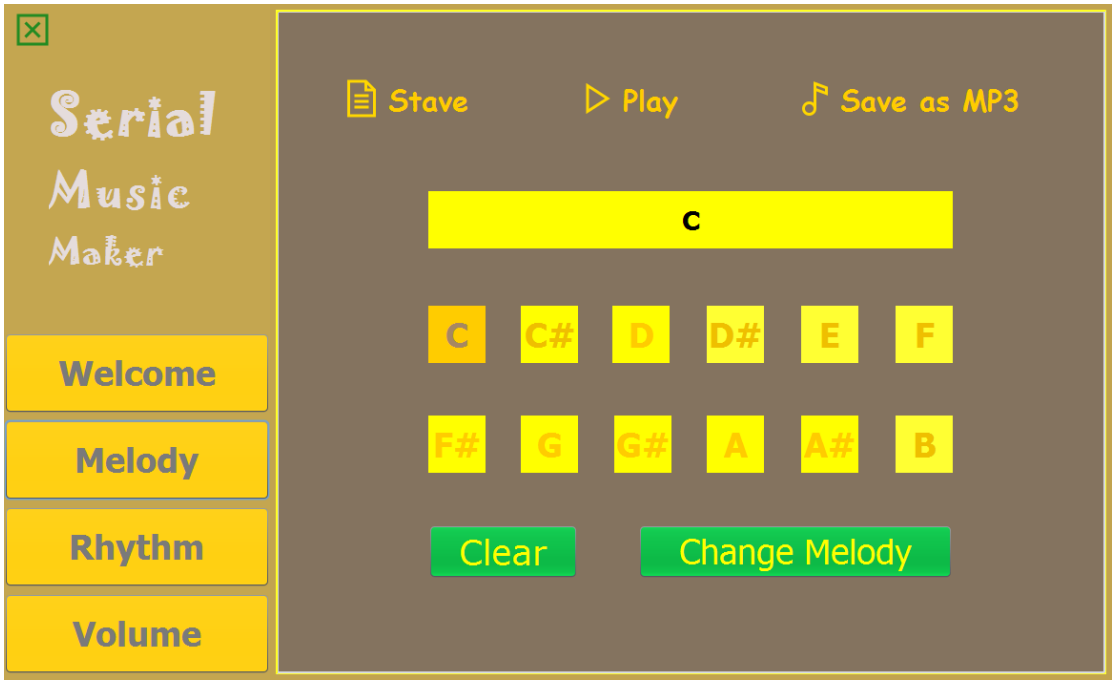


Figure 4 Melody View – mouse clicked on the pitch button

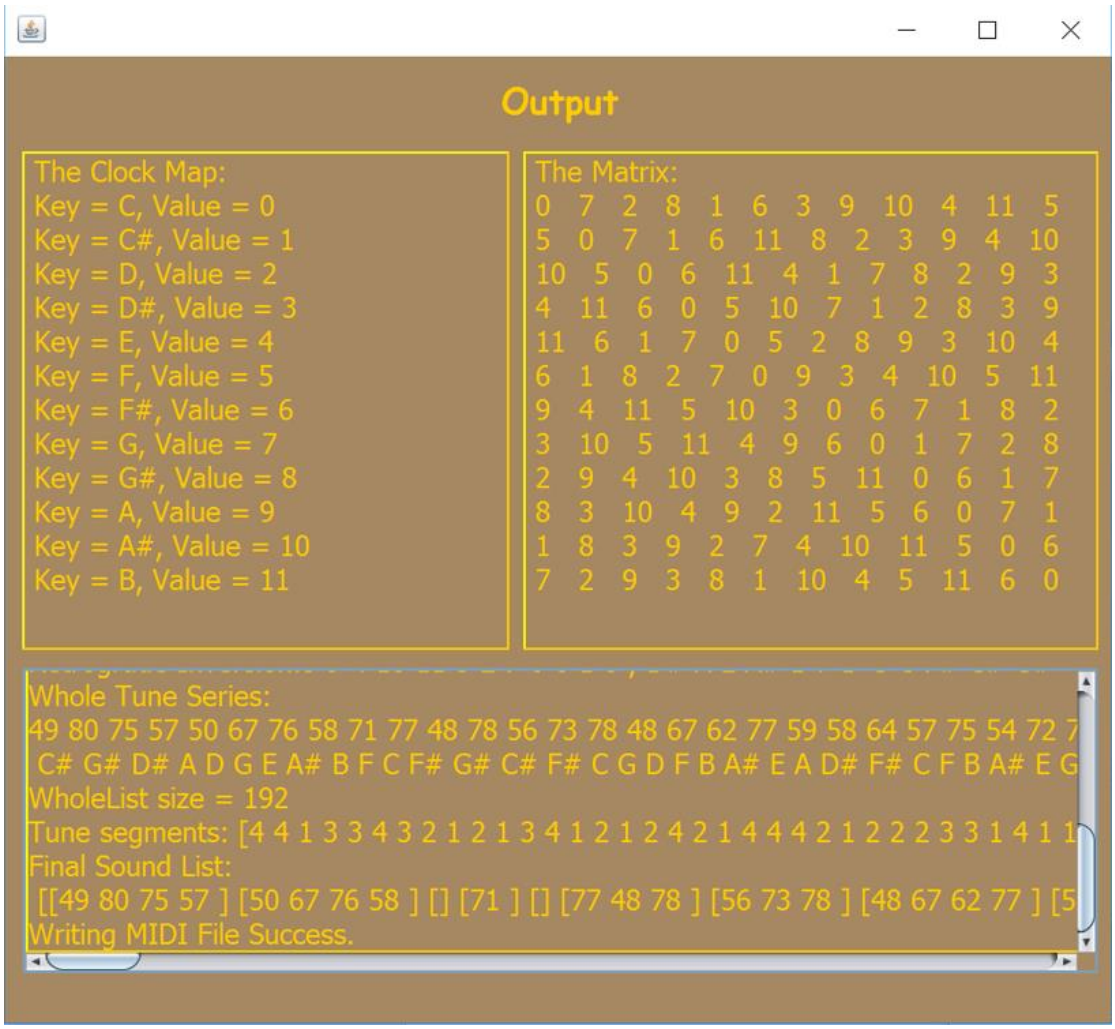


Figure 5 logFrame – for all log of the software including pitch clock and the matrix

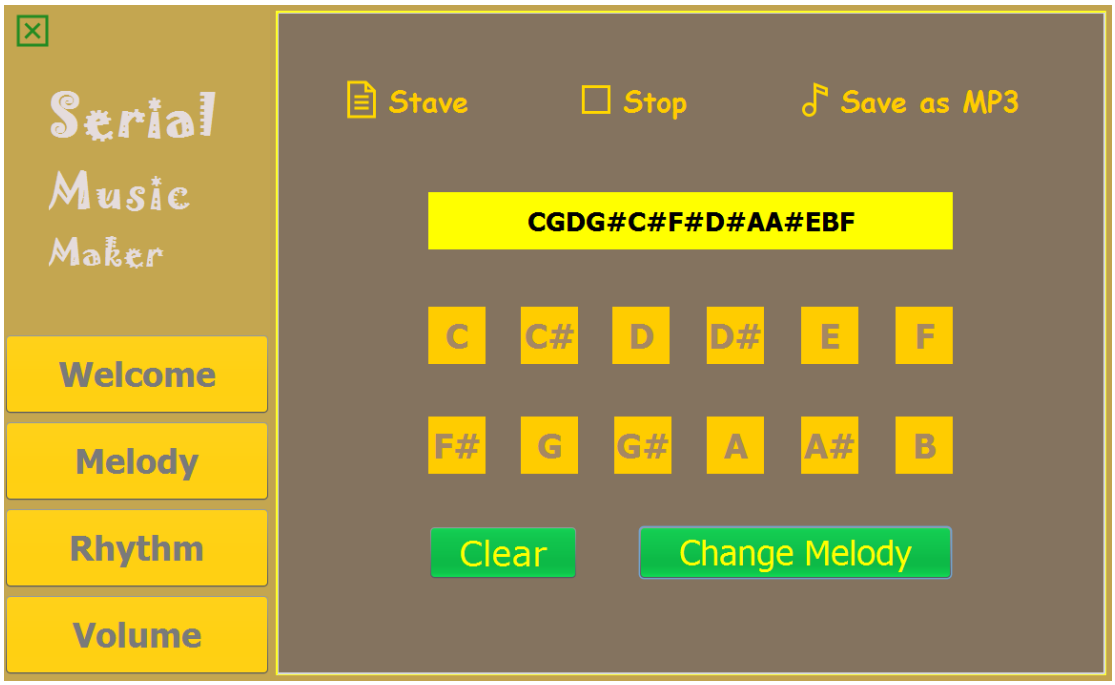


Figure 6 Melody View – after clicking on Play

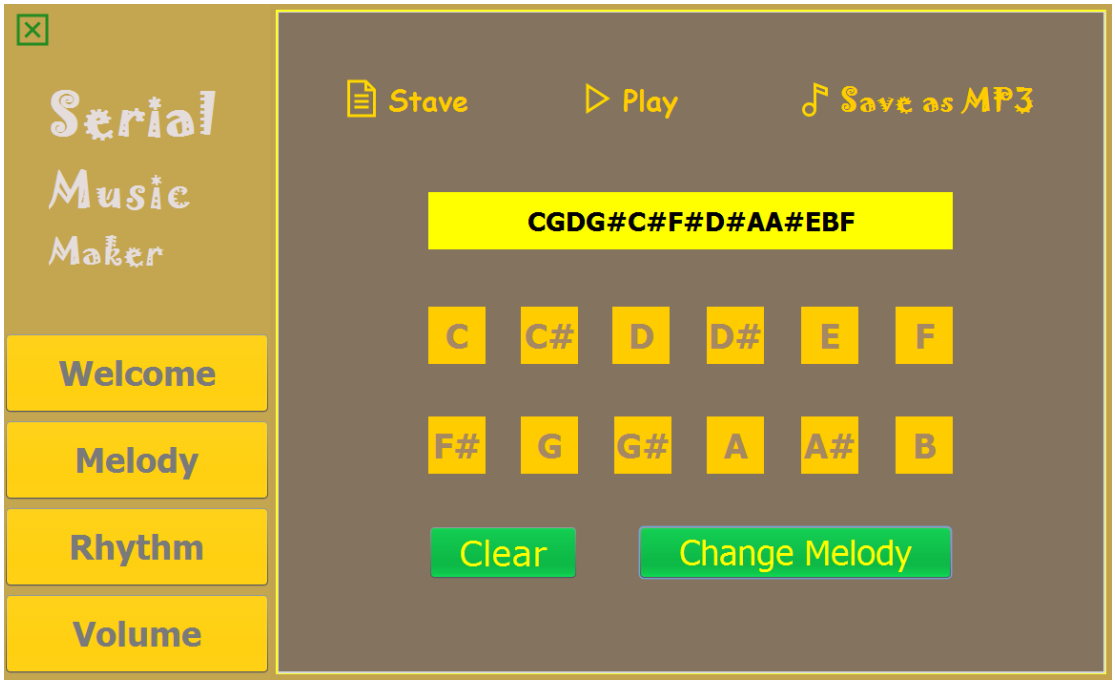


Figure 7 Melody View – mouse hovers on the “Save as MP3” button

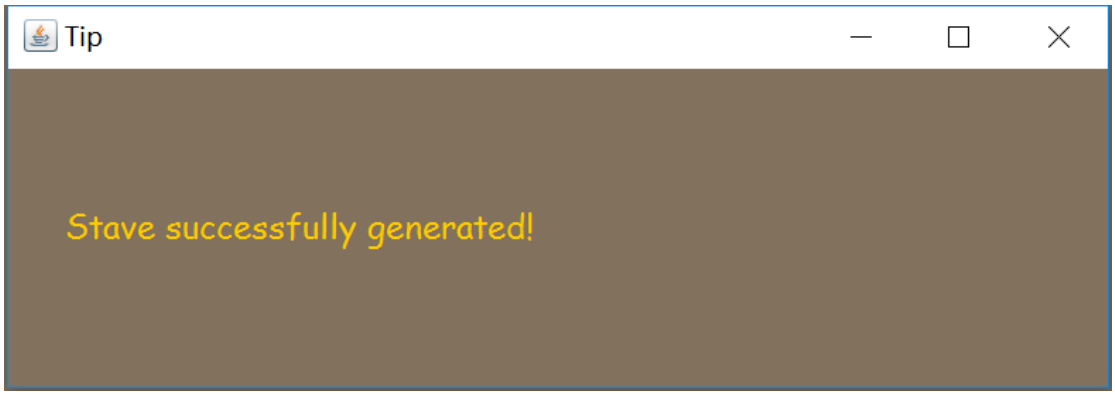


Figure 8 TipFrame – after the mouse clicked on “Stave” and the sheet is generated successfully.

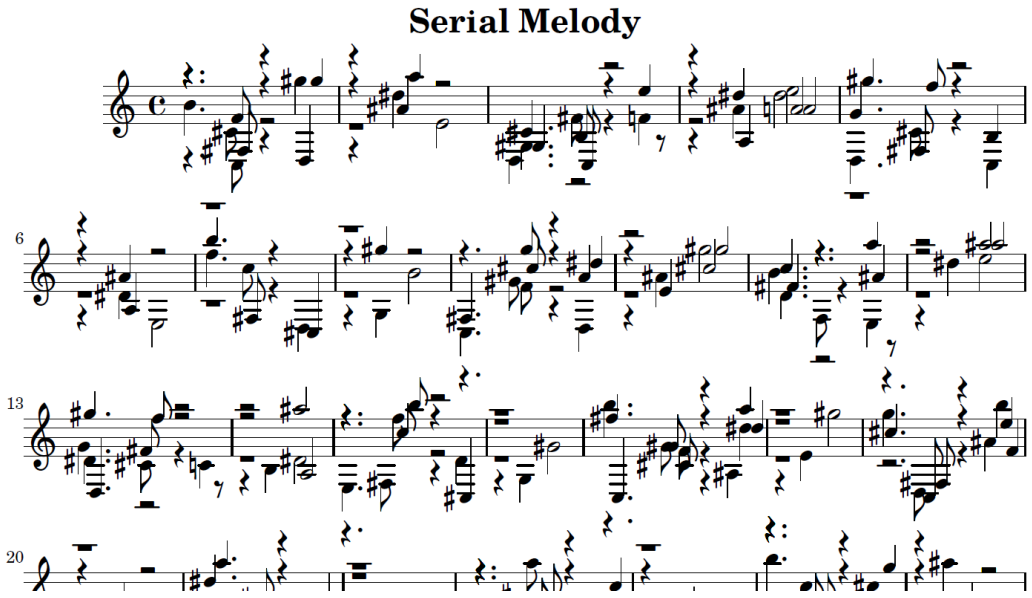


Figure 9 The PDF sheet music for Serial Melody

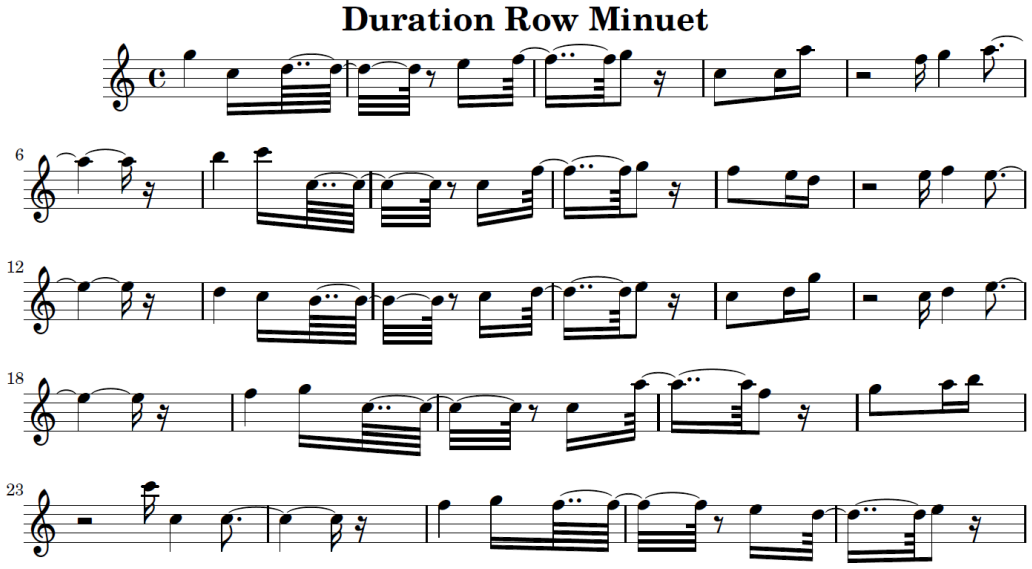


Figure 10 The PDF sheet music for Duration Row Minuet

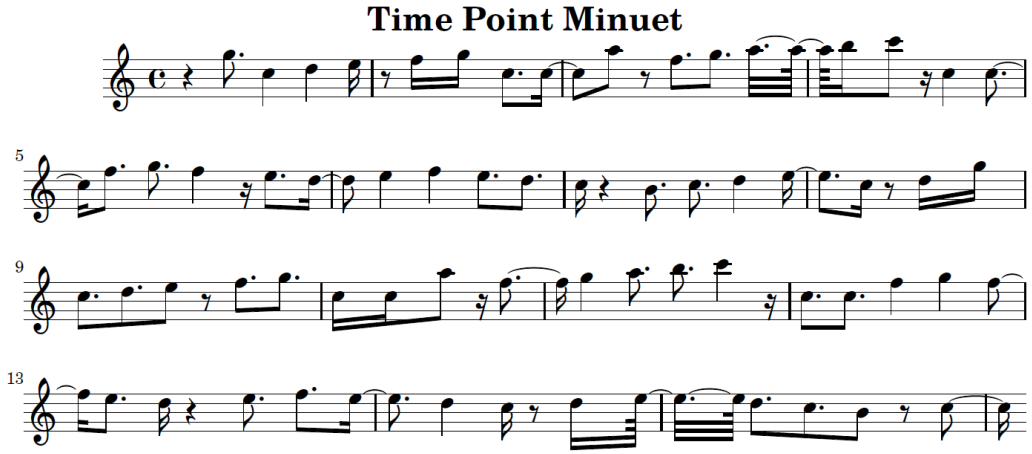


Figure 11 The sheet music for Time Point Minuet

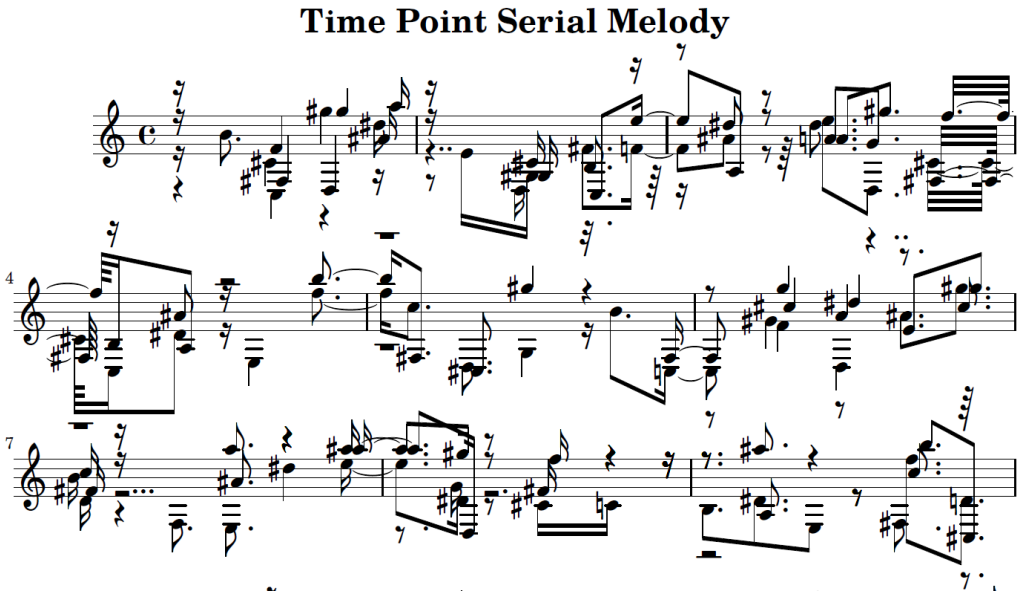


Figure 12 The sheet music for Time Point 12-tone Melody

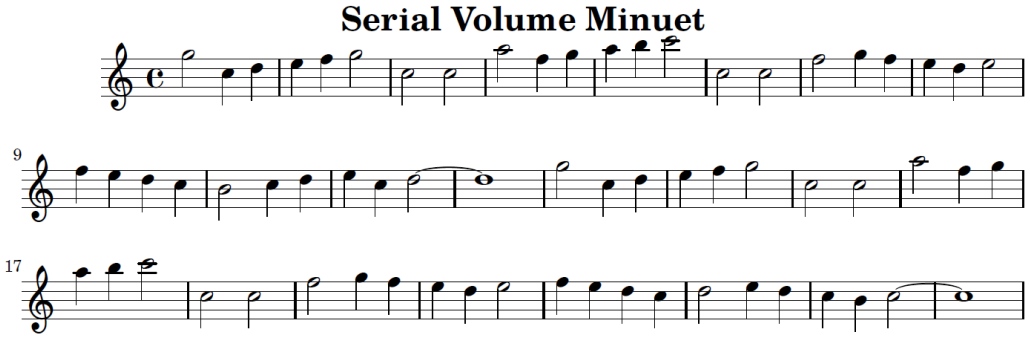


Figure 13 The sheet music for Serial Volume Minuet

(The display of volume marks is currently disabled)

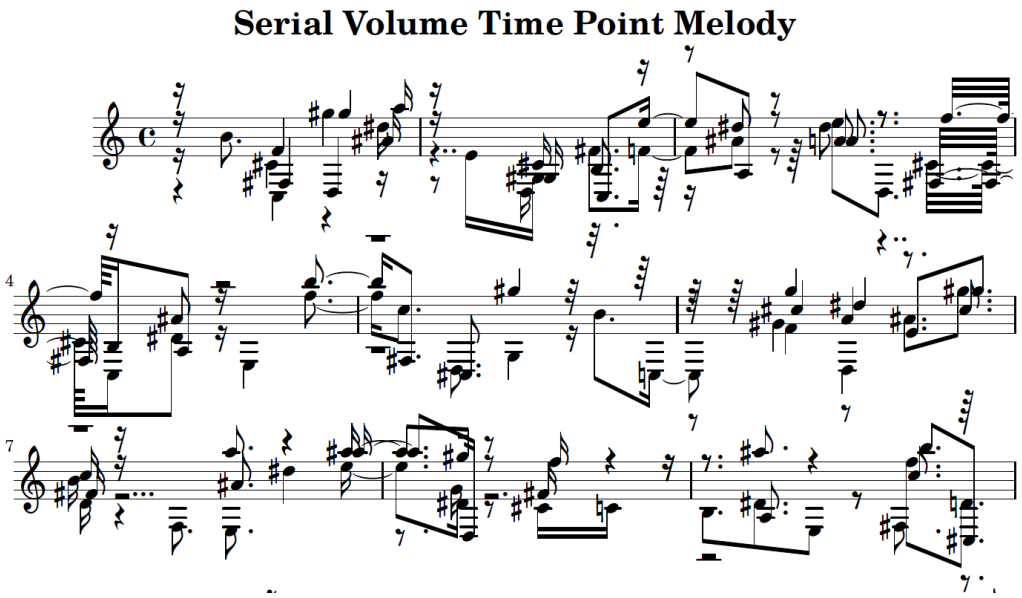


Figure 14 The sheet music for Serial Volume Time Point 12-tone Melody

(The display of volume marks is currently disabled)

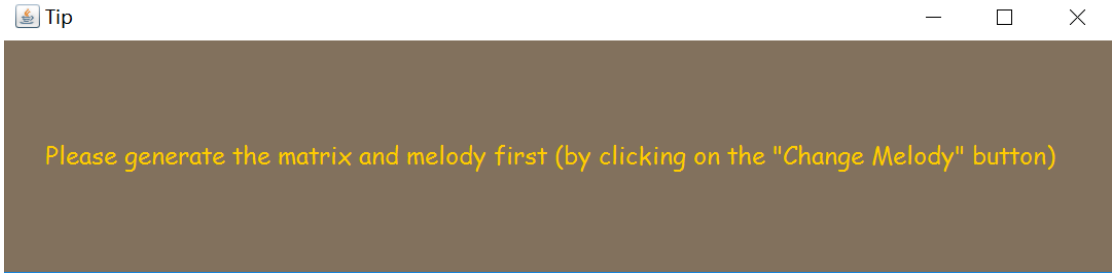


Figure 15 The tipFrame shown after clicking on “Play”/ “Stave” before “Change Melody”

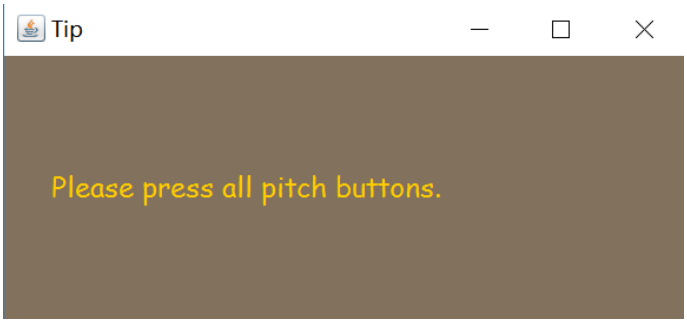


Figure 16 The tipFrame shown when the user clicks “Change Melody” before clicking all the pitch buttons

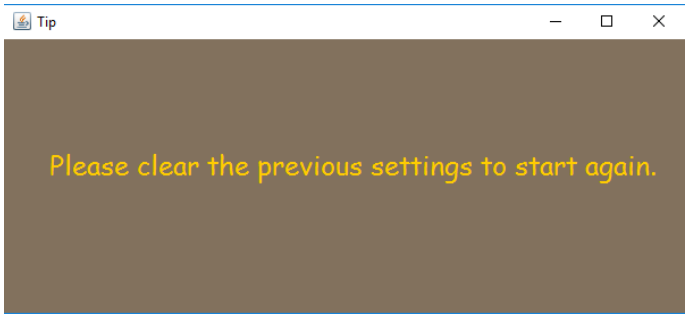


Figure 17 The tipFrame shown if the user wants to regenerate the melody (Clicking on “Change Melody” twice)

Figure 18 The tipFrame shown if any of the duration/volume boxes is empty

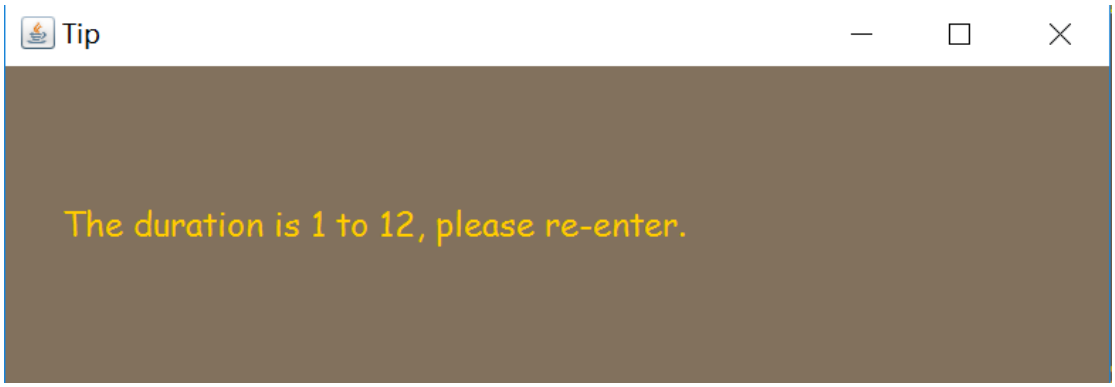


Figure 19 The tipFrame displayed if a duration number that exceeds range [1,12] is entered

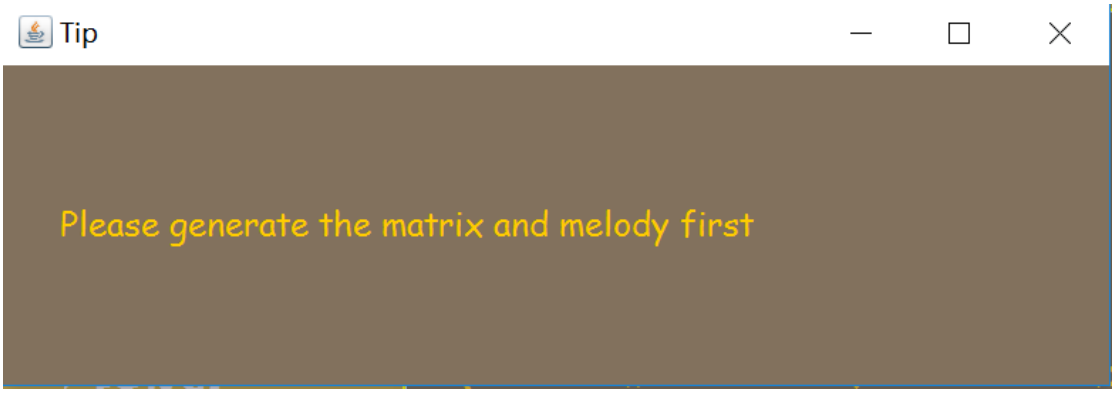


Figure 20 The tipFrame displayed after clicking on “Time Point 12-tone Melody” or “12-tone Melody with Rhythm and Volume Serialism” before clicking on “Change Melody”

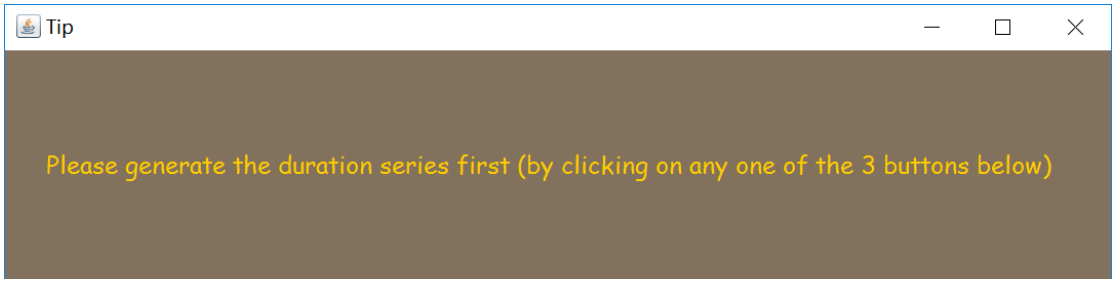


Figure 21 The tipFrame shown after clicking on “Play”/ “Stave” before “Time Point 12-tone Melody” is clicked

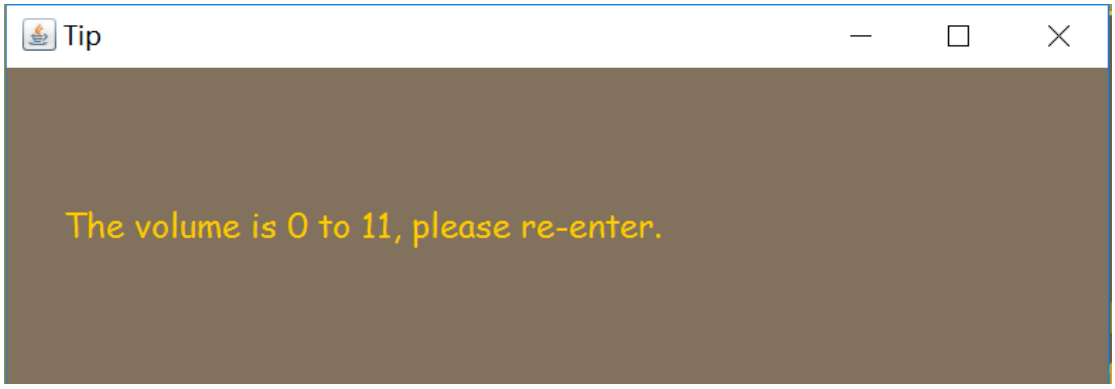


Figure 22 The tipFrame displayed if a volume number that exceeds range [0,11] is entered

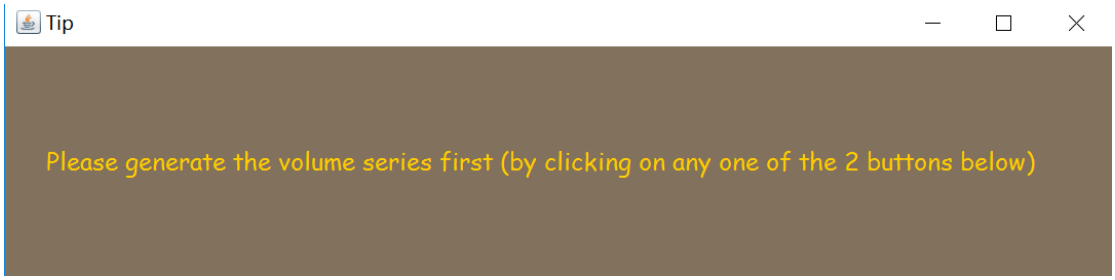


Figure 23 The tipFrame shown after clicking on “Play”/ “Stave” before “12-tone Melody with Rhythm and Volume Serialism” is clicked

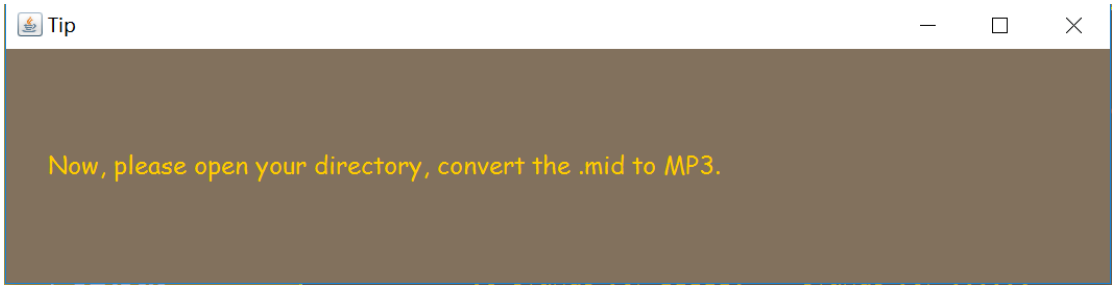


Figure 24 The tipFrame shown after clicking on “MIDI to MP3”

Appendix II - Full Code Listing

Please see the external ZIP file (Code.zip) for the full code.

Appendix III - The Original Design Documentation

Design Documentation

Contents

1. Summary of Proposal....2
1.1 Backgrounds, Aims and Objectives....2
1.2 Summary of Current Research and Analysis....3
1.3 Changes to the Original Proposal....5
2. Project Design....6
2.1 Description of Anticipated Components....6
2.2 Description of the Algorithms....6
2.3 Design of the User Interfaces....14
2.4 Description of the Evaluation of the System....15
2.4.1 The Criteria Used to Evaluate....15
2.4.2 The Approaches to Assess the Criteria....15
2.4.3 The Testing to be Done....15
2.5 Ethical Use of Data....15
3. Review against Plan....16
4. Bibliography....18

1. Summary of Proposal

1.1 Backgrounds, Aims and Objectives

The purpose of this design documentation is to introduce and document the designs of this project, including the design of the intended software and the evaluation.

Recall the Aim: The aim of this project is to develop a composition application software named "12-Tone Music Maker" for musician or researcher to generate a small piece of Schoenberg's 12-tone music using the original user-entered 12 tones' series.

Objectives

- To create a computer (Windows 10 System) composition application software.
- The software should have a graphical user interface.
- The software should implement some algorithms to transfer the original 12-tone series into a new piece of music which with different rhythm and volume being applied.
- The software should realise the MIDI output of the new music pieces.

From the initial specification document:

Essential Features

- The User Interface should enable the user to enter the original 12-tone series (using the Latin alphabet to represent the pitch class [C, C#, D, D#, E, F, F#, G, G#, A, A#, B], better have buttons for user to select from).
- The User Interface should have buttons for user to choose to different composing styles, such as "change melody only", "change melody and volume", "change melody, volume and rhythm".
- The algorithm should at least enable the alteration of 12-tone series order (as the serial ordering of tones is the basic intention of Schoenberg [1]), as well as divide each 12-tone into several segments (each contains 1 to 4 pitches), each segment of pitches will start to play at the same time.
- The output music piece should at least be a standard MIDI file (*.mid).

Desirable Features

- Implement algorithm for the volume alternation on the 12-tone series.
- Implement algorithm for rhythm alteration on the 12-tone series.

- ✧ Convert *.mid files to standard sound output files (*.mp3).
- ✧ Output the music score sheet of the new music piece.

1.2 Summary of Current Research and Analysis

Materials that have been read:

✧ **For how to construct the 12-tone matrix:**

CREATING A 12 TONE MATRIX **[5]**

This material clearly describes the steps and mechanism to construct a 12-tone matrix, which is the core part of 12-tone theory and helped me to generate an algorithm for 12-tone matrix construction.

- **For how to generate patterns using the 12-tone matrix:**

THE SERIALISM OF MILTON BABBITT **[6]**

This material describes many aspects of Milton Babbitt's research and contribution in the field of serialism, including how to apply the series from 12-tone matrix to a real piece of music. The picture below is an example from Schoenberg's Violin Concerto, Op36 which represent that *as long as the notes are appeared in the order of the given series (no octave difference), the durations can be random, and one can even combine two or more notes as a chord and play them simultaneously (See the blue part, the series is D C# G# C G F, while C# G# C [D♭ A♭ C] are played together).*

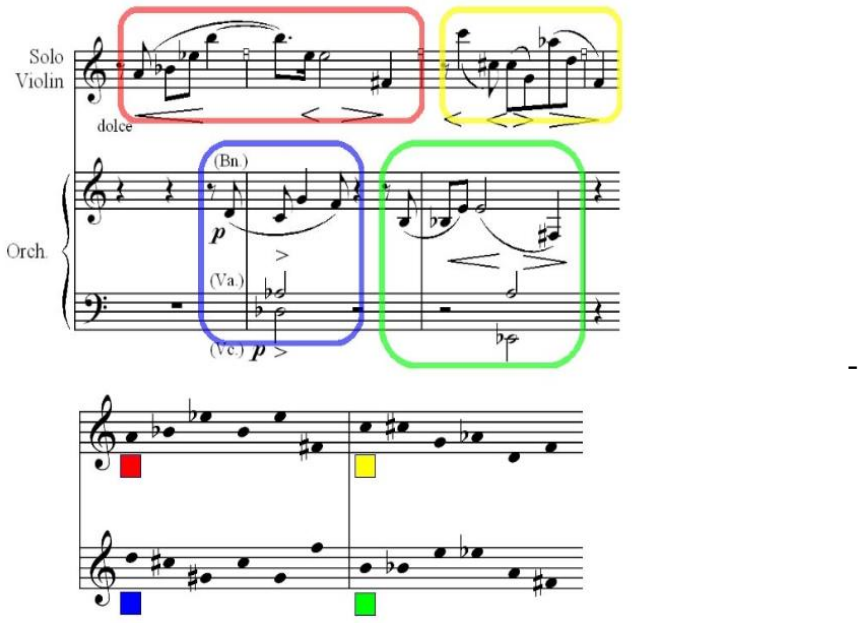
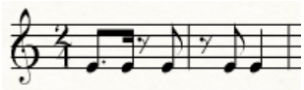


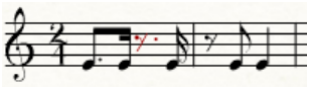
Figure 1:Opening of Schoenberg's Violin Concerto, Op. 36 – underlying structure [6]

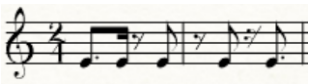
- **About Popular Rhythms**

The World's Most Popular Rhythm (Video) **[7]**

This video introduces one of the world's popular rhythms are called Clave and its derivatives.

Son Clave: 

Rumba Clave: 

Bossa Nova Clave: 

This gives an alternation of the serialised rhythm algorithm, which means that *before the rhythm algorithm Java method is developed, these traditional popular rhythms can be used to test the "Change Melody Only" function instead of using the dull beats with equal durations.*

✧ **For algorithm of rhythm alternation:**

Twelve-Tone Rhythmic Structure and the Electronic Medium 1962 [8]
Time-Point Sets and Meter [9]
THE SERIALISM OF MILTON BABBITT [6]
These materials introduce the way to construct a rhythmic serial system technically and help me with my rhythm alternation algorithm. Babbitt has developed two ways to construct the rhythm: The Duration Row System and The Time-Point System.

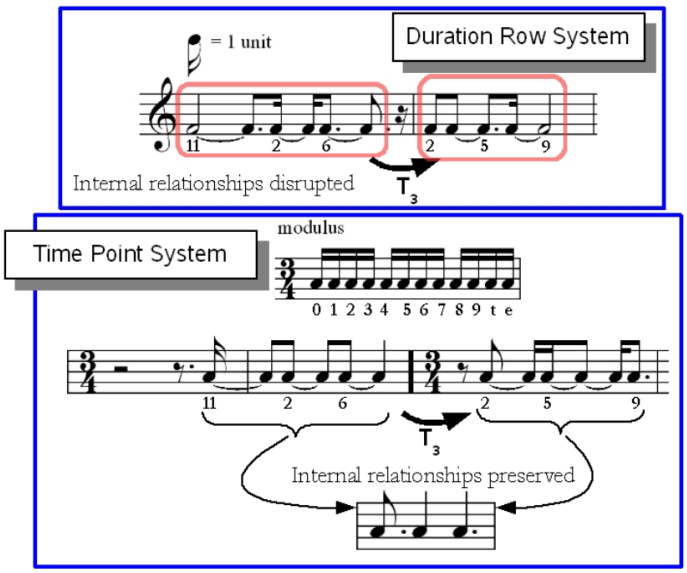


Figure 2: EXAMPLE FOR BABBITT'S RHYTHM SYSTEMS

In order to test this serial rhythm algorithm, I am now intended to add two buttons on the user interface called "Change Rhythm By Duration Row" and "Change Rhythm By Time Point" to test these two rhythm algorithms respectively. Also another input field should be added to the User Interface view to take in the original "three-note rhythm" number. Similar to the original pitch row of the 12-tone matrix, the "three-note rhythm" duration row is the original row to conduct rhythm serialisation using either "Duration Row System" or "Time Point System". More-note rhythm may be added in the future.

What have been tested:

- ✧ The construction the 12-tone matrix has been tested by creating a demo Java project, where I defined a Matrix Object by analysing the reading material and converted the concepts into formulas to assign value for each cell in the matrix.
- ✧ The MIDI sound output has been tested by importing javax.sound.midi.* in the demo Java project, reading midi file and creating thread to output sound.
- ✧ The basic construction of User Interface using NetBeans IDE 8.2 by creating JFrame using Java Swing, however, many functional issues need to be tested later.

1.3 Changes to the Original Proposal

The above reading and testing have generally confirmed the feasibility of the original proposal. The main changes to the original proposal demonstrated in the Specification will be

1. *Adding two buttons on the user interface called "Change Rhythm By Duration Row" and "Change Rhythm By Time Point" to test these two rhythm algorithms respectively.*
2. *Adding another input field to the User Interface view to take in the original "three-note rhythm" number.*

2. Project Design

2.1 Description of Anticipated Components

- ✧ User Interface – Theoretically, the user interface is everything which enable the interaction between user and the computer system. In this project, the user interface contains "12-Tone Music Maker" software front-end view, keyboard to enter, mouse to click on the buttons and audio output devices.
- ✧ Algorithm – The algorithms implemented by the back-end process for the software is the key point in this project. Using Java programming, methods for generating music patterns, combining patterns, distributing time-points values to each sound, distributing volume values to each pattern or single sound will be developed.
- ✧ MIDI Output – The new music pieces is intended to output as .midi files or .mp3 files, which needs the help of Java MIDI programming technology using Java Sound API.

2.2 Description of the Algorithms

- ➢ Melody Generation Algorithm:
  - Purpose:
    To generate music patterns and combine them into a small music piece.
  - Problems:
    Generate 12-tone matrix from the original user entered series;

    Use patterns in the 12-tone matrix to make a music segment;
    Set durations for each sound (in Son Clave pattern for example).
  - Existing Algorithms and Evaluation:
    The algorithm developed by Schoenberg for generating the 12-tone matrix is existed, which has been used in my testing.

    The algorithm is: for a given original 12-tone series, we first generate its Clock Map that match the first tone of the original series with number 0, and match the rest of the tones in the other of chromatic scale (C C# D D# E F F# G G# A A# B) with number 1 2 3 4 5 6 7 8 9 10 11. Then, we place the original series in the row 0 of the 12*12 matrix. Next, we can fill out the first column of the matrix by the formula:

$$Matrix\ [i][0] = 12 - Matrix\ [0][i]$$

Now we can fill out the rest cells of the matrix by the formula:

$$Matrix[x][y] = \begin{cases} Matrix[x][0] + Matrix[0][y] & (Matrix[x][0]+Matrix[0][y]<12) \\ Matrix[x][0] + Matrix[0][y] - 12 & (Otherwise) \end{cases}$$

Here is the test program:

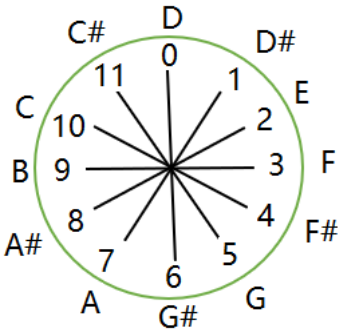
```
public class Matrix {

    public int[][] matrix;

    public Matrix(int[] list) {
        if(list.length!=12) {
            System.out.println("The original pitch row must contain 12 tones!");
        }
        else {
            this.matrix = new int[12][12];
            matrix[0]=list;
            for(int i=1; i<12; i++) {
                this.matrix[i][0]=12-this.matrix[0][i];
            }
            for(int y=1; y<12; y++) {
                for(int x=1; x<12; x++) {
                    if(this.matrix[y][0]+this.matrix[0][x]<12) {
                        this.matrix[y][x]=this.matrix[y][0]+this.matrix[0][x];
                    }
                    if(this.matrix[y][0]+this.matrix[0][x]>=12) {
                        this.matrix[y][x]=this.matrix[y][0]+this.matrix[0][x]-12;
                    }
                }
            }

        }
    }
```

Enter the original series: D C F# F C# D# A B G# A# G E

The output is:

```
The Clock Map:
Key = C#, Value = 11
Key = A, Value = 7
Key = A#, Value = 8
Key = B, Value = 9
Key = C, Value = 10
Key = D, Value = 0
Key = E, Value = 2
Key = F, Value = 3
Key = G, Value = 5
Key = G#, Value = 6
Key = F#, Value = 4
Key = D#, Value = 1
```

This Clock map each tone with an integer number, which can be transferred to:



And the integer representation of the original pitch class and the whole 12-tone matrix is:

```
The original row pitch class:
0.10.4.3.11.1.7.9.6.8.5.2.
The Matrix:
0,10,4,3,11,1,7,9,6,8,5,2,
2,0,6,5,1,3,9,11,8,10,7,4,
8,6,0,11,7,9,3,5,2,4,1,10,
9,7,1,0,8,10,4,6,3,5,2,11,
1,11,5,4,0,2,8,10,7,9,6,3,
11,9,3,2,10,0,6,8,5,7,4,1,
5,3,9,8,4,6,0,2,11,1,10,7,
3,1,7,6,2,4,10,0,9,11,8,5,
6,4,10,9,5,7,1,3,0,2,11,8,
4,2,8,7,3,5,11,1,10,0,9,6,
7,5,11,10,6,8,2,4,1,3,0,9,
10,8,2,1,9,11,5,7,4,6,3,0,
```

This matrix generation follows Schoenberg's approach, and the test program has correctly implemented this approach.

- Approach to be Used to Solve the Problem

  Algorithm to generate melody from the 12-tone matrix:

  Assume we want to make a small piece using only one row and one column with their Prime (left to right), Retrograde (right to left), Inversion (up to down) and Inversion Retrograde (down to up) patterns plus one duplication.

  Now the series is P, R, I, IR, P, R, I, IR with 12*8 = 96 notes in total. We can divide each of the 12 tones into segments of 1 to 4 notes (as any random chord with 5 or above notes will be a "mess" and hard to distinguish, so we limit it to 4), each segment will make a chord with notes being played simultaneously. We can do this by generating random integer numbers.

  Then, the Son Clave rhythm can be applied on this 96 notes' piece. Take the sixteenth note as a durational unit, the Son Clave can be represented as 3 1 2 2 2 2 4 durations (- is the duration for a pause).

  Here is the pseudocode:

  Melody_Generation (Matrix m)

  //Randomly choose a row and a column.

  r ← random (0 to 11)

  c ← random (0 to 11)

  Let new array rArr ← m[r]

```
Let cArr[12] be a new array

For i from 0 to 11
    cArr[i] ← m[i][c]
End Loop
//Generate array for Prime, Retrograde, Inversion and Retrograde Inversion
Let new array prime ← rArr
Let new array inver ← cArr
Let retro[12] be a new array
For i from 0 to 11
    For j from 11 to 0
        retro[i] ← rArr[j]
    End Loop
End Loop
Let reIn[12] be a new array
For i from 0 to 11
    For j from 11 to 0
        reIn[i] ← cArr[j]
    End Loop
End Loop
//Generate the whole list of tune series
Let new List wholeList ← (prime + inver + retro + reIn)*2
/*details are ignored here*/
//Divide prime, inver, retro, reIn into segments
Let segments be a list of integers //List of the number of notes in each segment
Let n ← 12 //The rest note number in an array
For i from 1 to 8 //There are 8 twelve-tone series patterns, so loop 8 times
    While (n > 0)
        Let x ← random (1 to 4)
        n ← n – x;
        If (n >= 0)
            segments.add(x)
        Else
            n ← n + x
    End Loop
End Loop
//Construct a list of sounds each of which is a list of notes
Let allList be a list of ArrayList
For i from 0 to segments.size() - 1
    Let ArrayList list ← []
    For j from 0 to segments.get(i) - 1
        list.add(wholeList.get(j))
    End Loop
    wholeList.removeRange(0, segments.get(i))
    allList.add(list)
```

```
End Loop
//Apply Son Clave (3122224) to the series in allList
/*Add pause as empty ArrayList [] to allList. The pause in Son Clave is in the 3rd, 5th, 10th, 12th, 17th,… position, they divide the sound as segments with 2, 1, 4, 1, 4, 1, 4,…sounds.*/
allList.add(2, [])
Let x ← 3
Let count ← 0 /* If count is odd, the next pause position will jump 2 indexes, otherwise jump 5 indexes*/
While (x < allList.size())
    count ← count + 1
    If (count%2==1) then
        x ← x + 2
    Else if (count%2==0) then
        x ← x + 5
    End If
    allList.add(x, [])
End Loop
```

Now we can match the durations in the order of 3 1 2 2 2 2 4 3 1 2 2 2 4… to the sound (ArrayList) in the allList when implementing the MIDI output.

- Methods for Analysing the Algorithm

  By output all the middle results generated in the above Melody_Generation Algorithm, such as the rArr, cArr, prime, retro, inver, reIn, wholeList, segments and allList, the correctness of the algorithm can be easily analysed.

➢ Serialised Rhythm Algorithm:

- Purpose:

  To generate a serialised rhythm using Babbitt's method.

- Problems:

  Use the user entered "three-note rhythm" to conduct rhythm serialisation using "Duration Row System" or "Time Point System".

- Existing Algorithms and Evaluation:

  Only some descriptions of Babbitt's rhythm theories can be found.

- Approach to be Used to Solve the Problem:

Similar to the durations list in the Melody_Generation Algorithm, here, a durations list can also be construct by take in the original duration row first and do T3 operation (add each number by 3, and extract by 12 if it is bigger than 12) recursively to construct the whole durations list and assign them to each sound. Again, we set the 16th note as the durational unit.

Pseudocode:

Duration_Row (Array trithm, int len) /*trithm: tri-rhythm, the user input original row, integer len is the size of the allList, the number of sounds

//Construct durations list

Let durations be a new List of Arrays with three elements

durations.add(trithm)

For i from 3 to len-1

    For j from 0 to 2

        trithm[j] ← trithm[j] + 3

        if trithm[j]>12 then

            trithm[j] ← trithm[j] – 12

    End Loop

    durations.add(trithm)

End Loop

For “Duration Row System” approach, every number in the rhythm row stands for the exact duration of each sound. Each duration row has 3 durations, while after each 3 durations, if there are still some durations left before the next bar-line, the left part will be a pause. We can map each sound duration with the number in each Array in the durations list and fill the rest of the bar-line with pause.

For “Time Point System” approach, what is different is that the numbers in the duration row do not stand for the duration of each sound but the duration between the sound and its previous bar-line. As a result, the

actual duration of that sound is the modular 12 of the duration rows, which preserves the internal relationship of the original row (the time signature could be stable 3/4 or 6/8).

Duration_For_Time_Point(List<Array> durations)

//Construct the time point duration from "durations"

Let tpDuration be a new List of Arrays with three elements

For x from 0 to durations.size()-1

Let arr [4]← durations.get(x) + [12]

Let temp be a new array for 3 integers

For y from 0 to 2

If arr[y+1] – arr[y]>0 then

temp[y] ← arr[y+1] – arr[y]

Else

temp[y] ← arr[y+1] – arr[y] + 12

End If

End Loop

tpDuration.add(temp)

End Loop

The number in the arrays in the tpDuration will be the actual durations for each sound under Time Point System.

- Methods for Analysing the Algorithm

  Output all the results in the algorithm including all elements in durations in **Duration_Row Algorithm** and the tpDuration in **Duration_For_Time_Point Algorithm** and analyse the correctness.

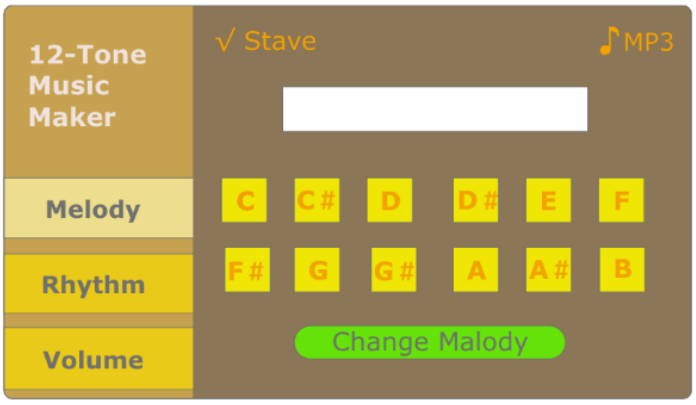
➢ Serialised Volume Algorithm:

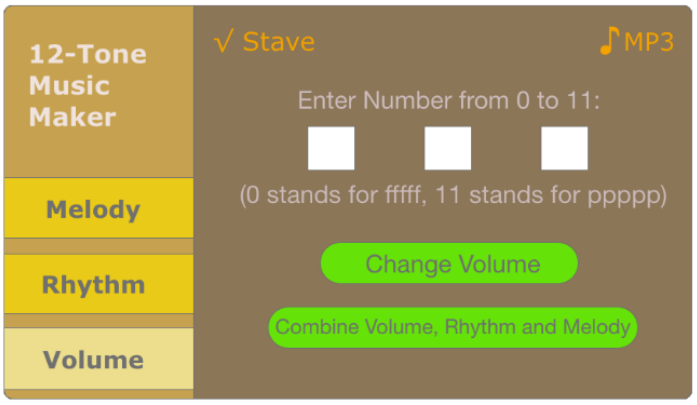
According to Babbitt's another contribution, the volume of the music can also be serialised, but the actual method is not clear by now. Maybe we can assign 0 to 11 to the volume mark "fffff", "ffff", "fff", "ff", "f", "mf", "mp", "p", "pp", "ppp", "pppp", "ppppp", and do some calculation like duration row system.

The dynamic combination of these algorithm may implement by MIDI output configuration.

2.3 Design of the User Interfaces

Apart from all the hardware user interfaces, the software views are shown by the following storyboards:





2.4 Description of the Evaluation of the System

2.4.1 The Criteria Used to Evaluate

Because the software is intended for academic or technical research use, the most important function is the correctness of the algorithm but not for public appreciation, and evidences have shown the dissonance character of this kind of music. Therefore, the evaluation criteria will only contain: how many functions are satisfied and how well each function is satisfied, which relates to the correctness and efficiency of the algorithm, as well as the Java Programming skills developed in this project.

2.4.2 The Approaches to Assess the Criteria

Check the correctness of the algorithms by analysing both console outputs and the generated music staves. Check the number of functions that have been accomplished.

2.4.3 The Testing to be Done

Check If: The components on the UI all work well

The algorithms all work well

The MIDI output can be implemented well

The music sheets can be made

The MIDI file can be converted to MP3

1.1.1 The Expected Conclusion

All the above functions can be achieved, and improvement can be made if the algorithm for volume serialism is developed.

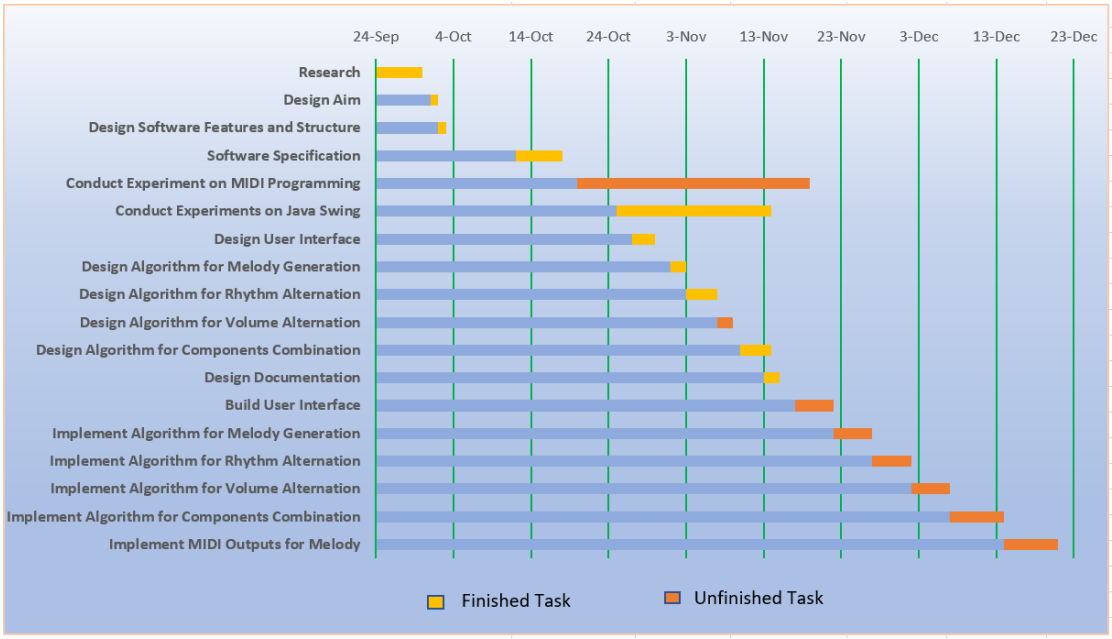
2.5 Ethical Use of Data

This project will only use real human data from the developer. No 3rd party evaluation will be used. The CS Department Ethical Procedure for Comp39x Projects 3rd Party Evaluation is confirmed to be followed.

3. Review against Plan

New Gantt Chart

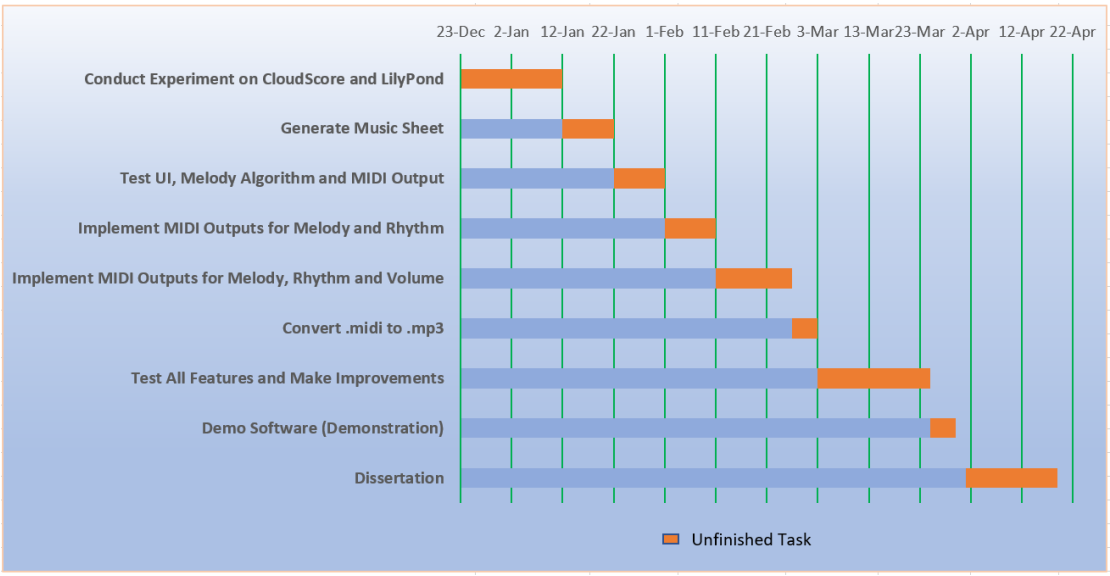
24 Sep 2018 ~ 23 Dec 2018:



Changes made in this period:

1. The experiment on MIDI Programming has started on 20 Oct, as I started to learn to use Java Sound API， but I can only implement the audio output of an existing .midi file. Experiments on audio output of a programmed sound piece still need to be done. The rest state of the tasks before Design Documentation is set to “Finished”.
2. The durations of the experiment on Java Swing, the implementations of Algorithms for Melody Generation/Rhythm Alternation/Components Combination and the implementation of MIDI Output for Melody have also been extended, due to the consideration of the complexity of these tasks.
3. Another task “Build User Interface” is added after the finish of Design Documentation, because the original plan to build Interface together with the Java Swing Experiment is not feasible.

23 Dec 2018 ~ 19 Apr 2019:



Changes made in this period:

The start date of each task is put forward to adjust with the duration prolongs of the previous period. On the other hand, because the Demonstration week will start on 25 Mar 2019, all the time before that can be used to improve the final software, some durations of the tasks are also extended.

4. Bibliography

[1] The Editors of Encyclopaedia Britannica. *Serialism: MUSIC*. (4 Feb 2014). [Online] Available: https://www.britannica.com/art/serialism

[2] Hybrid Pedagogy Inc., (2012 - 2018). *Pitch (class)*. [Online] Available: http://openmusictheory.com/pitch(Class).html

[3] Dr. Thomas Pankhurst *Introduction to Tonality*, [Online] Available: http://www.tonalityguide.com/introoverview.php

[4] George Perle (1915-), *Serial Composition and Atonality: An Introduction to the Music of Schoenberg, Berg, and Webern by George Perle*, (1991) [Online] Available:https://books.google.co.uk/books?id=4C8RjEaBRf4C&printsec=frontcover&dq=0520052943&hl=en&sa=X&ved=0ahUKEwj7v9e1-s7eAhXCDcAKHd5FDpYQ6AEIUzAH#v=onepage&q&f=false

[5] James Gholson. *CREATING A 12 TONE MATRIX: "My music is not modern, it is merely badly played." A. Schoenberg.* [Online] Available: https://unitus.org/FULL/12tone.pdf

[6] Paul Riker. *THE SERIALISM OF MILTON BABBITT.* [Online] Available: http://paulriker.com/words/The_Serialism_of_Milton_Babbitt.pdf

[7] Patreon: https://www.patreon/12tonevideos. *The World's Most Popular Rhythm.* [Online] Available: https://www.youtube.com/watch?v=Ye7d5mPNfYY

[8] Stephen Peles, Stephen Dembski, Andrew Mead and Joseph N. Straus: The Collected Essays of Milton Babbitt. *Twelve-Tone Rhythmic Structure and the Electronic Medium 1962.* (Available on 10 May 2018). [Online] Available: http://www.jstor.org/stable/j.ctt7rfx5.17

[9] William Marvin Johnson, Jr: Perspectives of New Music, Vol. 23, No. 1 (Autumn - Winter, 1984), pp. 278-293. *Time-Point Sets and Meter.* (Available on 10 May 2018). [Online] Available: http://www.jstor.org/stable/832917