Making Serial Music Based on Schoenberg's 12-tone Theory with Java

Specification & Design Document

By Ziwei. Lin

15th November 2018

Supervisor: Prof. Paul. E. Dunne

Contents

Project Specification

1. Project Description

1.1 Claim

This project is being done as part of the requirements for year three's degree award but not on behalf of any individual.

1.2 Aim of the Project

To simplify the exploration and research of the 20th-century music style especially the serial music of Schoenberg's for academic and technical purposes, we need a visualised way to introduce the composing mechanism under the serialism. Therefore, the purpose of this project is to develop a PC-based (Windows system) composition application software named "12-Tone Music Maker" 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 tones' series with the alternation of other elements such as rhythm and volume, and output the new piece of music.

1.3 Proposed Solution

As many musical and historical knowledges are required for this project, loads of background materials need to be read and understood first. While the software development will be based on Java programming, which will contain: use Java Swing to create the software GUI; perform algorithms to transfer a random entered 12 tones' series into a whole piece of 12-tone music; use Java Sound API to conduct MIDI programming and output the soundtrack.

1.4 Data to be Used

All data used in this project will only come from the developer.

2. Statement of Deliverables

2.1 Description of Anticipated Documentation

The anticipated documentation for this project contains this project specification, the design document, the interim report, and the final report. This specification will mainly introduce the features and structure of the anticipated software as well as some necessary music theories and components. The evaluation of the software, the plan for the whole project will also be described latter.

2.2 Description of Anticipated Software

The target software is consisted of three parts mainly: the design of the graphical user interface, the implementation of several algorithms (to transfer the original 12-tone series into a new piece of music which combines new 12-tone series with different rhythm and volume), as well as the realisation of MIDI output (to translate the digit figures into sounds, generate a complete soundtrack and output it).

2.2.1 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).

2.2.2 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.

2.2.3 Components and Technologies

- User Interface – designed by Java Swing.
- Key Algorithm for generating music patterns, combining patterns, distributing time-points values to each sound, distributing volume values to each pattern or sound – Java programming.
- MIDI Output – use Java Sound package.

2.3 Description of Anticipated Experiments

The anticipated experiments will contain the test of Java Swing features using NetBeans, the test of implementation of Java MIDI programming features (especially the conversion between number and tone, the output of sound), the test of music sheet generation using ScoreCloud and LilyPond.

2.4 Description of Methods for Evaluation of the Work

Because the core purpose of the "12-Tone Music Maker" is to implement the 12-tone theory which is on a very technical level, the individual opinion for the output appears to be unreasonable or meaningless as the music will sound unpleasant generally. The correctness of the output depends on the pitch order, so the best way to evaluate is to analyse the stave of the music piece to see if it follows the 12-tone composing strategy. ScoreCloud and LilyPond could be used, the former has the feature of generating music sheet from a recorded sound and the latter by text input. No 3rd party evaluation will be used. The CS Department ethical procedure for Comp39x projects 3rd party evaluation Procedure is confirmed to be followed.

3. Conduct of the Project and Plan

3.1 Preparation

3.1.1 Music Theories

- **The Pitch Class**

A pitch is a tone with its individual frequencies, and each pitch is given an alphabetical name to distinguish, which does not imply octave equivalence (means C1 is difference from C2). While a pitch class is pitches under octave equivalence (means C1 is regard equally to C2, C3…). [2]

- **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 in each of the chromatic scale in western music culture. [3]

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". [3]

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

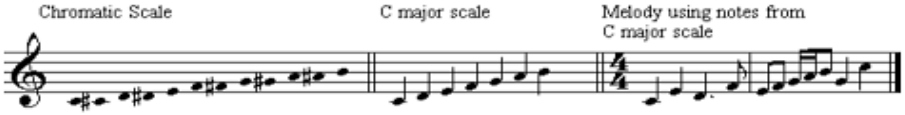


Figure 1: THE CHORDS C MAJOR [3]

- **Serialism and 12-Tone Music (Atonality)**

In contrast to tonal music, the general idea of serialism is to repeat a pattern over and over, which can be applied to different music components like the serialism of rhythm, volume and melody. The 12-Tone Music which is developed by Arnold Schoenberg during the pre-20th-centry is the most typical implementation of serialism [1]. The main concept of 12-tone music is to make all 12 notes equal (no note in a chromatic scale is more important than any other). Schoenberg's approach is to govern linear ordering of a given 12 notes of the semitonal 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 in a pattern before the next pattern, and some notes in a pattern can be grouped together to make a chord [4].

3.1.2 Knowledge Learning

- **Knowledges that already known:**

- Basic knowledge of music including terms and notations.
- Object-oriented technique for programming in Java.
- The algorithm to generate the matrix for 12-tone series.

- **Still need to learn:**

- Read materials and study the theories for serialism and Schoenberg's 12-tone, including its derivative use in other dynamics of music components.
- Study the Java Swing technique for GUI design
- Java MIDI program (totally new).
- How to use ScoreCloud or LilyPond to generate music score sheet for the programmed music piece (totally new).

3.2 Design Stage

- Design Method
  Draw prototype for UI design. Create demo of the 12*12 matrix construction, and design algorithms for each feature.
- Design Documentation
  The design will be documented by storyboards of UI, pseudocode or Java code for each algorithm.

3.3 Implementation Stage

- Hardware：The 12-Tone Music Maker software is developed on a laptop with Windows 10 operating system, and is expected to run on a Windows10 PC device enabling audio input and output.
- Software: NetBeans IDE 8.2, ScoreCloud/LilyPond.
- Testing: Component testing (test each component using real data from the developer), integration testing, white box testing and interface testing.

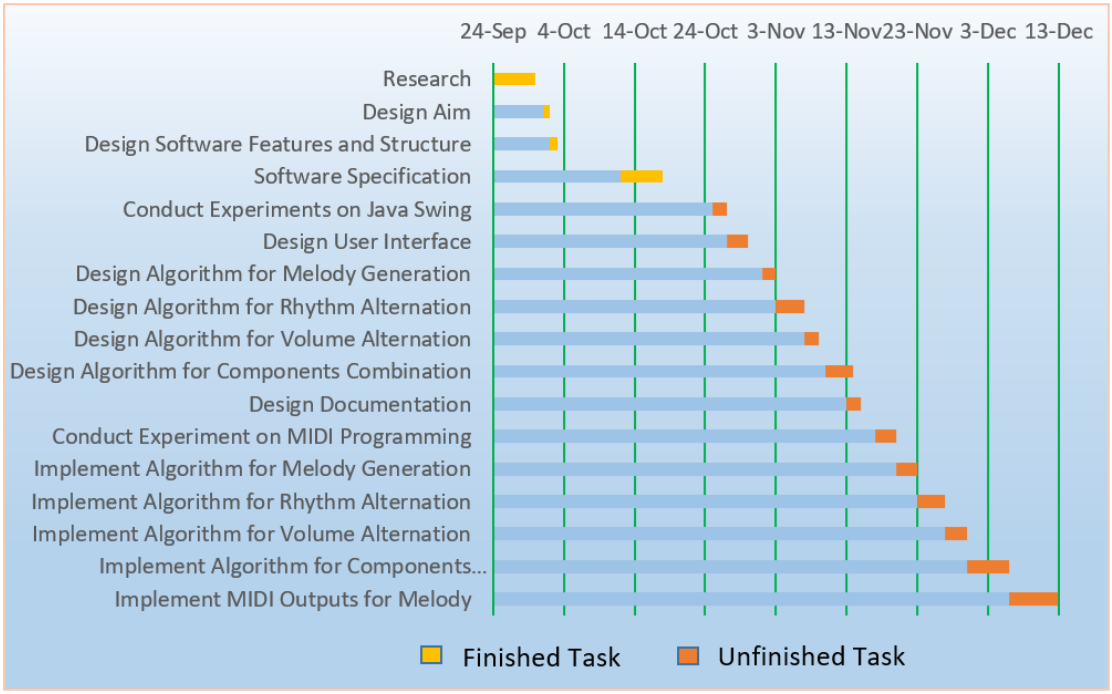
3.4 Risk Assessment

- Major Challenges: Java Swing coding to implement UI functions; Implement the algorithm for time-points and combine different alternations together; Generate MIDI file and MP3 file for the new music pieces; Unsure about the feasibility and reliability of generating music score sheets.

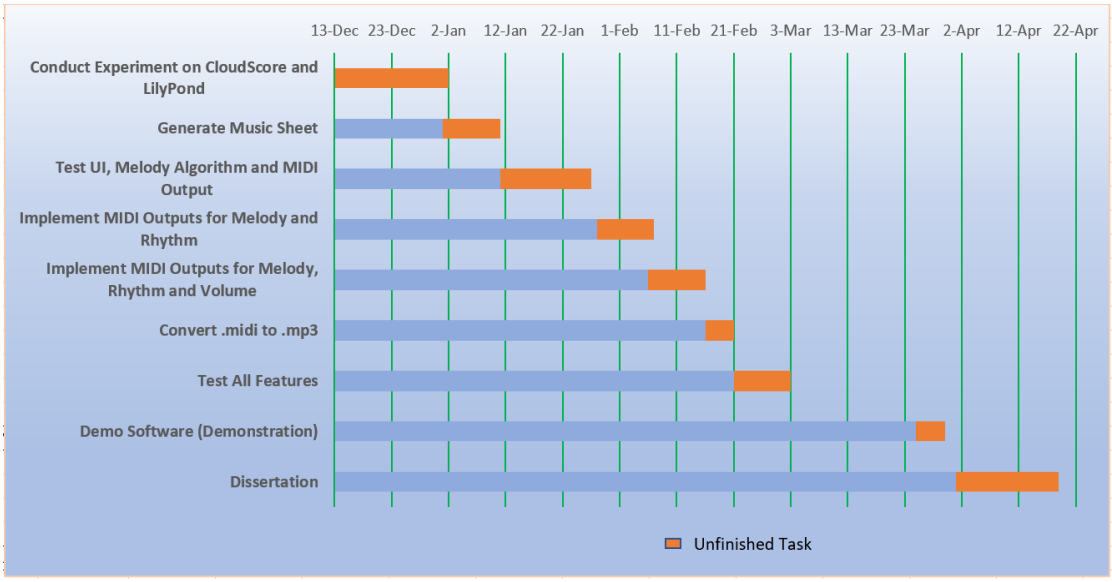
- New Skills: Use NetBeans to design UI; Java MIDI Programming; Use ScoreCloud or LilyPond to generate music sheets.

3.5 Project Plan (Gantt Chart)

From 24/09/2018 to 13/12/2018:



From 13/12/2018 to 22/04/2019:



Design Documentation

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).*

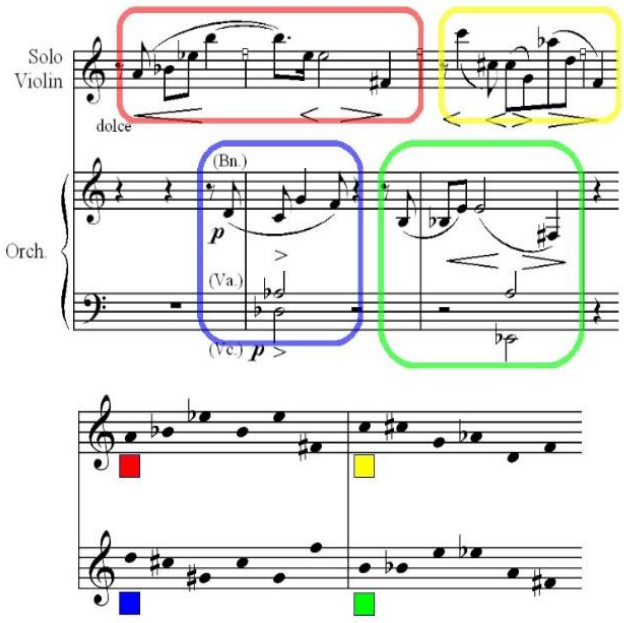


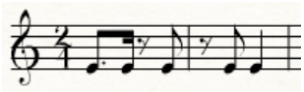
Figure 2:Opening of Schoenberg's Violin Concerto, Op. 36 – underlying structure [6]

- **About Popular Rhythms**

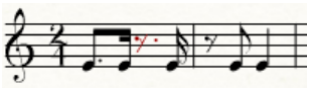
The World's Most Popular Rhythm (Video) **[7]**

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

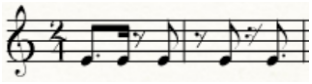
Son Clave:



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.

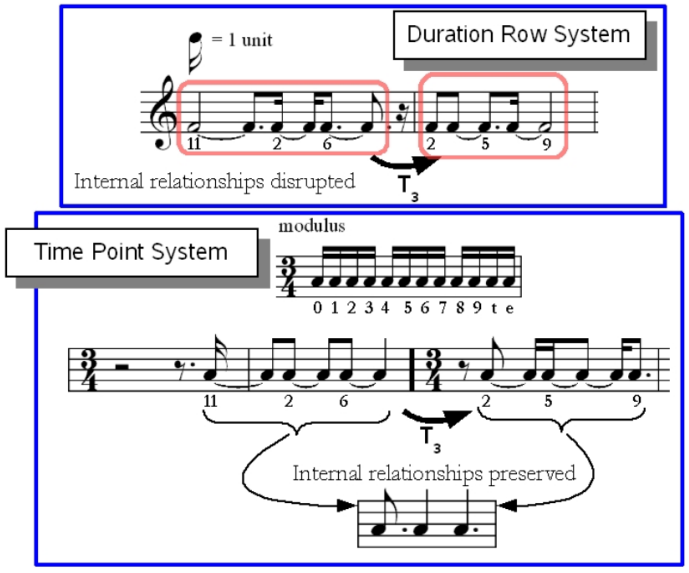


Figure 3: EXAMPLE FOR BABBITT'S RHYTHM SYSTEMS [6]

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 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:

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

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

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

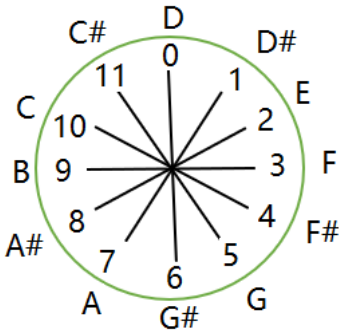
Here is the test program:

```
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;
                    }
                }
            }

        }
    }
```

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
Let new List wholeList ← (prime + inver + retro + reIn)*2 /*ignore the details here*/
//Divide prime, inver, retro, reIn into segments
Let segments be a list of integers //List of the number of notes in each segment
For i from 1 to 8 //There are 8 twelve-tone series patterns, so loop 8 times
    Let n ← 12 //The rest note number in an array
    While (n != 0)
        Let x ← random (1 to 4)
        segments.add(x)
        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.

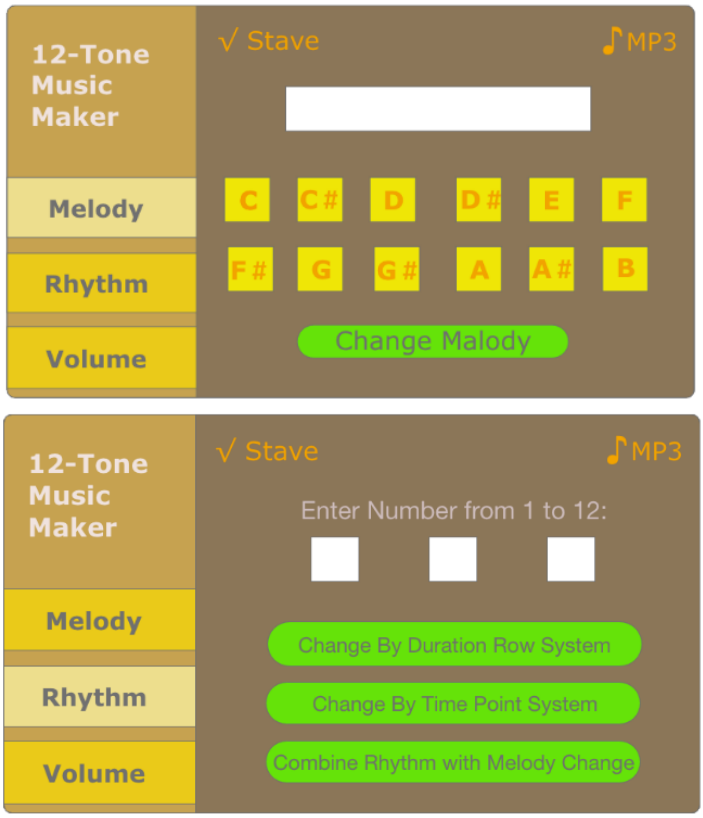
➢ Serialised Volume Algorithm:

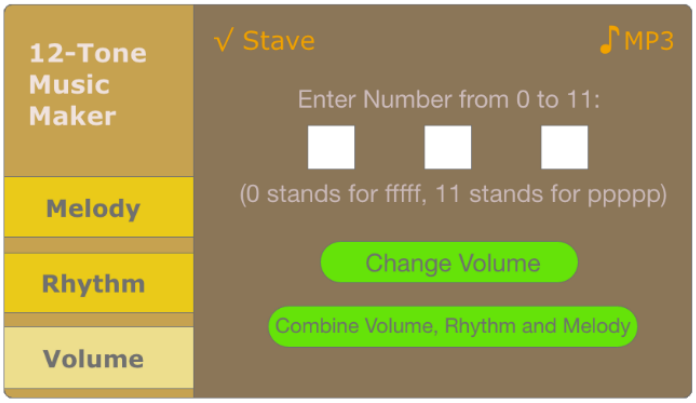
According to Babbitt's another contribution, the volume of the music can also be serialised, but the actual method is not clear by now. Maybe we can assign 0 to 11 to the volume mark "fffff", "ffff", "fff", "ff", "f", "mf", "mp", "p", "pp", "ppp", "pppp", "ppppp", and do some calculation like duration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

2.4.4 The Expected Conclusion

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

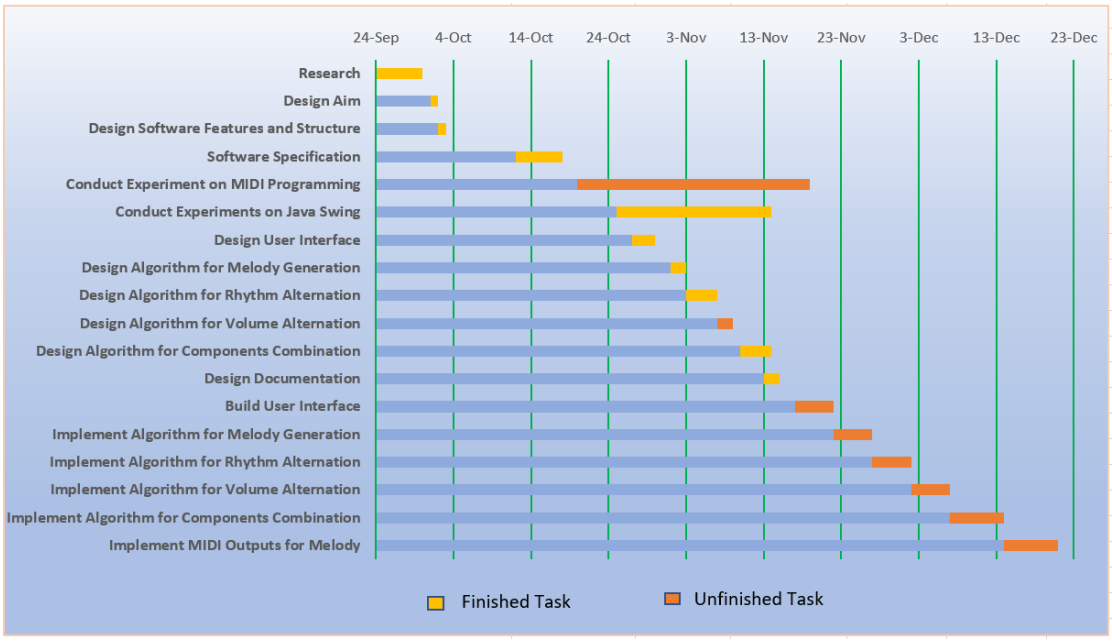
2.5 Ethical Use of Data

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

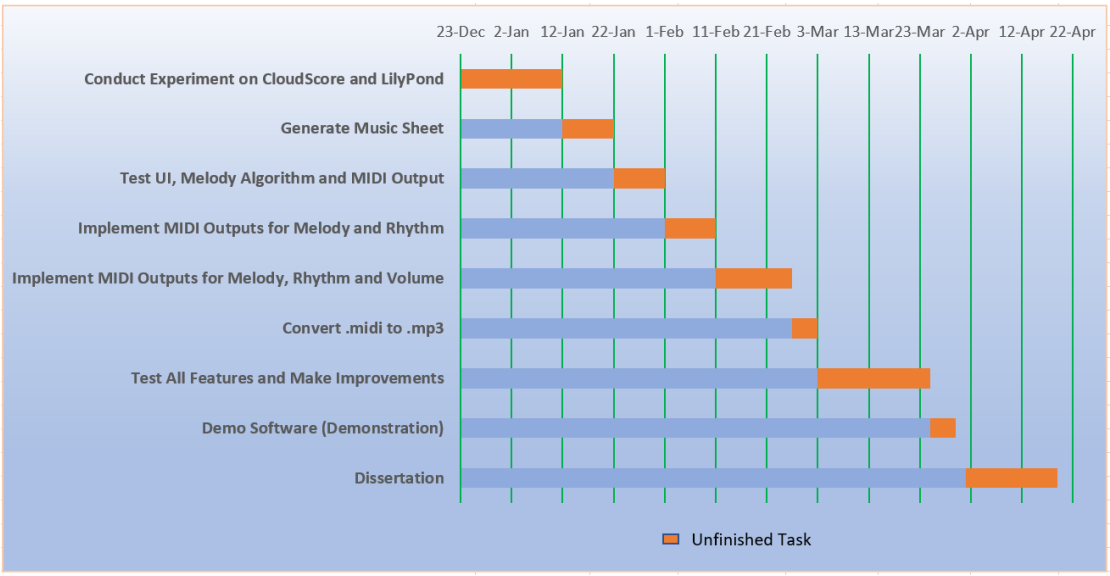
24 Sep 2018 ~ 23 Dec 2018:



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.

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] Unitus by 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