Contents

1.0.0 Introduction

1.1.0 Aims and objectives

The aim of the project is to combine music, art and computer science to create an illustration of music in the form of a static image. The system will analyse the structure of a score using electronic forms of a music score. The analysis of music will include analysis of music theory and apply them to art, when combined will represent and reflect the intended or possible mood of the score.

The analysis will require an understanding of music theory, art theory and algorithms used in computer science. However attempting to connect musical and for this matter art with feelings, is not as well established as many wold think, and there is no justification behind the connection between music and feelings as said by Eduard Hanslick, stating '*an interpretation of music based on feelings cannot be acceptable to art or science*'[1].

Taking this into account this project will not attempt to justify that colours, feeling and music share a connection, however it will attempt to visualise using the subjective opinions of how people may perceive music and feelings.

It is also important to understand the effects of colours and how colours are used to represent information. The system will attempt to understand and follow the research on 'envisioning information' – Eduard R. Tufte [2]. This will be achieved by understanding the rules of colours building on the rules stated by Eduard Imhof [3], who set to codify what combinations of colours and how colour is placed on a canvas can help demonstrate a variety of moods and feelings. Ann Driver also said '*if every note, word or movement is stressed, the result has even less meaning*' [4]. Taking influence from this quote, the project will not attempt to portray 'every note' or every musical notation, it will analyse and portray the overall picture of a score.
As art primarily is an opinion based field, where how the piece is perceived by the user is equally as important as the intended mood that an artist is attempting portray. To follow the rules and to create a work which accurately represents a mood or feeling is complex as no persons opinion can truly be justified, therefore it is import to emphasise – *Above all, do no harm* – Paul Kee [5].

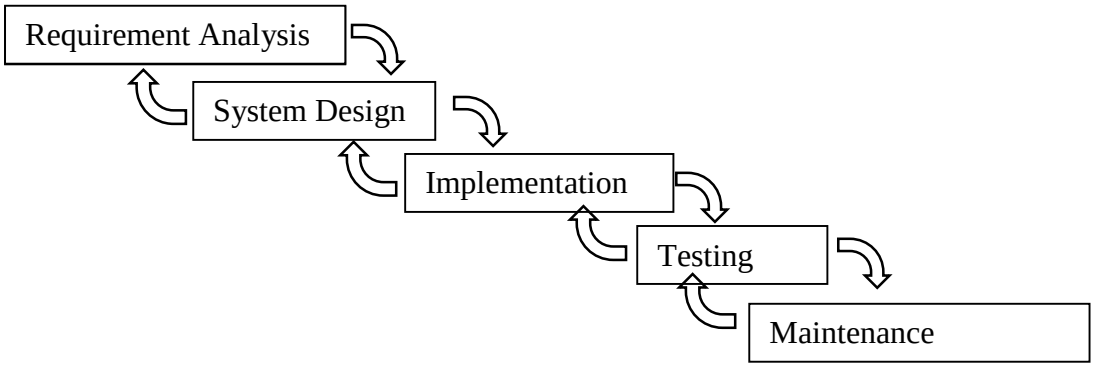
Artists in the past have tried to visualise poetry and music, David Hockney created art based on the poems he read, his work always had a minimalistic design to describe the overall picture of the poem, he did emphasise the use of colours however only painted the details which expresses the poem- David Hockney: two boys aged 23 – 24[6].
Artists such as Wassily Kandinsky, visualised music directly in his famous piece – Composition VIII [7]. In this particular painting Kandinsky wanted to paint what music sounds like, he did this by making the use of geometric shapes and combining them with effective use of graphic variables in order to paint what he heard, while not focusing on the structure of the music but rather the overall picture of the music.

1.1.1Intended Audience

The intended audience of this project are primarily, artists, musicians, computer scientists and generally anyone who is interested in the arts and sciences. There are really no boundaries in which the audience is limited to. On a daily basis people listen to music, see works of art and use a device which includes computer science. Therefore anyone who takes an interest are the intended audience of this project.

1.2.0 Approach to carrying out the project.

The general approach implemented while carrying the project is the waterfall model. The waterfall model is a sequential design process in the software development process. It contains a series of steps which needs to be carried out in order to create a software. Each step in the waterfall model should only be taken when the preceding phase is reviewed and verified, however if a problem occurs then it is possible to cycle back to previous steps in order to fix and refine the system.



1.3.0 Requirement Analysis

The requirement analysis will consist of a MoSCOW analysis and functional and non-functional requirements of the system.
The MoSOW analysis is a list of requirements which range from must have to the wont have functions of the system.

- Must have
- Should Have
- Could Have
- Won't have

The functional requirements specifies essentially something the system should do, or the behaviour of the function.
For example, in terms of business:

- Business rules
- Administrative functions
- Authentication
- External interfaces

The non-functional requirements describe how the system works, or how the system should behave.
For example, in terms of business:

- Performance
- Scalability
- Capacity
- Reliability
- Data integrity

1.3.1 System Design

The system design section paints a clear picture of the system this project plans to create, it will tell the reader what the system is required to do and top level details of how the system meets the requirements while also including the constraints on the software solution.

The designs will be represented primarily through the use of diagrams such as UML, activity etc.

These diagrams will describe:

- the dynamic behaviour of the system,
- How data flows through the system,
- What data types are implemented,
- What algorithms are implemented,
- The static architecture of the system.

This section will also justify the design of the system, discussing the constraints of the proposed solution and different design choices.

1.3.2 Implementation

The implementation section is the realisation of concepts and ideas developed in the requirements and design section. Here is where the system is built, the design is applied using a programming language. This part will also describe the problems that occurred during the implementation and any unforeseen events which may occur during the implementation.

1.3.3 Testing

The purpose of testing is to test for any faults and failures of the system, this is done by making use of test cases which clearly emphasise the results of each requirement.

1.3.4 Maintenance

The purpose of maintenance is to enhance the product so that better versions are released. This could include details on how new features may be implemented in order to fix issues and improve on the current system. It also includes details of possible future works that may be implemented into the system.

2.0.0 Background

To fully understand and appreciate the aims and objectives of the project. Firstly it is important to understand the basics of music theory and how western music is represented in a score. Aspects such as note types, duration, the role in which clefs play etc. are crucial to fully appreciate the outputs and of the aim of the project.

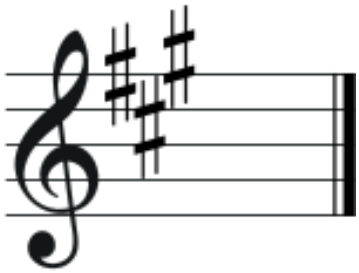
It is also important to understand, how music and any forms of art are subjective to the users who are either listening, seeing, feeling a piece of artwork, however there have been attempts to objectify the relevant aspects required in this project.

This section will also discuss the importance of music, colour, graphics, and graphic variables, how it plays a role in expressing the outcome of a piece of art work.

2.1.0 Understanding the basics of Music theory [8].

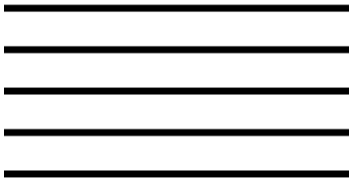
2.1.1 Symbols notation

2.1.2 Key signatures



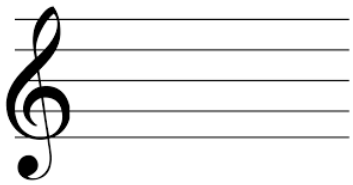
Western music is written in a particular major or minor scale, the tonic or the first degree of the scale is referred to as the scale. Each key has their own specific notes which relate to them. For example the key of 'G major' contains the notes G, A, B, C, D, E, F#, G.
The key signature is placed at the beginning of a score after the clef sign, it also determines the number of sharps or flats in the piece of music.

The Staff



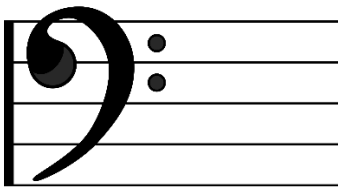
Consists of 5 lines and 4 spaces, as seen below. Each lines and space represents a different note ranging from A-G, each note sequence moves alphabetically up the staff.

Treble Clef (G)



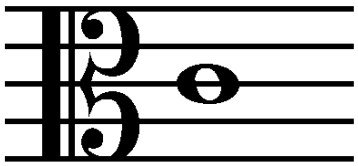
The treble clef notates the higher registers of music, for example if an instrument has a higher pitch such as a flute, violin or saxophone, the sheet music is written in treble clef.

Bass Clef (F)



The bass clef as opposed to the treble clef notates the lower registers of music, for example if an instrument has a lower pitch such as, bassoon, tuba or cello, the sheet music is written in bass clef.

Alto(C)



The alto clef notates the middle registers of music, for example if an instruments plays at a pitch in between a Treble and Bass such as viola, alto trombone and mandola, the sheet music is written in alto clef.

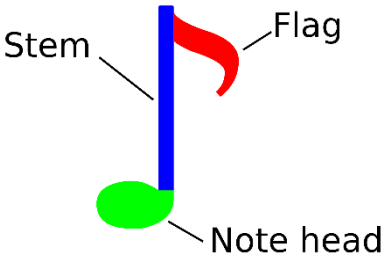
2.1.3 8va and 8vb

The characters 8va, 8vb may be placed above a note:

- **8va:** Indicates a note or series of notes will be played octave higher than written on the staff
- **8vb:** Indicates a note or series of notes will be played an octave lower than written on the staff.

2.1.4 Notes

The notes placed on the staff tells which note letter to play and how long to play it. The note consists of 3 parts: note head, stem and flag.



Note head:

- Each note is filled (black) or open (white), where the note is placed determines which note to play.

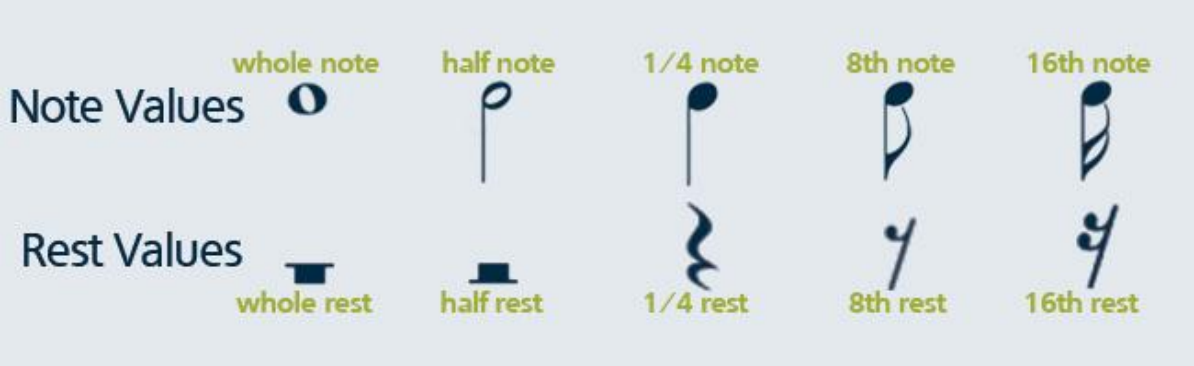
Note stem:

- Extends a thin line either up or down, from the right side of the note head, the purpose of the stem is to make notes easier to read while allowing them to fit neatly in the staff.

Note Flag:

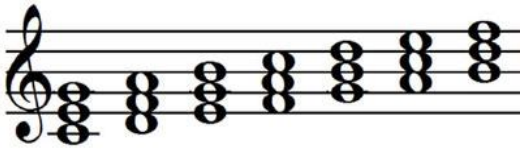
- The purpose of the flag is to tell how long to hold the note, multiple flags in a stem can make it shorter.

2.1.5 Note / rest values



The whole note/ rest values gets held for four beats, the half nots for 2 beats, quarter 4 beats and so on.

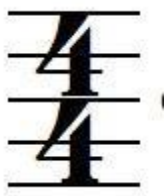
2.1.6 Chord



A chord is any harmonic set of usually 3 or more notes that is used simultaneously

2.1.7 Time signatures

The time signatures mentioned below aren't limited to the ones mentioned, others such as 2/4, 2/2, 9, 8 etc. can be used, depending on how many beats is required in the piece of music.



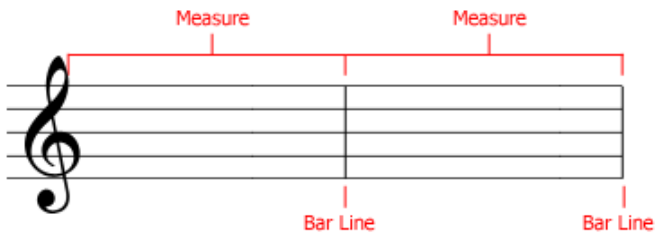
4/4 time signature
The 4/4 time signature means there are 4 beats per bar and every quarter note gets one beat. It is counted as 1,2,3,4 – 1,2,3,4

3/4 time signature
The 3/4 time signature means there are 3 beats per bar and every quarter note gets one beat. It is counted as 1,2,3 – 1,2,3

2.1.8 Tempo / metronome marks

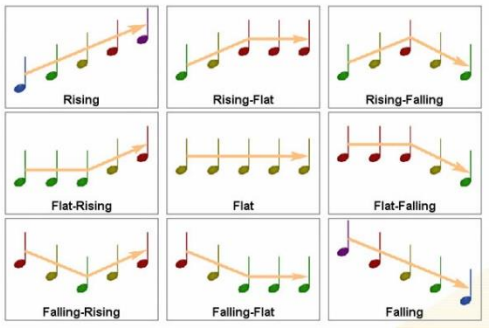
In addition to the note values and time signatures, it is also important to know the tempo or beats per minute BPM. Tempo determines how fast or slow the intended piece is to be played, it is often placed at the top of the sheet of music. For example 60bpm means a single note is played every second, 120 2 notes every second etc.

2.1.9 Measures



The measure is a segment of time which corresponds to a specific number of beats, which are indicated by vertical bar lines. The purpose of measures is to reference points to pinpoint locations within a piece of music.

2.1.10 Contour [14]



The contour line gives the melodic line of the music, this helps get a good idea of the shape of how the melody progresses and how pitches may go up or down. This can be seen by looking at the melody a piece of music is written in or you can hear it as you listen.

2.2.0 Understanding the role of colours

Humans envision information with the important use of colours. A typical human can see around 20,000 colours, trained eyes are able to envision more. However the use of many colours when representing information, the use of 20-30 colours produce rather unpleasant outcomes.
The fundamentals that colours should follow in envisioning abstract information are:

- To Label (*colour as noun*)
- To Measure (*colour as quantity*)
- To represent or imitate reality (*colour as representation*)
- To enliven or decorate (*colour as beauty*)

The project will follow the fundamentals of envisioning abstract information, however it will appreciate that it is difficult to avoid catastrophe such that it produces unpleasant outcomes, therefore as the first principle in bring colour to information the project will attempt to – *Above all, do no harm.* (Envisioning information: Eduard R. Tufte [3])

Eduard Imhof describes 4 major rules which describes the design practices for Swiss maps. The first, third and fourth rules are most important for this project, seek to minimize colour damage. [3]

2.2.1 First Rule

- But *extraordinary effects* can be achieved when they are used *sparingly on or between dull back-ground notes*.
- If one *limits strong, heavy, rich, and solid colours* to *small areas of extremes*, then *expressive and beautiful patterns occur*
- If one *gives all, especially large areas, glaring, rich colours*, the pictures have *brilliant, distorted, confusing and unpleasant effects*.

2.2.2 Third Rule

- *Large area background or base-colours should do their work most quietly.*

2.2.3 Fourth Rule

- *If a picture is composed of two or more large, enclosed areas in different colours, then the picture falls apart.*
- *Unity will be maintained, however, if the colours of one area are repeatedly intermingled in the other, if the colours are interwoven carpet fashion throughout the other*

Ann Driver stated that '*the colour logic is similar to that for emphasis in music*', '*if every note, word or movement is stressed, the result has even less meaning*'.[5]

In most systems of colour organization, every colour is located in 3 space: hue, saturation, value and other spatial-perceptual classifications; red, green, blue etc.
The colours multidimensionality can also be used to express multidimensional information, this can be used so that users can learn, understand and inform about the visualisation they are about to be faced with.

2.3.0 Combining music keys, feeling and colour

The project requires mapping keys to certain colours, being able to understand what type of feeling or colour a certain key may represent is a fundamental aspect, the problem here is how to map feelings and colour to music? And whether connecting feelings, colour with music is even justifiable. The system will loosely be based upon the research mentioned below.

2.3.1 Keys and feelings

According to Hugo Reinmann 'each key, through its type of derivation from the fundamental scale, has a particular character' [9].

- All steps upwards (in pitch) make the character brighter and more radiant
- All steps down make it darker and cloudier
- Major keys as opposed to minor, already has a bright and radiant effect, the brightest keys are the major keys with many sharps, and the darkest are the minor keys with many flats.

This is further expressed by Albert Lavigance [10]:

- Sharp side majors (i.e. major keys with a sharp), the general mood was joyous and brilliant
- As number of sharps increased, degree of moods became more intense
- General mood for flat side minor groups are sad and sombre, and the number of sharps increases the characteristics gain intensity.

Although Albert and Hugo have attempted to categorise music with feelings, Albert expresses that the characteristics of music are 'subjective'[10].

How Albert Lavigance categorised keys and feelings [10].

Key	Feeling
C# major	No entry
F# major	Rugged
B major	Energetic
E major	Radiant, warm, joyous
A major	Frank, sonorous
D major	Gay, brilliant, alert
G major	Rural, merry
C major	Simple, naïve, frank or flat and
A# minor	common place
D# minor	No entry
G# minor	No entry
C# minor	very sombre
F# minor	brutal, sinister, or ver
B minor	savage or somber, but vigorous
E minor	sad, agitated
A minor	simple, naïve, sad rustic
F major	Pastoral, rustic
B-flat major	Noble and elegant, graceful
E-flat major	Sonorous, vigorous, chivalrous
A-flat major	Gentle, caressing or pompous
D-flat major	Charming, suave, placid
G-flat major	Gentle and calm
C-flat major	No entry

D minor	serious, concentrated
G minor	melancholy, shy
C minor	gloomy, dramatic, violent
F minor	morose, surly, or energetic
B-flat minor	funereal or mysterious
E-flat minor	profoundly sad
A-flat minor	doleful, anxious

2.3.2 Keys and colour

Harry Farjeon, attempts to categorise keys with colours by combining his own opinions along with the results of his students [11].

Key	Colours: Farjeon	Colours: students
C major	White	Majority white, cream/ gray
G major	Spring green	Mostly green: yellow brown
E major	Warm orange	Several yellows and blues
F#major	Red	Yellow and deep rose pink
G flat major	Electric blue	
F major	Browns	Oyster – white
E flat major	Sky blue pink, light mauve	Browns/ oranges some pink
A flat major	Real mauve	Lost
D flat Major	Deepness into purple	Purple
C minor	Silver, steel and iron	Mostly gray
G minor	Green or blue with silver	Only few opinions mostly red
D minor	Sand color	Varied inclination
F#minor	Light red	Scarlet
B flat minor	Black	Mostly black

As the table shows 7 out of 11 agree with each other, the ones that agree were [E-flat major, E major, F major, F#major] however these keys were accordingly exist in close approximation to each other and are all in major mode. However in the overall results it is important to take into account that the opinions of the students varied more, showing that music cannot be fully be justified with colours and feelings.

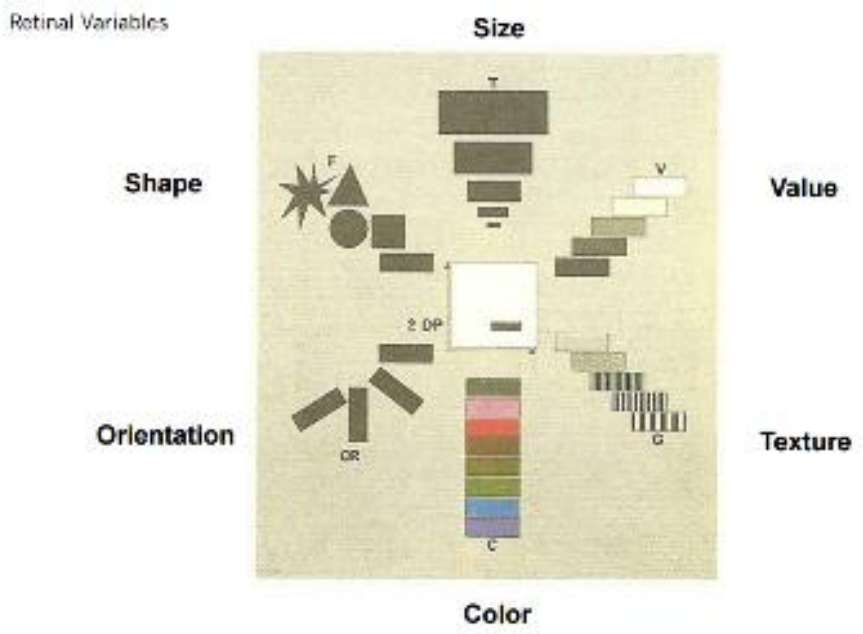
2.4.0 Understanding graphic variables.

Graphic or visual variables are characteristics of graphics which describes the characteristics of a certain graphic. Jacques Bertin categorises these characteristics into 'retinal variables'. [12]

2.4.1 Retinal variables

- Position: changes in the x,y, (z) location
- Size: change in length, area ore repetition
- Shape: infinite number of shapes
- Value: changes from light to dark
- Colour: changes in hue at a given value
- Orientation: changes in alignment
- Texture: Variation in pattern

Music itself is not stationary, the tempo, beat, instruments played etc changes throughout a piece of music. Graphic variables can help describe the change in, for exapmle a change in key may be described by a change in colour, or instruments may be represented as different shapes, or the value and colour of the graphic may change according to the beat or tempo of the music.



The system will attempt to implement all graphic variables mentioned by Jacques Bertin [12], the values of the graphic variables will change accordingly to the results of the survey conducted, combined with the opinions of the persons mentioned above.
The final outcome of this project will combine the understanding of musical notations, keys, feelings, colour and generate effective analysis in order to create a visualisation.

The music score to be analysed will be of type 'MusicXML' and will be analysed using 'Music21'; a python module developed by MIT for the purposes of electronically writing and analysing music. To create the visualisation Pythons Pygame will be used.

3.0.0 Tools used to create the visualisation system

3.1.0 Music21 [13]

Music 21 is a python toolkit for computer-aided musicology. It is a set of tools for music listeners and enthusiasts to read, write and analyse music electronically. The system has been available since 2008 and is constantly growing and expanding. It builds on pre-existing frameworks and technologies such as, MusicXML, MuseData, Midi etc, it uses an object-orientated skeleton which makes it easier to handle complex data, and it keeps code simple by reusing existing code available in python. With music21 a module can be easily adapted or built upon to solve dozens of similar problems.

Its origins as a project nurtured at MIT, the music departments of MIT along with Harvard, Smith and Mount Holyoke collages, helped bring this toolkit from its easiest roots to a mature system.
In terms of analysis of music, music21 allows extraction of high level musical data through its modules. Information such as, which key a certain piece is written in, the length of a particular note or measure, which bar line a clef resides in, time signatures, and overall any information written by hand to a music score has been made available as a module in Music21.

Other than recognition of high level musical notation, it also provides modules which analyse the frequency of code in each chord is played, displayed as a matrix, and a module which automatically generates frequency maps and graphs. To do this music21 makes use of other python modules in order to ensure simplicity in the software:

- Matplotlib: Graphing and visual displays
- Pygame: Allows for realtime MIDI performance.
- Pyaudio: Allows for recording within python.
- PIL (Python Imaging Library): resources foor transforming and editing graphics files.

To put the capabilities of music21 to perspective, with small amount code it is possible to google every motet in in your database that include the work ‘exultavit’ in the soprano part to see how common the motet’s text is.

```
import webbrowser
for motet in listOfMotets:
  superius = motet[0]
  lyrics = text.assembleLyrics(part)
  if 'exultavit' in lyrics:
    webbrowser.open('http://www.google.com/search?&q=' + lyrics)
```

To put it simply music21 is a powerful software, it is what Photoshop is to creating digitalised images. It is simple to use, simple to learn and able to create and analyse high levels of music data. It is important to emphasise that since Music21 is a relatively new toolkit for python, documentation on music21 is limited, for the most parts most documentation relies on analysing the information already imbedded into MXL or MIDI files, however do not extract and create new types of information (visualisation) using music21.

This system will use the latest version of Python 3.5 and import all available modules in music21 in order to extract and analyse a score.

3.2.0 MusicXML [15]

MusicXML was invented by Michael Good in 2000, its purpose was to publish interactive musical scores on the internet, and it allows musicians to easily collaborate with each other who use different music applications. The main purpose of MusicXML is to share sheet music files between applications, and to archive sheet music in the future.
MusicXML is similar to how PDF file is the common for sharing documents from one person to another, and just as MP3 files have become synonymous with sharing recorded music, MusicXML is the norm for sharing written music. It is widely supported by over 200 applications, including Sibelius package, muse score Finale.
Music is represented using the semantic concepts behind common western music notation. MusicXML includes both how a score looks and how it plays back, it is written as the name suggests in XML, and the advantages of using XML format are:

- Files can be opened in any computer text editor
- Fully internationalized via Unicode
- Files are human-readable as well as machine-readable
- Can use all standard XML tools developed by larger industries than the music industry.

With this type of diversity MusicXML allows continued development of the format by anyone, not just MakeMusic.

3.3.0 MusicXML vs MIDI [16]

MIDI (Musical Instrument Digital Interface) is a way to allow a computer to read data from an external instrument, and to send data to external instruments, it represents the basis of all sequencers and notation products currently in the market.

However music educators are often aided by the use of notation, vs just playing something into a computer (MIDI), and with MIDI it is difficult to process the writing of music. The problem with MIDI files for this project is that it often becomes difficult to set up or share between notation packages, for example if you were working with Finale or Sibelius, the original MIDI file may be a mess, barely legible and full of errors, from instrumentation to time/ key signatures, and it is not possible to import any printed diacritical markings, lyrics or text.

MusicXML provides a solution to this, if a song is written in MusicXML, for any instrumentation group, it is simple to export and import it to any other notation app, while maintaining all original score elements. Unlike MusicXML, MIDI cannot tell the difference between 2 notes such as F# and G-flat, nor does it represent the stem direction, repeats, slurs, measures and many other forms of notation. Being able to extract detailed notation is essential for this proposed system, as it needs to analyse each key, clef, measure in order to generate a visualisation which creates a certain mood. Also the system must make use of open source music files, in order to analyse the music, as it would be time consuming to write a new score. The security which MusicXML provides in terms of maintaining the originality of the score is vitally important in order to create the visualisation stated in the aims and objectives. Hence the reasons for using MusicXML over popular formats such as MIDI.

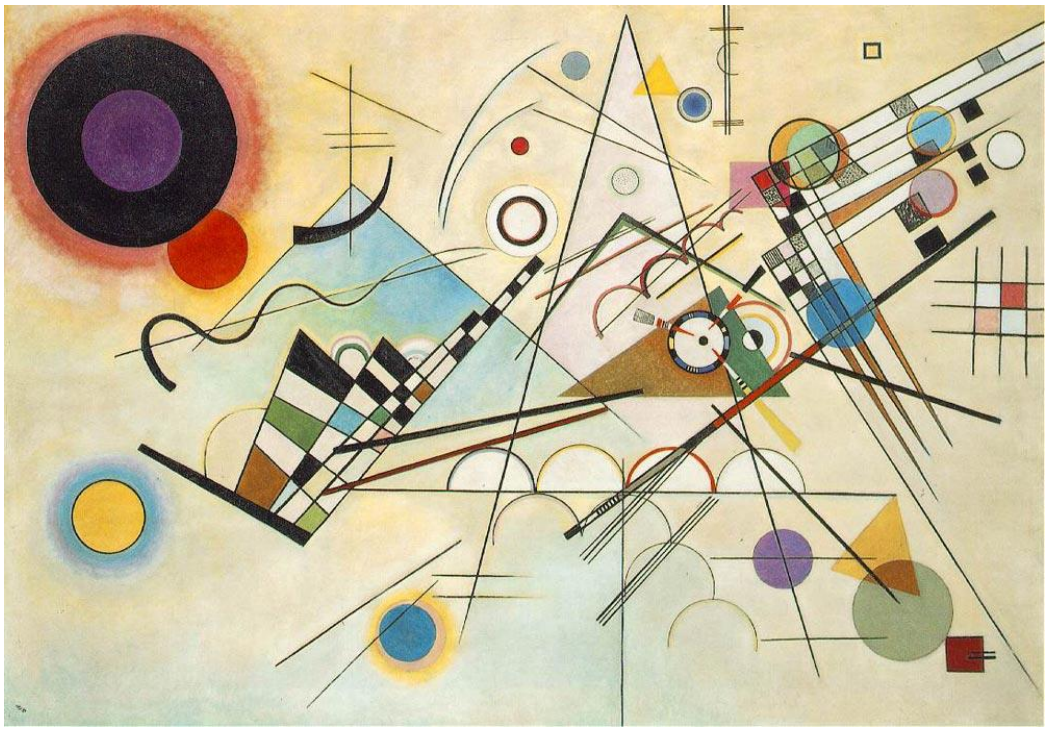
3.4.0 MuseScore v3.0 [16]

Musescore is a free and open source score writer available for windows, OS X and Linux, it is comparable to other similar software such as Finale and Sibelius, and it provides a wide variety of formats and input methods including MusicXML.

Other than MusicXML MuseScore provides support of MIDI, Guitar Pro and overture formats, it can export MusicXML and MIDI files into a variety of sound formats such as WAV and MP3, scores can be exported to PDF, PNG and other documents.
For sharing scores, MuseScore provides a website in which it is possible to view scores, listen and download the scores in a variety of formats including MusicXML.
The scores that is required for the system will be downloaded from 'musescore.com' as it provides an open source format to share and use files generated by the users of the software.

4.0.0 Review of existing solutions

There have been countless solutions to visualising music, through the use of a paintbrush and a pencil, before the existence of computers, and recently through the use of computer software. To begin it is important to look at visualising music through art specifically.

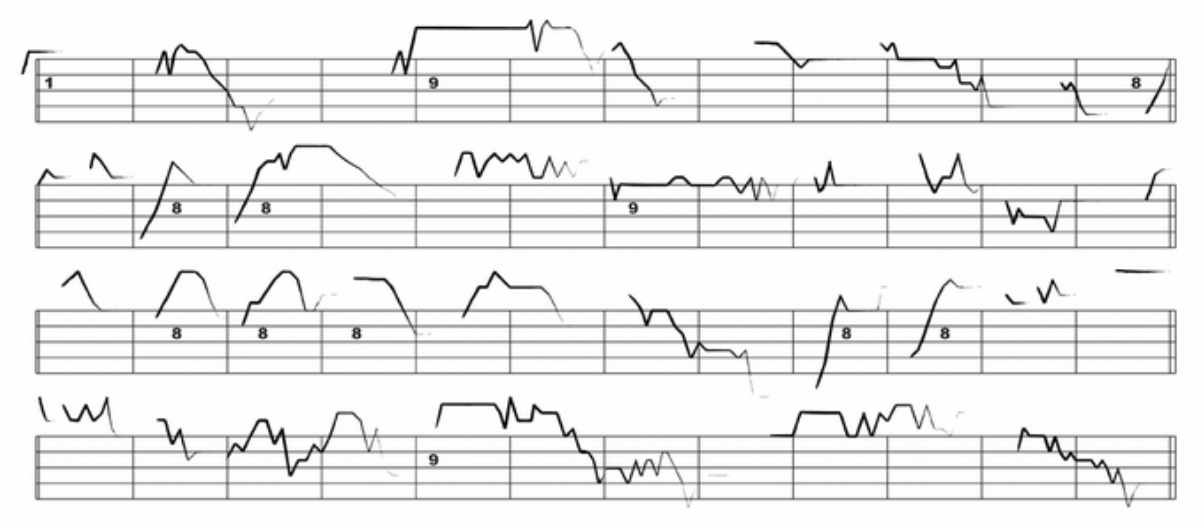


4.1.0 Wassily Kandinsky: Composition VIII: 1923 [7]

This visualisation as seen uses simple geometric shapes, triangles, lines, squares, and triangles etc. Kandinsky makes effective use of graphic variables described by Bertin [12], to describe the mood of the music. The piece above has no structure to it, it is almost random, and there is no start middle or end point, all of which are essential in writing music. Rather he focuses the overall painting rather than the structure of the music.

4.2.0 Jon Snydal: Visualisation of John Coltrane's solo in Miles Davis 'all Blues' [16]

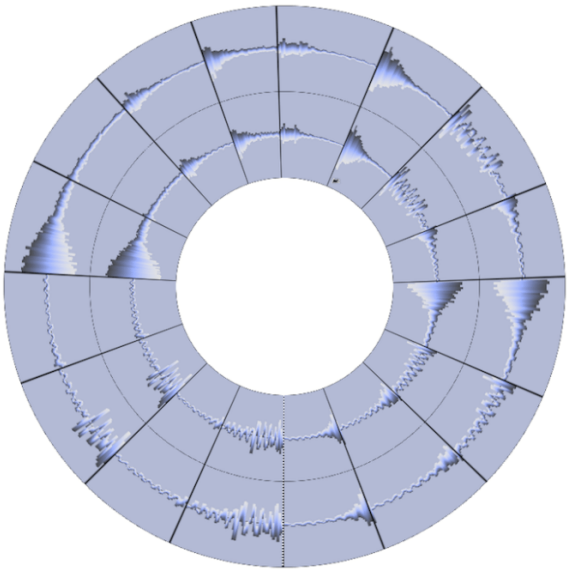
In contrast to Kandinsky, Syndal focuses more on the actual structure of a score, the contour is clearly visible for each measure. The pitch levels of instruments are greatly emphasises by the use of lines and their variance. The piece directly takes aspects of how music is written on paper and uses to create a visualisation which does not reflect the mood or feeling of a score, however it's a direct visualisation of music.



4.3.0 Similar Products

With the emergence of computers, so came the use of computers to visualise music, predominantly with the use of MIDI files in the late 20th century and early 21st century came the start of visualising music using computers, countless visualisations have been created using the MIDI format, most of which focus on real time illustrations of music or a moving illustrations of music. The most popular method of visualising music were waves, audio software's primarily used waves to represent the pitch, loudness of a digitalised piece of music.

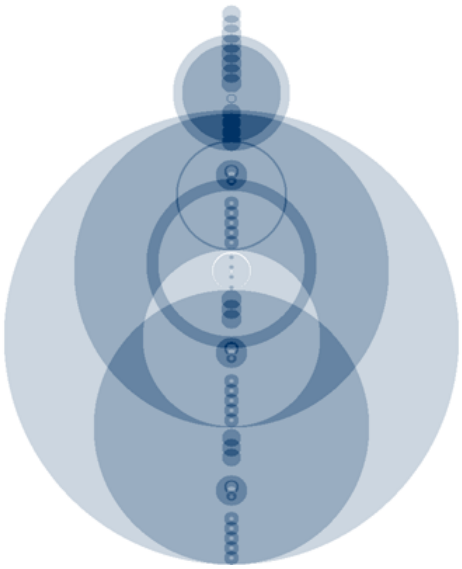
4.3.0 Ethan Hein: Funky Drummer as its looping [18]



This piece shows how a piece of music repeats with the effective us of a circle as a primary base to emphasise 'looping'.
The visualisation was created using the music software 'Recycle'. The amplitude blobs represent the timing and volume of a drum being played.

However it doesn't make effective use of colour or any other graphic variables in order to emphasise the overall mood generated by music.

4.3.2 Martin WattenBurg: Like a prayer by Madonna [19]



Martin WattenBurg uses the analysis of MIDI file along with java. By using repeated, identical passages of a compassing, the diagrams illustrate the deep structure of a composition. Where the arc or circle begins and ends determines where each part is repeated throughout the score. Creating an accurate illustration of the tempo or beat of music. The illustration as you can see is easy to follow, it is clear where the repeating beats occur and where individual notes are placed.

The overall picture painted is a calm feeling, with the use of subtle changes in lightness or value in the colours, key aspects of the music are clearly visible. However all his works seem to be represented using only circles and one particular colour, hence creating a similar mood for every piece.

Modern visualisations of music primarily make the use of midi, mp3 or other audio associated files. There seem to be little to no illustrations created with the use of MusicXML and music21 files. It seems that Music21 and MusicXML are primarily used to only analyse, read and write music.

Although visualisations have been created for music, they do not seem to focus the use of colours, rather on shapes and graphics in order to visualise the structure or rhythm of the music. The mood is generally not taken into consideration in the visualisations.

5.0.0 Specification

The specification is the description of the system to be developed, it describes what the system should or should not do through the use of analysis such as the MoSCoW method and listings of functional and non-functional requirements.

The MoSCoW method describes what the system Must do, Should do, Could do and will not do. MoSCoW is a prioritization technique, it helps focus and prioritize on the most important requirements, and this system will require a MoSCoW analysis of the survey to be conducted and the visualisation system.

The functional requirements define a function for a system and its component, it describes a set of inputs, the behaviour and the outputs of the system. Generally these requirements express what a system must do, for this particular system, a functional requirement for the visualisation system is that it must be able to extract and output many types of musical notation.

In contrast to functional, non-functional requirements are requirements that specifies a criteria that can or is used to judge the operation of the system. Generally these requirements express what the system shall be or the quality attributes of the system, they can refer to the qualities, quality goals, constraints etc. a non-functional requirement for the visualisation system is that it should reflect the key aspects of a score using graphic variables.

Below are the MoSCoW analysis, functional requirements and non-functional requirements for firstly the survey and then the visualisation system. The functionality will be later tested using test cases determining what requirements were successful and those that were not.

5.1.0 MoSCoW(survey)

Must	• Have checkboxes available for users to check feelings and colours • Have music available to listen to • Have next button to proceed to next song • Be quick and not time consuming • Add data to database
Should	• Error checking to limit users to 3 checkboxes for feeling and 3 for colour. • Be pleasing to the eye, UI should not attraction to anything in particular so users can focus on song and selecting colours or feelings. • Fit in one page • Have select optional colour option
Could	• Prompt users to say that their choices are final • View and change what they have chosen • View overall results as graphs
Won't	• Include complex JavaScript to make the experience feel better

5.2.0 Functional and non-functional requirements (survey)

5.2.1 Functional requirements (Survey)

Req.ID	Requirements Description	Priority	Complexity
FR.1	Songs related to a particular key and beat will be played, looped automatically for the users to hear. • Users should also be able to pause the song by clicking the 'pause' button. • Users should also be able to skip to any part of the song by moving the 'seek' button	High	Low
FR.2	Users will be able to select colours that are displayed by ticking the 'checkbox' button.	High	Low
FR.3	Users will be able to select feelings that are displayed by ticking the 'checkbox' button.	High	Low
FR.4	If users want to select more colours they should be able to click a 'select colour' button, which opens a window of colours.	Medium	Medium
FR.5	The users will be limited to 3 checkboxes for feelings and colour. • An alert should pop up stating users can only select a maximum of 3 checkboxes for each section.	High	High
FR.6	Upon clicking the 'Next' button, the page should load a new song which fulfil FR.1 along with refreshed checkboxes.	High	Medium
FR.7	Upon clicking the 'Next' button, the data users selected should be added to the database.	High	Low

5.2.2 Non-Functional requirements (Survey)

Req.ID	Requirements Description	Priority	Complexity
NFR.1	Upon completion, information will be shown on the webpage to show completion of survey	High	Low
NFR.2	The website should behave the same on all browsers	Medium	Medium
NFR.3	The survey must remain unanimous of the user	High	Low
NFR.4	The survey should be quick and design should maintain focus on the song and colours and feelings checkboxes	High	Medium
NFR.5	The amount of people that carry out the survey should not be limited	High	Medium

5.3.0 MoSCoW (creating visualisation)

Must	• Read MXL file • Extract low level musical concepts • Implementation of graphics • Implementation of graphic variables • Create static visualisation • Create 2d visualisation
Should	• Extract more than a few types of data from score • Generate analysis of music • Automatically select graphic variable methods • Automatically generate visualisation given a score
Could	• Machine learning methods to learn and recognise music • Rhythm and • 3d visualisation
Won't	• Real-time or moving visualisation

5.4.0 Functional and non-functional requirements (visualisation system)

5.4.1 Functional Requirements (Creating visualisation system)

Req.ID	Requirements Description	Priority	Complexity
FR.1	System should be able to extract all required information from MXL file. • Keys • Notes • Chords • Clefs • Rests • Offsets • Measures • Metronome mark	Medium	High
FR.2	Implement the results of the survey and map to relevant keys.	High	High
FR.3	Create an output for any mxl file	Medium	High
FR.4	Segment score into instruments played • All information should still be available for analysis	High	Medium
FR.5	All extracted data relating to notes, note types, rests should be ordered to the offset they appear in. • Apply insertion sort	High	High
FR.6	Analyse the key of the score and of each instrument.	High	Low
FR.7	Map a certain colour to a key: • If no match is found do a similarity analysis and apply the appropriate change according the results of the survey	Medium	High
FR.8	Draw Shapes relevant to each clef	High	Medium
FR.9	Set width and height of canvas automatically • Use pitch for height • Use offset for width	Low	Low
FR.9	Colours declared in text ‘white’ should be mapped to their RGB values ‘255,255,255’, as for any other colour declared that is in the database	High	Medium

5.4.2Non Functional Requirements (Extracting musical information)

Req.ID	Requirements Description	Priority	Complexity
NFR.1	The system should only require the input of a MXL file, otherwise analysis and visualisation is automated.	Medium	Low
NFR.2	The correct RGB values are placed for each colour	High	Low
NFR.3	Time complexity should not be taken into consideration	Low	High
NFR.4	Reflect the music score and data accurately as graphic variables	High	High
NFR.5	Should not try to justify that the system is a solution for mapping music with keys and colour.	High	Medium

6.0.0 Design: UML class Diagrams

Class diagrams describe the static structure of the overall system, showing the systems class, attributes, operations (methods or functions) and the relationships between objects. It is used for both the conceptual modelling systematics and detailed modelling to translate into programming code.

This section will firstly describe the class, attributes, operations and relationships of objects for the survey and then the visualisation system. For full class diagrams of the attributes and methods see Appendix 1.0.0(visualisation system) and Appendix 2.0.0 (survey).

6.1.0 Design of survey

The purpose for creating the survey is to test the theories presented by Albert Lavigance [10] and Harry Farjeon [11]. Albert lavigance attempted to classify keys with feelings and Harry with keys and colours. The survey will conduct a test with modern music and users, and see if their theories have any justification to them. If the results of the survey do indeed agree with the theories set by Harry and Albert then the system will be used based on their findings. If the survey shows no correlation with the results pointed out by the two then the system will attempt to find a correlation between keys, colours and feelings and use that as the base for creating the visualisation.
However if no evident pattern or insufficient data is presented by the survey, the system will again apply the methods placed by Harry and Albert.

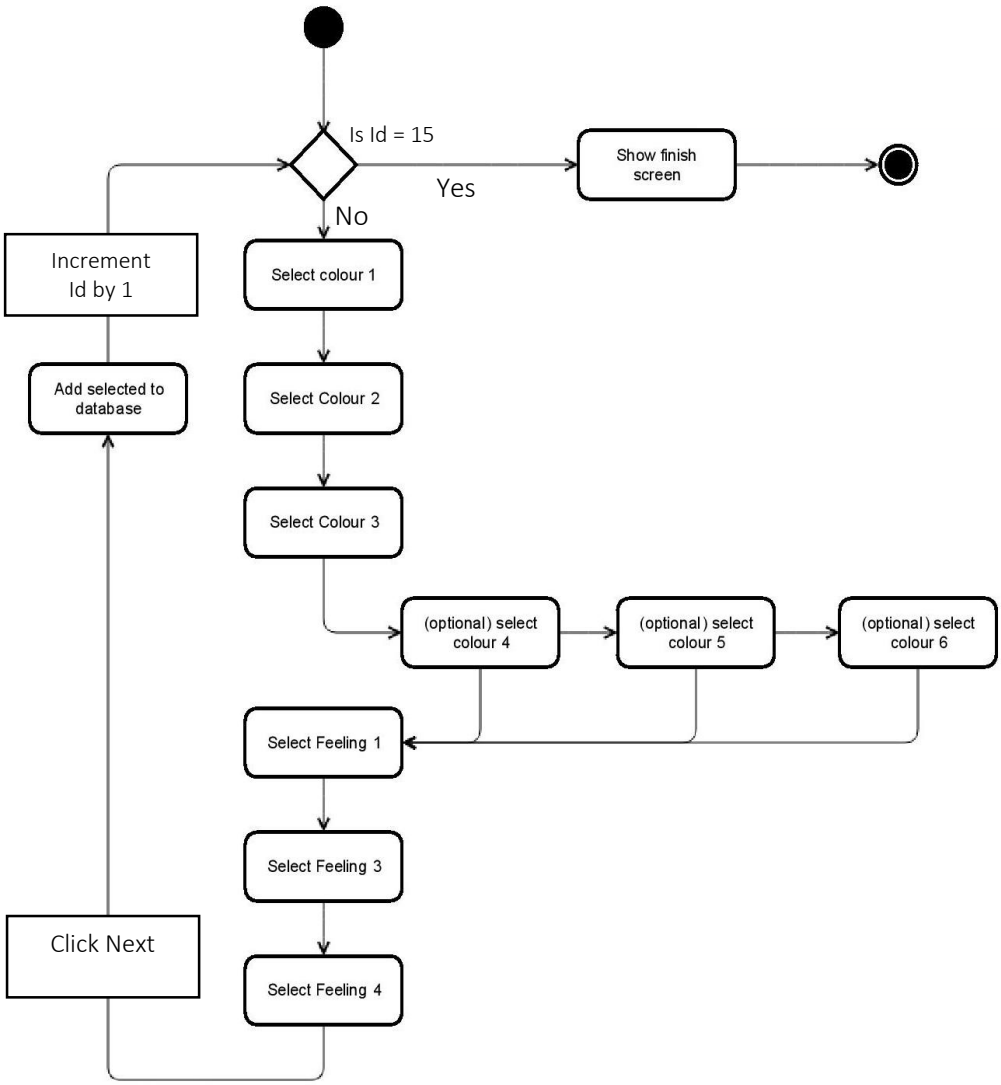
The user interface of the survey will be minimalistic and if possible be fit into 1 page. The colours used for the background will be colours which do not attraction to it, the background for the content will be white so that the colours in each table and the text will stand out. The choice of this design is also so that the user does not get distracted from listening to the song, so that they can give full concentration into listening and selecting the colours and feelings with all honesty.

The colours and feelings for the system and for the visualisation system will be from the tables presented by Albert and Harry, as the surveys purpose is to find some justification behind their hypothesis.

6.1.1 Activity diagrams

An activity diagram captures the dynamic behaviours of the system, where as UML class diagrams show the show the static state of the system. It helps visualise dynamic nature of the system, describe the sequence from one activity to another, describe the parallel,

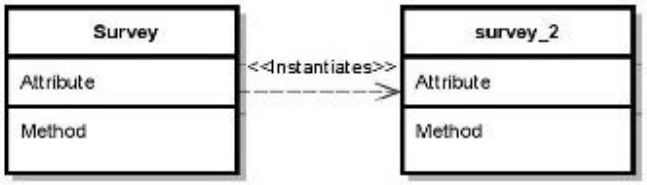
6.1.2 Activity diagram Survey



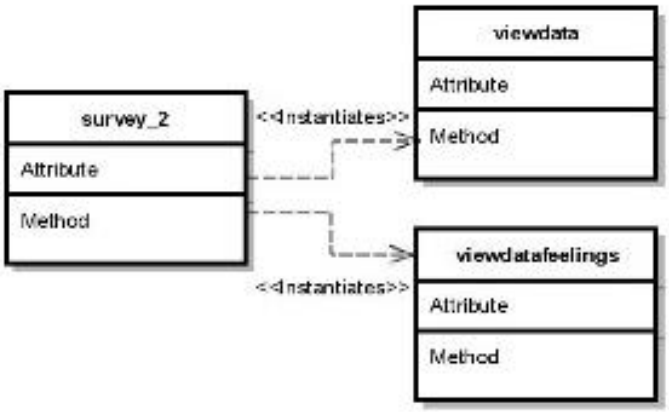
The diagram above shows how they survey behaves dynamically. Firstly it checks whether the song is 15, as when the song id reaches 15 the system reaches the finish screen and the process ends.
If the song id is not 16 then the song in the background plays according to the Id. The user has to select at least 3 colours and is given the option to select an extra 3, they can skip this process.
Then the user must select 3 feelings, then click next, upon clicking the next button, the data is then added to the database, and the Id value is incremented by 1, this process repeats until Id is 15.

6.2.0 UML class diagrams for Survey

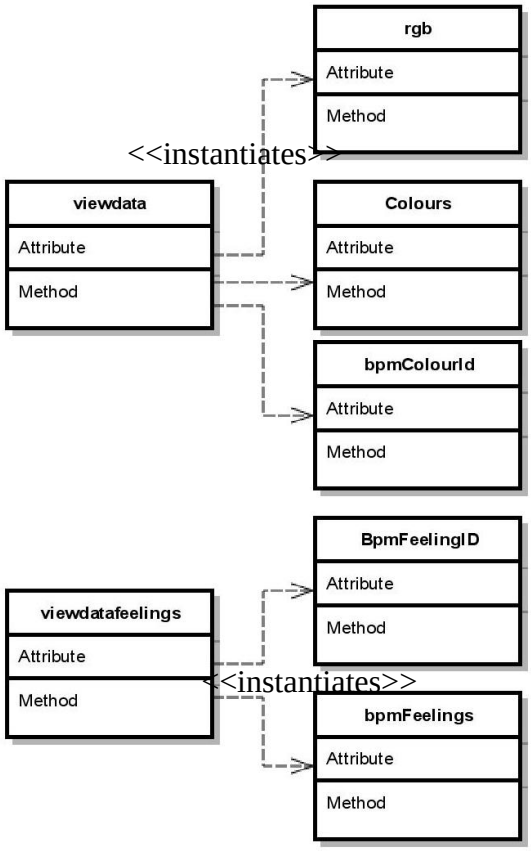
6.2.1 Getting data from and adding data to database



6.2.2 Getting data from database and count as required



6.2.3 Adding the counted data into separate tables in database



6.2.4 Explanation

The diagram for 6.2.1 shows that the table survey_2 is an instance of the class survey, the Survey class simply adds the related information to the database table survey_2. This means the occurrence of the table will only exist when the user has confirmed that they have finished selecting for that particular song.

Similarly for 6.2.2 the classes ‘viewdata’ and view ‘datafeelings’ are instances of the database survey_2, the classes view data and view data feelings get the data from survey_2 and echo them out. As a change in the database will impact on the data of class’s ‘viewdata’ and ‘viewdatafeelings’.

The class view data are instances of tables, rgb, colours and bpmColourId are of class ‘viewdatafeelings’ are instances of tables ‘BpmFeelingID’. The ‘viewdata’ and ‘viewdatafeelings’ classes simply take the data from the relevant tables and echo them out depending, on what is required. Either as just BPM count or both BPM and Key counts.

Function of each class diagram

6.2.1:

- The website (survey) takes the user input data and adds the data to the table (survey_2)

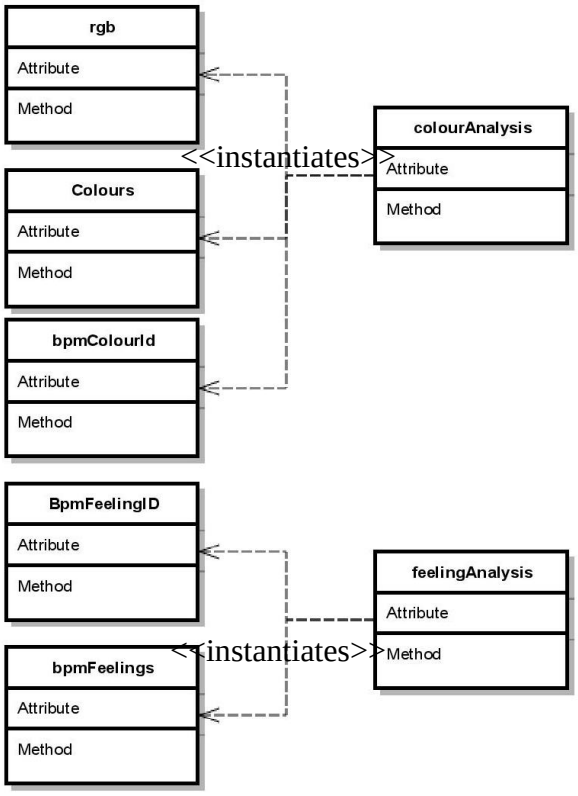
6.2.2:

- The classes ‘viewdata’ and ‘viewdatafeelings’ take the data from the database and count each occurrence of a colour or feeling and output them into tables either as:
    - BPM (80, 100, 120)
    - BPM and Key:
        - 80bpm and C
        - 80bpm and C#
        - Similarly for all keys and beats.

6.2.3

- ‘viewdata’ adds to:
    1. Table ‘rgb’: values of each colour in the colour array
    2. Table ‘Colours’: the count of colours for each BPM
    3. Table ‘bpmColourid’: the count of colours for each BPM and Key
- The class ‘viewdatafeelings’ repeats the process of 2 and 3 of class ‘viewdata’ and adds the data to ‘bpmFeelings’ and ‘BpmFeelingId’ respectively.

6.2.5 Getting data from relevant tables and creating analysis tables



6.2.6 Explanation

The class ‘colourAnalysis’ is an instance of tables:

- Rgb
- Colours
- bpmColourid

The class ‘feelingAnalysis’ is an instance of tables:

- BpmFeelingId
- bpmFeelings

The role of class ’colourAnalysis’ is to analyse the data received from its tables.

1. Counts which colours increased or decreased in frequency as the beat increases.
2. Takes each colour and gets their RGB values form table ‘rgb’.
3. Calculate the average RGB values of colours which showed an increase and decrease in frequency.

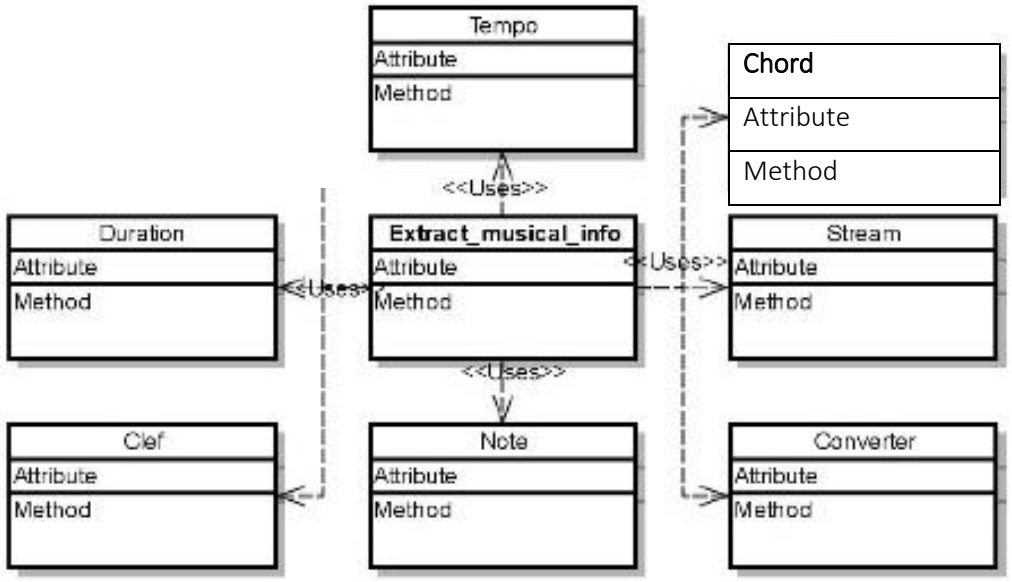
‘feelingAnalysis’:

1. Repeat No.1 of colourAnalysis but for from table ‘bpmFeelingId’ and bpmFeelings
2. Show 10 most common feeling for all BPM
3. Show 10 most common feeling for all BPM and Keys

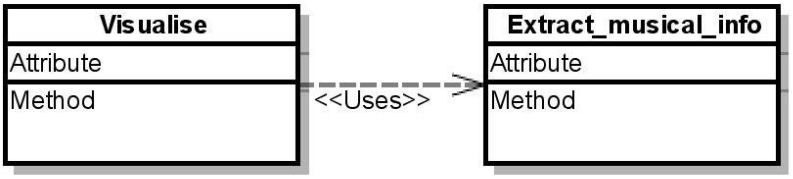
6.3.0 UML for system to create visualisation

To extract musical information from the 'MXL' file, the class 'extract_musical_info' uses classes available in 'music21' 6.2.1.

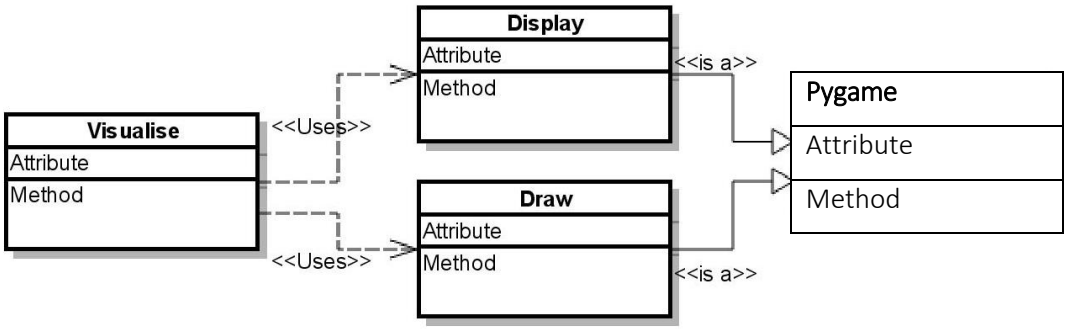
6.3.1 Get relevant data from music21 toolkit



6.3.2 Get functions used in Extract_musical_info



6.3.5 Use relevant pygame modules to create visualisation



6.3.6 Explanation

The class 'extract_musical_info' has a realisation relationship with the 'music21' classes to extract the required musical information (6.2.1):

1. Tempo
2. Chord
3. Stream
4. Converter
5. Note
6. Clef
7. Duration

The visualisation system 'Visualise' has a realisation relationship with the 'Extract_musical_info' class. It imports all the functions available in 'Extract_musical_info' and uses it to calculate:

1. Height and width of canvas
2. The shape to be drawn
3. The colours to be used
4. Other graphic variables of the shape (Bertin) [12]

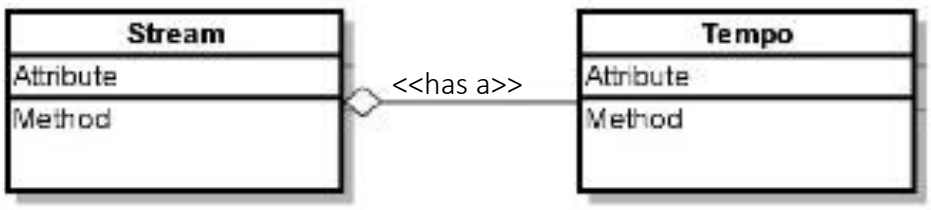
It then compiles all the data imported and creates a visualisation.

The class 'visualise' has a realisation relationship with the Pygame module in order to draw the data into the canvas.

1. Display
2. Draw

Both are subclasses of the main class 'Pygame'. As seen using the '<<is a>>' notation.

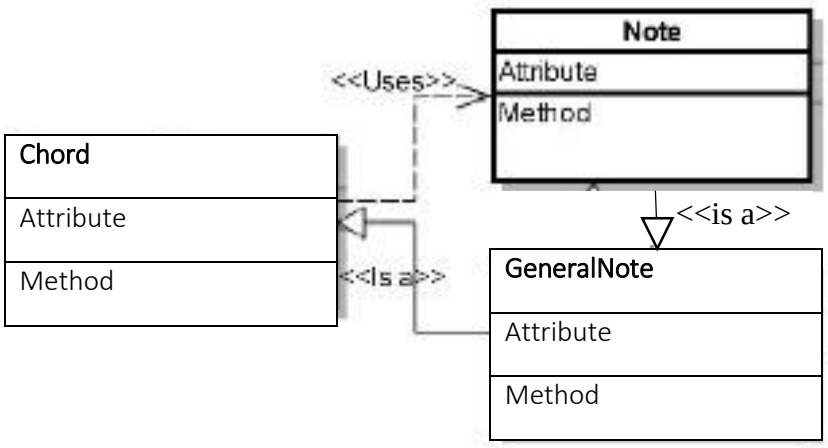
6.3.7 Extracting metronome marks



6.3.8 Explanation

The 'Music21' class 'Stream' has a aggregation relationship with the class 'Tempo'. As seen using the '<<has a>>' notation. Where a Stream of the score has a tempo within it. It is used to extract the metronome marks of the score.

6.3.9 Extracting Chords



6.3.10 Explanation

The Note and Chord classes are subclasses of General Note, note is a type in itself, the chord is a combination of notes in order to create a chord. Therefore they have a realisation relationship with each other.

6.3.11 Reasons behind choices of graphics and graphic variables

With the 'draw' class it is possible to draw shapes for our system: Circle, diamond and rectangle. The circle will represent the bass clef 'F', diamond represents the tenor clef 'C' and the circle represents the treble clef 'G'.

The length of the square and diamond will be determined by the length of each note 'type of note', it represents the radius of the circle. The where on the canvas the shapes will be placed will be determined by the pitch of the notes.

If the shape is hollow 'not filled' then it represents full and half notes, if the shapes are not filled then it represents all other notes.

The background colour will be the overall key of the whole score. Firstly the system will take each RGB values associated with each instrument key and generate an average RGB value of the keys.

The changes in colour throughout will be based on the metronome marks, if the metronome mark shows BPM decreases or increases then the RGB values will change according to the factor found in the results of the survey.

All pitch values will be divided by 10 as to keep the values smaller so all pitches are able to fit into the height and width of the canvas.

All Offsets will be multiplied in order to determine the width, this is so that the graphics have sufficient space to stand out and it does not look unpleasant.

7.0.0 Implementation of survey and analysis

7.1.0 Creating the Survey

The survey was created using a combination of Html, CSS, JavaScript, PHP and MySQLI. Each language used had its own role to fulfil within the website.

7.1.1 HTML

HTML5 provides a built in source tag allowing music to be played, where you determine the source and type using '*<source src= "(URL/ LOCATION).mp3*" type = "*audio/mpeg*">'. This simple function allows easy controls such as '*autoplay*' and 'loop' to be put inside the '<audio controls>' tag, before determining the source of the audio files.

Another useful HTML5 feature essential for the survey was the '*<input type = 'color'>*' tag. This new html5 feature only supports the latest versions of Chrome, Firefox and Opera. However this tag proved useful if users wanted to choose colours which I had not made available. The other input types that were used: checkbox and submit.

The colours and feelings were all echoed out dynamically using PHP, simply taking each index of an array e.g. colours, hex or feelings, and echoing each one as table row. For the colours section it was essential that users were able to see the colours (2.3.2). Tables can have specific background colours assigned to them by using the *<td bgcolor ='(hex value)'>*, and assigning a hex value as the argument for '*bgcolor*'.

To collect user input the website made use of the'<form>' tag available, the '*POST*' method was used to send data for processing.

7.1.2 JavaScript

The only JavaScript code present in the website was for error checking; to limit the users to 3 checkboxes each for both the feelings and colour section.

Pseudocode for error checking

```
var checkgroup = INPUT checkbox []
var limit = INPUT limit
FOR i in length(checkgroup)
    ONCLICK run function checkboxlimit
    SET var checkedcount to 0
    FOR i in length(ceckbox)
        IF checkgroup[i] == 1
        Checkcount +=1
        IF checkedcount > limit
        PRINT error message
        SET checked to 0
        END IF
    END FOR
ENDFOR
```

7.1.3 PHP

The role that PHP played in the survey was to process the data sent by the '*<form>*' tag, and to add the data received to the database.
Firstly for processing purposes, the PHP function '*isset($_POST['name'])*' was used to check if there were any incoming data, if true then for each incoming data a unique variable was created for each feeling and colour (9 in total) received in order to store the data into the database.

7.1.4 PHP + MYSQL

Firstly PHP was used to connect to the database using the '*mysqli_connect()*' function, an if statement was placed for error checking to check if connection was successful to database, if not the system would echo out the type of error occurred.

Secondly PHP was used in combination with MYSQLI to insert data into the database, after the relevant variables had been created by checking if data had been received.
To insert the data into the database:

- Insert INTO table_name (column names) VALUES (Variable names)

An if statement stating if the query had been successful echoes out 'New record created successfully' if query was successful or 'ERROR' and the type of error if query was not successful.

7.1.5 View Data

The SQL statement used to get data from the database:

- SELECT * FROM survey_2 WHERE ID = $index

This statement returns all the items where ID is equal to the index, the index value is originally 1 and incremented each time a data has been received from the database. This would fetch the data one by one and the each colour\ feeling, ID and song_key column would be appended to an array.

Example PHP (A while loop is placed to check add to array until MySQLI results returns false):

- Array_push($Colour1, $items['Colour1']);\\for each colour\ feeling column
- Array_push($id, $items['ID'])
- Array_push($key, $items['Song_Key'])

The data form the survey with a unique id for each row, a key value from 1,2,3,4,5, the column 'bpm' would have either 80, 100, or 120 the other columns stored the feelings and colours values selected by the user. The overall purpose of this class was to get the total for each selected colour or feeling, firstly according to BPM and then according to BPM and KEY.

7.1.6 Group According to BPM

```
var bpm
IF bpm == 80
  SET var offset to 1
END IF
IF bpm == 100
  SET var offset to 2
END IF
IF bpm == 120
  SET var offset to 3
END IF
FOR i in range length(key)
  IF key[i] == offset OR IF key[i] == offset + 3…(continue adding 3 until offset + 12) THEN
    APPEND to array beatsperm colours1,2,3[i]
  END IF
END FOR
```

The algorithm above groups all the colour according to BPM, the first index of each colour/ feeling array would be of 80 BPM, second 100 BPM and third 120BPM. Each increment of 3 after the index would be the relevant BPM

- Index 1,4, 7 … would be 80 BPM
- Index 2, 5, 8 … would be 100 BPM
- Index 3, 6, 9 … would be 120 bpm

The algorithm looks for each number 1,4,7 in the key value and according to index of the value it appends to variable 'beatsperm' accordingly. The resulting would be all the colours associated with 80BPM (9.1.1).
The next step was wo count how many times each colour occurred, this was done by simply creating a for loop the size of the returned 'beatsperm' and increment the values set for each colour (originally 0) of each colour every time it was found.

7.1.7 Group According to BPM and Song Keys

The survey consisted of 15 songs and 5 types of key in total, 3 songs for each key. The algorithm here was similar to the one described above however firstly they system groups each colour\ feeling by key.

The keys are: C, C#, F, G, G# (6.0.0), for grouping by bpm see algorithm above
If the value of colour is:

- 1,2,3 (song key is C)
- 4,5,6 (song key is C#)
- 7,8,9 (song key is F)
- 10,11,12 (song key is G)
- 13,14,15 (song key is G#)

To return the respective key and their BPM values:

- For C and 80bpm
  - Only return colours which have key value 1
- For C# and 80bpm
  - Only return colours which have key value 4
- For F and 80bpm
  - Only return colours which have key value 7

For others return respectively to the method stated above, and count how many times each colour appears for each key and BPM.

7.2.0 Analysis of colours and feelings

To analyse the RGB values of each colour, the algorithm mentioned above must be used, after each colour has been grouped by BPM or BPM and key.

The database 'colours' contains all the colours selected and their respective values. In this class the, the case here is to simply match the colour variable with the one in the database and replace the colour with RGB values.

The first analysis needed was to see how colours increased or decreased with the increase of bpm (9.1.1). Using the returned data from Group by bpm, firstly calculate the difference of each colour from 80 to 100 bpm, then 100 – 120, then calculate the average Red, Green, Blue values for the colours which showed a decrease and an increase for each, then calculate the average for the change in 10bpm.(9.1.1), similarly for the ones returned by Group by bpm and key as well.

The analysis of feelings simply consisted of how feelings increased or decreased according to change in BPM and BPM and key.

7.3.0 Extracting music information

Extracting musical information made vast use of the modules available in music21 and Python 3.5. As there is little documentation available on the music21 and analysis of music using Music21 and MusicXML, the system had to make most use of the document made provided by MIT in the official Music21 Website.

Firstly the system would need to read and parse any MusicXML file, for this Music21 uses the '*converter.parse(value, *args, **keywords)*' function. This function takes in a URL or a filepath reads and converts the score into a stream. From this it is possible to extract and analyse musical information.

The first analysis carried out by the system is analysing the overall key a music score is written in. The '*stream.analyze(*args, **keywords)*' takes the stream returned from the '*parse()*' function (see above) and returns the best key for the score, this function can also be used to return the key of each instruments in the score, by returning each instrument as a stream(see below). The function analyses the frequency in which some notes are used as opposed to others and returns a key either as a key object or a string object if specified using the '*.name*' function available as a Music21 module.

The system also extracts the name of all the instruments used in a particular score, this is done by using the parts function. MusicXML is split into parts depending on each instrument. Therefore by using the stream returned by the '*parse()*' function, and using the '.parts' function resulting in '*stream.parts*' which returns all the instruments used in the score.
To extract all the chords used by an instrument firstly the stream needed to be flattened using the 'stream.flat' function. This function returns a new Stream where no elements nest within other elements.

To return a stream of just the instrument the '*stream.getElementByID(instrument)*' is used. At first the '*stream.flat.notesAndRests*' seemed a viable option to extract all the chords used, however although it was capable of returning all the notes and rests, it would only return if the type specified in the stream was either 'note' or 'rest'. In a score, chords this function does not allow the extraction of such data. Returning an error (see figure below). This error is because 'chords' are stored in a different way from notes and rests. Chords are essentially multiple notes playing together, therefore chords are stored as an array of notes simply extracting the chord would not provide all the information about the chord.
Figure (1.0)

```
rest
Traceback (most recent call last):
  File "<pyshell#19>", line 2, in <module>
    print(i.name)
AttributeError: 'Chord' object has no attribute 'name'
>>>
```

Therefore to get around this problem, Music21 provides the function '*stream.getElementsByClass(classFilterList)*'. This function is a stream iterator where it iterates over all elements that match the one or more classis in the '*classFilterList*'. However by using this function, each aspect had to be returned one by one, e.g. to return all the notes the function would be '*stream.getElementsByClass('Notes')*' equally for 'Rests' and 'Chords' as well. Using this function with 'chords' returned all the notes used within the chord, allowing to further analyse the notes. Although this would increase the time complexity of the algorithm it would allow extraction of all information needed in order to create a visualisation.

7.3.1 Algorithm for extracting notes, rests and chords:

```
Var instruments = instruments_used(read_mxl)
Var notes = [[]*length(instruments)]
FOR in length(instruments)
    Var inst = readmxl().getElementByID(instruments[i])
    FOR j in inst.flat.getElementsByClass('Note')
        Add to notes name of note
    END FOR
    FOR k in inst.flat.getElementsByClass('Rest')
        Add to notes name of rest
    END FOR
    FOR l in inst.flat.getElementsByClass('Chord')
        FOR m in l.pitches
            Add to notes name of chords
        END FOR
    END FOR
END FOR
```

The algorithm above uses as list of list to add chords respectively according to the index of instruments. E.g. *instruments = [instrument1, instrument2, instrument3]* therefore pitch would be *[[notes of instrument1],[notes of instrument2],[notes of instrument3]]*. The first FOR loop, loops around the first instrument, the second, third and fourth FOR loop loops through all the 'notes', 'rests' and 'chords' respectively and adds each to the respective index in the notes array. The FOR loop inside the 'Chord' loop, iterates through the notes in the 'chord' and appends to the notes array.

This algorithm is essential for all the algorithms used to extract information about the notes such as offset, pitches and note types. Currently the algorithm only adds the name of the note using the '*note.name*' function. In order to extract and add contents such as pitch, note type, and offset (2.1.1) the fundamentals of the algorithm would stay the same with only a change in functions for the notes. To extract the pitch, note type and offset, the functions *'note.pitch.frequency', 'note.duration.quarterLength', 'note.offset',* needs to respectively replace the *'note.name'* function.

- Note.pitch.frequency
    - Gets or sets the frequency of the pitch in hertz
- Note.duration.quarterLength
    - Returns the quarter note length or sets the quarter note length to the specified value
- Note.offset
    - Returns the offset of the note as integer

The quarterLength, offset and frequency are essential pieces to creating the visualisation, they determine the length of a graphic, the length and the height of the visualisation respectively. As the system extracts each information independently and needed to be sorted according to the offset of each note, the objects were not returned in order (lowest to highest). The insertion sort algorithm sorts the offset in ascending order and the objects accordingly as demonstrated in the pseudocode below.

The quarterLength function returns either a string ('*half*') or a fraction as (1 / half), this type of fraction would cause a problem when creating the visualisation, however by simply converting the return to float '*float(1/ half)*' would return the correct value needed for calculations in the visualisation.

7.3.2 Pseudocode for insertion sort \ sorting of offset, quarterLength and frequency

```
Var arr[] allOffsets // offsets not in order
var arr[] myReturn // example pitches to sort according to offset
For index in range(1, length(allOffsets))
        var currentvallue = allOffsets[index] //offsets
        var returnValue = myReturn[index] //pitches
        var position = index
        While (position > 0) and allOffsets[position -1] > currentvalue:
                allOffsets[position] = allOffsets[position -1]
                myReturn[position] = myReturn[position -1]
                position = position -1
        allOffsets[position] = currentvalue
        myReturn[position] = returnValue
```

The algorithm above simply sorts the offsets in order and if a swap is made in the offset, swap the index of array frequency and quarterLength as well whichever array is passed through the argument for the insertion sort function.

To return the clefs and their signs and lines (2.1.1), the system again uses the '*stream.getElementsByClass(classFilterList)',* where '*classFilterList*'= '*Clef*'. Clefs contain sign and lines (2.1.1) to return each respectively the use of 'MensuralClef.line' which returns the staff line the clef resides on and '*MensuralClef.sign*' which returns the sign of clef.
The '*stream.getElementsByClass(classFilterList)'* where '*classFilterList*' = '*MetronomeMark*', returns the metronome mark (2.1.1). To return the offset of the offset of 'metronomeMark' use the function '*.offset*' as mentioned above.

To return the measures or bar (2.1.1) and offset of the measures Music21 provides the '*stream.measures(numberStart, numberEnd)*' function. This function allows the return of all the measures given the numberStart and numberEnd variable. To get all the measures '*numberStart = 0*' and '*numberEnd = none*', to get the offset use *'.offset()*' (see above) function with the measures function.

All the functions mentioned above were implemented in the final version of the software, however these were not the original methods used to extract information. A brief summary of how the previous methods worked and why they were not chosen above the final version is stated below.

7.3.3 Getting pitches

As seen in figure (1.0) above, trying to get the name of the chords using the '*.name*' function would throw an error. The error here was similar to the one explained above, also using the function '*.notesAndRests*' it would return a rest however, rests have no frequency, therefore an if statement stating: IF j.name == 'rest' THEN continue, had to be written. Rather than attempting to extract all the information at once, getting each information one by one proved more stable and reliable, therefore the decision to use '*getElementsByClass()*' was made.

7.3.4 Getting note types

The function go get note types again tried to extract all notes and rests at once using 'notesAndRests'. This function used 'duration.fullName' which returned the full name of the note (2.1.1) as a string. The system required the duration of these notes to be either float or integer, to do this a function to convert the string to a float e.g. half = 2.0 is used. The function to convert simply has an array the full names of notes [whole,half, quarter,….., 128^{th}], another array with each of the notes number associates [4.0, 2.0, 1.0,……,0.03125] and an array dotType ['none', 'dotted'…..'triple'].

The string passed in the function argument would first be split 'double dotted half' would be ['double', 'dotted', 'half']. Now half would return as 2.0 and double as 'double' and passed through the formula: $a_n = a(1 + ½ + ¼ + … + 1/2^n) = a(2 – 1/2^n)$.

Where a = 2.0 (type of note) and n = 2 (double), this would return the actual duration in which each note should be played, this had to be applied to every note returned. This solution also didn't take into account the amount of dots can range to greater than 'triple' and the note duration could range from less than '0.03125'. However a better solution which was more efficient shorter replaced the current one. By simply returning '*quarterLength*' and converting it to float using pythons '*float()*' function.

7.3.5 Getting Clefs

This method for getting clefs is more accurate than the one I decided to use in the final implementation. Here I simply gathered all the possible clefs and put them in an array and for each clef I stored their Music21 counterpart into another array.

Here the function would only select the first measure of each instrument and save each line of the measure as a string, then it would search for that string in the array if found, it would return the found string counterpart. E.g. <music21.clef.TrebleClef> would return '*TrebleClef*'.
Then it would search the returned string for 8va or 8va (2.1.1) it would return the appropriate sign / line for the clef.

However the problem with this was, the sign and line for '*TrebleClef*' or '*Treble8vaClef*', '*Treble8vaClef*' and for other clefs were exactly the same. Therefore this concluded that it would be better to use '*getElementsByClass()*' function, the unused method however could be used for future works(10.0.0).

7.4.0 Creating Visualisation

The implementation of creating visualisation importing all functions of extracting musical information.

Firstly by importing the '*get_measure_offset()*' and '*allPitchesnotes()*' function, the functions '*maxHeight()*' and '*maxLength()*' determines the maximum height and length of the surface in which the visualisation needs to be created.

To determine the height of the surface the 'maxHeight()' takes the highest frequency returned by '*allPitchesNotes()*' and divides it by 10 and multiplies it by the number of instruments returned by '*instruments_used()*'. This means that all frequency values that is to be used in the implementation needed to be divided by 10, this is done in the 'calculatePitches()' function where the imported pitches from 'allPitches()' are divided by 10 and returned, as the pitches would also contain string values for 'rest' this needed to be ignored and stored as 'rest' in the returning array.(6.3.11)

Pseudocode

```
FOR i in length(allPitches())
        IF k == 'rest' THEN
                ADD to actualPitches 'rest'
                  CONTINUE
        ELSE
                ADD to actualPitches VALUE allPitches[i] / 10
        END IF
END FOR
```

A similar algorithm is used for measures, to determine the length of the surface the *'maxLength()*' function takes the highest offset returned by '*get_measure_offset()*' and multiplies the result by 5. As stated in the algorithm above, however this time '*allNoteTypes()*' would replace '*allPitches()*' and instead of dividing by 10, the values would be multiplied by 5(6.3.11).

7.4.1 Determining the colour of keys and dynamically changing RGB values according to metronome marks.

Pseudocode for function keysAndColour(theKey)

```
VAR overall = theKey
Array [] instKeys = keyOfInstrument(read_mxl())
ARRAY [] = keys [] \\ all keys mentioned in background section*
ARRAY [] = colour[] \\ all keys mentioned in background section
VAR greatest = 0
\\ get most similar key by comparing the value in keys and overall
FOR j in length(keys)
        a = overall
        b = keys[h]
        VAR seq = difflib.seQuenceMatcher(a= a.lower(), b = b.lower))
        ratio = seq.ratio()
        IF ratio >= greatest
                greatest = ratio
                index = j
END FOR
chosen = keys[index]
chosenColour = colour[index]

\\ choosing how to change RGB values according to the keys
IF greatest == 1.0:
        ADD to array toRETURN[0] value of chosenColour
        ADD to array toReturn[1] value of chosen
END IF
IF greatest < 1.0:
        IF overall[1] == '-' and chosen[1] == '-'
                ADD to toReturn[0] value of chosenColour
                ADD to toReturn[1] value of 0
        \\ Do this for all combinations of chracters (- , \ and ' ') and add respective changing values
from 0 to -2 as explained below.
```

The purpose of the above pseudocode is to firstly find the similarity ratio between the key passed as argument in the function and the keys available (2.1.1). This is done by using the importing the '*re*' function available for python. It simply finds the similarity between two strings and returns value between 0 and 1 where 0 is not same at all and 1 being exactly similar.

If the ratio value returned is 1 then no changes need to be made to the colour selected and therefore the value to be added to the empty array would be 0 indicating a change in the RGB values need to be made(*7.4.1*).

The values chosen are according to the results of the survey conducted (9.1.1).
If the ratio is less than 0 then changes may need to be made, if the 2nd value of variable overall and chosen are the same respectively:

- '-' and '-'
- '#' and '#'
- ' ' and ' '

Then the value added to toReturn[1] is 0
If they are not the same and the values are respectively:

- ' ' and '#'

- '-' and ' '

Then the value addd to toReturn[1] is 1
If they are not the same and the values are respectively:

- ' ' and '-'
- '#' and ' '

Then the value addd to toReturn[1] is -1
If they are not the same and the values are respectively:

- '-' and '#'

Then the value addd to toReturn[1] is 2
If they are not the same and the values are respectively:

- '#' and '-'

Then the value addd to toReturn[1] is -2

7.4.2 Pseudocode for function ColourOfKey()

```
Array [] = colour[] \\ all colours mentioned in background section*
Array [] = red[] \\ the RED values for all colours mentioned in colour array
Array [] = green[] \\ the GREEN values for all colours mentioned in colour array
Array [] = blue[] \\ the BLUE values for all colours mentioned in colour array
FOR i in keysAndColour
        getKey = keysAndColour(instruments_used[i])
        select = getKey[0]
        color = select[0] \\ from the array returned from function keysAndColour()
        toMul = [1] \\ from the array returned from function keysAndColour()
        multiply = toMul[0]
        FOR j in colour
                IF color = colour[j] THEN
                        r = red[j]
                        g = green[j]
                        b = blue[j]
                END IF
                        IF multiply == 0 THEN
                                r = r
                                g = g
                                b = b
                        END IF
                        IF multiply == 1 THEN
                                r = r / 0.47
                                g = g /1.07
                                b = b / 1.07
                        END IF
                        IF multiply == -1 THEN
                                r = r * 0.47
                                g = g * 1.07
                                b = b * 1.07
                        END IF
                        IF multiply == 2 THEN
                                r = r / 0.47^2
                                g = g /1.07^2
                                b = b / 1.07^2
                        END IF
                        IF multiply == -2 THEN
                                r = r * 0.47^2
                                g = g * 1.07^2
                                b = b * 1.07^2
                        END IF
        END FOR
END FOR
```

The purpose of the algorithm mentioned above is to detect changes in the key and apply appropriate changes to RGB values, if there isn't a perfect match between the key of the instrument and the keys available with colours (7.1.1).

7.4.3 Pseudocode for function getRGB(re, gr, bl, metOne, metTwo)

```
Array RGB = []
VAR bpmDiff = (metTwo – metOne) / 10 (to determine the power of value: see results section)
IF re == 0 and bl == 0 and gr == 0 THEN
        VAR myRed, myGreen, myBlue = 1.22, 2.53, 2.15
END IF
IF bpmDiff > 0
        r = myRed / 1.22^abs(bpmdiff)
        g = myGreen / 2.53^abs(bpmdiff)
        b = myBlue / 2.15^abs(bpmdiff)
END IF
IF bpmDiff > 0
        r = myRed * 1.22^abs(bpmdiff)
        g = myGreen * 2.53^abs(bpmdiff)
        b = myBlue * 1.2.15^abs(bpmdiff)
END IF
```

This function applies changes to RGB values according to the difference found between metOne, metTwo, the resulting value of the difference must always be positive. (see results sectoin)
The functions '*OverAllColour()*', '*colourOfKey()*' and '*getRGB()*'contain if statements to cap R,G,B values to 255 as RGB values do not exceed 255.

7.5.0 Drawing the graphics and determining the graphic variables

To draw a rectangle, circle and polygon, Pygame provides functions to do so 'pygame.draw.rect(Surface, color, Rect, width=0)', 'pygame.draw.circle(Surface, color, pos, radius, width=0)' and 'pygame.draw.polygon(Surface, color, pointlist, width=0)' respectively.
The 'pygame.draw.rect()' function takes 4 arguments:

- Surface :- chose which surface to draw on
- Color: - chose the colour of the shape as RGB values
- Rect:- x and y coordinates of top left hand corner of rectangle set as [x,y] also takes width and height values set as [x,y,width,height]
- Width = 0: Leaving the width value as 0 will fill the shape with colour specified, 1 will only colour the border as colour specified.

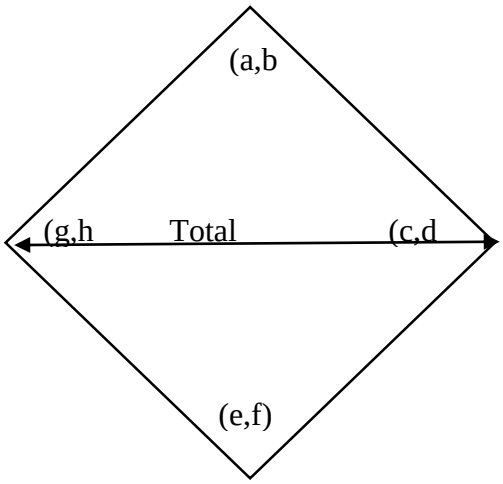
The 'pygame.draw.circle()' function takes 5 arguments:

- Surface :- chose which surface to draw on
- Color: - chose the colour of the shape as RGB values
- Pos:- x and y coordinates specifies the centre point of the circle of rectangle set as [x,y]
- Radius: The radius of the circle
- Width = 0: Leaving the width value as 0 will fill the shape with colour specified, 1 will only colour the border as colour specified.

The 'pygame.draw.polygon()' function takes 4 arguments:

- Surface :- chose which surface to draw on
- Color: - chose the colour of the shape as RGB values
- Pointlist:- ((a,b),(c,d),(e,f),(g,h)) specifies each point of the polygon (able to have more) .
- Width = 0: Leaving the width value as 0 will fill the shape with colour specified, 1 will only colour the border as colour specified.

The polygon that was required was a diamond, while drawing a diamond in a set of points proved simple, however dynamically changing it according to the length of note proved challenging. Below is the solution for enlarging the polygon while maintaining the same shape.



- Total distance = length of note (6.2.0)
- (g,h) = (g, h) argument from function diamond
- (a,b) = (g + (Total distance /2)), (h - (Total distance /2))
- (c,d) = ((g + Total distance), h)
- (e,f) = (g + (Total distance \2), h + (Total distance \2))

7.6.0 Creating the overall visualisation

Pseudocode for function drawShapes() broken down into subsections
Set all variables:

```
VAR duration = durationOfNote()
VAR all Colourkeys = colourofKey()
VAR pitches = calculatePitches()
VAR instKeys = keyOfInstrument(read_mxl())
VAR total_Height = maxHeight() / length(instruments_used(read_mxl))
VAR clefsSignsAndLines = allClefs()
VAR clefsOnly = clefsSignsAndLines[0]
VAR instPart = 0
VAR myMetronome = metronomeMark()
VAR myMetOffset = []
FOR i in myMetronome[1]
    ADD to myMetOffset values (i * 5) \\ as mentioned above all offset values need to be multiplied
by 5
END FOR
```

Set temporary variables

```
FOR i in length(pitches)
        tempDuration = duration[i]
        tempPitches = pitches[i]
        currentClef = clefsOnly[i]
END FOR
```

Summary

As duration, pitches and clefs are returned in a list of lists e.g. [[], [], []] each iteration of the list of lists is the instrument that needs to be visualised, setting each iteration as individual temp arrays will help visualising each instrument easier, each temp array will determine the type of graphic and its graphic variables.

Chose graphic variables, colour and line or filled shape:

```
    FOR j in length(tempDuration)
            IF tempDuration[j] == 20 or tempDuration[j] == 10 THEN
                    line = 1
            END IF
            ELSE
                    line = 0
            IF tempPitches[j] != 'rest':
                    myRed = allColourKeys[0]
                    myGreen = allColourKeys[0]
                    myBlue = allColourKeys[0]
                    for i in length(instKeys)
                            IF instPart == (total_height * i)
                                    r = myRed[i]
                                    g = myGreen[i]
                                    b = myBlue[i]
            END IF
```

Summary

The first if statement checks for 20 and 10 in the tempDuration set in the above algorithm, it simply states that if the value is 10 or 20 then don't fill, else fill the graphic with colour.
The second if statement checks if any values in tempPitches are 'rest', if the value is 'rest' then don't draw anything, else draw object. If the statement returns true then set temporary myRed, myGreen, and myBlue values. Again the allColourKeys is a list of lists where the first, second and third index of the list of list are red, green and blue values respectively. Then if statement inside the for loop checks to see if the system is till drawing variables for the same instrument. Then the r, g, b values are set for the current instrument.

Determining if clef is G and drawing Rectangle

```
IF currentClef == 'G'
                        gettingColor = getRGB(r,g,b, myBPM[myCounter -1], myBPM[myCounter])
                        color = [gettingColor[0], gettingColor[1], gettingColor[2]]
                        DRAW rectangle(totDuration, tempPitches[j], instPart, color,
                                        tempDuration[j],line)
                        IF totDuration >= bpmOffset and myCounter < length((myBPM) -1)
                            mycounter+=1
                        END IF
                       totalDuration+=tempDuration[j]
                END IF
        END FOR
END FOR
```

Summary

The first if statement here recognises the type of clef and the type of graphic to draw. If currentClef == 'G' then draw a rectangle, if currentClef == 'F' draw a circle, if currentClef == 'C' draw a diamond/ polygon.
the gettingColor variable is returned as an array where the RGB galues are input to the functionRGB and also the values of the metronome mark (myBPM) at index 'j' and the one before it. This is to detect change in tempo and change the RGB values according to the increase and decrease in tempo.
To draw the shapes, each shape takes the exact same arguments where:

- xval = totDuratoin
- yval = tempPitches[j]
- instPart = instPart
- color = color
- shapeW, theRad, theSize = tempDuration[j]
- width = lineSize

The arguments for each shape (see above) is as such

Rectangle

- pygame.draw.rect(visualDisplay, color, [xval, (instPart + yval)], shapeW, shapeW)

Circle

- pygame.draw.circle(visualDisplay, color, [xval, instpart + yval] theRad, line)

Diamond/ polygon

First map all the x and y (see above) values and then draw

- variable polyp = (xval + (theSize / 2) , yval + instPart - (theSize/ 2)), (xval + theSize, yval + instPart), (xval + (theSize/2), yval + instPart+(theSize/2)),(xval , yval + instPart)
- pygame.draw.polygon(visualDisplay, color, polyp, lineSize)

The final if statement checks to see if a tempo change has occurred, if so then the variable '*myCounter*' is incremented by 1. The '*totalDuration +=tempDuraion*' sets the place for the next graphic to be drawn.

8.0.0 Testing

The purpose of testing is to validate and verify the system against requirements gathered from the system specifications. The tests will targets the systems:

- Errors: actual coding errors made during implementation, the difference between the desired output and system output
- Fault: bugs in the system, basically any type of error that prevents the system from fulfilling its requirements
- Failure: the inability of the system to perform the desired task.

The purpose of validation is to:

- Ensure system developed is as the requirements
- Emphasise requirements

The purpose of verification is to:

- Ensures system is according to design specifications
- Verifies concentrates on the design and system specifications

The tests will be carried out with the use of test cases, test cases are documents which has a set of test data, preconditions, expected results and post conditions, developed in order to verify fulfilment of a specific requirement. The tests will be carried out for the requirements of both the survey and the visualisation system.

8.1.0 Testing implementation of survey

Project Name: Visualisation of music
Test Case
Test Case ID: Su_1
Test Priority (Low/Medium/High): High
Module Name: Survey
Test Title: Survey
Description: Survey to determine whether colours and feelings can be mapped to a certain key
Pre-conditions: Database should be live in order for data to be saved

Test 1

Test Step:

Songs related to a particular key and beat will be played, looped automatically for the users to hear.

- Users should also be able to pause the song by clicking the 'pause' button.
- Users should also be able to skip to any part of the song by moving the 'seek' button

Test data: '1.MP3'

Expected Result	Actual Result	Pass/Fail
Upon loading of screen, the song is played automatically	Song played automatically successfully	**Pass**
If Play/Pause Button is clicked song performs play/pause action	Upon clicking of button, song played or paused	**Pass**
Skip to any part of song	Song skipped to relevant part	**Pass**

Test 2

Test Step:
Users will be able to select a colour that are displayed in a table by ticking the 'checkbox' button.

Test data: 'Red'

Expected Result	Actual Result	Pass/Fail
The checkbox changes from 'empty' to 'ticked'	Checkbox is successfully ticked	Pass

Test 3

Test Step:
Users will be able to select a feeling that are displayed in a table by ticking the 'checkbox' button.

Test data: 'Brilliant'

Expected Result	Actual Result	Pass/Fail
The checkbox changes from 'empty' to 'ticked'	Checkbox is successfully ticked	Pass

Test 4

Test Step:
If users want to select more colours they should be able to click a '*select colour*' button, which opens a window of colours.

Test data: 'Button'

Expected Result	Actual Result	Pass/Fail
A window pops up showing an array of colours	A window successfully popped up showing an array of colours	Pass

Test 5

Test Step:
The users will be limited to 3 checkboxes for feelings and colour.

- An alert should pop up stating users can only select a maximum of 3 checkboxes for each section.

Test data: 3 feelings, 3 colours

Expected Result	Actual Result	Pass/Fail
An alert box shows up stating they can only check 3 maximum colours	An alert box successfully showed up stating they can only check 3 maximum colours	Pass
An alert box shows up stating they can only check 3 maximum feelings	An alert box successfully showed up stating they can only check 3 maximum colours	Pass

Test 6

Test Step:
Upon clicking the 'Next' button, the page should load a new song which fulfil FR.1 along with refreshed checkboxes.

Test data: '2.MP3'

Expected Result	Actual Result	Pass/Fail
New song automatically loads upon clicking of 'Next' song	New song successfully loaded upon clicking of 'Next' song	Pass

Test 7

Test Step:
Upon clicking the 'Next' button, the data users selected should be added to the database.

Test data: All data selected by user

Expected Result	Actual Result	Pass/Fail
Data added to database	Successfully add to database	Pass

Post Condition

An output should be created for each test in the form of a JPEG file or printed on screen.

Evaluation

The test cases show that the survey fulfils its requirements without any fails, the test cases determines that there were no errors while, users selected their chosen data, the automation and freedom of playing and seeking songs.
They also show that the data were all saved successfully without errors into the database for further analysis.

8.2.0 Testing implementation of visualisation system

Project Name: Visualisation of music
Test Case
Test Case ID: Vis_1
Test Priority (Low/Medium/High): High
Module Name: Extraction and Visualisation
Test Title: Visualisation
Description: The system automatically extracts information of given MXL file and analyses it to create a visualisation
Pre-conditions: The mxl file is already loaded

Test 1

Test Step:

System should be able to extract all required information from MXL file.

- Instruments
- Keys
- Notes
- Chords
- Clefs
- Rests
- Offsets
- Measures
- Metronome mark

Test data: 'The_legend_of_korra.mxl'

Expected Result	Actual Result	Pass/Fail
The Instruments are saved into the array are saved to an array	Successfully extracted and saved 5 instruments into an array	**Pass**
The Keys are saved into the array are saved to an array	Successfully extracted and saved 5 key into an array	**Pass**
The notes are saved into the array are saved to an array	Successfully extracted and saved notes into an array	**Pass**
The Chords are saved into the array are saved to an array	Successfully extracted and saved Chords into an array	**Pass**
The Clefs are saved into the array are saved to an array	Successfully extracted and saved 5 clefs into an array	**Pass**
The Rests are saved into the array are saved to an array	Successfully extracted and saved rests into an array	**Pass**
The Offsets are saved into the array are saved to an array	Successfully extracted and saved offsets into an array	**Pass**

The Measures are saved into the array are saved to an array	Successfully extracted and saved measures into an array	**Pass**
The Metronome marks are saved into the array are saved to an array	Successfully extracted and saved metronome marks into an array	**Pass**

Test 2

Test Step:
Implement the results of the survey and map to relevant keys.

Test data: 'string key'

Expected Result	Actual Result	Pass/Fail
Upon selection of key, a colour should be selected form an array of colours	Successfully returned a colour, when key was selected	**Pass**

Test 3

Create an output for any mxl file as JPEG image

Test data:

- TS.1: The_legend_of_korra.mxl
- TS.2: let_it_go_By_James_Bay.mxl
- TS.3: Layla.mxl

ID	Expected Result	Actual Result	Pass/Fail
TS.1	Visualisation output for 'The_legend_of_korra.mxl'	Successful output of mxl file as JPEG file	**Pass**
TS.2	Visualisation output for 'let_it_go_By_James_Bay.mxl'	Successful output of mxl file as JPEG file	**Pass**
Ts.3	Visualisation output for 'Layla.mxl'	Error while producing JPEG file.	**Fail**

TS.3 error output

```
[]
Traceback (most recent call last):
  File "seconddrawing.py", line 479, in <module>
    drawShapes()
  File "seconddrawing.py", line 398, in drawShapes
    bpmOffset = myMetOffset[myCounter]
IndexError: list index out of range
```

Test 3

Draw Shapes relevant to each clef

Test data:

- TS.1: G
- TS.2: F
- TS.3: C

Expected Result:

ID	Expected Result	Actual Result	Pass/Fail
TS.1	Should draw 'square' on canvas	Successful drawing of 'square' on canvas	Pass
TS.2	Should draw 'circle' on canvas	Successful drawing of 'circle' on canvas	Pass
Ts.3	Should draw 'diamond' on canvas	Successful drawing of 'diamond' on canvas	Pass

Test 4
Colours declared in text 'white' should be mapped to their RGB values '255,255,255', as for any other colour declared

Test data:

- TS.1: White
- TS.2: Black
- TS.3: Red

ID	Expected Result	Actual Result	Pass/Fail
TS.1	Return 255,255,255	Successfully returned 255, 255,255	Pass
TS.2	Return 0,0,0	Successfully returned 0, 0,0	Pass
Ts.3	Return 255,0,0	Successfully returned 255, 0,0	Pass

Evaluation of tests for Visualisation of music

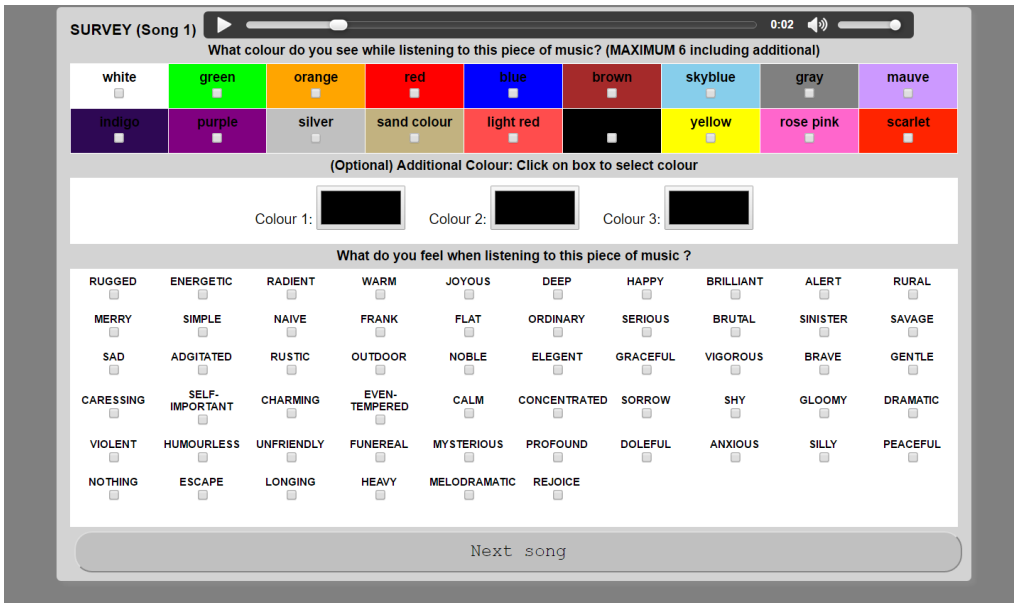
Overall the test cases show that the system fulfils most requirements successfully. The system showed passes for all tests except from TS.2 in test 3.
Test 3 tested to see if the system produced an output for 3 different mxl files. It is crucial that the system produces an output successfully to illustrate a piece of visualisation as it is an important functional requirement for the system.

The test id TS.3 failed, the expected result was an output of visualisation, the actual output as seen on TS.3 error output is a 'list index out of range' in the function' draw shapes. From the error shown it is unclear as to what is causing this error. However as the job of the function is to simply draw a shape from the gathered data, the cause of the error may be within the class 'extracting musical information'. As the classes' main function is to gather and store data appropriately, this indicates that some type of data has not been extracted and stored properly into a list.

9.0.0 Results and evaluation

The survey tests the hypothesis given by Harry Farjeon [11] and Albert Lavigance [10]. They have attempted to connect keys with colour and feelings respectively. The survey simply played a song written in a particular key, the results of the survey would determine the colours used for the system and possibly mapping colours with feelings to ensure that the visitation gave a certain feeling.

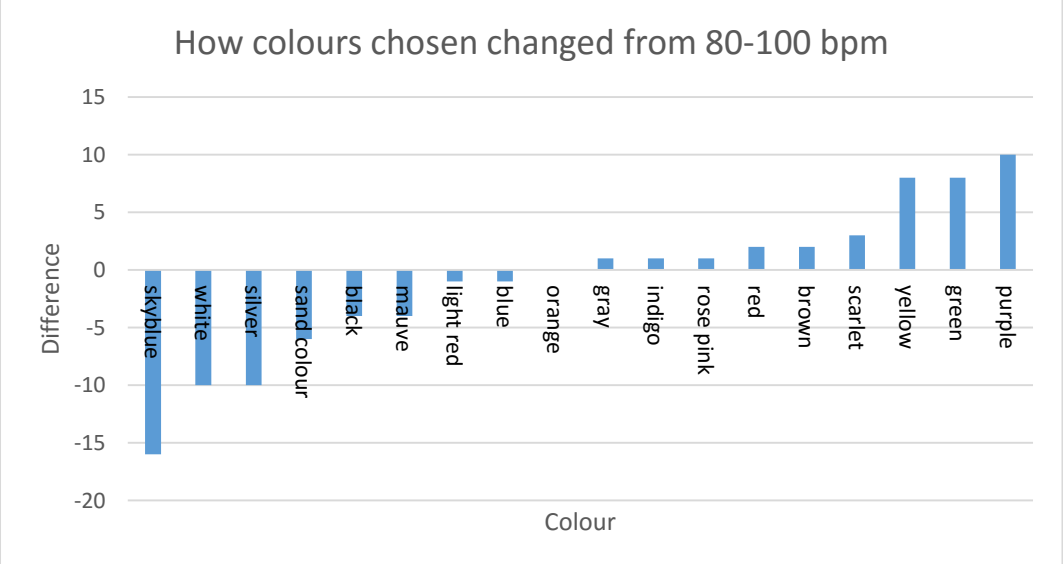
How the survey looks:



9.1.0 Results of survey for colour

Detecting the change in RGB values according to increase in BPM

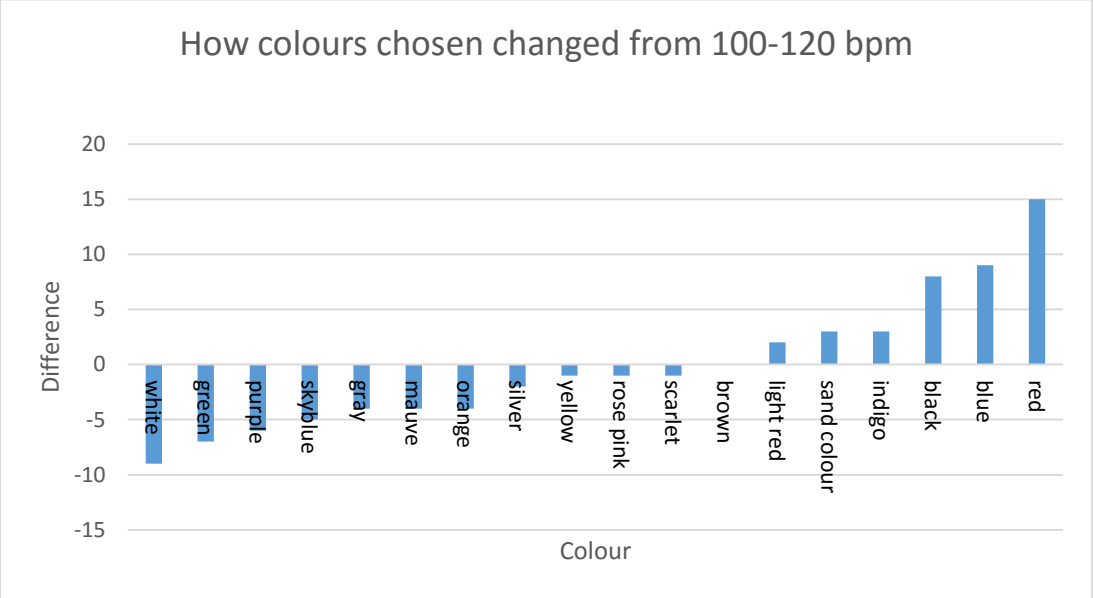
This graph shows how the total colour selected changed as the beat increased from 80-100 BPM



This table shows the average RED, GREEN and BLUE values for the colours which showed negative and positive change for 80-100 BPM.

	Negative	Positive	Change Of
RED	154.4	165.2	0.93
GREEN	132.6	91.8	1.45
BLUE	174.6	61.8	2.83

This graph shows how the total colour selected changed as the beat increased from 80-100 BPM



This table shows the average RED, GREEN and BLUE values for the colours which showed negative and positive change for 100-120 bpm.

	Negative	Positive	Change Of
RED	187.5	125.0	1.50
GREEN	158.8	43.8	3.62
BLUE	127.0	85.7	1.48

For both 80-100 and 100-120: Change Of = Negative/ Positive

Calculating the average Change of for a change of 10BPM.

	AVERAGE CHANGE
RED	1.22
GREEN	2.53
BLUE	2.15

This shows how the RGB values change as the bpm increases, the results mean that for every RED, GREEN, BLUE values if the BPM increases by 10 divide the original R,G,B by 1.25, 2.53 and 2.15 respectively.

As for decreasing by 10 BPM multiply the RED, GREEN, BLUE values by 1.25, 2.53, 2.15 respectively.

Essentially for any increase or decrease, multiply or divide by the RGB values by the respective change, to the power of BPM change abs(10) divided by 10 (1).

- RED value = 100
- BPM CHANGE = abs(-20) therefore 20
- 20* 10 = 2
- Since the BPM change is a negative change we multiply
- New RED value = 100 * 1.25^2

This is true for all RGB values if a change of tempo is detected otherwise the RGB value remains the same.

9.1.1 Detecting the change in RGB values between normal keys and sharp counterparts.

This table shows the average RED, GREEN and BLUE values for the colours which showed negative and positive change for 80 -120 bpm and C.

	Negative	Positive	Change of
RED	149.00	154.00	0.97
GREEN	173.14	68.25	2.54
BLUE	161.14	53.63	3.00

This table shows the average RED, GREEN and BLUE values for the colours which showed negative and positive change for 80 -120 bpm and C#.

	Negative	Positive	Change of
RED	218.13	106.13	2.06
GREEN	166.00	70.00	2.37
BLUE	168.25	59.88	2.81

This table shows the average RED, GREEN and BLUE values for the colours which showed negative and positive change for 80 -120 bpm and G.

	Negative	Positive	Change of
RED	123.00	213.17	0.58
GREEN	111.75	102.00	1.10
BLUE	121.13	87.50	1.38

This table shows the average RED, GREEN and BLUE values for the colours which showed negative and positive change for 80 -120 bpm and G.

	Negative	Positive	Change of
RED	147.73	193.14	0.76

GREEN	121.55	102.14	1.19
BLUE	70.82	167.71	0.42

This table below shows by how much the Change of increased or decreased while changing from C to C#

RED	0.470736
GREEN	1.069774
BLUE	1.069386

This table below shows by how much the Change of increased or decreased while changing from G to G#

RED	0.754404
GREEN	0.920697
BLUE	3.278318

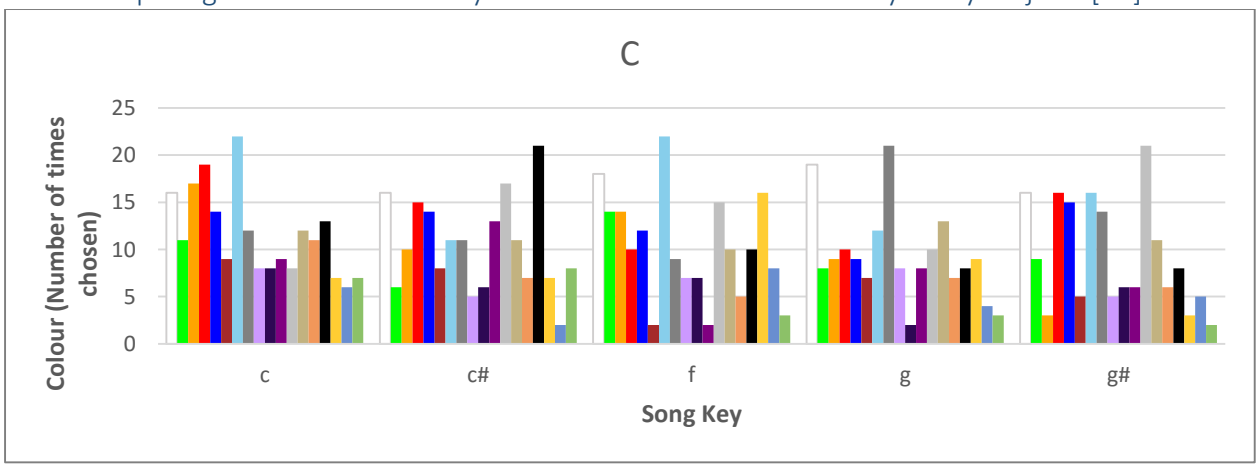
By taking the average of each increase or decrease of each RGB values from C to C# and G to G# we can determine how much the RGB values should increase or decrease by if the key changes from a normal key to a # key, as seen in the table blow.

RED	0.61257
GREEN	0.995235
BLUE	2.173852

Then we follow the same principle as stated in (above) meaning if the key changes from G to G# divide the RGB values by the ones stated above. With this result it is also possible to determine how the RGB values change if the key changes from e.g. G# to G, to do this we multiply the RGB values by their respective results in the table above.

From this we can also determine the change in RGB values if the key changes from a '#'(sharp) to '-' (flat) or vice versa. (see background). This is done by squaring the RGB values stated above and multiplying and dividing the RGB values appropriately.

9.1.2 Comparing the results of survey with the research conducted by Harry Farjeon [11]



The two most common colour for each key

Key	Colour	Total	Colour	Total
C	Sky blue	22	Red	19
C#	Black	21	Silver	17
F	Sky Blue	22	White	18
G	Grey	21	white	19
G#	Silver	21	White, sky blue	16

9.1.3 The table below shows how the results of the survey compared with the opinions of Harry Farjeon and his students.

Key	Colour: Harry Farjeon	Colour: Students of Harry Farjeon	Colour: Survey
C	White	White, cream, grey	Skyblue, red
F	Brown	White	Sky blue, white
G	Green	Red	White

9.1.4 Evaluation

The purpose of the survey was to test the hypothesis placed by Harry Farjeon, and to see if it is possible to objectively say that a certain key belongs to a certain colour, (see design and specification). Although the data collected by the survey may not be sufficient, the data does suggest that as suggested by (name here), any form of art may always be truly subjective. By comparing the results of Farjeon, his students' and the survey, the data shows:

- Only one match between Farjeon and his students(Key: C, colour: White)
- Only one match between Farjeons students and the survey (Key: F, colour: White)
- No matches between the survey and Farjeon

However the system still uses the colours and key stated by Harry Farjeon, although the data has little justification to it regarding the connection between keys and colour, it would be most reliable to use Harry's subjection to generate the visualisations as the survey did not test all the possible keys.
It is important to note that due to the design restrictions of the survey, and that in the future it may be possible to generate results which justify the connection between keys and colours if more keys were used in a survey.

9.2.0 Results of survey feelings.

The purpose of this part of the survey was to test the theories presented by Hugo Reinmann [10] and Albert Lavigance [10], in order to see if a key could be connected to a particular group of feelings:

- All steps upwards make the character brighter and more radiant
- Sharp side major the general mood was joyous and brilliant

The tests intended to see whether a change in tempo had a more 'brighter and radiant change '. And whether a change from normal major to sharp did indeed create a more 'joyous and brilliant' feeling.

9.2.1 Frequency of feeling as BPM increases

80bpm

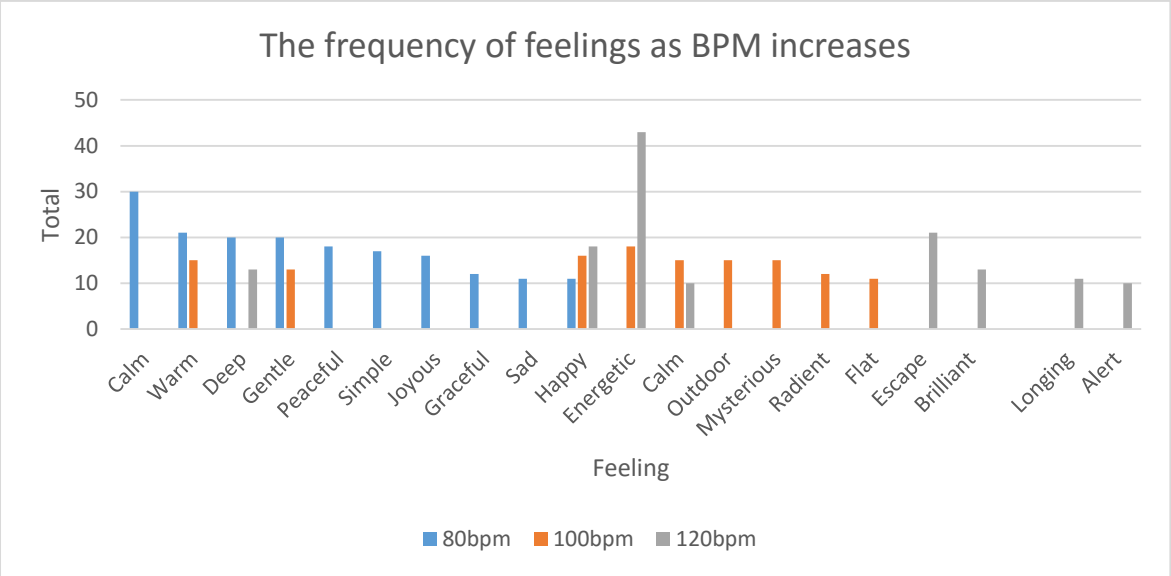
CALM	30
WARM	21
DEEP	20
GENTLE	20
PEACEFUL	18
SIMPLE	17
JOYOUS	16
GRACEFUL	12
SAD	11
HAPPY	11

100bpm

ENERGETIC	18
HAPPY	16
CALM	15
OUTDOOR	15
MYSTERIOUS	15
WARM	14
JOYOUS	13
GENTLE	13
RADIENT	12
FLAT	11

120bpm

ENERGETIC	43
ESCAPE	21
HAPPY	18
JOYOUS	16
OUTDOOR	13
BRILLIANT	13
DEEP	13
LONGING	11
CALM	10
ALERT	10



9.2.2 Evaluation

The results show that at a beat of 80bpm, the general moods were moods generally associated with calmness: Calm, warm, deep, peaceful etc.

As the beat increased from 80-100 bpm, the frequency in which the moods appeared in 80bpm decreased, and the moods generally associated with 'bright' saw an increase: Happy, energetic, outdoor, radiant.

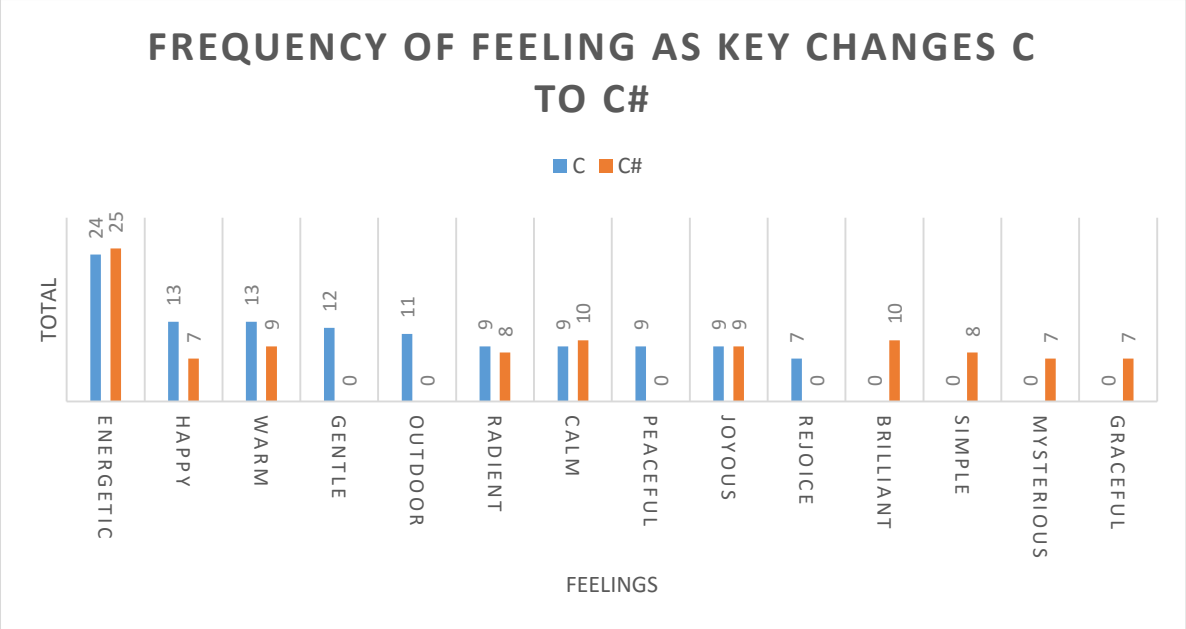
At 120bpm the moods associated with upbeat saw a dramatic increase, with moods such as Energetic showing a large increase from 100bpm, of which showed no appearance at 80bpm. Feelings such as, escape, brilliant and alert also make an appearance at 120bpm.

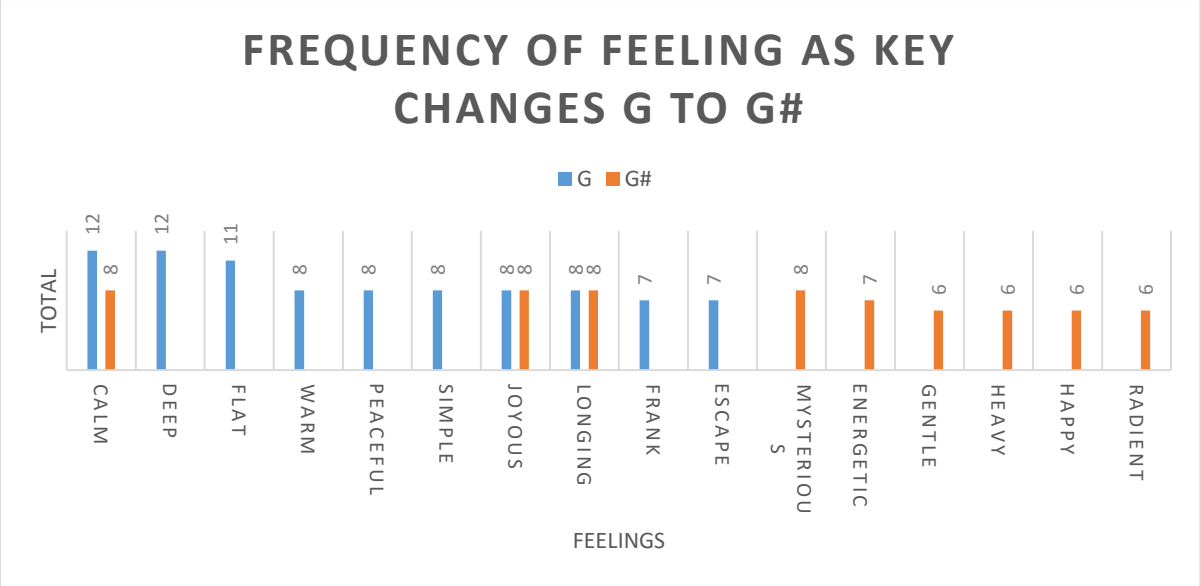
This result shows overall that as the beat increases does indeed make the character of a piece 'brighter and more radiant'.

9.2.3 Frequency of feelings as key changes to sharp (#)

Feeling	C	C#
Energetic	24	25
Happy	13	7
Warm	13	9
Gentle	12	0
Outdoor	11	0
Radient	9	8
Calm	9	10
Peaceful	9	0
Joyous	9	9
Rejoice	7	0
Brilliant	0	10
Simple	0	8
Mysterious	0	7
Graceful	0	7

Feeling	G	G#
Calm	12	8
Deep	12	0
Flat	11	0
Warm	8	0
Peaceful	8	0
Simple	8	0
Joyous	8	8
Longing	8	8
Frank	7	0
Escape	7	0
Mysterious	0	8
Energetic	0	7
Gentle	0	6
Heavy	0	6
Happy	0	6
Radient	0	6





9.2.4 Evaluation

Albert Lavigance [10] stated that 'Sharp side major (keys), the general mood was joyous and brilliant'. According to the data above for C, the most common moods were energetic, happy, warm, gentle outdoor and radiant. As for the sharp side major of the key (C#) the most common moods were: Energetic, calm, brilliant, simple and radiant.

While comparing the moods for C and C#, moods such as: Energetic brilliant and radiant saw an increase, while others show a decrease in frequency.

For G the most common moods were, calm, deep, flat and warm, as for G# the most common moods were, calm, joyous, longing and mysterious.

However comparing the increase and decrease in frequency, moods such as deep, flat and warm saw a dramatic decrease, and moods such as radiant, mysterious and energetic saw a rapid increase in frequency.

According to the data presented, Sharp side majors do have a more joyous and brilliant mood, or moods that are generally associated with the mentioned moods. Therefore it may be possible to justify that certain keys do create a certain feeling unlike the results shown through the results of colour.

9.3.0 Visualisation outputs of the system.

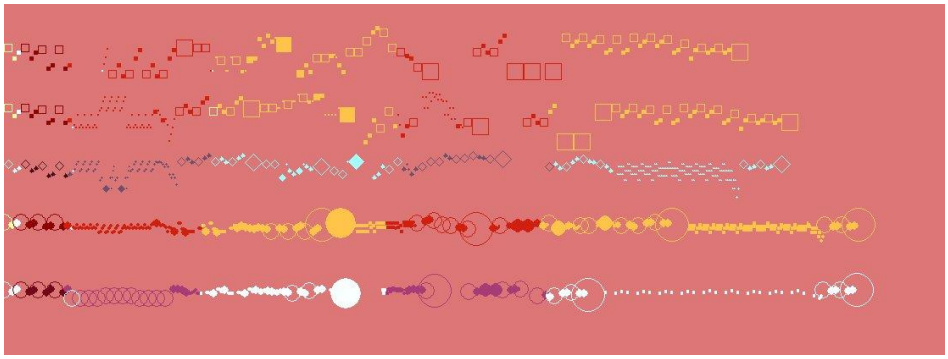
Below are the outputs of the visualisation for 8 scores of music (MusicXML files), each is divided into 3 subgroups:

- Effective outputs which illustrate the music score well (effective outputs).
- Outputs which failed illustrate the music score very well (non-effective)

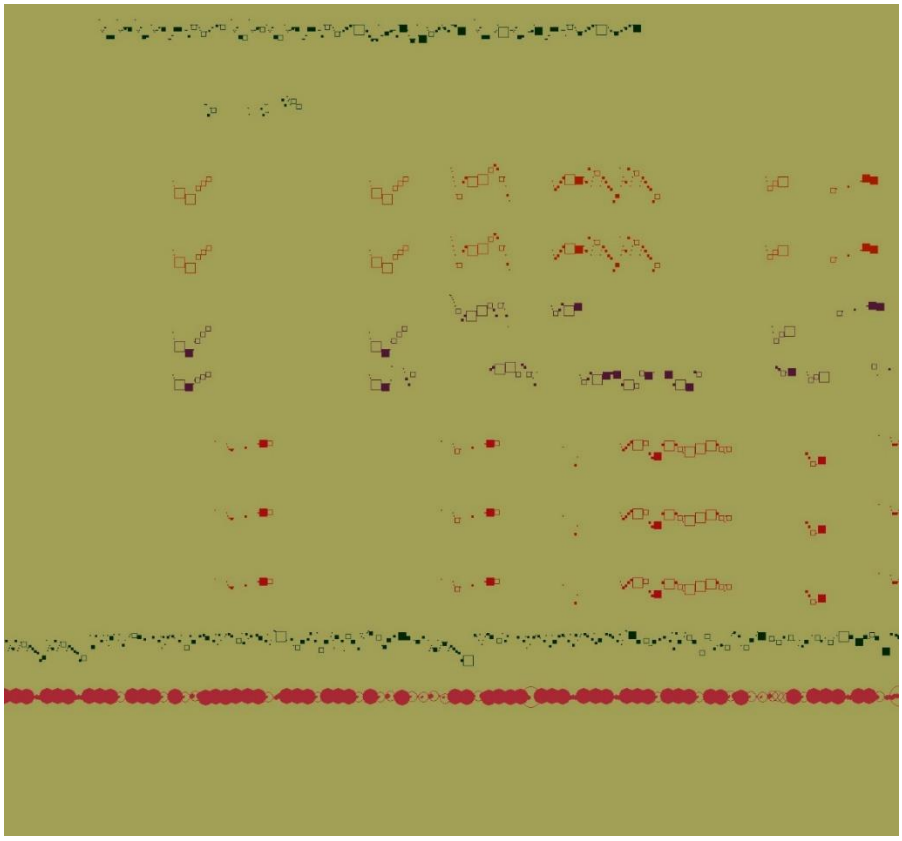
Each segment represents a particular instrument played, however this does not state that the instrument is different in particular, it indicates that the instrument is played by a different user.

9.3.1 Effective outputs

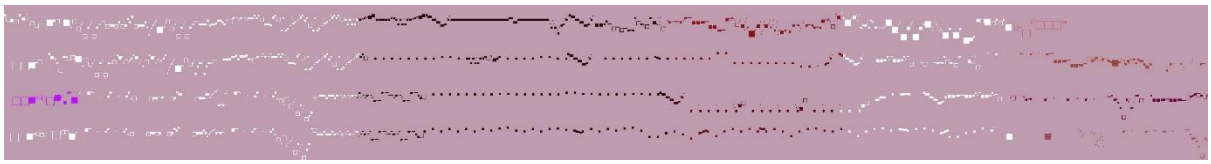
Legend of kora (The_legend_of_Korra.mxl)



Wonderful Tonight (Wonderfull_Tonight.mxl)



Star wars (Star_Wars_Horn_Quartet.mxl)



The figures above overall illustrate the pieces of scores named, very well. The background is created by mapping the keys to their respective colours, according to the theory presented by Harry Farjeon [11]. If the colour isn't found directly to a certain key then a new colour is generated according to the similarity of keys, and combining it with the results found of the survey.

9.3.2 Evaluation

As it is present above, the illustrations expresses traits of graphic variables stated by Jacques Bertin [12]. Position, Size, shapes, value, colour. The graphic variables do well to represent the overall music of each score, where the notes played, along with their type and frequency is well represented. Each instrument is represented in their own segment, with signs of tempo where present is visible, each clef is represented with their respective overall it does well in accurately representing a piece of music score.

Shape

- Shape is recognised by the type of key used in each particular instrument, the first piece uses all 3 types of shapes (square, circle, polygon) and the second piece uses 2 shapes (circle, square) each indicating that the particular instrument is played in a different key.
- Shape is also expressed by the latter 2 illustrations however not to the extent of the first. This indicates that all keys are played in the same key.

Size

- The size of each shape is determined by the length of each note, all three illustrations express the use of size very well. The sizes of each shape can be seen as a small dot (representing very short notes), large shapes (representing long notes). The variance in the size of the shapes illustrate clearly the varying length of notes in the respective score.

Position

- The position is determined by the overall offset in which a note is played in, or how far they are placed respective of the start of the piece, the positions represented by the square and diamond are accurate according to the piece, this is seen as notes do not overlap each other.
- However, circles are seen overlapping each other, this can be confusing as it could represent a chord playing or ties etc.

Colour

- Colour or the change in colour inside an instrument is represented by the change in tempo of a piece of music, as seen clearly in the first and third illustrations. The change in tempo can clearly be seen by the change in colours in the respective illustrations.
- As seen in the first piece, there are a variety of colours throughout the representation of each instrument, stating that beat changes frequently throughout the particular score, a sudden change in colour indicates that the change in beat was quite significant, as opposed to a colour which shows a light change in hue and intensity, which shows a slow change in beat, which is illustrated in the third illustration.
- The third illustration only shows a change in colour depending on the instrument, indicating that the colour is unique to the instrument. As the instruments do not show a change in colour it indicates that the piece is played in the same tempo throughout.

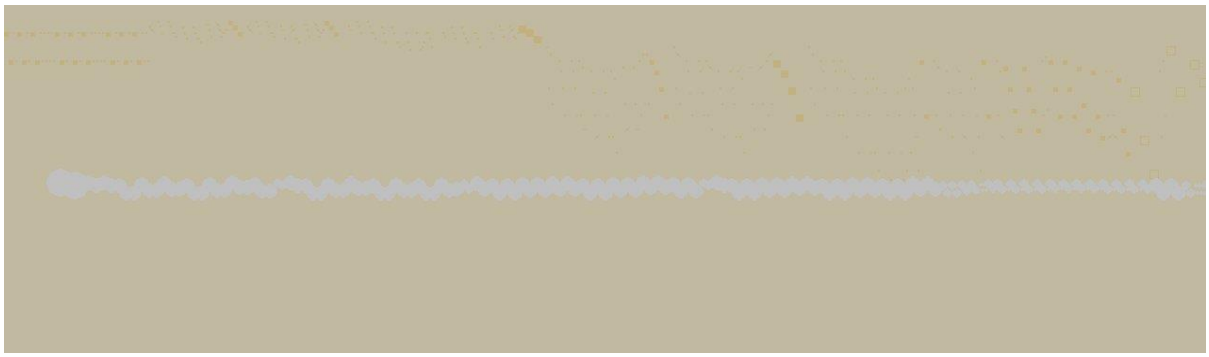
Texture

- The texture factor however is the weakest point of the particular piece. The illustrations only show that it is filled with a colour or not, however it is important to note that a note can only have so many characteristics related to them, and depending on the characteristics it is difficult to determine a variety of texture within each shape.

9.3.3 Non effective outputs

The outputs shown below are represented as non-effective outputs as they do not paint a clear picture of a particular score through the use of graphic variables. The illustrations shown below, do not show effective outputs of the graphic variables stated by Bertin [12]. The shapes are not clearly visible, the colour, size and shape are not clearly visible. Although it is somewhat clear that an instrument is being played and for how long, the combination of colours make it hard to determine the notes the instruments which are played.

Pirates of the Caribbean (Pirates_of_the_Caribbean_Hes_a_pirate_594.mxl)



What do you mean (what_do_you_mean.mxl)

Pokémon surf theme tune (Surf_Theme-_Pokemon_Gold_Silver_and_Crystal.mxl)

9.3.4 Evaluation

Shape

- As seen in the illustrations above, what type of shape used to emphasise what clef the piece is written is not clearly visible. Although in the second visualisation it is clear that squares are used, the size of the shapes do not emphasise the aspect of the music enough. With the third visualisation showing no shapes at all rather a black canvas painted light blue.
- In the first illustration, due to the colour used in the background and the colour the shape is filled in, the shapes are not clearly visible it is only when looked at from a close range that the shapes can be seen. It is also difficult to determine what shapes are being used, for example the grey coloured shape is meant to represent a bass clef, however it is difficult to determine whether it is a circle or a diamond.

Size

- As seen in the first two visualisations the size of the shapes show little expression. This could show that the length of the notes are played are of similar duration throughout, however it is difficult to recognise if that is the case with the visualisations.

Position

- The position variable is emphasised well in the first two visualisations, however due to the colour being relatively similar to the background, it is somewhat difficult to determine the position of the position of the graphic variable.
- The position x represents the pitch and y represents the duration of the music, here the duration is emphasised well, however the difference in pitch is not clear.

Colour

- The colour aspect does is not represented well by the illustrations, either because the colours are too similar to the background or not present at all. The colours are meant to represent the key of the instrument or the score. Which in turn represents the overall mood generated by the score. However this could mean that the keys are the same or similar for each key and instrument and that tempo stays consistent throughout, there may have been a better method to represent the uniqueness of the particular instrument.

Texture

- As with the effective visualisations, the texture here is not emphasised very well. Although the effective visualisations it was clear to see if the shape was filled or not, representing either whole, half and other types of noes here it is unclear what notes are being played in the particular instrument.

Overall the visualisations generated effective visualisations, clearly representing key aspects of music notations used throughout the score, and with the use of graphic variables, the changes which occur throughout the score is also evident, the colours used represent a certain mood, and the change in colour for a particular instrument clearly represents the change in tempo. However the system also produced ineffective illustrations for some scores, failing to recognise the distinction between key, instruments played and tempo.

However it is important to consider that the mxl files used were generated by users who may not be proficient in writing music in '*MusicXML*' and that they may not have used all the correct or effective notations for the piece of score.

9.4.0 Evaluating through the response of people

This evaluation asked 2 people to choose from 3 different outputs while a music was being played in the background.
The choices were (9.3.1):

1. Legend of kora (The_legend_of_Korra.mxl)
2. Wonderful Tonight (Wonderfull_Tonight.mxl)
3. Star wars (Star_Wars_Horn_Quartet.mxl)

The correct music being played in the background was song No.3 (star wars).

The names of the outputs and the song played would remain anonymous to the 2 people.
The questions asked were:

- Choose which output you feel is the correct output for this song.
- What were your reasons for choosing the outputs?
- Did the shapes and their variables play a part in your choice?

The response given by both were fairly similar, neither mapped the correct output to the song correctly, however their answers to the last 2 questions were intriguing.

Both people stated their reasons behind their choices were because the song overall is quite intense, and that the music sounds like people are marching, and the colour red gives an intense feeling.

When asked the third questions they stated that, No the variables did not play a huge part in their choices. Stating that they mainly focused in the overall background as that is what sticks out the most, also they did not understand what the graphic variables are intended to do, therefore the background is the only part which they understood.

According to this evaluation it seems that minor details of a piece of illustration do not play a huge part behind how they perceive the overall picture, and that the background colour, which is the most crucial part of the illustration is justified as the main reason behind the mood of the illustration.

10.0.0 Future Work

The system created shows only illustrations in the form of a static image. The system currently uses documentation found on the official 'Music21' website. As 'Music21' and 'MusicXML' are relatively new concepts and 'Music21' is a tool mainly for creating scores and analysing the score, the combination of 'Music21' and 'MusicXML' has little to no 3rd party documentation to it. The popularity and use of MIDI is common upon creating visualisations however 'MusicXML' is still in its infancy and shows great potential over MIDI. As it contains much higher levels and details of music notation, it is possible to analyse the music in greater detail in order to create an effective illustration of a score.

The ideas listed below are possible methods of future works that may be used to improve upon the current system.

Attempt to accurately find some type of justification with the analysis of greater amount of keys and colours.

- The survey conducted only analysed the colours and feelings of 5 keys, a total of 18 colours and 52 feelings, due to design and time restrictions.
- There are a vast array of keys which are not restricted to the ones stated by Albert Lavigance [10] and Harry Farjeon [11].
- There are also many more feelings and colours not restricted to the ones stated by Albert and Harry.
- By analysing the input of greater number of people, it may be possible to justify that colours and feelings have some connection with keys.
- The survey conducted for this system showed that there is little connection between keys and colours however the analysis of feelings and colours showed similarities between the hypotheses by Hugo Reinmann [10].

Attempt to map feelings with colours:

- Due to time restrictions the current system analyses keys and colours and keys and feelings separately.
- Even though there may not be a connection directly between colour and keys, there may be connection between colours and feelings.
- By mapping colours with feelings, it may be possible to map keys with colours and feelings with justification.
- Possibly through the implementation of machine learning and AI.

Analyse segments and repeating patterns similar to (Martin WattenBurg: 4.3.2)

- The current system maps the use of all chords depending on pitches and offsets.
- Martin uses midi files to analyse repeating patterns and places circles through each repeating patterns.
- With the level of detail 'Music21' is able to extract, far greater than that of MIDI it may be possible to implement other shapes to illustrate segments and generate texture (graphic variables)

Create Moving Visualisations (live or animated):

- Although live visualisations of music are common, it does not fully represent the music itself, rather they are used just for aesthetic purposes, mostly through the use of MIDI and MP3 files. MusicXML is a more powerful tool to extract detailed information about a score.
- Art as it has throughout history has been represented statically in the form of paintings and illustrations, however with the emergence of video and 'GIF' images, art can be expressed in more ways than static images.
- With the analysis of scores it may be better to represent music as an animated form. As music it is not stationary and in order to fully appreciate music one must listen to it over time. It may be the same in order to fully appreciate the art illustrated music.
- Being able to differentiate the difference between keys, notes, time signatures etc, it is possible to generate animated images using 'MusicXML'.

Possibly implement 3d graphics.

- Two dimensions can only illustrate and create a limited amount of feeling.
- Being able to visualise in 3 dimensions may bring a whole new way to illustrate music. Being able to see the depth of an object, the hue, saturation or lightness changing according to the depth, width, and height is a really exciting prospect for the system.

11.0.0 Conclusion

The aim of this project was to create visualisation of music through the analysis of an electronic score, in the format of 'MusicXml' (MXL) file. The system uses a python toolkit 'music21' developed by MIT to extract and analyse a variety of musical information. Through the hypothesis presented by Albert Lavigance [10] and Harry Farjeon [11] the system would analyse keys and attempt to map keys to feelings and colour. Other information such as notes, offsets, measures, instruments, clefs etc. would determine the graphic (visual) variables of the visualisation.

The system successfully managed to create visualisations for most of the open source MXL files downloaded from musescore.com. The outputs overall showed good implementation of the results of the survey and correctly managed to implement the change in music e.g. note duration, metronome marks, and assigned correct variables in order to create a unique and accurate visualisation of musical notations of a particular score.

The survey (8.1.0) helped determine how to assign colour to keys and how to change the RGB values of the colours, as the beat of the music changed. The overall intention was to visualise the mood of the music. Although the visualisation makes effective use of colours, it still is difficult to determine whether it creates the intended mood of the music. This is primarily due to the fact that music, colours, and feelings are subjective matters, therefore it is difficult to determine any justification behind the use of colours to determine such feelings.
However in the future, if a survey is taken for more than 50 users, i.e. 1,000 or 10,000 it may be possible to accurately map feelings with colours and keys.

The visualisation system successfully (8.2.0) managed to extract the relevant data for most mxl files i.e. keys, instruments, clefs, metronome marks etc, required for analysis in order to apply graphic variables relating to the relative data. The visualisation also successfully managed to implement the results of the survey. For example what happens to RGB values if the tempo changes by 10BPM, or how the RGB values change if the key changes from Normal to Sharp side. However it is difficult to evaluate whether the intended mood was successfully visualised by the system, as each person may feel different about the choice of colours, it is difficult to determine what mood exactly is created by the user.

The outputs do well to represent the graphic variables stated by Bertin [12] and each choice is justified to their respective musical notations. However according the evaluation in 9.4.0, it seems that people do not pay much attention to the finer details, rather are more interested in the overall output, therefore in the future it would be best to visualise only the main aspects of a song rather than all of it, in order to represent the mood of the music in the output.

In conclusion, the project was a success, the system successfully manages to recognise and extract musical notations and graphic variables are assigned according to the change in the score. The system also has justification behind the reasons of assignments through the results of the survey. Although the system generates outputs successfully, it is still difficult to determine whether it was an effective output, in terms of generating the overall picture of the painting. This could however be further emphasised in the future works that are carried out.

12.0.0 Reflection

This project has helped me to apply the knowledge that I have learnt over my time of university, it allowed me to apply all the skills I learnt during group project exercises, apply the methods learnt in lectures and allowed me to think originally in order to create content which is unique and which fulfils a sense of achievement.

The skills I have learnt throughout will of course help me in my professional career, where I am able to communicate my ideas effectively with the use of reports, diagrams etc. It has also showed me the importance of research and inspiration. In order to build a system, unless the idea is completely new, then researching through the use of books, journals, blogs, documentation helps find suitable answers in order to find the solution to the problem. The research also helps find inspiration, without inspiration, the will to do something seems to diminish, and the work seems boring, however by seeing how others have tackled similar problems and being able to see how their ideas worked, helps in the process of learning and solving a problem, these are the skills which I feel will carry on and help me in a lifelong process of discovery and problem solving.

This project also helped me appreciate what the subject 'computer science' is capable of. The assumption that computer science is a subject where you build apps, websites, or just spend your time programming something which is of only has commercial value is an assumption which I will no longer hold. By being able to combine my interests of art, music and computer science, I am able to see the subject in a different perspective. It shows that computer science is a broad, powerful tool which can fulfil the desires of any subject, if you follow the right steps and use the right tools to execute it. By creating a visualisation which showed a variety of colours, shapes, graphic variables, I am now able to appreciate that computer science may be, with the appliance of AI and Machine learning, could become art itself, where something can be created without having to justify the reasons behind it, without having to programme something to do everything. Maybe in the future visualisations can automatically be created by computers which are considered 'art'.

The skill of time keeping, or managing time effectively is a lesson which I will hold in both my professional career and personal life, the importance applying given time to complete your goals is already an established one. However what I learnt from this project is the importance of giving time for something to grow, giving time to reflect upon your ideas. This concept is effectively double loop learning, where you constantly reflect and possibly question your goals in order to make the right decisions. From this I will not be afraid to question the assumptions I have made before, be able to think what has made my project successful so far and how if I changed my approach to carrying out tasks, then it may help me build a biter system or make better choices is a lesson which I will take on for the coming future.

To conclude, this project has not only helped me realise how to create an effective system, but has also helped me improve as a person itself. I am now able to appreciate the subject in a far greater way than I was able to before, an appreciation which I hope will allow me to create effective software and build further into my skills in the lifelong journey of discovery and self-improvement.

I would like to add that the outputs of the system not be called 'art' in itself, but a process in which it may help create art by the use of computer science in the future. The system right now, does not analyse music in detail to find, patterns, rhythm etc. also it does not apply the emotions that humans feel, of which if not applied to an illustration, cannot be called 'art'.

References

[1]Eduard Hanslick, The Beautiful in Music, tr. Gustav Cohen (New York: The Liberal Arts Press, 1957), vii-ix.

[2]Tufte R.E (2011). Envisioning information. Connecticut: Graphics Press LLC. 81-100.

[3]Tufte R.E (2011). Envisioning information. Connecticut: Graphics Press LLC. 82.

[4]Driver, A (1936). Music and Movement . London: Thomspon Press. 34.

[5]Tufte R.E (2011). Envisioning information. Connecticut: Graphics Press LLC. 83.

[6]Hockney, David. Two Boys Aged 23 Or 24. Liverpool: Tate Britain's Prints and Drawings Rooms, 1966. Print.

[7]Kandinsky, Wassily. Composition VIII. New York: The Solomon R. Guggenheim Museum, 1923. Print.

[8]Music Notes. (2014). How to Read Sheet Music: Channel Your Inner Musician with These Simple Steps!. Available: http://www.musicnotes.com/blog/2014/04/11/how-to-read-sheet-music/. Last accessed 6th May 2016.

[9]Ishiguro, M. (2010). Section 1: Music Scholars. In: The affective properties of keys in instrumental music from the late nineteenth and early twentieth centuries. Massachusetts: University of Masschusetts. 60-62.

[10]Ishiguro, M. (2010). Section 1: Music Scholars. In: The affective properties of keys in instrumental music from the late nineteenth and early twentieth centuries. Massachusetts: University of Masschusetts. 77-78.

[11]Ishiguro, M. (2010). Section 1: Music Scholars. In: The affective properties of keys in instrumental music from the late nineteenth and early twentieth centuries. Massachusetts: University of Masschusetts. 86-89.

[12]BERTIN, J. (1967|1983) Semiology of graphics: Diagrams, networks, maps, Madison, WI, University of Wisconsin Press.

[13]Cuthbert, M. (2006). music21: a toolkit for computer-aided technology. Available: http://web.mit.edu/music21/doc/about/what.html. Last accessed 6th May 2016.

[14]Jones, C. (2016). The Shape or Contour of a Melody. Available: https://www.boundless.com/users/232513/textbooks/5656/definitions-2/melody-12/the-shape-or-contour-of-a-melody-65-13540/. Last accessed 6th May 2016.

[15]MusicXML. (2014). FAQ. Available: http://www.musicxml.com/tutorial/faq/. Last accessed 6th May 2016.
[16]Guy, C. (2012). MIDI vs. MusicXML. Available: https://techinmusiced.wordpress.com/2012/05/26/midi-vs-musicxml/. Last accessed 6th May 2016.

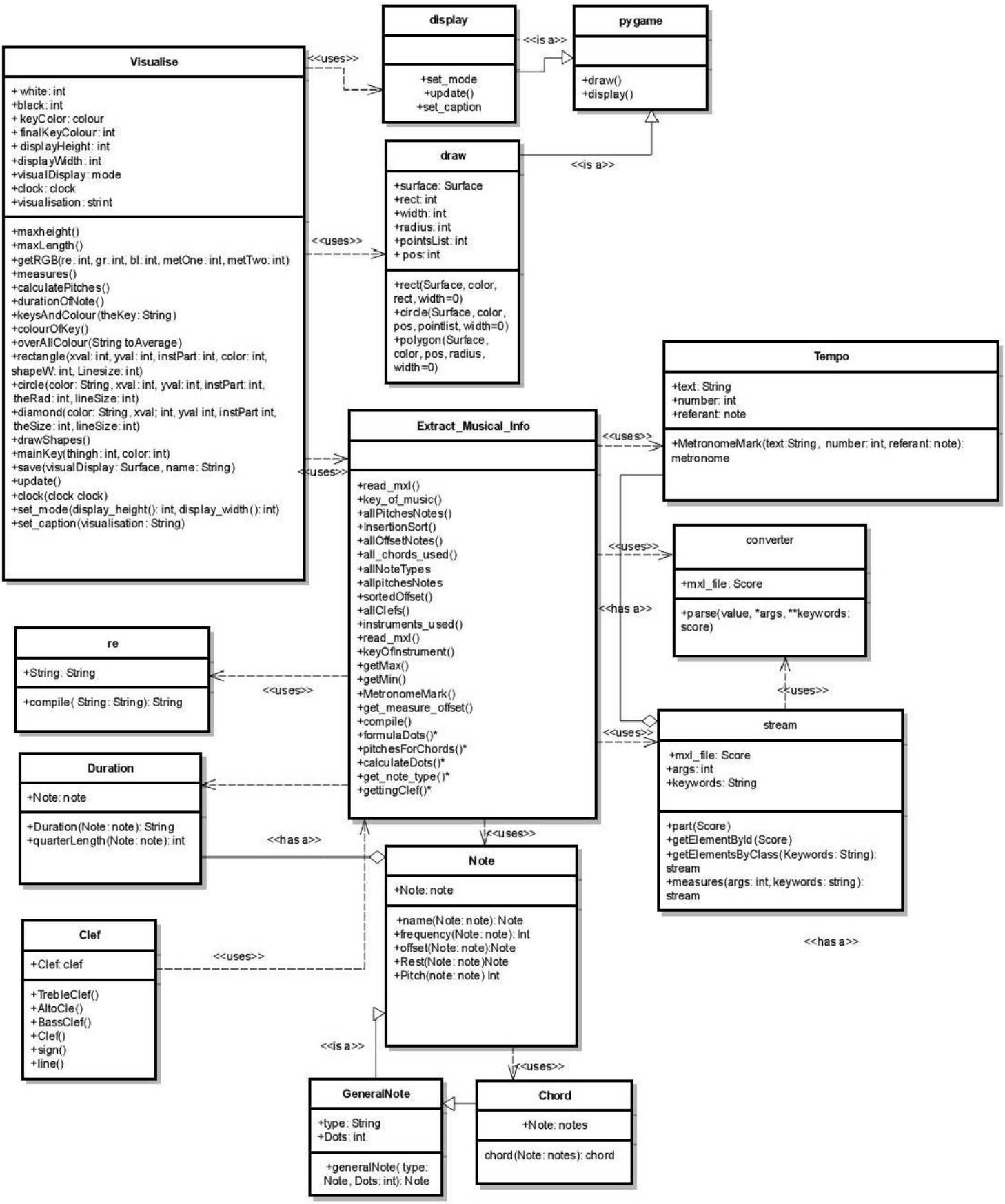
[17]MuseScore. (2016). Handbook. Available: https://musescore.org/en/handbook. Last accessed 6th May 2016.

[18]Hein, E. (2011). Visualizing music. Available: http://www.ethanhein.com/wp/2011/visualizing-music/. Last accessed 6th May 2016.

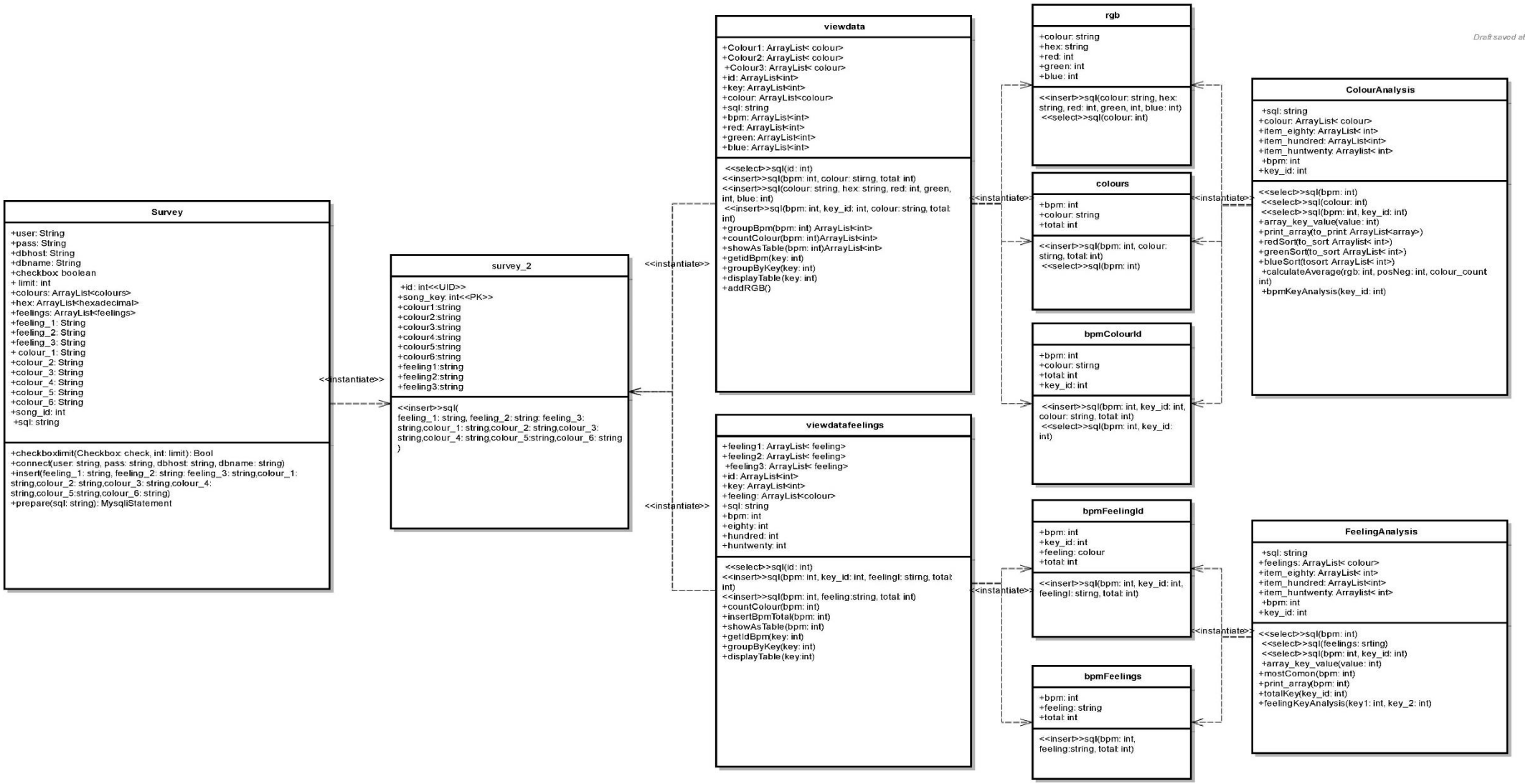
[19]Martin Wattenburg. (2001). The Shape of Song. Available: http://www.bewitched.com/song.html. Last accessed 6th May 2016.

Appendix

1.0.0



2.0.0


Survey
+user: String
+pass: String
+dbhost: String
+dbname: String
+checkbox: boolean
+ limit: int
+colours: ArrayList<colours>
+hex: ArrayList<hexadecimal>
+feelings: ArrayList<feelings>
+feeling_1: String
+feeling_2: String
+feeling_3: String
+ colour_1: String
+colour_2: String
+colour_3: String
+colour_4: String
+colour_5: String
+colour_6: String
+song_id: int
+sql: string
+checkboxlimit(Checkbox: check, int: limit): Bool
+connect(user: string, pass: string, dbhost: string, dbname: string)
+insert(feeling_1: string, feeling_2: string: feeling_3: string,colour_1: string,colour_2: string,colour_3: string,colour_4: string,colour_5:string,colour_6: string)
+prepare(sql: string): MysqliStatement
<<instantiate>>
survey_2
+id: int<<UID>>
+song_key: int<<PK>>
+colour1:string
+colour2:string
+colour3:string
+colour4:string
+colour5:string
+colour6:string
+feeling1:string
+feeling2:string
+feeling3:string
<<insert>>sql(
feeling_1: string, feeling_2: string: feeling_3: string,colour_1: string,colour_2: string,colour_3: string,colour_4: string,colour_5:string,colour_6: string)
<<instantiate>>
<<instantiate>>
viewdata
+Colour1: ArrayList< colour>
+Colour2: ArrayList< colour>
+Colour3: ArrayList< colour>
+id: ArrayList<int>
+key: ArrayList<int>
+colour: ArrayList<colour>
+sql: string
+bpm: ArrayList<int>
+red: ArrayList<int>
+green: ArrayList<int>
+blue: ArrayList<int>
<<select>>sql(id: int)
<<insert>>sql(bpm: int, colour: stirng, total: int)
<<insert>>sql(colour: string, hex: string, red: int, green, int, blue: int)
<<insert>>sql(bpm: int, key_id: int, colour: string, total: int)
+groupBpm(bpm: int) ArrayList<int>
+countColour(bpm: int)ArrayList<int>
+showAsTable(bpm: int)ArrayList<int>
+getidBpm(key: int)
+groupByKey(key: int)
+displayTable(key: int)
+addRGB()
viewdatafeelings
+feeling1: ArrayList< feeling>
+feeling2: ArrayList< feeling>
+feeling3: ArrayList< feeling>
+id: ArrayList<int>
+key: ArrayList<int>
+feeling: ArrayList<colour>
+sql: string
+bpm: int
+eighty: int
+hundred: int
+huntwenty: int
<<select>>sql(id: int)
<<insert>>sql(bpm: int, key_id: int, feelingI: stirng, total: int)
<<insert>>sql(bpm: int, feeling:string, total: int)
+countColour(bpm: int)
+insertBpmTotal(bpm: int)
+showAsTable(bpm: int)
+getIdBpm(key: int)
+groupByKey(key: int)
+displayTable(key:int)
<<instantiate>>
<<instantiate>>
rgb
+colour: string
+hex: string
+red: int
+green: int
+blue: int
<<insert>>sql(colour: string, hex: string, red: int, green, int, blue: int)
<<select>>sql(colour: int)
colours
+bpm: int
+colour: string
+total: int
<<insert>>sql(bpm: int, colour: stirng, total: int)
<<select>>sql(bpm: int)
bpmColourId
+bpm: int
+colour: stirng
+total: int
+key_id: int
<<insert>>sql(bpm: int, key_id: int, colour: string, total: int)
<<select>>sql(bpm: int, key_id: int)
bpmFeelingId
+bpm: int
+key_id: int
+feeling: colour
+total: int
<<insert>>sql(bpm: int, key_id: int, feelingI: stirng, total: int)
bpmFeelings
+bpm: int
+feeling: string
+total: int
<<insert>>sql(bpm: int, feeling:string, total: int)
<<instantiate>>
<<instantiate>>
<<instantiate>>
ColourAnalysis
+sql: string
+colour: ArrayList< colour>
+item_eighty: ArrayList< int>
+item_hundred: ArrayList<int>
+item_huntwenty: Arraylist< int>
+bpm: int
+key_id: int
<<select>>sql(bpm: int)
<<select>>sql(colour: int)
<<select>>sql(bpm: int, key_id: int)
+array_key_value(value: int)
+print_array(to_print ArrayList<array>)
+redSort(to_sort Arraylist< int>)
+greenSort(to_sort ArrayList< int>)
+blueSort(tosort ArrayList< int>)
+calculateAverage(rgb: int, posNeg: int, colour_count int)
+bpmKeyAnalysis(key_id: int)
FeelingAnalysis
+sql: string
+feelings: ArrayList< colour>
+item_eighty: ArrayList< int>
+item_hundred: ArrayList<int>
+item_huntwenty: Arraylist< int>
+bpm: int
+key_id: int
<<select>>sql(bpm: int)
<<select>>sql(feelings: srting)
<<select>>sql(bpm: int, key_id: int)
+array_key_value(value: int)
+mostComon(bpm: int)
+print_array(bpm: int)
+totalKey(key_id: int)
+feelingKeyAnalysis(key1: int, key_2: int)
Draft saved at