

A Narrative Music Generation System

Submitted April 2018, in partial fulfilment of
the conditions for the award of the degree **BSc (Hons) Computer Science.**

Praneil Kamat

4263680

School of Computer Science

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated
in the text:

Signature



Date: 24/04/2018

1 Abstract

Music generation systems provide a variety of uses, including generating music that can accompany narratives in visual media such as videos and video games. However the music generation systems available to visual media creators are hosts to issues that may make them undesirable to use in such situations. In this project, the Narrative Music Generation System (NMGS) has been created. The system allows users to shape the music it generates by using the emotions that are conveyed by a narrative, as input, attempting to solve some of the aforementioned issues and provide visual media creators with a system that is more desirable than those that are currently available.

2 Acknowledgement

I would like to express my gratitude to my supervisor Dr. Chris Greenhalgh for his guidance throughout the course of the project. He helped me to stay focused on and achieve my goal, despite my common tendency to get distracted and go on a tangent that leaves me far from it.

Contents

1	Abstract	2
2	Acknowledgement	2
3	Introduction	5
3.1	Project Aims	5
3.2	Objectives	5
4	Motivation	6
5	Related Work	7
5.1	Systems designed for research	7
5.2	Systems designed/available for use	8
6	Description of Work	9
6.1	Functional Specification	10
7	Methodology	11
7.1	Software Development Methodologies	11
7.2	Programming languages, libraries, frameworks and plugins	12
7.3	Version Control	13
8	Design	13
8.1	Music Generator	13
8.2	Generating Emotional Music	15
8.3	Transitioning Between Emotions	16
8.4	GUI	17
8.5	System Architecture Plan	18
9	Implementation	18
9.1	JMusic	19
9.2	Back-end	19
9.3	Front-end	22
9.4	Testing	25
10	Evaluation	26
10.1	Method	26
10.2	Results	27
10.3	Discussion	32
11	Summary and Reflections	37
11.1	Project Management	37
11.2	Contributions and Reflections	38
A	Glossary of Musical Terms	41

B IntelliJ IDEA Project Structure	43
C Unit Test Results	44
D Input Parameters Used to Generate Music for Clips A-E	44
E Stage 1 Brief and Questionnaire	46
F Stage 2 Brief and Questionnaire	47
G Responses for Questionnaires 1 & 2	50
H Development Plan	52
I Sprint Retrospective Notes	54
J Old Prototype	55
K Hansoft Project Backlog	56
L Hansoft Sprint Burn-down charts	56
M Information Sheet	58
N Consent Form	59

Some chapters of this report will include musical terms. See appendix A for relevant musical terms and definitions.

3 Introduction

In this chapter, the main aims and objectives of the project will be outlined, as well as how these aims and objectives will be met.

3.1 Project Aims

The aim of this project is to develop a music generation system that allows the user to shape the outputted music by giving the music narrative. For this project, narrative music is defined as music that helps convey a narrative by expressing the emotions that characterise narrative situations and events [1]. Using this definition, the aim of the project can be simplified to developing a music generation system that allows the user to shape the outputted music by using emotion as input. The system could also be defined as an emotional music synthesis (EMS) system, which is a system that generates music that has a recognisable emotion to set a mood [2]. The system designed and developed for this project is referred to as the Narrative Music Generation System (NMGS).

The main target audience for NMGS are creators of visual media, for example, creators of video games or film. It should allow them to generate music to accompany their work. Due to this, it is considered essential that prior musical knowledge is not necessary to use the system.

To give an example of how NMGS may be used, a game developer may want music to accompany a scripted sequence in their game. The sequence could begin with the intention of being characterised as happy, and an event could occur which intends to change this to sad. The developer could use NMGS to generate music that attempts to give a happy feeling before the event and a sad feeling following the event. The music should transition compositionally, in a way that is perceived as smoothly, from happy to sad at the time of the event.

3.2 Objectives

The key objectives of this project are to:

1. Investigate, based on previous work, generative music algorithms and approaches to algorithmically incorporating narratives into music.
2. Design and develop an algorithmic narrative music generation system that:
 - Allows the outputted music to express target emotions, which are inputted by the user.
 - Can transition compositionally between different target emotions.
 - Is capable of conveying a narrative when matched to a form of visual media.
 - Provides a simple graphical user interface (GUI) that makes the system easy to learn.
 - Outputs algorithmically generated music in Musical Instrument Digital Interface (MIDI) format.
3. Review and evaluate the success of the system against the system's sub-objectives outlined above and suggest further developments.

To ensure the aims and objectives are met:

- A functional specification, outlining and prioritising the minimum requirements was created, and a project backlog was created from this specification.
- A software development methodology that best fits the project to provide smooth development was chosen. It was essential that the methodology provides flexibility to account for coursework deadlines and workloads that could not be fully known ahead of time.

4 Motivation

There are several generative music systems that exist and are readily available to the public. The potential uses of these systems vary widely, from musical inspiration for musicians to accompanying and enhancing the emotional experience of a linear narrative, which could exist in multiple forms. For example, it has been argued that music is an important part of the emotional experience in a classic narrative film [3]. It has also been argued that the functions of music in film are equally applicable to video games [4]. Following this, it would be possible for the music generated from these systems to be used by video game developers to accompany a non-linear narrative, through the use of techniques such as looping. See Collins [5] for an overview of music techniques used by developers in video games. Generative music systems could be of particular use to independent video game developers with small budgets, with an interest in this existing [6]. Generative music systems could also be seen as useful in virtual reality (VR) games, as it has been shown that adaptive music increases immersion and time spent in VR [7].

It is worth noting that if continuing development on NMGS after this project, I would eventually like to interface it with the sound API of a video game engine, allowing for dynamic, real time music generation during gameplay. Due to this, the systems ability to generate music quickly enough to be eligible for use in a real time environment is considered important.

NMGS attempts to solve three problems in the area of generative music systems available to the public.

The first of these problems is ease of use and learnability. Many existing systems are arguably not easy to use for those who are not musicians, have overly complex user interfaces, or have a significant learning curve. By using emotion as the input provided to the user, and by providing a simple GUI, the system can be used without musical knowledge and be immediately useful to the user. Emotion as an input will also be specifically intuitive for visual media creators, as they will often be aware of the emotions that they want their work to evoke in their audience.

The second problem is that many existing systems do not allow the user to create compositions with multiple different sections. Instead, these systems either allow the user to generate a section of music or a composition with one set of input parameters. If the user wants to transition between sections or different sets of input parameters, they are limited in how they can achieve this, as the music has already been generated. To address this, methods are investigated to allow the system to compositionally transition between different sections.

The third problem is that existing systems often require a great deal of experimentation with the input parameters to reach a desired result, for example, a desired result being music that matches or 'fits' alongside a video. With enough experimentation, this can be reached, as existing systems can be powerful. However, a narrative based solution could lead to much less experimentation needed, leading to a system that could be more desirable to visual media creators.

5 Related Work

The existing work in the area of generative music systems can be separated into two categories:

- Systems built to expand and/or improve on existing research on computer generated music, and
- Systems built/available for use, either public or commercial

NMGS is intended to fall into the latter category. Therefore, instead of expanding on existing research, the system uses existing research to solve the problems outlined above to fill a gap in the existing music systems available for use.

In this chapter, notable and relevant existing systems are outlined below, with reference to their achievements and how this project aims to use these to solve the weaknesses of systems available for use.

5.1 Systems designed for research

Wallis et al [2] developed an EMS system that uses a rule based algorithm based on a circumplex model (a 2D model to represent emotion - see description in chapter 8.2.1) for modelling emotions, with rules for the musical features, to generate music. They suggest further uses of EMS systems such as in therapeutic systems for emotional imbalances and autism. Their results showed that listeners perceived the desired emotions. An acknowledgement that they make, which should also be made for this project, is that limited variation in the music in terms of genre leaves the potential for emotional bias in the music. However, they suggest that due to their results, this was unlikely to be the case. As the aim of the system was to evaluate how well a target emotion matched user evaluations of emotion perceived, no efforts were made to allow the system to transition between different emotions.

The main takeaway for the design of this project is the use of a circumplex model and it's effectiveness, when paired with the rules for musical features, in causing listeners to perceive a target emotion.

Prechtel [1] developed a music generation system that was interfaced with a video game to dynamically generate music in real time. A Markov chain model (a stochastic, probability based model - see description in chapter 8.1.1) was used to generate the music. A circumplex model was used, however this model was used to represent tension instead of specific emotions, with tension in the generated music being mapped to the proximity to an enemy in the game. By using filters based on musical features to dynamically build the transition/probability matrix in the Markov model, the system was able to smoothly transition between low tension and high tension. Conclusions were made that the system produced music with emotional expression and that supporting a game's narrative through varying emotional expression of the music positively impacted the playing experience. An acknowledgement is made that using only varying tension to support an emotional narrative is simplistic. This leaves the system unable to support more complex narratives.

The main takeaways here are the use of a Markov model, the ability to represent tension within the circumplex model, and the ability to manipulate the Markov model to smoothly transition between different states.

The systems outlined above have been included in this report as they are narrative/emotion based and therefore the most relevant, and because they have provided the most knowledge for

this project, with little new information being learnt from other systems. However, many other systems relevant to this project exist. See Herremans et al [8] for a summary of existing generative music systems, categorised both by the generative algorithms/techniques used and their functional aspects (e.g. Narrative).

5.2 Systems designed/available for use

The details of the design and implementation choices for these systems are not always known, therefore their functionality is focussed on. All of these systems provide functionality to play generated music within the application and allow the user to export their generated music as a MIDI or MP3 file, while most give the user access to the seeds for the random number generators used in the system.

Abundant Music [9] is a rule-based music generation system. The system is web-based and prior musical knowledge is not essential to generate music, although due to musical terms used and musical parameters provided, it would help the user. A graphical music visualiser is used to help the user visualise their song. This resembles a graph, with time in the x-axis and pitch in the y-axis. Instruments playing produce a drawn line and each instrument is colour coded. Several parameters are provided, such as the multipliers, probabilities and ranges of musical attributes, including the instruments used, to shape the music. While this provides a great deal of control, the user must experiment with a large number of parameters to reach their desired goal, whereas control of a narrative could allow the user to generate music closer to their desired goal, with less experimentation. In addition to this, an arguably overly-complex GUI could deter new users.

Wolfram Tones [10] is a cellular automata based music generation system. The system is also web-based. Musical knowledge is again not essential to generate music, but, for the same reasons as before, would be helpful. An image is used to help the user visualise the music that will be generated, in a way that is similar to Abundant Music. The user is given control of several musical parameters, as well as each instrument and their roles (e.g. lead, chords, bass), in a simple GUI. The system also allows the user to pick from a selection of musical styles such as 'Classical' and 'Blues' (while interesting, due to time limitations, this goes beyond the scope of this project). However, no parameters for musical structure are given, and the maximum length of the output is 30 seconds, meaning only sections of music can be created. This can make it difficult to accompany a narrative. It also can require a lot of experimentation to get a desired result.

cgMusic [11] is a music generation system available as a Windows application. As previously, musical knowledge is not essential for music generation, but, for the same reasons as before, would be helpful. The user is able to design the structure for the song, and they are able to adjust multiple musical parameters in each 'part' of the song in the structure. Once again, user experimentation is key to acquiring the desired results, such as following a narrative, though a simple GUI and limited number of parameters make the system more accessible than those previously mentioned. However, a narrative based solution could allow the user to get closer to their desired result in less time.

Jukedek [12] is a neural network based music generation system. The system is web-based, and requires no previous musical knowledge to use. The system, which requires a user account to use, allows users to select a genre and a mood, as well as select the instruments used. Each genre has a limited set of moods to choose from. There is also functionality for a video to be loaded into the webpage, syncing playback of the generated music to the video, which could be helpful for visual media creators. The system does not, however, provide any functionality for transitioning between genres or between moods, which could make it difficult to generate music that follows a

narrative that is characterised by multiple moods.

The systems outlined above have been included in this report as they were the systems with the largest number of positive aspects that could be applied to this project. These aspects include:

- A simple GUI (Wolfram Tones, cgMusic, Jukedek)
- Providing functionality for the user to structure the music (Abundant Music and cgMusic)
- Allowing the user to choose instruments and their roles (Wolfram Tones, Jukedek and to a certain extent Abundant Music)
- Using graphical visualisation to represent the music that will be generated (Abundant Music and Wolfram Tones)
- Providing functionality for a video to be loaded into the application to allow for generated music to played alongside it (Jukedek)

A functional summary of these systems, in regards to the functional aspects deemed as important to this project, is given below, to further highlight the gap in the market.

System	Simple GUI	Usable with no musical knowledge	Allows generation of full compositions	Ability to transition between sets of input parameters	Support for narrative
Abundant Music	No	Somewhat	Yes	Somewhat	No
Wolfram Tones	Yes	Somewhat	No	No	No
cgMusic	Yes	Somewhat	Yes	Yes	No
Jukedek	Yes	Yes	Yes	No	Somewhat
NMGS	Yes	Yes	Yes	Yes	Yes

6 Description of Work

Based on the aims and objectives outlined in the introduction, and what has been learnt from existing systems, a functional specification for the system was created.

For this functional specification, MoSCoW prioritisation is used. In accordance with this prioritisation method, each requirement is categorised as:

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

See the DSDM Atern Handbook [13] for a detailed definition of what should fit into each category. The MoSCoW method is used for this project because, as the handbook suggests, it implies what failing to deliver a requirement will result in. This helps to ensure the Minimum Viable Product (MVP) is delivered.

6.1 Functional Specification

Requirement	Function
Must Have - The system cannot be delivered without meeting these requirements	
Efficient algorithmic music generator	Algorithmically generates musical output at a computational speed that would be eligible for use in a real-time system.
Inputs for target emotion	The user can change the target emotion, which alters the musical output accordingly.
Minor user-control of structure	The user can separate the music into sections and set the duration and target emotions for each section.
Ability to transition between target emotions	The user can generate a full musical composition with changing emotions, allowing the music to follow a narrative.
Simple GUI that doesn't presume prior musical knowledge	The system is accessible to users with no musical background and is easy to learn.
Exporting to a MIDI file	Generated music can be exported as a MIDI file.
Ability to play generated music within the application	The user can listen to the the generated music without the need to export
Should Have - These requirements would be painful to leave out but are not vital	
Further user-control of structure	The user can label and slot sections into the overall music, giving a simple way to repeat sections.
Visual representation of music that will be generated	A visual representation of the output that will be generated is displayed as changes are made.
Choice of instruments	The user can select the instruments used in the generated music, and the instrument's role in the music
Access to seeds for random number generators used	The user can set the seeds for random number generators used in the system, allowing them to exactly recreate previously generated music when the parameters are the same.
Saving project state	The user can save their progress to be able to return to it after changes have been made or the application has been closed.
Video integration	The user can load a video into the application and play generated music alongside it, without the videos original audio.
Could Have - These requirements are desirable but will not significantly impact the system if left out	
User-control of musical features	Access to control of musical features, such as tempo, pitch, key ect. are provided (in an 'advanced' tab) to allow more control over the generated music for those with musical knowledge or the desire to experiment.

Availability in a web-based form	The system is available in a web-based environment to allow use without the need for a download.
User-control of transitioning between target emotions	The user has control over elements regarding transitioning between target emotions, such as weighting the transition towards one section.
Exporting to file formats other than MIDI	Generated music can be exported to file formats other than MIDI, such as MP3, WAV, and FLAC.
Won't Have - For clarification of scope, these requirements will not be delivered in this version of the system	
Integration with Sound API of video game engine	The system provides a library for video game developers and integrates with the Sound API of a video game engine, allowing for dynamic, real-time generation that reacts to changes in the game's narrative during gameplay.
Multiple musical styles	Multiple musical styles/genres, such as 'Classical', 'Blues' etc, can be generated.
User accounts	The user can create an account and log in, providing access to their saved projects

Backlog items for requirements from the 'must have' and 'should have' categories were created, with only 'must have' requirements being planned into the project timeline ahead of time. This was to acknowledge that 'should have' requirements may not be included in the system due to time constraints.

7 Methodology

In this chapter, the methodologies, techniques, tools, algorithms and technologies relevant to the project topic will be described, and the choices made justified.

7.1 Software Development Methodologies

A software development methodology is a model that includes the process, tasks and activities to develop software projects within project constraints such as time [14]. A methodology has been chosen to ensure effective management and therefore success in meeting the aims and objectives.

For this project, an Agile methodology is followed. An Agile method is one that follows the principles outlined in the Agile Manifesto [15]. Agile methodologies often consist of iterative plan-develop-test cycles (or 'sprints') and a backlog to track progress. The methodology was chosen for this project as it allowed for flexibility, providing the opportunity to constantly refine and reprioritise the backlog [16]. Due to varying and potentially unknown workloads from other modules throughout this project, a flexible model such as this was more desirable than rigid methodologies such as a waterfall model.

Sprint retrospectives, used in the scrum framework (a subset of the agile methodology), were also included in the project. This involves, at the end of each sprint, outlining what went well in the sprint, what didn't go well, and what will be improved for the next sprint. This allowed for continual improvement through development as it helped to regularly iron out issues and ensure that good practices were continued.

Following the Agile methodology, the development phase of the project included:

- A project backlog
- Sprints lasting 2 weeks each
- Plan-Develop-Test cycles throughout each sprint
- A retrospective at the end of every sprint

To support this workflow, Hansoft, an Agile Project Management tool that is free for teams of up to 9, was used. Hansoft provides support for a project backlog, sprints, and further features that were deemed potentially useful to the workflow of the project, such as automatically generated burn-down charts.

7.2 Programming languages, libraries, frameworks and plugins

Java, alongside the use of JMusic, an open source Java library that provides music data structures for composition and audio processing, was used for development. The library was used by Plans and Morelli [17] in their music generation system that was interfaced with a video game engine. This shows that while Java's use of features such as garbage collection make it slower than languages such as C++, it can produce a music generation system efficient enough to be eligible for generating music in a real-time system. Java and the JMusic library were chosen for development because they provided the following benefits to this project:

- Applications run on a Java virtual machine, making it multi-platform.
- Object-oriented programming provides modularity for easier troubleshooting, and reuse of code [18].
- Java provides features such as memory management and garbage collection, making it quick to develop in and less error prone, which is useful for a project with time constraints
- Java applications can be embedded in web pages, further improving the accessibility of the system
- Java provides multiple GUI frameworks such as Swing and JavaFX
- JMusic has been well documented [19], which will be useful during the development of the system.
- JMusic supports MIDI I/O, which is a functional requirement of the system
- JMusic is open source which means extensions can be made if they become necessary for the project

Alongside JMusic, the following libraries, frameworks and plugins were used to aid in development and testing:

- JUnit, a unit testing framework for Java, was used to write unit tests for the system.

- JavaFX was used to design the GUI.
- TestFX, a testing library for JavaFX that supports JUnit, was used to write unit tests for the GUI.
- SonarLint, a plugin that lints source code to find bugs and quality issues, was used to help with code quality and consistency.

7.3 Version Control

Version control is a system that records changes to a set of files so that they can be reverted to a previous state. This was seen as useful as several parts of the development of the system required some exploration to find good solutions. For example, some exploration was needed to find the best method for transitioning between emotions. Version control provides an environment to confidently explore these ideas [20]. It also provides a backup for these files on a server, and a history of all committed changes, with the ability to move between these changes, which proved helpful during bug fixing. Git version control was used, due to the fact that unlimited private repository storage is provided to students.

8 Design

To make the design of the system more manageable, it has been separated into four segments. There was a focus on building a complete system in the available time, therefore some sacrifices were made in each segment of the system to ensure that no segment was neglected. The segments are outlined below.

8.1 Music Generator

This segment focuses on the design of a music generator that is the foundation for entire system. This involves choices on the algorithms/methods used to generate the music, the instrument roles that were included, and musical parameters for the system.

8.1.1 Algorithms

Multiple algorithms exist for the purpose of music generation including Markov models, cellular automata, neural-networks and genetic algorithms. It was decided that since genetic algorithms could lead to issues with optimisation, and neural networks require large amounts of existing musical data to train, these algorithms could prove too time consuming for one segment of the system. In Nierhaus [21], it is stated that the use of cellular automata is well suited for genuine sounding composition (music sounding as if it was composed by a human), but less suited for style imitation. This may be why it has not been used in any narrative based systems, if we consider music that is designed to express a specific emotion as a style. Nierhaus also states that the use of Markov models are a well-established paradigm of algorithmic composition, and are used for genuine composition and style imitation. For this reason, Markov models have been employed in the system.

A Markov model is a stochastic model of a system's state transitions. The model consists of a transition matrix that contains the probabilities for each state transition. In NMGS, the states are chords, and the transition matrix is used to find the probability of moving from one chord to the

next. The music can be shaped by altering the values in the transition matrix. Like in Prechtl’s revised music generator, the transition matrix used for NMGS consists of 48 chords (48 rows and 48 columns), containing all major, minor, diminished and dominant seventh chords. Unlike Prechtl’s music generator, all chords are relative to the key that is provided to the music generator, allowing it to be possible to give the user access to setting the key, or perhaps even for key changes during the music.

8.1.2 Parameters

Musical parameters, in form of musical features, are used to shape the music generated. It was essential for these musical features to have been linked to emotions in previous research, to allow for the generation of emotional music further in development. The features, based on existing music generation systems [1, 2, 22], which were based entirely or partially on the work done by Gabrielsson and Lindstrom [23], are as follows:

- Major chords
 - Min (0) = Transitions to all major chords don’t scale at all.
 - Max (1) = Transitions to all major chords scale by 3.
- Minor chords
 - Min (0) = Transitions to all minor chords don’t scale at all.
 - Max (1) = Transitions to all minor chords scale by 3.
- Diminished chords
 - Min (0) = Transitions to all diminished chords don’t scale at all.
 - Max (1) = Transitions to all diminished chords scale by 3.
- Dominant chords
 - Min (0) = Transitions to all dominant 7th chords don’t scale at all.
 - Max (1) = Transitions to all dominant 7th chords scale by 3.
- Mode
 - Min (0) = Minor (Aeolian).
 - Max (1) = Major (Ionian).
 - Depending on the mode and the current chord, different chords/transitions are preferred by scaling all other transitions towards 0.
- Tonality
 - Min (0) = Mode has no effect on the music.
 - Max (1) = Mode has full effect on the music.
- Diatonic

- Min (0) = Transitions to all non-diatonic chords are not scaled at all.
- Max (1) = Transitions to all non-diatonic chords are scaled to 0.
- Pitch register
 - $0 < 1$ = All note pitches are moved down by an octave.
 - $1 < 2$ = All note pitches are left unchanged.
 - $2 < 3$ = All note pitches are moved up by an octave.
- Tempo (beats per minute)
- Velocity (loudness/softness of all midi notes)
 - Min (0) = 40.
 - Max (1) = 70.
 - The velocity of a MIDI note ranges from 0 - 127, with 0 being the softest and 127 being the loudest.

These features were, in general, chosen for their simplicity to implement in JMusic. This was to prevent too much time being spent on this segment of the system. All effects of continuous parameters were scaled linearly. The details for the preferred chords/transitions depending on mode were based on Precht's revised music generator, and are outlined in table 3.7 and table 3.8 in his work [1].

8.1.3 Instrument Roles

Three instrument roles are used in the music generator; a chords role, a bass role and a lead role. A new chord is selected each bar. The chords role fills this full bar with the selected chord, while the bass role follows the root note of the chords one octave lower with four quarter notes. The lead role plays the root, third, fifth and then third of the chord in succession as quarter notes. The roles were based on roles found in Wolfram Tones. A piano was used for the chords role, a bass for the bass role, and a guitar for the lead role, with these sounds existing in the JMusic library. These act mostly as placeholders for within the application, as these sounds can be changed once imported into a Digital Audio Workstation (DAW).

8.2 Generating Emotional Music

This section focuses on how music that expresses a target emotion is generated. This involves choosing a model for the input of emotion, and how musical features are mapped to this model.

8.2.1 Representing Emotion

The circumplex model [24] is a 2D model with emotional valence (or pleasure/displeasure) corresponding to the x axis and emotional arousal (or excitement/calmness) corresponding to the y axis. Common emotion words can be placed on the axis, with happiness in the upper-right, sadness in the lower-left, anger in the upper-left, and serenity in the lower right [24]. However, multiple emotions can fall in the same area, for example, anger and fear are very close together, which would cause

the music generated to sound very similar, despite their perceived emotional differences [2]. To solve this, another dimension could be added to distinguish between these emotions. More complex models that do this exist [25], however adding more dimensions could lead to a confusing interface for the user to select their emotion. To allow for a simple 2D interface, and due to its common use in existing narrative based music generation systems, the circumplex model was chosen to represent emotion.

8.2.2 Parameter Mappings

To be able to generate music that has a target emotion, the parameters previously discussed must be mapped to emotions. This may be done by mapping parameters to the dimensions in the circumplex model [2] (For example, as valence increase, so does pitch register) or by mapping parameters to quadrants of the model [22]. It was decided that the parameters for the music generator would be mapped to the valence and arousal dimensions, and in the following way, based on existing music generation systems [1, 2, 22] and some minor experimentation:

Parameter	Valence Min	Valence Max
Major chords	Low (0.0)	High (1.0)
Minor chords	High (1.0)	Low (0.0)
Dominant chords	High (0.5)	Low (0.0)
Mode	Minor (0.0)	Major (1.0)
Pitch	Low (0.0)	High (3.0)
Tonality	Low (0.75)	High (1.0)
Diatonic	Low (0.5)	High (1.0)
Parameter	Arousal Min	Arousal Max
Tempo	60 BPM	120 BPM
Velocity	Soft (0.0)	Loud (1.0)

The values shown above refer to the parameter values discussed previously. As limited research could be found that explored the role of diminished chords in perceived emotion, the diminished chords parameter was omitted from the parameter mappings, always remaining at 0.0, leaving all diminished chords unscaled.

8.3 Transitioning Between Emotions

This section focuses choosing the algorithm/method used to transition between sets of parameters. Each parameter will need to transition from one state to another.

Interpolation is the technique of estimating transitional values between two points. It would be possible to interpolate the values for each parameter when transitioning between emotions. It would also be possible to gradually move the current valence and arousal towards the new values to simulate interpolation with less effort [1]. However, this would mean that the underlying music generator would have no way of transitioning between sets of parameters, which could cause issues such as preventing the ability to give the user access to individual parameters in later versions of the system. Markov morphing [26] could also be employed, which is an algorithm used to morph two transition matrices. As the algorithm was designed for music generation with Markov models, this could result in very natural sounding compositional transitions, but could also be time consuming

to implement. In the end, linear interpolation between each parameter was chosen. This provided a simple way to transition between sets of parameters, and allowed the length of the transition to easily be adjusted. During transitions, a new transition matrix would have to be generated for each bar, as each bar would have a different set of parameters. Due to this, rows in the transition matrix are generated individually, improving efficiency and the systems eligibility for use in a real-time environment.

8.4 GUI

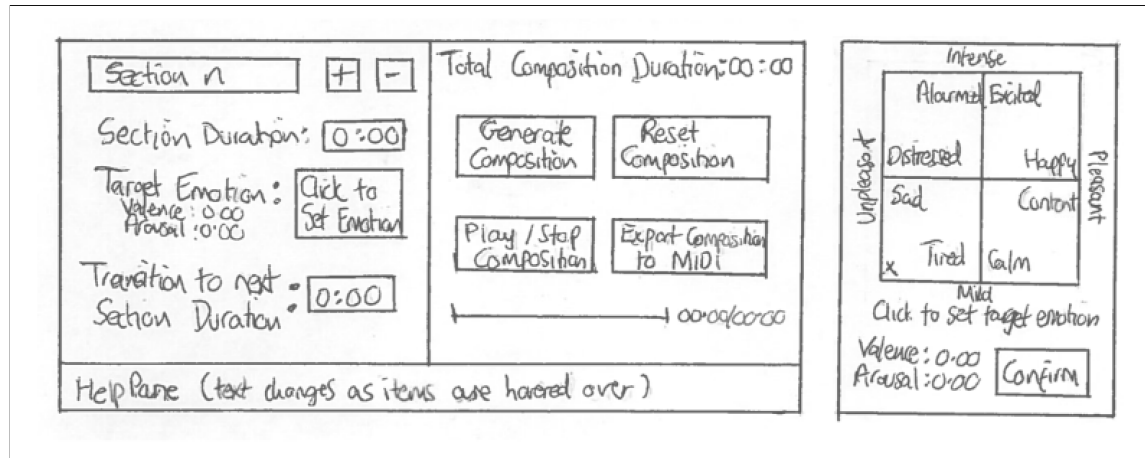


Figure 1: Prototype

A low fidelity prototype (as this was quicker to produce than a high fidelity prototype), shown in Figure 1, was created. This was created while taking into account the aspects of the GUIs of existing systems that were considered positive, and the minimum requirements for the GUI. The minimum requirements, according to the systems 'must have' requirements, state that the GUI should provide:

- A 2D interface for the circumplex model, for emotion input, with common emotion words shown on the model to guide the user.
- Input for duration and target emotion for each section.
- A generate music button and an interface to play the generated music.
- A button to export the generated music to a MIDI file and a window to let the user decide the location they would like to save the file.

A help area that provided tips while the user used the application was also planned, as it was deemed potentially useful in making the system easy to learn.

The emotion words placed on the circumplex model in the prototype were based on the work done by Russell [24]. These were placed on the model to guide the user, but were not used as a guide when designing the parameter-emotion mappings. This means the emotion words on the model are not necessarily representative of the musical output.

8.5 System Architecture Plan

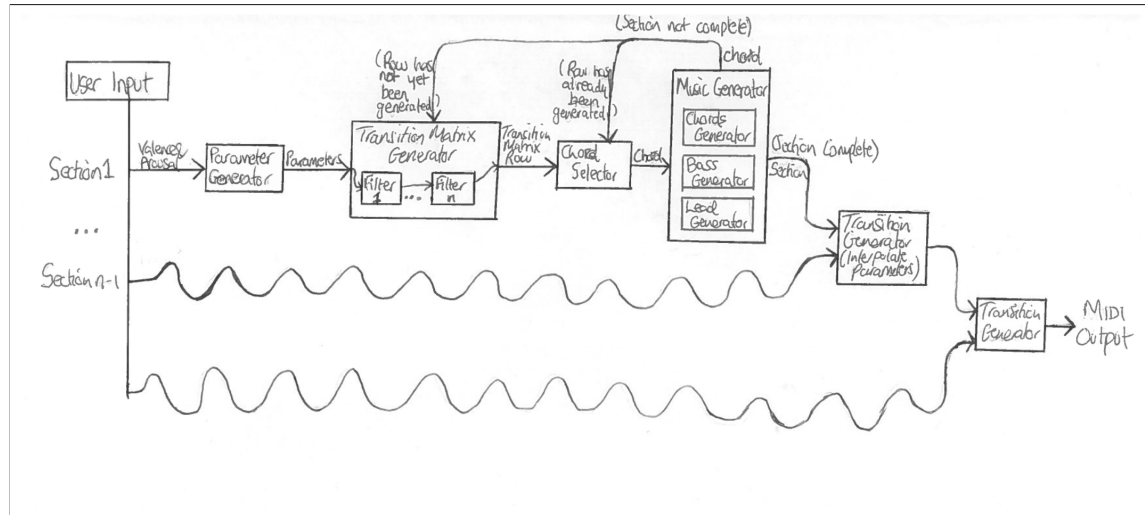


Figure 2: System Architecture Diagram

A system architecture diagram, shown in Figure 2, was created before development to break the system into components, to provide a guideline for the classes that could be created and how they may interact with each other. It must be emphasised that the purpose of this diagram is not to illustrate exactly how the final system functions.

The design of the system was iteratively given more detail during the planning phase of each sprint, as one of the purposes of agile is to reduce large initial planning phases [14].

9 Implementation

In this chapter the classes that make up the system will be outlined, their functionality and interactivity between one other explained, and the difficulties faced through development highlighted. All class names will begin with a capital letter to indicate that the class, rather than the concept the class is named after, is being referred to. IntelliJ IDEA, a Java IDE, was used for the implementation. The project structure, which includes all classes and test classes described in this chapter, can be found in appendix B.

The Implementation of the system was completed with modularity, and loose coupling between classes in mind. This means that each component in the system should have as little knowledge as possible of the definitions of other components. This is to make the system easy to build upon and to allow components to be easily updated or even completely changed without affecting other parts of the system.

9.1 JMusic

JMusic provides several data structures and helper classes to aid in creating computational compositions that were used during development. These are outlined below.

The **Note** class is the data structure used for a musical note, containing useful information such as the note's pitch, length, and dynamic (velocity).

The **Phrase** class is used to hold a list of notes, while the **CPhrase** class is used to hold a list of chords (groups of Notes that share the same onset time and length).

The **Part** class is used to hold a list of Phrases or CPhrases. A part can also have an instrument associated with it.

The **Score** class is used to hold a vector of Parts. These Parts are layered on top of each other such that each Part that is added to the Score will start at the same time and play simultaneously.

The **Write** class provides the means to turn a Score into a MIDI file, writing the file to the current directory.

JMusic also provides constants, making it easier to use and making the code more readable. These constants include note pitches (e.g. C4) and note lengths (e.g. QUARTER_NOTE).

9.2 Back-end

The classes outlined here make up the back-end of the system. They are the fundamental building blocks used to generate the music in NMGS. These classes were built mostly based on the system architecture diagram outlined in Figure 2.

9.2.1 Parameter Generator

The **ValenceArousalModel** class takes valence and arousal values, and generates a set of musical parameters, based on the parameter mappings outlined in the Design chapter. It is described as the 'Parameter Generator' on the architecture diagram.

9.2.2 TransitionMatrixGenerator

The following classes make up the transition matrix generator outlined in the architecture diagram.

The **ChordTransitionMatrix** class is a wrapper for the 48 x 48 transition matrix used for chord transitions. This class was not originally on the architecture diagram but was added as it made the TransitionMatrixGenerator class easier to read and more modular. It initialises all values in the matrix to 1 and provides a means to reset the transition matrix back to this once it has been altered. The probabilities in a row of the transition matrix are relative to each other. This means that if the value for the transition to chord C major is 1 and a value for the transition to chord D major is 2, it will be twice as likely that a transition to D major is made than to C major.

The **TransitionMatrixFilter** class is an abstract class that provides the basis for transition matrix filters. A transition matrix filter can be used to alter the values in the ChordTransitionMatrix. All transition matrix filters consist of a filter value, that decides the degree to which the filter will have an effect on the ChordTransitionMatrix. The filters are used to implement many of the musical parameters in the system outlined in chapter 8.1.2.

The **ChordsFilter** class consists of four nested classes: MajorChordsFilter, MinorChordsFilter, DiminishedChordsFilter and DominantChordsFilter. These classes scale up all major chords, minor chords, diminished chords or dominant seventh chords, by the amounts specified chapter 8.1.2.

The **TonalityFilter** class handles two musical parameters, tonality and mode. Tonality is used as the filter value, and decides the degree to which the filter has an effect on the transition matrix (which in this case is the degree to which the mode parameter has an effect, as mode will have no effect when tonality is set to 0). Mode is used to determine the chords that are scaled. There are preferred chords (chord transitions that are preferred regardless of the current chord) and preferred transitions (chord transitions that are preferred based on the current chord) for both major and minor modes, based on Prechtl's revised music generator as previously stated. If the mode is more major (> 0.5), preferred major mode chords/transitions are scaled down less and less, while their minor mode counterparts are scaled down more and more. The opposite applies for when the mode is more minor (< 0.5). All other chords (apart from those that are diatonic, which are scaled down the same amount as the preferred chords/transitions that are scaled down the least) are scaled towards 0 as tonality increases. A full major (1) or full minor (0) mode value would scale all the opposite mode's preferred chords/transitions down by the same amount as all other chords, leaving it's own preferred chords/transitions (and diatonic chords) unscaled.

The **DiatonicFilter** class scales all non-diatonic chords towards 0 as the filter value increases, as specified in chapter 8.1.2.

The **TransitionMatrixGenerator** class consists of a ChordTransitionMatrix and all of the transition matrix filters. It takes the major chords, minor chords, diminished chords, dominant chords, tonality, mode, and diatonic parameters, and uses them to apply each filter to the transition matrix. The class generates one row of the matrix at a time, depending on the chord that is needed. This improves efficiency, as the transition matrix will need to be reset and all the filters applied again if the musical parameters change, and many rows in the transition matrix may not have been used by this point. The efficiency is further improved by storing the generated row so that it does not need to be generated more than once.

9.2.3 MusicGenerator

The following classes make up the music generator outlined in the architecture diagram.

The **ChordGenerator** class provides methods for generating an array that contains the JMusic integers for the note pitches that make up a chord. This class did not originally exist on the architecture diagram, but was added to prevent the MusicGenerator class from becoming too large. Methods are provided for the four chord types used in the system, and require the root note to generate the chords. An obstacle here is that due to there being four notes in a dominant 7th chord and three notes in the other chords used, and due to the randomness of the system, testing was proving to be extremely difficult (e.g. checking that a chord was added to each bar in the composition, as there is no constant number of notes to check for). To solve this, all chords have four notes, with the final note being set to a note with no sound if the chord should have three notes.

The **ChordSelector** class is a helper class that takes a row in the transition matrix and selects a chord based on the probabilities in the matrix. This is done by generating a random number between 0 and the sum of all probabilities in the row, then iterating through the row, while summing each probability, and stopping at the row index where the random number is reached. This class was originally planned as separate to the music generator, but it was moved to being part of the music generator as this simplified the interaction between the music generator and transition matrix generator.

The **MusicGenerator** class uses a ChordSelector object and a ChordGenerator object to create

three Part objects, one for each instrument role. This is also where the instruments (guitar, piano and bass) are selected. The class is used to add one bar to the Parts at a time, from a given row (corresponding to the current chord) in the transition matrix, and returns the selected chord. This allows it to be used to create a section of music by passing a row from the transition matrix to it to add a bar of the composition, then passing the the row with the index it returns back to it to add another bar, and repeating for as long as necessary. CPhrase objects are used to add to the chords Part, while Phrase objects are used to add to the bass and lead Parts. The ChordSelector is used to select the chord from the given row, and the ChordGenerator it used to find the notes to populate the Phrases and CPhrase.

9.2.4 Using the TransitionMatrixGenerator and MusicGenerator to create sections and transitions

The **Section** class's functionality is roughly captured on the architecture diagram from the point that parameters are passed to the transition matrix generator up to the music generator outputting a complete section. It holds a set of musical parameters (including the key), the length of the section, and the transition lengths (set in bars) for the start and end of the section. The class uses a TransitionMatrixGenerator object and a MusicGenerator object to create a Score object for the section, consisting of all three instrument roles. One limitation here is that the transitions between sections are considered as part of the section, and therefore a transition length is needed for the start and end of every section (apart from the start of the first section and the end of the last section). These transition lengths make up half of the full transition (a full transition being the transition length for the end of one section combined with the transition length for the start of the next section), meaning that all full transitions must be even in length. This limits the detail in transition lengths, as each even increment in transition length can be anywhere between 5 and 8 seconds, instead of half this amount. This issue could have been easily avoided if transitions were originally designed to be separate entities to sections; this is arguably a downside to the lack of initial planning associated with an agile workflow. Fortunately, the modularity of the system and loose coupling between classes allows for this issue to be easily rectified in future versions.

The **TransitionHelper** class acts as the TransitionGenerator on the architecture diagram, however it acts as more of a helper class to aid in the creation of transitions between sections. It takes two sets of musical parameters and interpolates a value for each parameter, based on the transition length and the current bar in the transition that is being generated. For example, if the section before the transition has a tempo of 60, the section after the transition has a tempo of 110, and the transition length is 4, the interpolated tempos for bars 1, 2, 3 and 4 will be 70, 80, 90 and 100, respectively. The TransitionHelper can be used to generate each bar of the transition, outputting each individual bar as a Score object. It does not generate the full transition by itself, as each Score must still be combined to create the full transition. Similarly to the Section class, the TransitionHelper class also uses a TransitionMatrixGenerator object and a MusicGenerator object to generate a score. The difference here is that, due to each bar in the transition likely having a different set of musical parameters (as the sections it is connecting will likely have different sets of musical parameters), the transition matrix must be reset for each bar.

9.2.5 Creating the full composition

The **Composition** class takes an array of Section objects and uses them to generate Scores for the sections, as well as using the TransitionHelper and the transition lengths from the Sections

to generate Scores for the transitions, combining all of these to create a full composition. As the TransitionHelper builds a transition one bar at a time, the Composition class is responsible for building the overall transition by appending the the Scores for each bar in the transition. The overall generation is done by generating a section, then generating a transition, and repeating, while appending Scores as it goes along. A significant difficulty faced here is in regards to JMusic's lack of support for variable tempos throughout a Score, leaving gaps and overlaps in the final composition if the tempo changes. JMusic does, however, provide the ability to manually set the start times, in bars, of each Phrase. Using this, the start times of each Phrase/CPhrase were manually set in the MusicGenerator class when they are added, which ensures the timings for an individual section, and each transition bar, are correct. In addition to this, in the Composition class, when a Section or whole transition is appended, the start times for every Phrase/CPhrase in the appended section/transition is updated to ensure that each whole section/transition is in time. Updating start times is done by choosing a tempo of 60 for the overall Score, and calculating, based on the tempo of each phrase, what it's start time in bars should be. This must be done constantly throughout the music, instead of when the whole composition has been created, to ensure that the system can still be eligible for use in a real-time environment. This was one of the most difficult and time consuming issues to solve during development.

9.3 Front-end

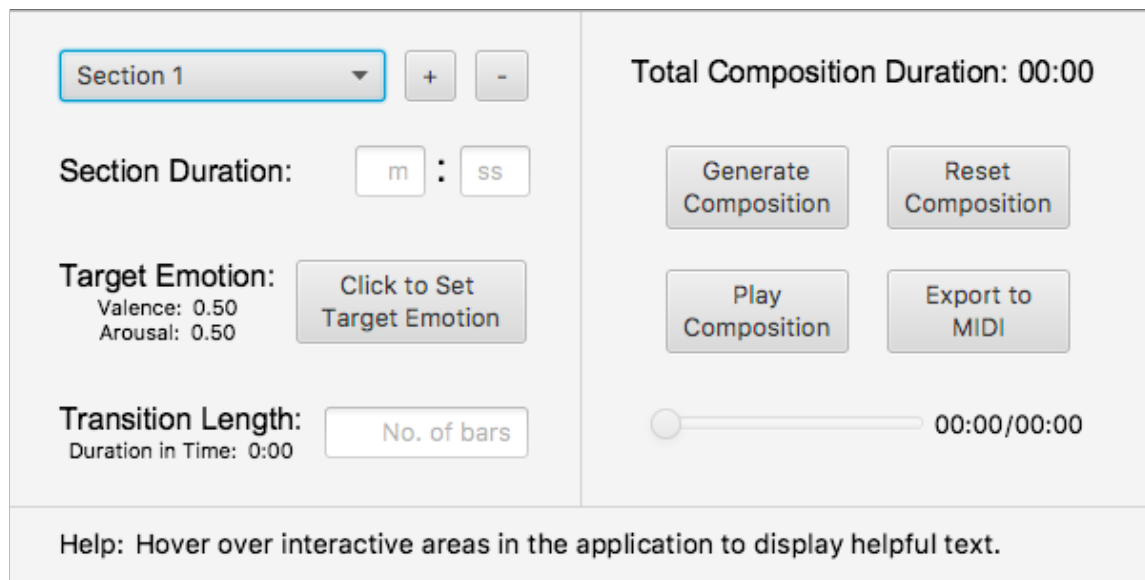


Figure 3: Main Window

A JavaFX application consists of FXML documents used to describe the design of the GUI (one per window), and Controller classes associated with the FXML documents to handle the logic. SceneBuilder, an application that provides a user interface that can be used to generate FXML documents, was used, saving time on development. The main window and target emotion selector



Figure 4: Target Emotion Selector Window

window, based on the low fidelity prototype, are shown in Figures 3 and 4 respectively. The classes related to the GUI, are outlined below.

The **SectionParameterContainer** class is used as a container for the musical parameters. It stores the valence and arousal values, the section duration, and start and end transition lengths, for a single section. It was designed to be used to store user input. To aid in generating music it also contains a **ValenceArousalModel** object, and uses it to generate parameters. An issue that was faced here was how to allow users to set any value for the section duration. This was an issue because the tempo is decided by the value of arousal, and therefore to prevent a section from ending mid-bar, the selection for the duration of the section must be limited to specific values that ensure that, based on the tempo, the number of bars is an integer. This would be very limiting on the system. The issue was solved by overwriting the tempo generated by the **ValenceArousalModel** based on the duration of the section. This is done by finding out the number of bars that would be generated (likely a decimal value) if the existing tempo was used, then rounding to the nearest integer, and calculating the tempo that would be required to make this number of bars fill up the duration of the section. This new tempo is returned when generating parameters using the **SectionParameterContainer**. The minimum section duration allowed by NMGS, which is six seconds, can also be explained by this. A section duration of six seconds was found, through experimentation, to be the lowest duration that allowed the arousal value to still consistently have an effect on the tempo. Any lower than six

seconds and a large selection of arousal values lead to the same tempo, with there being a sudden change in tempo at a threshold in the arousal value (decided by the duration). As tempo is shown to be an important factor in the emotion perceived [27], and as this behaviour could be confusing, a minimum section duration of six seconds was enforced.

The **MainWindowController** class handles all the logic for the main window. It contains a list of `SectionParameterContainers`, with one `SectionParameterContainer` object per section that the user adds (up to a maximum of 9). A section can be selected by using a drop-down list, and all details for the currently selected section are saved in the relevant `SectionParameterContainer`.

There are multiple buttons in the main window, for which the `MainWindowController` handles the logic. The '+' button adds a section to the end of the list of sections, while the '-' button removes the currently selected section. The 'Click to Set Target Emotion' button opens the target emotion selector window for altering the current section's valence and arousal. The 'Generate Composition' button uses the list of `SectionParameterContainers` to generate parameters for a list of `Section` objects, which are passed to a `Composition` object to generate a `Score` that matches the user's input. It then uses the `Write` class to write the `Score` to a MIDI file and ends by loading the file for playback by using Java's `Sequence` and `Sequencer` classes, which are designed for MIDI playback. The 'Reset Composition' button deletes all sections, leaving one empty section, as when the application is first opened. The 'Play Composition' (or 'Stop Composition') button plays or stops the `Sequence` that is loaded in the `Sequencer`, with a progress bar below to show the current time in playback and allow the user to move playback to the point that they would like (functionality that is conveniently provided by the `Sequencer`). A background thread regularly checks on the state of the `Sequencer`, updating the current time in playback on the GUI properly stopping the `Sequencer` once the `Sequence` has ended. The 'Export to MIDI' button uses Java's `FileChooser` class to open a dialog box that allows the user to choose a location to save their MIDI file. As Java is cross-platform, the `FileChooser` class automatically handles the logic of opening the dialog box used by the platform the application is running on. Here, Java's `MidiSystem` class is used to save the file from the `Sequence`, as the `Write` class does not provide the functionality to write a file to a specific location.

The `MainWindowController` also handles the logic for what text is allowed in each text field. Java's support of regular expressions was useful here, as they are used to restrict text to only numbers and limit values. In addition to this, text fields are checked when the focus leaves them (when the user clicks on another element of the GUI), and their values potentially updated. This is to solve issues that regular expressions cannot. For example, as the minimum section duration is six seconds, but the regular expression used for the seconds text field cannot enforce a minimum of six, as less than this minimum should be allowed when the minutes text field contains a value of one or larger. Other functions that occur when the focus leaves a text field include updating elements of the GUI, such as the total composition duration, and saving values to the relevant `SectionParameterContainer`.

The transition area, which allows users to set transitions for the section following the currently selected one, is only visible when the currently selected section is not the final one. A significant issue that was faced here, is that there was an oversight in how the duration of transitions could be set. As the tempo of each bar in the transition is interpolated from the section before the transition and the section after it (as illustrated in the description of the `TransitionHelper` class in chapter 9.2.4), setting the duration of the transition would likely cause the transition to end mid-bar, similarly to the issue faced in the `SectionParameterContainer` class. Unlike that issue, the tempo here is changing for each bar in the transition, meaning the same solution cannot be

applied. Due to this problem, the user can only set the transition length in bars, which limits the user from entering exact durations for transitions, and could become confusing for users with no musical knowledge, which is an important issue the system is attempting to solve. Another issue that this caused, was that the transition duration (in minutes and seconds) needed to be calculated to be displayed for the user and to show the total composition duration. The transition duration is calculated through interpolation in the exact same way that tempo is interpolated from the sections before and after the transition in the `TransitionHelper` class. This means that if the `TransitionHelper` class is updated to do this in a new way, the way that the transition duration is calculated will also need to be updated to not be inaccurate. This dependancy and tight coupling with the `TransitionHelper` class could cause extending and updating this area of the software to be more difficult.

The final part of the main window is the help area. This area provides help text that changes as the user hovers over different elements in the application. The text provided here was chosen based on what was considered potentially confusing. For example, it is stated when hovering over the transition length text field that the transition length is for the transition to the next section. The help area is also responsible for displaying any issues that may affect the user, for example if there is an issue generating the music when the user clicks the 'Generate Composition' button, the text displayed will notify the user that the generation was unsuccessful and attempt to advise the user on why this is (e.g. Section n is not complete).

The **`TargetEmotionSelectorController`** handles all the logic for the target emotion selector window. Initial valence and arousal values are passed to it to initialise the location of the crosshair on the circumplex model graph displayed in the window. Users are able to click on the graph to change the valence and arousal values, with the location of the crosshair providing feedback to the user. Clicking the 'Set Target Emotion' button will use the index of the currently selected section to set the new valence and arousal values in the relevant `SectionParameterContainer`, and close the target emotion selector window. It was decided that while the target emotion selector window was open, the main window should be disabled. While it could be a useful feature to be able to change the currently selected section while the target emotion selector window is open, as it would allow for the valence and arousal values of different sections to be quickly changed without closing the window, the logic for this was deemed too time consuming given the time constraints.

9.4 Testing

Unit testing was performed throughout development as it can help to ensure that the software behaves correctly, and that it is robust [28]. Test-driven development, which requires writing tests first and writing software that causes these tests to pass, was not followed, as it can be time consuming up front [29]. JUnit was used to perform unit testing during development. One test class, consisting of multiple unit tests, was created per class, with every class being tested. The tests were designed to see if the class being tested performed it's overall function correctly, without being concerned with how this was achieved. The tests for the Controllers were written using TestFX alongside JUnit, which tests the GUI by simulating user interaction. This allows tests to be written for things such as text field restrictions.

Further levels of testing, such as integration testing and system testing, would have been useful as it could have identified issues with the interaction between modules and the system as a whole, which unit testing cannot accomplish. However, this level of testing would have been too time consuming for the project. As such, there are likely some issues that have not yet been identified.

The results of a final run of the unit tests can be found in appendix C

10 Evaluation

Upon completion of the development phase, the system was evaluated to review whether it had been successful in meeting its aims and objectives, and solving the problems outlined in the Motivation chapter. Several aspects of the system should be evaluated (based on the system's sub-objectives outlined in the design chapter and the issues outlined in the motivation chapter) and many of these aspects would have been very difficult to evaluate without users or listeners. There also needed to be a way to quantitatively measure the success of the system. As criteria-based assessment gives a quantitative measurement of quality in a number of areas, including usability [30], and users/listeners were needed, criteria-based user evaluation was employed. The aspects of the system that were evaluated, and the criteria that were drawn from them, are outlined below:

- The ability for the system to produce music that follows a narrative:
 1. The music generated can express target emotions.
 2. The music can reflect the narrative of a visual sequence.
 3. The transitions between target emotions are natural sounding.
- Usability for the purpose of generating music that follows a narrative:
 4. The GUI is simple.
 5. The system is easy to learn.
 6. Using the system requires no musical knowledge.
 7. Minimal need for experimentation is needed to achieve a desired result.

An email was sent out to all computer science students for recruitment for participants, however due to minimal responses, most were friends or family.

10.1 Method

To attempt to assess whether these criteria had been met, the following user evaluation, consisting of two stages, was planned.

10.1.1 Stage 1

The user was asked to watch five versions of a 32 second video clip. Each version of the video clip (labelled A-E) had different background music, each generated by NMGS. A segment from "Salt and Pepper - Romantic 1 Minute Stop Motion Animation" by Tatu Pictures Ltd [31] was chosen as it contained clear emotional changes. One of the compositions (used in clip B) was created to mirror the emotions and narrative of the clip, while another (used in clip E) was created to do the opposite. These compositions were each created within three minutes and within three clicks of the 'Generate' button. This was to attempt to further solidify, should the results be positive, that an intended result can be achieved with minimal experimentation. The remaining three pieces of generated music (used in clips A, C and D) were generated with random valence and arousal

values, with all five clips having the same number of sections and the same (as much as was possible) section and transition durations (The input parameters used to generate the music for each clip can be found in appendix D). This allowed the changes in the different compositions to be focussed on the target emotions of each section, and ensured that the timing of sections did not have an affect on the results. Users were asked to answer a questionnaire, the questions for which are shown in appendix E, alongside watching the clips. This questionnaire was designed to evaluate the system against criteria 1, 2 and 3. Users were told that they could rewatch/listen to any of the clips as many times as desired to help them answer the questionnaire. The letters a, b, c, d and e in the questions refer to the clips A, B, C, D, and E, respectively, that were viewed by the participant. A five-level likert scale was used for the questions here, with 1 being the most negative response and 5 being the most positive (refer to the questionnaire for details).

10.1.2 Stage 2

The user was asked to attempt to use NMGS to generate music for the clip shown to them in stage 1. They were provided the clip with no audio to help them complete the task. No instructions for using the application were given, and it was made clear that if the user was struggling to use the application, they can ask questions. The user was asked to, once they decided they were done with the task, complete a second questionnaire, shown in appendix F. This questionnaire was mainly designed to evaluate the system against criteria 4, 5, 6 and 7, but also briefly provide further evaluation to criteria 1,2 and 3 by asking the user the same questions asked about the clips in stage 1, but about the music that they had generated. This questionnaire also asked for qualitative feedback on the usability of the application and the music generated by it. Likert scales continued to be employed for most questions, with both three-level and five-level scales being used (refer to the questionnaire for details).

A note to be made on questionnaires is that the actual questionnaires filled out by the users were created in SmartSurvey, an online survey and questionnaire tool. The questions shown in the questionnaires in the appendices are identical to those asked in the study.

10.2 Results

The statistical analyses of questionnaire responses are shown below. The analyses were mostly done using SPSS Statistics. The reasons for the chosen analyses should become apparent in the discussion following the results.

10.2.1 Stage 1

For each question in the stage 1 questionnaire, a Friedman test was performed to find out if there were any significant differences between the answers for each clip. After this, as there was a significance ($\text{Sig} < 0.05$) for all three questions, post hoc two-tailed Wilcoxon tests were done between all paired combinations of clips for each question. The mean answers and standard deviations for each question and each clip are also shown alongside the results of these tests.

In question 1 ("How well do you feel the emotions conveyed by the music matched the emotions conveyed in the clip?"), the results, shown in Figure 5, indicated a significant difference between answers for clip B and all other clips, showing that clip B performed the best, with the strongest significance between clip B and clip E. A significant difference was also seen between clip E and all

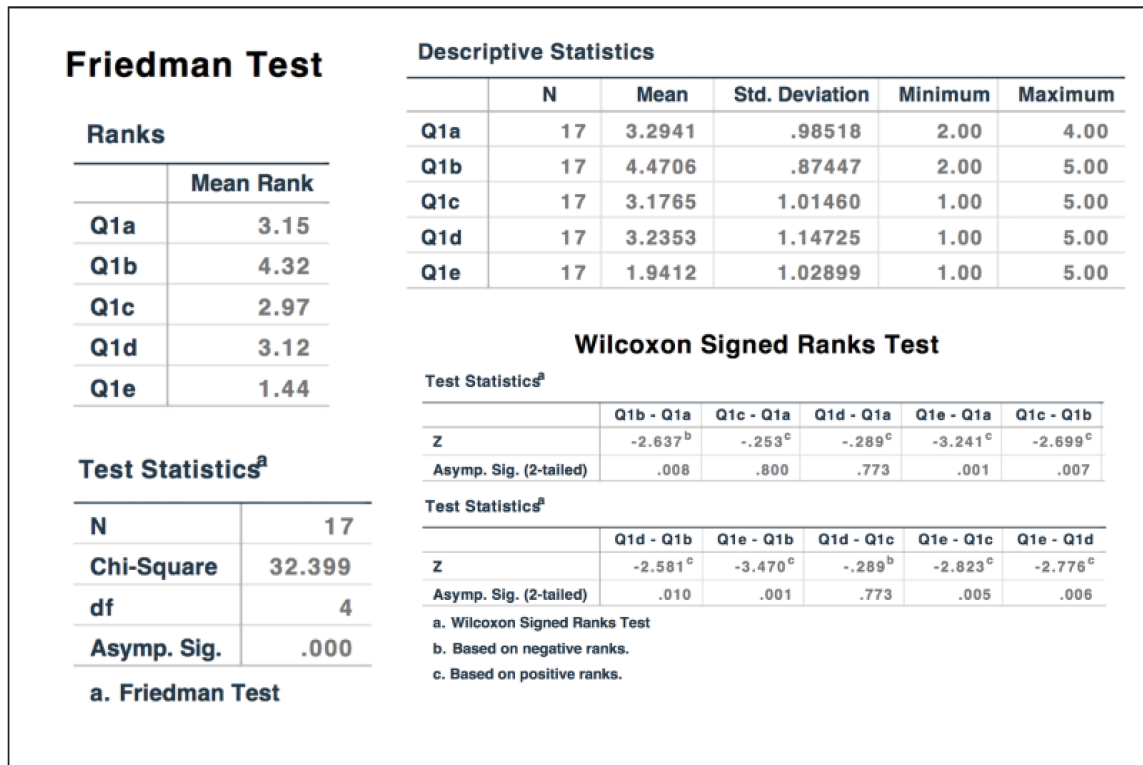


Figure 5: Stage 1 Question 1 - Descriptive Statistics, Friedman Test and Wilcoxon Test Results

other clips, showing that clip E performed the worst. No significance was seen between clips A, C and D. Clip B had the highest mean (4.47) with the lowest standard deviation (0.87).

In question 2 ("How well do you feel the music matched/reflected the narrative/story of the clip?"), results, shown in Figure 6, indicated a significant difference between clip B and all other clips except clip D, showing that clip B outperformed most clips. Again, the strongest significance was seen between clip B and clip E, and a significant difference was seen between clip E and all other clips, showing that clip E performed the worst. No significance was seen between clips A, C and D. Clip B had the highest mean (4.24), but also had the highest standard deviation (1.03).

In question 3 ("The music transitioned between different intended emotions during the clip. How natural did these transitions sound?"), results, shown in Figure 7, indicated that clip B outperformed clips E and C, and clip A outperformed clips E and C. The strongest significance was seen between clip A and clip C.

10.2.2 Stage 2

For stage 2, questions 4, 5 and 6, which are similar to questions 1, 2 and 3 from stage 1 but are asked about the music that the user generated alongside the clip (referred to from now on as clip F), had two-tailed Wilcoxon tests run on them, comparing the answers to the questions in stage 2 to their counterpart questions (for all clips) in stage 1.

Friedman Test		Descriptive Statistics				
Ranks		N	Mean	Std. Deviation	Minimum	Maximum
	Mean Rank					
Q2a	3.18	17	3.3529	.93148	2.00	4.00
Q2b	4.09	17	4.2353	1.03256	2.00	5.00
Q2c	2.79	17	3.0588	.96635	1.00	4.00
Q2d	3.35	17	3.4706	.94324	2.00	5.00
Q2e	1.59	17	2.0000	1.00000	1.00	5.00

Wilcoxon Signed Ranks Test					
Test Statistics ^a					
	Q2b - Q2a	Q2c - Q2a	Q2d - Q2a	Q2e - Q2a	Q2c - Q2b
Z	-1.987 ^b	-.890 ^c	-.541 ^b	-3.168 ^c	-2.409 ^c
Asymp. Sig. (2-tailed)	.047	.374	.589	.002	.016

Test Statistics ^a					
	Q2d - Q2b	Q2e - Q2b	Q2d - Q2c	Q2e - Q2c	Q2e - Q2d
Z	-1.709 ^c	-3.289 ^c	-1.645 ^b	-2.982 ^c	-3.425 ^c
Asymp. Sig. (2-tailed)	.087	.001	.100	.003	.001

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.
c. Based on positive ranks.

Figure 6: Stage 1 Question 2 - Descriptive Statistics, Friedman Test and Wilcoxon Test Results

The comparison between answers for question 4 from stage 2 and question 1 from stage 1, the results of which are shown in Figure 8, indicated a significant difference between clip F and clip B, showing that clip B performed better. There was also a significant difference between clip F and clip E, showing that clip E performed worse, with there being a larger significance between clip F and clip E than between clip F and clip B. No other significance was seen.

The comparison between answers for question 5 from stage 2 and question 2 from stage 1, the results of which are shown in Figure 9, indicated a significant difference between clip F and clip E, showing that clip F outperformed clip E. No other significance was seen.

The comparison between answers for question 6 from stage 2 and question 3 from stage 1, the results of which are shown in Figure 10, indicated that there was no significant difference seen. Friedman tests should have been run these questions to avoid needing to doing this set of post-hoc tests, but to save time they were not, which evidently proved to be counterproductive.

Question 1 ("Do you have any music theory knowledge?"), had a chi-squared test run on it alongside Question 7 ("How pleased are you with the result of the task?"). This was to find if there was any relationship between the two sets of answers. However, one of the conditions for pairs of variables that produce cross-tabulations larger than 2x2 is that "No more than 20% of the expected counts [should be] less than 5 and all individual expected counts [should be] 1 or greater" [32, p. 734]. In the cross-tabulation for answers to question 1 and question 7, 88.9% of expected counts were less than 5, with several being smaller than 1, meaning the results of the test are not

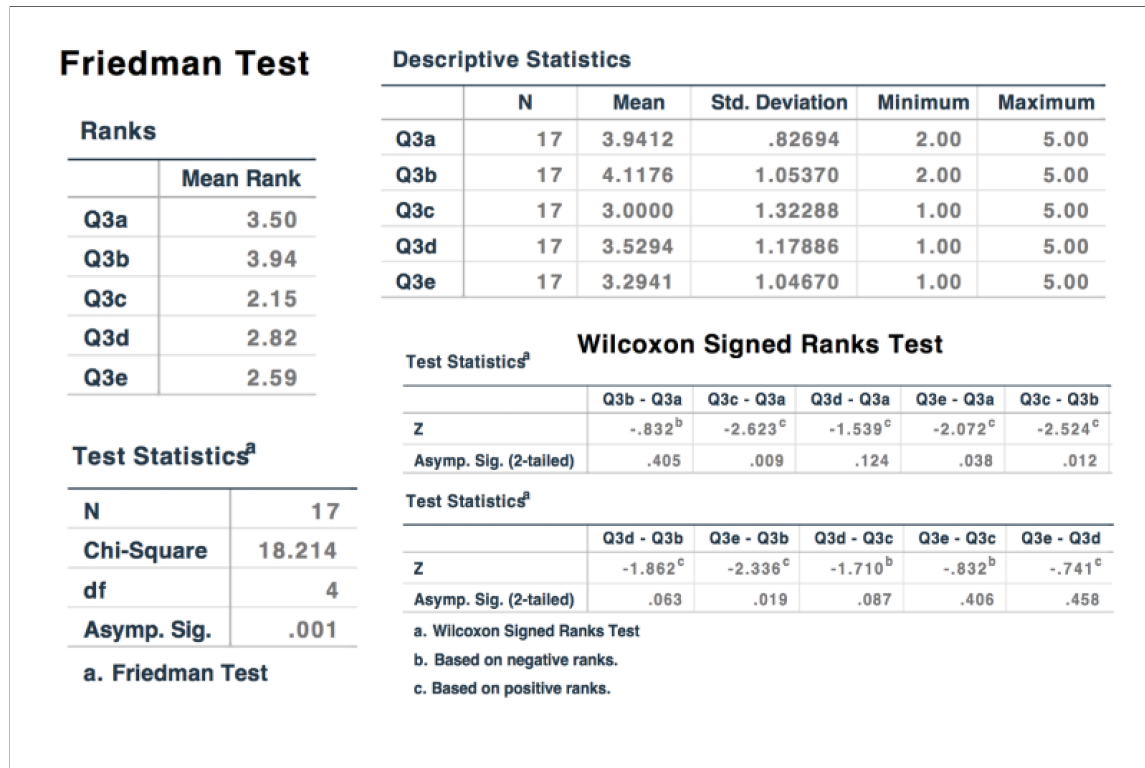


Figure 7: Stage 1 Question 3 - Descriptive Statistics, Friedman Test and Wilcoxon Test Results

a good approximation. Following this the mean answers for question 7 based on the answers for question 1 were calculated. This showed that users who stated that they had little to no musical theory knowledge were on average no less pleased with the result of their task than users who knew some or lots of musical theory. The chi-squared and mean results are shown in Figure 11.

For questions 2, 3, 6, 7, 8 and 9, distributions are shown in Figures 12, 13, 14 15, 16 and 17 respectively. For question 2 ("How simple/complex did you find the graphical user interface to use/navigate?") the majority of users stated that the GUI was 'somewhat simple' to use/navigate, with no users stating that they found it 'somewhat complex' or 'very complex' to use/navigate. For question 3 ("How easy/difficult did you find learning how to use the application?"), the majority of users stated that the application was 'somewhat easy' to learn, with one user stating that it was 'somewhat difficult' and no users stating that it was 'very difficult'. For question 6, the majority of users stated that transitions sounded 'mostly natural', closely followed by 'I'm not sure'. No users stated that transitions sounded 'not natural at all'. For question 7, the majority of users were 'somewhat pleased' with their result, with no users being 'somewhat displeased' or 'very displeased'. For question 8 ("How much experimentation/tweaking do you feel you needed to do to get to your result?"), the majority of users stated that 'some experimentation was required', with the distribution being skewed towards there being less experimentation needed. For question 9 ("Did you ask for help using the application to complete the task?"), the majority of users stated

Descriptive Statistics					
	N	Mean	Std. Deviation	Minimum	Maximum
S2Q4	17	3.8235	.80896	2.00	5.00

Test Statistics ^a Wilcoxon Signed Ranks Test					
	S2Q4 - Q1a	S2Q4 - Q1b	S2Q4 - Q1c	S2Q4 - Q1d	S2Q4 - Q1e
Z	-1.718 ^b	-1.990 ^c	-1.761 ^b	-1.490 ^b	-3.141 ^b
Asymp. Sig. (2-tailed)	.086	.047	.078	.136	.002

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.
c. Based on positive ranks.

Figure 8: Stage 2 Question 4 - Descriptive Statistics and Wilcoxon Test Results

Descriptive Statistics					
	N	Mean	Std. Deviation	Minimum	Maximum
S2Q5	17	3.7647	.75245	2.00	5.00

Test Statistics ^a Wilcoxon Signed Ranks Test					
	S2Q5 - Q2a	S2Q5 - Q2b	S2Q5 - Q2c	S2Q5 - Q2d	S2Q5 - Q2e
Z	-1.374 ^b	-1.848 ^c	-1.849 ^b	-.924 ^b	-3.089 ^b
Asymp. Sig. (2-tailed)	.169	.065	.064	.355	.002

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.
c. Based on positive ranks.

Figure 9: Stage 2 Question 5 - Descriptive Statistics and Wilcoxon Test Results

they didn't ask for help during the task, with three users out of the seventeen asking for help once and no users asking for help multiple times.

The raw responses for the questionnaires, including the responses to the text fields provided in the stage 2 questionnaire can be found in appendix G.

Descriptive Statistics					
	N	Mean	Std. Deviation	Minimum	Maximum
S2Q6	17	3.5294	1.12459	.00	5.00

Test Statistics ^a Wilcoxon Signed Ranks Test					
	S2Q6 - Q3a	S2Q6 - Q3b	S2Q6 - Q3c	S2Q6 - Q3d	S2Q6 - Q3e
Z	-1.100 ^b	-1.779 ^b	-1.244 ^c	-.107 ^c	-.679 ^c
Asymp. Sig. (2-tailed)	.271	.075	.213	.915	.497

a. Wilcoxon Signed Ranks Test
b. Based on positive ranks.
c. Based on negative ranks.

Figure 10: Stage 2 Question 6 - Descriptive Statistics and Wilcoxon Test Results

Q1 * Q7 Crosstabulation

			Q7			
			3.00	4.00	5.00	Total
Q1	1.00	Count	1	6	4	11
		Expected Count	.6	7.1	3.2	11.0
	2.00	Count	0	3	1	4
		Expected Count	.2	2.6	1.2	4.0
	3.00	Count	0	2	0	2
		Expected Count	.1	1.3	.6	2.0
Total	Count	1	11	5	17	
	Expected Count	1.0	11.0	5.0	17.0	

Means

Q7

Q1	Mean	N	Std. Deviation
1.00	4.2727	11	.64667
2.00	4.2500	4	.50000
3.00	4.0000	2	.00000
Total	4.2353	17	.56230

Chi-Square Tests

	Value	df	Asymptotic Significance (2-sided)
Pearson Chi-Square	1.967 ^a	4	.742
Likelihood Ratio	2.820	4	.588
Linear-by-Linear Association	.299	1	.585
N of Valid Cases	17		

a. 8 cells (88.9%) have expected count less than 5. The minimum expected count is .12.

Figure 11: Stage 2 Question 1 & Question 7 - Chi-Squared and Mean Results

10.3 Discussion

Based on the results of the analyses of the questionnaires, whether the system has been successful in meeting the evaluation criteria is discussed below. It is worth pointing out that stronger, more

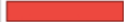


			Response Percent	Response Total
5	Very simple		23.53%	4
4	Somewhat simple		52.94%	9
3	Neither simple nor complex		23.53%	4
2	Somewhat complex		0.00%	0
1	Very complex		0.00%	0

Figure 12: Stage 2 Question 2 - Distribution of Results




			Response Percent	Response Total
5	Very easy		23.53%	4
4	Somewhat easy		70.59%	12
3	Neither easy nor difficult		0.00%	0
2	Somewhat difficult		5.88%	1
1	Very difficult		0.00%	0

Figure 13: Stage 2 Question 3 - Distribution of Results





			Response Percent	Response Total
1	Not natural at all		0.00%	0
2	Mostly unnatural		6.25%	1
3	I'm not sure		37.50%	6
4	Mostly natural		43.75%	7
5	Completely natural		12.50%	2

Figure 14: Stage 2 Question 6 - Distribution of Results




			Response Percent	Response Total
1	Very unpleased		0.00%	0
2	Somewhat unpleased		0.00%	0
3	Neither pleased nor unpleased		5.88%	1
4	Somewhat pleased		64.71%	11
5	Very pleased		29.41%	5

Figure 15: Stage 2 Question 7 - Distribution of Results

valid conclusions could have been made with a larger number of participants.






			Response Percent	Response Total
5	Very little to no experimentation was required (I reached the result almost immediately)		5.88%	1
4	A little experimentation was required		29.41%	5
3	Some experimentation was required		52.94%	9
2	A large amount of experimentation was required		5.88%	1
1	A very large amount of experimentation was required		5.88%	1

Figure 16: Stage 2 Question 8 - Distribution of Results



			Response Percent	Response Total
3	No		82.35%	14
2	Once		17.65%	3
1	Multiple times		0.00%	0

Figure 17: Stage 2 Question 9 - Distribution of Results

10.3.1 The music generated can express target emotions

As clip B outperformed all other clips in question 1 from stage 1, it can be suggested that the music generated for clip B expressed the emotions conveyed in the video clip better than the music generated for clips A, C, D and E. As the intention when generating the music for clip B was to match the emotions conveyed by the clip, it can be argued that this shows that music generated by NMGS can express target emotions. This is further backed up by the high mean answer for clip B and the low standard deviation, suggesting that, in general, users strongly believed that the emotions conveyed in the music match the emotions conveyed in the clip.

This is also highlighted by clip E underperforming, for which the results suggested that the emotions conveyed by the music did not match the emotions conveyed in the video clip. Given that this was the intention, it can be argued that the target emotions for the generated music were reached.

The comparison between question 4 in stage 2 and question 1 in stage 1 attempted to take this further by showing that the user was, themselves, able to use NMGS to generate music that they believe expressed target emotions. As the results showed that clip B outperformed clip F (the clip with the user's own music), it is not clear whether this is the case. This difference could be due to clip B being generated with the knowledge of how the system functions, and that users may be able to get closer to results that they desire with continued use of the system. One user stated "my composition didn't directly reflect the narrative of the movie mostly due to confusion regarding the timings of the sections", which could also be a possible explanation for the result, as this was brought up more than once.

Another comment on the emotions conveyed by the music was that "[the generated music] conveys sad and stressed emotions well[...] it didn't convey happy/ excited emotions well". However why they believed this was the case was not stated so it is difficult to analyse this further. Another

user stated that "[the] emotion selection chart could use more dimensions", however, as already discussed, this would likely add a great deal of complexity, as was acknowledged in the same comment.

Overall, while issues were pointed out, the results suggest the the music generated is able to express target emotions, meaning that this criteria has been met.

10.3.2 The music can reflect the narrative of a visual sequence.

Question 2 from stage 1 and question 5 from stage 2 were included to show that the music can follow the narrative of a visual sequence. However, as the narrative is characterised by the emotions it conveys, the questions are almost identical to question 1 from stage 1 and question 4 from stage 2. Due to this the results are also almost identical, and therefore similar conclusions can be drawn from them, suggesting that the music generated is able to reflect the narrative of a visual sequence and that this criteria has been met.

10.3.3 The transitions between target emotions are natural sounding

The aim here was to show that there were no differences between transitions in the different clips and that in general, users believed that the transitions sounded natural. However, this was not the case. Clip B performed better than some others, which could suggest that users believing that the emotions conveyed in the music matched the emotions in the clip may have affected their answers for this question. However, the small sample size could also mean that this is a coincidence, as, unlike results from questions 1 and 2 from stage 1, clip E did not perform the worst. The results aren't as sharply defined as in questions 1 and 2, and no strong conclusions can be made here.

The average mean for all clips, after calculation, is 3.58, showing that in general, people believed that transitions sounded more natural than unnatural, but not to the extent that was desired. As there was no significant difference between stage 2 question 6 and stage 1 question 3, the same conclusion can be made for the users generated music. However as 37.5% of users selected 'I'm not sure', it is possible that users did not notice they could use transitions as they are hidden when the currently selected section is the last section, which will continue to be the case if new sections are added without going back. Some comments seem to support this, with one user stating "[I] didn't notice I had any control over 'transitions' - missed that bit". This is frustrating as it makes it difficult to accurately judge the results. It would have made more sense to have separate choices for 'I'm not sure' and 'neither natural nor unnatural', to make the results clearer.

Overall, no significant conclusions can be made regarding how natural the transitions sounded.

10.3.4 The GUI is simple and the system is easy to learn

The answers for question 2 in stage 2 suggest that, in general, users believed that the GUI was simple to use/navigate. The comments provided for question 2 appear to support this, with comments such as "once you got used to the program it was easy to [...] manipulate the sounds/emotions the way you wanted them" and "[it is] very straightforward and easy to explore". From this, it can be argued that the system has been successful in having a GUI that is simple.

The answers for question 3 in stage 2 suggest that, in general, users believed that it was easy to learn how to use the system. Given that no instructions were provided and the learning was through exploration, this hypothesis is further supported. Most of the comments provided for question 3 also support this, however some comments suggest issues in this area. For example, one user stated

”[it was] hard to understand where to start the first time around. The help text isn’t very eye grabbing and I ignored it most of the time”. This will be considered in terms of improvements to be made, however, most of the results suggest that the system has been successful in being easy to learn.

As most people did not ask for help during the task, the success of these criteria are further highlighted. As everyone was able to complete the task without needing to ask questions, it can be suggested that no participants severely struggled to understand how to use the application.

10.3.5 Using the system requires no musical knowledge

As the mean answer for how pleased users were with the task was no higher for users who stated they had some or lots of musical knowledge than for those who stated they had little to no musical knowledge, it can be suggested that having musical knowledge did not benefit users in using the application to achieve a desired result. However, the number of users with either some or lots of musical knowledge were very low, meaning that this conclusion may be weak. Overall, as all users with little to no musical knowledge were able to complete the task and the mean for how pleased they were with their result was high (4.27), this criteria has arguably been met.

10.3.6 Minimal need for experimentation is needed to do achieve a desired result

The answers to how much experimentation users believed was needed to achieve a desired result appear to suggest that, in general less experimentation is needed than more. This was, however, not by a significant amount, and therefore no strong conclusions can be drawn from this. It could have been useful to have had users also generate music with other music generators and compare how much experimentation was needed, with the aim of showing that NMGS requires significantly less experimentation than other systems, but this would have been too time consuming to be practical.

10.3.7 Final criteria

In addition to the evaluation criteria, it should be possible for the user to:

- Generate a composition with multiple different sections
- Export their generated composition as MIDI file

These cover the last of the system’s sub-objectives outlined in the design chapter and the issues outlined in the motivation chapter. NMGS allows compositions to be created that have sections with different sets of input parameters, and generated compositions can be exported to a MIDI file, therefore these criteria have been met.

10.3.8 Improvements

Suggested improvements/features from the comments in the stage 2 questionnaire have been summarised below. These are based on direct suggestions, as well as the criticisms:

- Add a ‘visual timeline’ to help with working out timings for section/transition starts.
- Move target emotion selector window to main window to make it easier to adjust.

- Add a preview feature for valence and arousal values so that you know what it sounds like.
- Sync to video, to make timings of the sections less confusing.
- Make it more obvious that transitions are available without needing to go back.
- Make help text more eye grabbing.
- Notify users of minimum values (e.g. section duration being six seconds).
- Improve flexibility of transition times and section duration.
- Improve emotion selection chart (e.g. add more dimensions) as it can be restrictive.
- Add more variety to the melody.
- Add some control over the ending (e.g. ending in a root note).
- Add more instruments.
- Add a tempo setting so that section lengths aren't constrained to 6 seconds.
- Add more variety in the melody

Most of these improvements, while they would provide quality of life improvements, do not prevent the system from meeting its aims. However, for some improvements, such as reducing the confusion of timings of sections and transitions by adding a visual timeline and/or adding the ability to sync the music to video, this is not the case. For example, a user stated "my composition didn't directly reflect the narrative of the movie mostly due to confusion regarding the timings of the sections". This can be considered a serious flaw in a system that aims to be for visual media creators. Due to this, the 'visual representation of music that will be generated' and 'video integration' requirements should become 'must-have' requirements, as they would solve this. Other improvements should be added to the 'should-have' or 'could-have' requirements. An updated functional specification would provide a new backlog for the next version of the system.

11 Summary and Reflections

In this chapter, the project management, achievements and contributions of the project will be reflected on.

11.1 Project Management

A project plan, shown in Figure 18, was created before development. The development phase was split up into sprints and the content of the sprints were planned, making it easy to know where development should be week to week. The Christmas holiday and exam period were explicitly outlined, with no work being scheduled during this time. Despite this, work, albeit at a slower pace, continued. This provided a head start on the development of the project, and reduced the risk of the project falling behind schedule.

Each sprint contained a sprint planning/retrospective, a development phase, and testing phase. The development plan, outlining the development phase for each sprint, based on the 'must have'

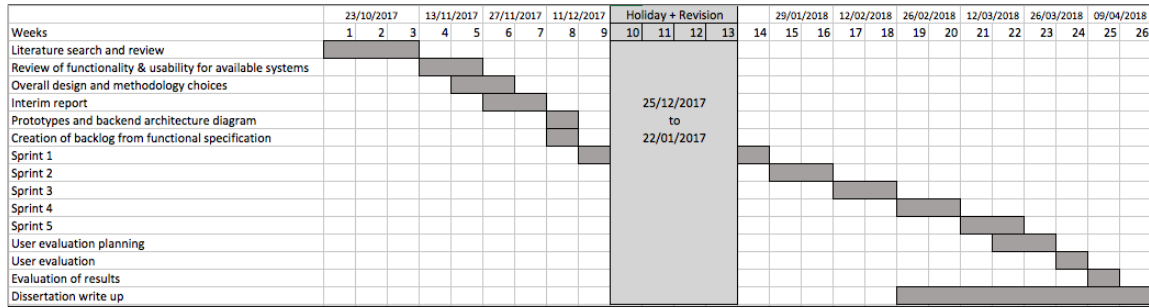


Figure 18: Project plan.

requirements in the functional specification, is shown in appendix H, with a simplified plan being showed below.

- Sprint 1: Efficient algorithmic music generator
- Sprint 2: Inputs for target emotion & Minor user-control of structure
- Sprint 3: Transitioning between target emotions
- Sprint 4: Simple GUI, Playback functionality & Exporting to a MIDI file

Changes were made to the plan throughout development. The main change is that development on transitions were originally planned for Sprint 4 and the GUI was planned for Sprint 3. These were swapped as it made more sense to build a GUI over a complete underlying system. The contents of a sprint also sometimes overflowed, for example, Sprint 1 was not completed on time, and backlog items had to be pulled into Sprint 2. Work over the Christmas break meant development didn't fall behind by a significant margin, and the setback was recovered from.

Sprint 5 was intentionally not planned. This was to provide space for setbacks in development to ensure the 'must-have' requirements were completed, and to potentially allow for some 'should-have' requirements to be included. However, when this sprint arrived, all 'must-have' requirements were met, and it was felt that not enough time was scheduled for the evaluation and write-up, so these were prioritised instead.

The changes that have been outlined above, and more, can be found in the sprint retrospective notes, in appendix I.

Overall, I feel that the project was well managed and the schedule provided enough padding to make it likely that all of the minimum requirements for the system would be met. While some changes were necessary, this is unavoidable and using an agile software development methodology allowed this to not become an issue.

A side note to make is that more time was spent working on the project in the second semester. This was to account for module choices that lead to the credits for the year being weighted 70/50 for the first and second semester respectively.

11.2 Contributions and Reflections

The project highlighted the gap in the market for a narrative based music generation system, and the uses of such a system, such as places where both linear and non-linear narratives exist. Research

on existing systems helped to create a functional specification to meet the system’s objectives, and allowed the MVP to be created. Evaluation of the system has showed that it can generate music that follows the narrative of a visual sequence and has solved some of the issues that hold back existing music generation systems from being used by visual media creators, arguably filling the gap that was highlighted. These issues could not all be shown to be successfully or unsuccessfully tackled, however, due to a lack of thoughtful planning regarding the user evaluation, and a low number of participants. Nevertheless, this evaluation did help to provide useful guidance in future development for the system. Some of the issues faced during development, such as tempo being controlled by the emotion causing several implementation issues, could help others designing similar systems to avoid these problems, and the research and implementation could also help others designing EMS systems for other purposes that were highlighted, such as for music therapy.

Overall, while mistakes were made along the way, the experience of the project has been a good one, and these mistakes are all part of the learning process. Going back, I would make several changes, creating clearer objectives, planning more before each area of the project, and having a less broad focus, which I believe would have solved most of issues faced.

References

- [1] A. Prechtel, “Adaptive music generation for computer games,” Ph.D. dissertation, The Open University, 2015.
- [2] I. Wallis, T. Ingalls, E. Campana, and J. Goodman, “A rule-based generative music system controlled by desired valence and arousal,” in *Proceedings of 8th International Sound and Music Computing Conference*, 2011.
- [3] A. J. Cohen, “Music as a source of emotion in film,” *Handbook of Music and Emotion: Theory, Research, Applications*, pp. 878–908, 1993.
- [4] A. Berndt and K. Hartmann, “The functions of music in interactive media,” in *Proceedings of the International Conference on Interactive Digital Storytelling*, 2008, pp. 126–131.
- [5] K. Collins, “An introduction to procedural music in video games,” *Contemporary Music Review*, vol. 28, no. 1, pp. 5–15, 2009.
- [6] M. Alexander, “Procedural music - a viable alternative?” Sep 2015. [Online]. Available: <https://www.yoyogames.com/blog/119/procedural-music-a-viable-alternative>
- [7] V. Velardo, “Adaptive music increases immersion and time spent in vr by more than 30%,” Feb 2018. [Online]. Available: <http://melodrive.com/blog/adaptive-music-increases-immersion-experiment/>
- [8] D. Herremans, C.-H. Chuan, and E. Chew, “A functional taxonomy of music generation systems,” *ACM Computing Surveys*, vol. 50, no. 5, pp. 1–30, 2017.
- [9] “Abundant music,” accessed on 02.12.17. [Online]. Available: <https://pernyblom.github.io/abundant-music/index.html>
- [10] “Wolfram tones,” accessed on 02.12.17. [Online]. Available: <http://tones.wolfram.com/>

- [11] “cgmusic - can computers create music?” accessed on 02.12.17. [Online]. Available: <http://codeminion.com/blogs/maciek/2008/05/cgmusic-computers-create-music/>
- [12] “Jukedeck,” accessed on 12.05.18. [Online]. Available: <https://www.jukedeck.com/>
- [13] A. Craddock, B. Fazackerley, S. Messenger, B. Roberts, and J. Stapleton, *DSDM Atern: the handbook*. DSDM Consortium, 2008.
- [14] A. Farrell, “Selecting a software development methodology based on organizational characteristics,” Master’s thesis, Athabasca University, August 2007.
- [15] M. Fowler and J. Highsmith, “The agile manifesto,” *Software Development*, vol. 9, no. 8, pp. 28–32, August 2001.
- [16] S. Technologies, “8 benefits of agile software development,” Jun 2016, accessed on 03.12.17. [Online]. Available: <https://www.seguetech.com/8-benefits-of-agile-software-development/>
- [17] D. Plans and D. Morelli, “Experience-driven procedural music generation for games,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 3, pp. 192–198, 2012.
- [18] R. Half, “4 advantages of object-oriented programming,” Nov 2017, accessed on 03.12.17. [Online]. Available: <https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming>
- [19] A. R. Brown, *Making music with Java: an introduction to computer music, Java programming, and the jMusic library*. Lulu, 2005.
- [20] S. Revisions, “Why you should use git,” May 2017, accessed on 03.12.17. [Online]. Available: <https://www.webpagefx.com/blog/web-design/why-you-should-use-git/>
- [21] G. Nierhaus, *Algorithmic composition: paradigms of automated music generation*. Springer, 2010.
- [22] S. R. Livingstone, R. Muhlberger, A. R. Brown, and W. F. Thompson, “Changing musical emotion: A computational rule system for modifying score and performance,” *Computer Music Journal*, vol. 34, no. 1, pp. 41–64, 2010.
- [23] A. Gabrielsson and E. Lindstrom, “The influence of musical structure on emotional expression,” in *Music and emotion: theory and research*, P. N. Juslin and J. A. Sloboda, Eds. Oxford Univ. Press, 2001, pp. 223–248.
- [24] J. A. Russell, “A circumplex model of affect,” *Journal of Personality and Social Psychology*, vol. 39, no. 6, pp. 1161–1178, Dec 1980.
- [25] M. Zentner, D. Grandjean, and K. R. Scherer, “Emotions evoked by the sound of music: Characterization, classification, and measurement,” *Emotion*, vol. 8, no. 4, pp. 494–521, 2008.
- [26] R. Wooler and A. Brown, “Investigating morphing algorithms for generative music,” in *Proceedings of the International Conference on Generative Systems in the Electronic Arts*, Dec 2005.

- [27] G. D. Webster and C. G. Weir, “Emotional responses to music: Interactive effects of mode, texture, and tempo,” *Motivation and Emotion*, vol. 29, no. 1, pp. 19–39, Mar 2005.
- [28] P. M. Watt, “The purpose of a unit test,” Feb 2014. [Online]. Available: <https://www.codeproject.com/Articles/727053/The-Purpose-of-a-Unit-Test>
- [29] S. Hill, “The pros and cons of test-driven development,” Feb 2015. [Online]. Available: <https://leantesting.com/test-driven-development/>
- [30] M. Jackson, S. Crouch, and R. Baxter, “Software evaluation: Criteria-based assessment,” in *Software Sustainability Institute*, Nov 2011.
- [31] T. P. Ltd., “Salt and pepper - romantic 1 minute stop motion animation,” Mar 2015, accessed on 10.05.18. [Online]. Available: <https://www.youtube.com/watch?v=jRUE4imKYSo>
- [32] D. S. Yates, D. S. Moore, and G. P. McCabe, *The Practice Of Statistics*. W.H Freeman, 1999.

A Glossary of Musical Terms

- Aeolian: The ‘natural minor’ mode of the major scale (A B C D E F G in the key of C)
- Bar: A section of the music that has a number of beats in it (4 in 4/4 time). Each bar in a piece of music often has the same number of beats in it
- Beat: The basic unit of time (what listeners will often tap to when listening to music)
- Chord: A set of two or more notes that are sounded simultaneously
- Composition: An original piece of music
- Diatonic: The use of notes within a scale without chromatic alteration (the use of notes outside of the scale)
- Fifth: A pitch 7 semitones higher than the root of a chord
- Ionian: The ‘major’ mode of the major scale (C D E F G A B in the key of C).
- Key: The scale that a piece of music revolves around. For example, a key of C revolves around the C major scale (C D E F G A B).
- Mode: An scale derived from the major scale, by starting on a different pitch, changing the role of each element of the scale
- Note: The pitch and duration of a sound
- Octave: The jump, or ‘interval’ between a pitch and another with half or double it’s frequency; moving 12 semitones down or up is the equivalent of moving by an octave
- Quarter Note: A note played for the quarter of the duration of a whole note. It is synonymous with a beat in 4/4 time

- Root: The pitch that is used as the name to represent a chord, for example, the root of chord C major is C
- Scale: A group of pitches arranged in ascending order
- Semitone: The smallest musical jump, or 'interval' between pitches.
- Tempo: The speed of the music, often measure in beats per minute (BPM)
- Third: A pitch either 3 or 4 semitones higher than the root for minor and major chords respectively
- Whole Note: A note equal to 4 beats, and taking up a full bar, in 4/4 time
- 4/4 Time: Used to denote that a bar should contain 4 beats each with the length of a quarter note