Addressing the Gap Between Machine Learning and Casual Music Creation

CS39440 Major Project Report

Author: Daniel Swift (das82@aber.ac.uk)

Supervisor: Dr Chuan Lu (cul@aber.ac.uk)

04th May 2022

Version 1.86 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science (G401)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name Daniel Swift

Date 28/04/2022

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Daniel Swift

Date 28/04/2022

Acknowledgements

I am grateful to…

I’d like to thank…

Abstract

Music is one of the most expressive and emotive forms of human art. Deeply ingrained in the roots of different cultures and communities, music is widely created and enjoyed in various forms throughout the world. The rise of personal computers and innovations such as the Musical Instrument Digital Interface (MIDI) standard has born a new era of digital music creation, capitalised on from the biggest artists in the world perfecting their sound, to curious users experimenting with music scales in their bedroom.

Coinciding with this is the rise in machine learning, which is also becoming more accessible to a wider, non-specialised audience. However, as of this report, there are still many areas of machine learning that are too complicated or expensive for the more casual user. My project has tried to address this problem, allowing people who may have some background in digital music creation but none with machine learning or AI in general, and allow them to combine these relatively new fields as seamlessly as possible.

In this project I will compare different generative algorithms regarding their suitability for music generation. Furthermore, I will attempt to create software that is user-friendly and intuitive whilst enabling users to generate music from different complex music generation algorithms.

Contents

1. Background, Analysis & Process 7

1.1. Background 7

1.1.1. Initial Motivations 7

1.1.2. Music and Sound Research 7

1.1.3. MIDI Standard and Useful Filetypes 8

1.1.4. Neural Network Models 8

1.1.5. Similar Existing Solutions 9

1.2. Relevant Tools and Libraries 10

1.2.1. Machine Learning Libraries 10

1.2.2. Development Environments and Package Managers 11

1.2.3. GUI Libraries 11

1.3. Analysis 12

1.3.1. Gather Data 12

1.3.2. Perform Tests with Provided LSTM Model Examples/Tutorials and Train with Custom Configurations and Architectures 12

1.3.3. Analyse the Magenta Pre-Trained GANs and Attempt to Train a Custom Model 13

1.3.4. Develop a Program based off the Previous Experiments 13

1.3.5. Potential Security Issues 13

1.4. Process 13

2. Machine Learning for Music Generation 15

2.1. Introduction 15

2.2. Data 15

2.3. RNNs/LSTMs 15

2.4. GANs 15

3. Design 16

4. Implementation Process 17

4.1. Sprint 1: 16th March – 19th March 17

4.1.1. Version Control 17

4.1.2. Software Setup and Package Management 17

4.1.3. Data Collection and Initial Colab Demos 17

4.2. Sprint 2: 4th April – 8th April (Mention Problems, Difficulties and Delays) 18

4.2.1. Plan 18

4.2.2. Design 18

4.2.3. Implementation 20

4.2.4. Testing 21

4.2.5. Review 21

4.3. Sprint 3: 21

4.3.1. Plan and Design 21

4.3.2. Implementation 21

4.3.3. Testing 21

4.3.4. Review 21

5. Testing 22

5.1. Overall Approach to Testing 22

5.2. Automated Testing 22

5.2.1. Unit Tests 22

5.2.2. User Interface Testing 22

5.2.3. Stress Testing 22

5.2.4. Other Types of Testing 23

5.3. Integration Testing 23

5.4. User Testing 23

6. Critical Evaluation 24

7. References 25

8. Appendices 27

A. Third-Party Code and Libraries 28

B. Code Samples 29

# Background, Analysis & Process

## Background

### Initial Motivations

My initial interest in pursuing this area for my project stemmed firstly from the fact that I love both listening and playing guitar music. I started playing guitar around five years ago, and have accrued some knowledge in how to write my own music and some of the theory behind it. Furthermore, I believe that in those five years, the way that I perceive and understand music that I listen to has changed drastically. Being able to play a musical instrument has made me appreciate skill and creativity a lot more, especially guitar music.

This is no coincidence, as musicians with different expertise selectively engage different sensorimotor regions in the brain when listening to the instruments they are able to play [1]. However, this is not to say that people with some kind of musical expertise are better at perceiving music than non-musicians, obviously non-musicians are able engage, understand, and build powerful connections with music they listen to. This is shaped by numerous factors, such as mood, setting, the piece itself[2], or the person’s familiarity with the piece and genre[3].

In terms of familiarity with machine learning, I spent my year in industry working in the marketing department for a company producing machine learning and AI related products, with emphasis on these products being relatively easy to program and deploy within businesses and schools. The focus on making AI and machine learning more user-accessible and widely accepted was another source of inspiration for me to partake in this specific project. I hoped that it might be an insightful way to get more people into both writing music and machine learning, as well as being a genuinely useful tool for basic music composition.

### Music and Sound Research

My initial research and reading predominantly regarded the different kind of neural network models that could be used for this project, the libraries that supported them, the languages or programming interfaces that I could use, and similar existing solutions that I could draw inspiration from.

Music itself is composed of many different elements, which combine to create the various melodies that we can perceive in a variety of ways. It can be conveniently described as a series of notes, which play out in a linear pattern from the beginning of a song until the end. These notes can be identified by their pitch (the frequency of the soundwave they produce), duration, velocity, volume, and many others. Different notes can be played at the same time to make up chords, and can be played in conjunction with other specific notes to make up scales.

It is important to understand what makes music sound “good”. Sure, pressing random keys on a piano or plucking at random strings on a guitar is probably not going to sound like a masterpiece, but in order to play something harmonious, notes need to stay in the same key, and make sure they are played within defined scale intervals.

Leading on, it could be easy enough to develop an algorithm where different scales are hardcoded, and it would select notes at random (or in a loosely determined pattern based on other inputs) in order to generate a piece that, while potentially sounding quite weird, would be in key. This would, in turn, reduce the sense of originality, as the piece would be the product of either over-determinism or pure randomness, where in reality we would require more of a balance.

This would justify a need for machine learning, as, for instance, a neural network model could be trained from music that is generally well regarded, and would likely follow different scale intervals and remain in key. The model would pick up on this so that when it generates its own music, the notes should all remain in key. However, remaining in key is not always the most important factor. It is worth acknowledging that notes can sound good sometimes when played out of key, and this should be reflected in the songs within the provided training datasets. Furthermore, obviously the concept of “good” music is subjective, and different individuals will form an impression based on the reasons aforementioned in 1.1.1.

### MIDI Standard and Useful Filetypes

MIDI is a technological standard that evolved to enable communication between digitally enabled instruments, such as 1980’s synthesisers. Later, it provided a platform for users with personal computers to record, store and edit music[9]. MIDI files themselves are normally relatively small, meaning that they do not require much disk space, due to the fact that they do not have any actual audio data. In place of this, they contain data regarding what notes are to be played, the time that they are played at, the duration of the note, and how loud it should be[10]. Due to this, many musical pieces that are converted to MIDI files will sound different to the original recording, and sometimes unpleasant.

### Neural Network Models

In terms of selecting a specific type of generation algorithm, there were a few options. Most early music generation techniques used Recurrent Neural Networks (RNN), such as Long Short-Term Memory (LSTM) networks, which would generate single-instrument music in the same way that natural language models work[4][5][6].

A big benefit of using RNNs is that it has the ability to apply contextual information when mapping between input and output sequences, however the drawback is that the range of this context is limited. To explain this further, within RNN architectures, the influence of a given input on the hidden layer, and then in consequence the output, either decays or blows up exponentially as it cycles through the network’s recurrent connections[7]. This is more commonly known as the vanishing gradient problem[8].

LSTMs have been developed to address these limitations, and have proven more successful at it[12]. Within the architecture of an LSTM is a constant error carrousel (CEC), which basically affords protection to input gates by perturbing irrelevant inputs [13].

Another potential model could be a Generative Adversarial Network (GAN). GANs work using unsupervised learning, however it frames itself as a supervised learning algorithm, using two sub-models, a generator model, and a discriminator model. The generator model is trained to generate new data from an existing dataset, and the discriminator model attempts to classify data that is real (from the given dataset), or generated by the generator model. The two models work at the same time until the discriminator fails to correctly determine the difference between real and generated data at a pre-determined rate[19], the original proposed rate being 50%[20].

### Similar Existing Solutions

#### SOUNDRAW

A screenshot of a computer

Description automatically generated with medium confidence“Soundraw” is a web-based music generator that uses pre-trained AI to generate songs at a selection of lengths, based on a user-given input of the song’s mood, genre(or style), tempo and instrument. The generated music can also be edited to cut its length, slow or quicken its beats per minute (bpm), change the key, and change the volume, amongst other advanced attributes at specific timings.

Figure 1: The GUI for soundraw.io[22], with its extensive plethora of features

The design of this app is very professional and fairly straightforward, and its use of well-developed pre-trained models is a good source of inspiration for how this project could work. However, this solution offers no way for users to train their own models, or select the different kind of generation algorithm they would like to use.

#### dopeloop.ai

The “dopeloop” melody generator is a web-based application that is able to generate melodies from three user-given inputs. The first is how many notes to generate (the length), the second the root note, and the third is the scale to generate the notes in. The notes are generated randomly within these given confinements. The user also has the option to change the instrument and the bpm. This application is also available as a mobile app.

Graphical user interface, application

Description automatically generated

Figure 2: The GUI for dopeloop.ai's[21] melody generator

Whilst the way that music is generated for this solution is not the same way that it will be generated in this project, the simplicity of the GUI design and the swiftness that it operates provides a large source of inspiration for the practical outcome of this project, as well as advocating for a web-based solution.

## Relevant Tools and Libraries

### Machine Learning Libraries

#### Keras/TensorFlow

TensorFlow is an open-source library built specifically to allow developers to create and train machine learning models. Created by Google, TensorFlow is predominantly used with Python and JavaScript, and it is one of the most widely used libraries in relation to machine learning[11]. Keras is an extension of TensorFlow which tries to make using it a little more user friendly.

#### PyTorch

Similarly to TensorFlow, PyTorch is an open-source library, but focused on accelerated deep neural network programming and developed by Facebook[14]. However, when compared to TensorFlow, it would probably fall a little short for this project. This is because TensorFlow has a superior and well-documented framework, more of an abundance of tutorials, and better visualisation of its processes, whereas PyTorch provides only limited visualisation. On the other hand, PyTorch gains optimal performance with native asynchronous execution within Python, whereas for TensorFlow this would have to be manually coded in[15].

#### Scikit-Learn

Scikit-Learn offers a wide range of machine learning algorithms, with alternatives to neural networks. Decision trees, Support Vector Machines (SVM), and linear regression, to name a few, are all available to use through this library.

#### Magenta

Magenta is a self-described open-source research project, created by Google using the TensorFlow library. It is specifically designed to explore the role of machine learning when undergoing the process of creating art and music, and is available for both Python and JavaScript[16]. Magenta can be used to train custom models, but also comes with a selection of pretrained models, of which can be changed certain parameters to affect the final output of the model[17].

### Development Environments and Package Managers

#### Anaconda

Anaconda is an open-source Python distribution platform, with a focus on making data science and machine learning easily accessible on one machine. It provides a user-friendly graphical user interface (GUI) which makes it easy to switch between different development environments, and features its own repository of thousands of open-source data science and machine learning related packages. This includes previously mentioned packages such as TensorFlow, PyTorch and Keras[18].

#### Google Colab

Google Colab is a Jupyter notebook environment that provides a free web-based python environment built specifically for research and data science. Whilst being able to run arbitrary Python code step-by-step in different coding chunks, the best feature of Colab is that it provides access to computing resources including RAM and GPUs for free, which allows users to perform large processes with ease that everyday PCs may struggle with, such as training a large complex model with a large dataset that requires strenuous pre-processing. Furthermore, Colab has plenty of built-in features that drastically help to improve the visualisation of data, such as being able to play midi files and plot graphs in-browser.

### GUI Libraries

#### PyQT5

PyQT5 is a GUI framework for Python written in C++, which is free and open source. Unlike some other GUI libraries for Python, it offers a visual framework that allows for developing GUIs via drag and drop of a wide selection of widgets to build these interfaces, obviously leaving the coding for the backend[23].

#### Tkinter

Tkinter is a built-in library for GUI development in Python. Alternatively to PyQT5, all of the GUI widgets and canvases are programmed manually, without the aid of a drag and drop visual framework. This may seem like it is trickier to learn, however I personally have had a lot of experience using it in the past. Furthermore, it is normally packaged in with the core Python libraries, so there is no need to externally install it[24].

## Analysis

Based on background research, a basic structure for the preliminary approach needed for this product was devised. The approach was decided as follows:

### Gather Data

Based on looking at numerous other models, it was clear that an abundance of data would need to be collected for training and validation purposes, and to increase the chance of the models producing music of a good standard. The data would need to be in the form of MIDI files, and sourced online. The option of creating my own MIDI files from my electric guitar was considered, however this would require specialist equipment, which was too expensive. Fortunately, there is a plethora of websites where midi files are freely available, so it would have been relatively easy to look through these websites and download as many as was needed.

Graphical user interface

Description automatically generated

Figure 3: A selection of thousands of available MIDI files from midiworld.com

### Perform Tests with Provided LSTM Model Examples/Tutorials and Train with Custom Configurations and Architectures

Initially, it was decided that the best libraries to work with for this project would be TensorFlow (and by extension Keras), and Magenta. This is because they are both very well-documented, have extensive, in-depth tutorials, and are both open source, meaning that it is easier to visualise how they are structured.

Consequently, this did mean that the majority of the development of this project would have to be conducted using Python, due to the fact that most of the provided tutorials and documentation use Python code (although Magenta is also compatible with JavaScript). On one hand, this removed the advantage of highly accessible and functional user interface that a web-based project would have, although the GUI libraries provided in Python are also well-documented, and personally I have experience with GUI development so it should not be as steep of a learning curve than learning JavaScript.

Furthermore, this opened up the strong advantage of being able to use Google Colab, as it was quickly realised that experimenting with machine learning would require some pretty powerful hardware, which is something that was not in my possession. It may have been possible to do something like log into a remote server and try to develop from that, but it was decided that this was unfeasible and unnecessary.

Using the example code provided by TensorFlow, a model would be trained, and the results analysed within Google Colab. The initial inputs and other configurations would then be altered and changed in order to get a better idea of what would cause the model to produce better outputs. Based off this, another model would be trained with the data gathered in the previous step, in order to evaluate the effectiveness of the model and the data that had been collected.

### Analyse the Magenta Pre-Trained GANs and Attempt to Train a Custom Model

Similarly to the step before, the examples given by Magenta would be analysed, and the results of the given pre-trained models would be tested. Based off background research of the pretrained models, at a glance they are very effective at generating good music. Next, Magenta would be used to create and train a custom model, in order to see if it would be more or less effective than the previously experimented-with LSTM models.

### Develop a Program based off the Previous Experiments

Initially the focus would be on setting up the necessary software required, including some of the relevant tools and libraries mentioned in section 1.2. Since the models were created in Python, it made sense for compatibility reasons that the GUI should be written in Python too. It was decided that Tkinter would be used, for two reasons. The first was that as mentioned before in section 1.2.3, it comes packaged in with the Python’s standard built-in libraries.

Secondly, as I have some previous experience working with Tkinter, it would save time and resources to just use it again, as opposed to having to learn how to use a new library. PyQT5 did appear to be easy to use and learn, and if it were the case that I had to learn both libraries from scratch, then it most likely would have been selected instead. However, despite the moderately higher complexity of Tkinter, it is able to do pretty much everything that PyQT5 is able to provide.

The GUI should be able to load and play MIDI files, and have options to generate new music using a selected model, and furthermore have to ability to generate some kind of music tablature, such as a guitar tab, from the given MIDI file. There should be further options to alter the configuration of the models so that the sounds produced can be altered accordingly.

### Potential Security Issues

There did not appear to be any relevant potential security issues or threats regarding this project. There is no handling or storing of sensitive data, and does not require the use of human or animal test subjects. A lot of the libraries used for this program are open source, which could leave them vulnerable to bugs and attacks, but most of them are now well-documented and continuously updated which reduces these risks to a minimum.

## Process

This project is a mixture of research and software engineering. Since a large part is taken up by research, the scale of the development side is reduced, and greatly affected by the results of the research. Therefore, the process of developing the GUI should reflect this. A methodology was needed that would allow for flexibility with the design and work through an iterative process. It was decided that an Agile approach would be used, with moderate elements of Scrum incorporated. This would work by having multiple sprints (as is common in Scrum[25]), and in each sprint plan, design, implement and test. Since the scope of this side of the project is relatively smaller, it was felt that it was achievable to develop it in this way.

# Machine Learning for Music Generation

## Introduction

The concept of first using computers to generate music was proposed in writing by Ada Lovelace in 1843, where she wrote:

*“Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent….”*[26]

However, much of the progress in the field of machine learning, such as deep learning, has only really been made in the last 25 years, and only in the past decade has there been a wider availability for using it outside of research purposes, and there continues to be limitations. As *L Casini et.al* argue, while models are able to produce pieces that are good enough to be considered music to the casual listener, there still remains a vital human element, especially when generating longer pieces, which generally lack structure and meaning to more musically inclined listeners[27]. Furthermore, deep learning structures do not have direct ways to control how they generate results, and generate music without human interaction[28].

## Data (unfinished)

The rise in machine learning has undoubtedly coincided with the rise in data gathering. Algorithms used within the field of machine learning, specifically deep learning, continue to become larger and more complex, requiring more and more data to perfect their results[28]. Throughout the process of experimenting with the different models and training data that have been used for this project, a consistent issue was Google Colab timing out after or during training the models. Obviously, this is necessary in order for Google to continue to provide the service for free, by freeing up space and sharing resources across everyone who wants to use it, but, consequently, it was difficult to train with larger datasets and complex models since it could not be left for more than a few hours at most.

## RNNs/LSTMs (unfinished)

Due to the poor results from the simple models I was using, I instead tried using a different approach. The new model would have multiple LSTM layers, and would take advantage of using the ADL Piano MIDI dataset, specifically the Folk genre. My hope was that using data that was in the same genre would generate more cohesion and structural integrity within the produced music.

The new approach that was tried was based off a third-party module, “Classical Piano Composer” by Sigurour Sigurgeirsson, which I mainly used to help with the data pre-processing and the general structure of the model, although adjustments were made to both of these attributes. This code was also used to generate techno music, in testament to the wide-ranging possible implications of his module[30].

## GANs(unfinished)

# Design

You should concentrate on the more important aspects of the design. It is essential that an overview is presented before going into detail. As well as describing the design adopted it must also explain what other designs were considered and why they were rejected

The design should describe what you expected to do and might also explain areas that you had to revise after some investigation.

Typically, for an object-oriented design, the discussion will focus on the choice of objects and classes and the allocation of methods to classes. The use made of reusable components should be described and their source referenced. Particularly important decisions concerning data structures usually affect the architecture of a system and so should be described here.

How much material you include on detailed design and implementation will depend very much on the nature of the project. It should not be padded out. Think about the significant aspects of your system. For example, describe the design of the user interface if it is a critical aspect of your system, or provide detail about methods and data structures that are not trivial. Do not spend time on long lists of trivial items and repetitive descriptions. If in doubt about what is appropriate, speak to your supervisor.

You should also identify any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.

Some example sub-sections may be as follows, but the specific sections are for you to define.

# Implementation Process

As per the chosen methodology for this project, the implementation of the GUI was done in three different sprints, with the second and third having their own planning, implementation, and testing stage. However, the first sprint was purely just to get things set up so did not need much discussion.

## Sprint 1: 16th March – 19th March

As previously stated, the first sprint was about setting up some basic tools and utilities in order to give myself a solid foundation to continue with the rest of the project. I also went about collecting data to use for training.

### Version Control

The project started off by the setting up the chosen form of version control, Git, initially GitLab, of which a specialised account is provided by the university. This had numerous advantages, mainly that it provides a place to store backups for every part of the project, including the Google Colab notebooks, the datasets I collected, the Python files I would work with, and the reports necessary for this project. Another is that it provides a useful timeline of the project’s history when it comes to reviewing it. It is worth mentioning however, that I had to move to GitHub, as I could only access GitLab when I was connected to the university network, so when I moved home for Easter I was unable to gain access.

### Software Setup and Package Management

From the background research, it had been found that Anaconda was a useful tool for package management, and through Anaconda it would be easy to install packages such as TensorFlow easily, rather than having to do it manually or through pip, which could have been confusing or arduous. Anaconda comes with a Python IDE installed called Spyder, which was useful as it had syntax highlighting and debugging properties. I also installed IDLE, which would load quicker on my PC and was useful if there needed to be small quick changes to the code.

### Data Collection and Initial Colab Demos

I also looked for suitable datasets to use throughout for this project to train the models I would create. As mentioned in section 1.3.1, there were plenty of available resources that could be used for this project. I initially collected around 100 songs that I could use to train the models, because the models were not planned to be too complex, and the way that the data pre-processing worked meant that the songs would be split into the notes that form them, which would expand the dataset widely.

The initial expectations that the simple models and relatively smaller datasets were ultimately proven false however, after the moderately poor results from the trials using this data (see section 2.3 regarding the performance of LSTMs). Therefore, I had to search for much larger quantities of datasets, such as the Maestro dataset provided by Google and the ADL Piano MIDI dataset. I decided to use the data I collected as data to use for testing purposes.

Despite the poor results produced by the initial models used in the Google Colab demos, the fact that the models were able to produce anything at all, and the data pre-processing worked with all the different datasets I used were a big bonus, and put me in good stead for future development.

## Sprint 2: 4th April – 8th April (Mention Problems, Difficulties and Delays)

### Plan and Objectives

Firstly, an initial design for how the GUI should look on paper was drawn up, then there was an attempt to replicate this through Python using the Tkinter library. The initial hand-drawn design was the first source of inspiration for the interface design for this project.

<Image>

As can be seen from the provided figure, the GUI is relatively simple, with a menu bar at the top, a display for visualisation purposes and some buttons for playing the MIDI files and to increase or decrease the volume. Furthermore, there are buttons that will allow the user to upload MIDI files to be played or experimented on, along with a button to generate sheet music for the uploaded MIDI file. The final major component of this design was a button that would use an uploaded MIDI file to generate music from a pretrained model, or allow the user to train their own model. This feature would be developed in a later feature to allow for the research to continue on Google Colab.

### Design

Since at this stage all that was being focused on was the GUI, it was decided against using an object-orientated approach (in hindsight this was a mistake, but not the worst-case scenario), and if it needed to be put into a class then it would not be too difficult to do so. The majority of the tasks performed via the GUI in this project would inherently be event-driven, such as the clicks of buttons. So, after the GUI was designed, I would attach each aspect to a different function. It was decided that the official name of the program would be “MidiGen”.

#### Window

The main window would be the main area for the application, as a Tk() object, called “window”. The window was given the title: “MidiGen by das82” and the default size when it was created was set to 700x500 pixels, as the application was only small so did not need to take up an entire screen.

#### Title, File, Volume and Button Labels

The labels for each aspect of the design were all Tkinter Label objects, and were named accordingly:

* title\_label: A big header displayed at the top of the window (underneath the menu-bar) . This label had a larger font as it was a header for the title.
* filename\_label: A label to display the name of an uploaded MIDI file, by default outputting the text “No song selected!” as when the application is opened there would not be any MIDI files loaded yet.
* volume\_label: A label used to display the current volume of the application, by default set to 0.5 which is half the maximum volume.

#### Buttons

The buttons were all Tkinter Button objects, and like the labels are named according to their function:

* pause\_b: Button to pause the current song being played.
* play\_b: Button to play the current song, if pressed twice it should restart the song.
* volume\_up\_b: Button to increment the volume by 0.1.
* volume\_down\_b: Button to decrease the volume by 0.1.
* upload\_b: Button to upload a new MIDI file that can be played and used by the program.
* generate\_b: Button to open a new window enabling the user to interact with the generation part of the program.
* sheet\_b: Button to generate sheet music from the given MIDI file.

#### Other Widgets

The menu-bar at the top of the window was a Tkinter Menu object, called “menubar”. The menu-bar had different cascading submenus:

* file\_menu: This menu had options to upload new MIDI files, to save MIDI files, and exit the program.
* tools\_menu: This menu was implemented but not fleshed out at this stage.
* view\_menu: This menu was implemented but not fleshed out at this stage.
* help\_menu: This menu was implemented but not fleshed out at this stage.

There was also a separator widget, which was a Tkinter Separator object, to provide a split down the middle of the window just to make the appearance a little nicer for the user.

#### Placement

The widgets on the window were placed using relative scaling, so that their size and/or position would change in real time according to the size of the root window, which could be dragged and changed by the user.

#### Functions

There were numerous event-driven functions that needed to be implemented, which were usually called whenever a button was clicked:

* save\_midi(midi): Function to save a loaded or generated midi file to the local storage area.
* upload\_file(): Function to upload a midi file to the program (which ended up in turn calling the “load\_music” function.
* load\_music(): Function to load a song into a Pygame mixer (See section 4.2.3).
* generate\_music(midi): Function to generate music from a given MIDI file, at this stage does not do anything.
* get\_sheet\_music(midi\_file): Function to generate guitar tab from a given MIDI file, using the “tayuya” library (See section 4.2.3).
* pause\_music(): Function to pause a current song.
* play\_music(): Function to play a current song, and also to replay it if called while a song is already playing.
* volume\_down(): Function to reduce the volume by 0.1.
* volume\_up(): Function to increase the volume by 0.1.

Graphical user interface

Description automatically generated

Figure 4: The final look of the GUI after the second sprint

### Implementation

Firstly, the outline of the GUI was created, and the major widgets were implemented in their basic forms. Following this, the placement of the widgets became the priority, as my original method of using Tkinter methods such as “grid()” and “pack()” were simply not cut out for specific placements of multiple widgets and changing window sizes. I therefore had to switch to using the “place()” methods, which had use relative scaling, meaning that the widgets would scale correctly with the size of the window. The downside was that if the window got too small that they overlapped, and if it got too big then some of them looked slightly off-centre, but other than those small problems it worked pretty well.

Focus then turned to the functions. I realised that I would probably need to use some third-party libraries to help with these, such as generating sheet music from MIDI files and getting them to play. I used two separate libraries for each:

#### Pygame

Pygame is a collection of modules for Python that were originally designed for programming video games, and can be installed via Anaconda or with pip[29]. The module within Pygame that was useful for this project is the “mixer” module, which is able to play MIDI sound files. This was therefore vital for the music-playing part of the project.

#### Tayuya

This Python library assists in the conversion of MIDI files to guitar tablature, which was useful for this project for generating the written music from generated MIDI files.

#### Problems

I encountered a few problems during the development of the GUI. Firstly, the Spyder IDE proved to be temperamental at best, however fortunately for what I was doing in this sprint I was able to switch to IDLE and use pip to install the necessary libraries. As mentioned, I had some general problems with the placement of widgets, but I managed to mostly fix these, although labels and buttons do overlap if the window is too small. I was unable to fully implement the “save\_midi” function, which allows the user to save MIDI files back to the local file area. This would be something to look at again in the third sprint. Furthermore, the rewind function does not work, but I believe this was a Pygame issue so was unlikely to be fixed.

#### Global Variables

I had to use some global variables throughout the application, which could have been eliminated had I used an OOP approach. These were the volume and the loaded midi file (and its name).

### Testing (unfinished)

### Review

In summary for this sprint, most of the objectives have been achieved to an acceptable standard. There were some problems as mentioned in section 4.2.3, but these were not major problems that affected the core functionality of the program, so this can be considered a successful sprint.

## Sprint 3: 29th April – 4th May

### Plan and Objectives

It was decided to use and modify part of a third-party library to help bridge the gap between my initial GUI program and the backend functionality of the generative model. This was the library regarding LSTM generation used in Section 2.3, however a different section, which allows for fitting models with pre-trained weights without the need to do the heavy processing of training a model from scratch. However, users should be able to fit their own pre-trained weights rather than the one that was generated by me if they want, it would just have to match the architecture of the model.

The plan, however, was to alter this module so that it could be turned into a class, with some extra methods and a helper class. There were a few reasons for this; firstly so that it was more structured than the original, which was a just a collection of functions. Secondly, it would allow for easier adjustments to the specific variables contained within the methods, as these could be assigned when the object is initialised, and then assigned accordingly. This removes the need for global variables, making the project cleaner as a whole. Thirdly, converting the function into class methods makes it easier for me or other people to make changes to the original code, model architecture, class attributes etc more easily, and it can be imported and used for other projects in the future.

In terms of the GUI, I needed to create a window for the music generation part of the program, and connect it to the aforementioned class so that the user could enter the values needed to generate each new song. There should be checks to make sure the values the user enters are reasonable, and that the generated midi is loaded back to the main player so that the user can listen to it back.

### Design

#### Generator Class

The Generator class is the class that combines all of the original functions into one callable object. It also has an extra method that was added to try and implement a custom input during the generation process. The Generator class therefore contains the following methods:

* generate(): A method initially from the library that uses the other methods in the class to generate the new MIDI, however it was altered so that it would return the file-path of the new generated midi that could be used by the GUI.
* prepare\_sequences(notes, pitchnames, n\_vocab): A method initially from the library that prepares the sequences to be run through my pre-trained model, which was mostly unaltered from the original.
* create\_network(network\_input, n\_vocab): A method initially from the library that builds the model architecture and loads the weights. This was modified so that the architecture would match the weights I produced and was slightly lighter so that it would not take so long to train.
* generate\_notes(model, network\_input, pitch\_names, n\_vocab): A method initially from the library that, as the name suggests, generates the notes for the new MIDI file. Part of this method was modified so that the starting sequence of notes could be user input.
* create\_midi(prediction\_output): A method originally from the library that converts the generated notes back to a MIDI file and saves it back to the local drive, although it was altered to also return the filename for use in the main GUI.
* midi\_to\_notes(midi): A method that I created and appended to the class based off another part of the library, which would convert a given MIDI file input by the user, and use it as the starting sequence of the generative model to try and vary the output.
* get\_midi\_info(): This is a small method I wrote that might help for testing purposes, which makes use of the MidiInfo class to generate information about each MIDI file provided.

#### MidiInfo Class

This is a small class written by me to provide information about given MIDI files. This class is used by the Generator Class. The MidiInfo class contains the following methods:

* get\_file\_info(): This method returns the information about a given MIDI file, output to the Python shell.
* get\_track\_messages(track): This method returns information about a specific track on a MIDI file, specifically the messages it contains, indicated by an input track number. The output is to the Python shell.
* Get\_num\_messages(track): This method returns the number of messages within a given track on the MIDI.

#### GUI Generation Window

The generation window is a small popup window that allows the user to input specific options for the generator. The title of the window is “Generation Settings” and its default initial size is 300x120 pixels. As there are three settings that the user can change, the window consists of three entry boxes, all labelled, and a button to confirm the values. All of the widgets featured in this window, as with the root window, are positioned relative to the size of the window.

#### Entry Labels

* temp\_label: Label to indicate the entry box for where the temperature can be altered.
* gen\_length\_label: Label to indicate the entry box where the user can enter the number of notes they wish to generate.
* seq\_length\_label: Label to a indicate the entry box where the user can alter the length of the sequences fed into the model to try and alter the output.

#### Entry Boxes

* temp\_entry: The entry box for the temperature.
* gen\_length\_entry: The entry box for the number of notes the user wants to generate.
* seq\_length\_entry: The entry box for the length of each sequence the user would like to use

#### Confirmation Button

This is the button that confirms all the values and calls a nested function to initiate the Generator object with the values in the entry boxes as the given input.

### Implementation (unfinished)

### Testing (unfinished)

### Review (unfinished)

# Testing (unfinished)

Detailed descriptions of every test case are definitely not what is required in this section; the place for detailed lists of tests cases is in an appendix. In this section, it is more important to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Provide information in the body of your report and the appendix to explain the testing that has been performed. How does this testing address the requirements and design for the project?

How comprehensive is the testing within the constraints of the project? Are you testing the normal working behaviour? Are you testing the exceptional behaviour, e.g. error conditions? Are you testing security issues if they are relevant for your project?

Have you tested your system on “real users”? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

Whilst testing with “real users” can be useful, don't see it as a way to shortcut detailed testing of your own. Think about issues discussed in the lectures about until testing, integration testing, etc. User testing without sensible testing of your own is not a useful activity.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

## Overall Approach to Testing

## Automated Testing

### Unit Tests

### User Interface Testing

### Stress Testing

### Other Types of Testing

## Integration Testing

## User Testing

# Critical Evaluation (unfinished)

Examiners expect to find a section addressing questions such as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

The questions are an indication of issues you should consider. They are not intended as a specification of a list of sections.

The evaluation is regarded as an important part of the project report; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things in the work and aspects of the work that could be improved. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

# References

This final section should list all relevant resources that you have consulted in researching your project.

1. Krishnan, S., Lima, C.F., Evans, S., Chen, S., Guldner, S., Yeff, H., Manly, T. and Scott, S.K., 2018. Beatboxers and guitarists engage sensorimotor regions selectively when listening to the instruments they can play. Cerebral Cortex, 28(11), pp.4063-4079.
2. Blood, A.J. and Zatorre, R.J., 2001. Intensely pleasurable responses to music correlate with activity in brain regions implicated in reward and emotion. Proceedings of the national academy of sciences, 98(20), pp.11818-11823.
3. Leaver, A.M., Van Lare, J., Zielinski, B., Halpern, A.R. and Rauschecker, J.P., 2009. Brain activation during anticipation of sound sequences. Journal of neuroscience, 29(8), pp.2477-2485.
4. Tham, I., 2021. Generating Music using Deep Learning. [online] Medium. Available at: <https://towardsdatascience.com/generating-music-using-deep-learning-cb5843a9d55e#:~:text=Recently%2C%20Convolutional%20Neural%20Networks%20(CNNs,convolutions%20to%20generate%20raw%20audio.> [Accessed 13 March 2022].
5. Skuli, S., 2017. How I Built A Lo-fi Hip-Hop Music Generator. [online] Medium. Available at: <https://ai.plainenglish.io/building-a-lo-fi-hip-hop-generator-e24a005d0144> [Accessed 21 March 2022].
6. Medium. 2020. How I Built A Lo-fi Hip-Hop Music Generator. [online] Available at: <https://ai.plainenglish.io/building-a-lo-fi-hip-hop-generator-e24a005d0144> [Accessed 19 March 2022].
7. Graves, A., 2012. Long short-term memory. Supervised sequence labelling with recurrent neural networks, pp.37-45.Hochreiter, S., 1998. Recurrent neural net learning and vanishing gradient. International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems, 6(2), pp.107-116.
8. Hochreiter, S., 1998. Recurrent neural net learning and vanishing gradient. International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems, 6(2), pp.107-116.
9. University, I., 2022. The MIDI Standard: Introduction to MIDI and Computer Music: Center for Electronic and Computer Music: Jacobs School of Music. [online] Cecm.indiana.edu. Available at: <https://cecm.indiana.edu/361/midi.html> [Accessed 29 March 2022].
10. Fisher, T., 2022. What's a MIDI File and How Do You Open One?. [online] Lifewire. Available at: <https://www.lifewire.com/midi-file-2621979> [Accessed 13 April 2022].
11. Yegulalp, S., 2019. What is TensorFlow? The machine learning library explained. [online] InfoWorld. Available at: <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html> [Accessed 17 April 2022].
12. Graves, A. and Schmidhuber, J., 2005, July. Framewise phoneme classification with bidirectional LSTM networks. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. (Vol. 4, pp. 2047-2052). IEEE.
13. Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.
14. Yegulalp, S., 2017. Facebook brings GPU-powered machine learning to Python. [online] InfoWorld. Available at: <https://www.infoworld.com/article/3159120/facebook-brings-gpu-powered-machine-learning-to-python.html> [Accessed 21 April 2022].
15. Terra, J., 2022. [online] Available at: <https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article#:~:text=TensorFlow%20is%20an%20open%2Dsourced,because%20it's%20built%2Din%20Python.> [Accessed 13 April 2022].
16. Magenta. 2022. Magenta. [online] Available at: <https://magenta.tensorflow.org/> [Accessed 10 April 2022].
17. Sidor, S., 2020. magenta/magenta/models at main · magenta/magenta. [online] GitHub. Available at: <https://github.com/magenta/magenta/tree/main/magenta/models> [Accessed 10 April 2022].
18. Dominodatalab.com. 2022. What is Anaconda? | Domino Data Science Dictionary. [online] Available at: <https://www.dominodatalab.com/data-science-dictionary/anaconda> [Accessed 25 April 2022].
19. Brownlee, J., 2019. A Gentle Introduction to Generative Adversarial Networks (GANs). [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/> [Accessed 24 April 2022].
20. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. Advances in neural information processing systems, 27.
21. McCormick, C., n.d. Melody Generator. [online] Melody Generator. Available at: <https://dopeloop.ai/melody-generator/?s=7019330677856559> [Accessed 1 May 2022].
22. Soundraw.io. n.d. Soundraw. [online] Available at: <https://soundraw.io/#about-soundraw> [Accessed 1 May 2022].
23. PyPI. n.d. PyQt5. [online] Available at: <https://pypi.org/project/PyQt5/> [Accessed 1 May 2022].
24. Amos, D., 2022. Python GUI Programming With Tkinter – Real Python. [online] Realpython.com. Available at: <https://realpython.com/python-gui-tkinter/> [Accessed 1 May 2022].
25. Sutherland, J., 1993. What is Scrum?. [online] Scrum.org. Available at: <https://www.scrum.org/resources/what-is-scrum> [Accessed 1 May 2022].
26. Lovelace, A., 1843. Analytical Engine :: Museum of Imaginary Musical Instruments. [online] Imaginaryinstruments.org. Available at: <http://imaginaryinstruments.org/lovelace-analytical-engine/> [Accessed 2 May 2022].
27. L. Casini, G. Marfia and M. Roccetti, "Some Reflections on the Potential and Limitations of Deep Learning for Automated Music Generation," 2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), 2018, pp. 27-31, doi: 10.1109/PIMRC.2018.8581038.
28. Briot, JP., Pachet, F. Deep learning for music generation: challenges and directions. Neural Comput & Applic 32, 981–993 (2020). <https://doi.org/10.1007/s00521-018-3813-6>
29. Pygame.org. n.d. *About - pygame wiki*. [online] Available at: <https://www.pygame.org/wiki/about> [Accessed 3 May 2022].
30. Surkis, L., 2019. *How to Generate Techno Music using Deep Learning*. [online] Medium. Available at: <https://medium.com/@leesurkis/how-to-generate-techno-music-using-deep-learning-17c06910e1b3> [Accessed 2 May 2022].

# Appendices (unfinished)

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

If you have used any 3rd party code, i.e. code that you have not written yourself such as libraries, then you must include Appendix A. In that appendix, you will provide details of the 3rd party code that you have used.

For most other items, it would be better to include them in your technical submission instead of including them as an appendix. For example:

* If you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document in the technical work. In this report, you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.
* If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification at the start of the project. Perhaps you used stories to keep track of the functionality and the ‘future conversations.’ If it isn’t relevant to include all those stories in the body of your report, you could detail those stores in a document in the technical work.
* If you have used manual testing, then include a document in the technical work that records the tests that have been done. In this report, you would talk about the use of those tests.

Documents included in the technical work or in the appendices are supporting evidence of the work done. Where you include documents, this report should refer to the documents. You should not be relying on detailed study of those documents in order to understand what is written in this report.

Speak to your supervisor or the module coordinator if you have questions about this.

* 1. Third-Party Code and Libraries

If you have made use of any third-party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third-party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what your original work is and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

The following is an example of what you might say.

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

Include as many declarations as appropriate for your work. The specific wording is less important than the fact that you are declaring the relevant work.

* 1. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Project Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

Only include code in the appendix if that code is discussed and referred to in the body of the report.

Random Number Generator

The Bayes Durham Shuffle ensures that the pseudo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs.

// Some example code here…