**Computer Science and Engineering, University of Nevada, Reno**

**MelodyBot**

**Team #19**

**Nicholas Harris**

**David Neilsen**

**Joseph San Nicolas**

**Stefan Stukelja**

**Instructors: Sergiu Dascalu & Devrin Lee**

**External advisor(s): Dave Feil-Seifer**

**Benjamin Brown**

**February 23rd, 2018**

# **Table of Contents**

Abstract……………………………………………….3

Recent Project Changes…………………………….3

Specification………………………………………….3

Summary of Changes in Project Specification…...3

Technical Requirements Specification…………….3

Use Case Modeling………………………………….6

Requirement Traceability Matrix…………………...10

Summary of Changes in Project Design………….10

High-Level and Medium-Level Design…………….10

User Interface Design……………………………….21

Glossary of Terms…………………………………...30

Engineering Standards/Technologies……………..32

List of References…………………………………...33

Contribution of Team Members……………………34

# **Abstract**

MelodyBot is is an artificial intelligence whose purpose is to create unique music samples based on the genre and other parameters that are specified by the user. MelodyBot is able to create the desired music sample thanks to its use of a Recurrent Neural Network (RNN). Currently, the neural network has been able to generate short music samples ranging from 6-30 seconds that are of a simple nature. The developers of MelodyBot have created a functioning user interface in Visual Basic that allows the user to load/remove sound files in a selectable list, play and pause said sound files, set preferences for the generated song, and invoke the artificial intelligence.

# **Recent Project Changes**

There are no recent changes to the project since the submission of the first report.

# **Specification**

# **Summary of Changes in Project Specification**

The only changes done to the project’s specification is changing the level of priority of one of the functional requirements. The specific requirement being “allowing the user to generate a new random ‘seed’, upload a new seed, and save a seed”. The team saw that this requirement would take up a large amount of work and believed that it would interfere with the finalization of the level 1 requirements.

## **Technical Requirements Specification**

**Functional Requirements (FR)**

The following table lists the updated functional requirements for MelodyBot. The requirements are put into three levels of priority (Level 1, Level 2, Level 3). Also, each requirement includes a rationale justifying the requirement.

|  |  |  |
| --- | --- | --- |
| **FR Number** | **FR Implementation** | **FR Description** |
| FR\_01 | Level 1 | * MelodyBot shall allow users to select a type of music genre. * Rationale: Having the user choose the genre of music is important so the user knows what they are getting. |
| FR\_02 | Level 1 | * MelodyBot shall have a functioning audio player that can play, pause, resume, fast-forward, and rewind music. * Rationale: Since MelodyBot’s main function is to generate music samples, it should have the functionality to play said music sample. |
| FR\_03 | Level 2 | * MelodyBot shall allow the user to download the generated music sample created by it. |
| FR\_04 | Level 1 | * MelodyBot shall allow the user to change the audio settings, i.e. volume, of the music it is playing. * Rationale: MelodyBot should have the functionality of a regular audio player. |
| FR\_05 | Level 2 | * MelodyBot shall allow the user to change the music settings, i.e. tempo, of the music that it is playing/generating. * Rationale: Users should have the freedom of changing the tempo of music and what instruments are being used. |
| FR\_06 | Level 3 | * MelodyBot shall allow the user to generate a new random “seed”, upload a new seed, and save a seed. * Rationale: MelodyBot uses neural networks, which uses these “seeds” to generate samples that are based-on the seeds. Giving the user the ability provide and save seeds gives them more interaction and use for MelodyBot. |
| FR\_07 | Level 1 | * MelodyBot should include information about its use for the user to read. * Rationale: If the user doesn’t know a product’s use, then they won’t use the product. |

**Non-Functional Requirements (NFR)**

The following table lists the non-functional requirements for MelodyBot. The requirements are put into three levels of priority (Level 1, Level 2, Level 3). Also, each requirement includes a rationale justifying the requirement.

|  |  |  |
| --- | --- | --- |
| **NFR Number** | **NFR Implementation** | **NFR Description** |
| NFR\_01 | Level 1 | * MelodyBot’s shall have a “How-To” section showing the user how to use MelodyBot. * Rationale: Every product needs an instruction manual. |
| NFR\_02 | Level 2 | * MelodyBot shall be published as a website, being compatible with most major web browsers. * Rationale: Having MelodyBot as a web application gives users access to it from any device that can access the internet. |
| NFR\_03 | Level 3 | * MelodyBot shall have a mobile friendly interface. * Rationale: If MelodyBot is not released as a mobile app, giving users a mobile friendly interface will allow users to view the website in an organized way on a mobile device. |
| NFR\_04 | Level 1 | * MelodyBot shall have a organized and pleasant looking user interface. * Rationale: Users enjoy a well made user interface. |
| NFR\_05 | Level 3 | * MelodyBot shall be able to process most major credit cards to pay for subscriptions. * Rationale: Allowing the users to have special access to MelodyBot for a small monthly fee may increase users as well as subscriptions. |
| NFR\_06 | Level 3 | * MelodyBot shall be updated with more genres using additional trained neural networks. * Rationale: Updating MelodyBot at a decent rate and adding more genres will increase users. |

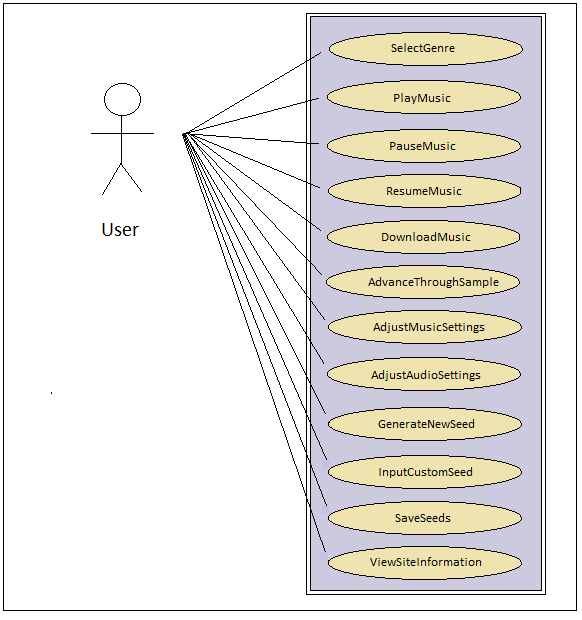
# 

## 

## 

## Use Case Modeling

**Use Case Diagram**



**Use Case Descriptions**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Use Case Description** |
| UC\_01 | SelectGenre | The user will be able to select between numerous genres that correspond to a pre-trained neural network made to dispense music of the selected genre. These may include classical, blues, rock, EDM, etc. |
| UC\_02 | PlayMusic | Once a genre is selected, the user should be able to play music generated by the trained neural network. This should function like an ordinary audio player, with an intuitive “play” button. |
| UC\_03 | PauseMusic | The user should be able to pause their music once it is playing, in the way a normal audio player functions, with an intuitive “pause” button. |
| UC\_04 | ResumeMusic | A paused player should not stop functioning entirely, so the user should be able to resume their music where they left off. |
| UC\_05 | DownloadMusic | If a user likes a particular set of music, they should be able to download a sample of it for their own use. This may be in the form of a MIDI file, or an alternative. |
| UC\_06 | AdvanceThroughSample | The user may want to “fast-forward” through a musical sample to skim it, as opposed to listening to it in its entirety. As in normal audio players, a simple bar allowing the user to change the time of the track will suffice. |
| UC\_07 | AdjustMusicSettings | Certain properties of the produced music will be easily changeable, such as the tempo, instrument selection, and others. Allowing these to be changed increases the customizability of the music. |
| UC\_08 | AdjustAudioSettings | As with ordinary audio players, certain settings such as the volume should be quick and easy to change to allow for a better user experience. |
| UC\_09 | GenerateNewSeed | The user can generate a new random “seed” to feed to the trained neural network to produce a new music track within the same genre. |
| UC\_10 | InputCustomSeed | Instead of relying on a random seed, the user will be able to input a seed of their own choice. This can be as simple as a string of characters. |
| UC\_11 | SaveSeeds | To allow the user to save music they like, they should have the option to save the seeds used to produce the music in whatever genre. A collection of saved seeds can be associated to the user. |
| UC\_12 | ViewSiteInformation | The website hosting this functionality may be a little confusing to some users, so a user should easily be able to find a page of information describing how to use the project’s functionality to its fullest. |

**Detailed Use Cases**

|  |  |
| --- | --- |
| **Use Case: SelectGenre** | |
| Use Case ID | UC\_01 |
| Actor | User |
| Precondition(s) | 1. The user has opened the web page or app. |
| Flow of Events | 1. The user opens the list of genres to browse. 2. The user decides on a genre and selects it, opening a region for the production of music in the selected genre. |
| Postconditions(s) | 1. An area is opened for the user allowing them to continue with the production of music in their choice of genre. |

|  |  |
| --- | --- |
| **Use Case: PlayMusic** | |
| Use Case ID | UC\_02 |
| Actor | User |
| Precondition(s) | 1. The user has opened the web page or app. 2. The user has selected a genre and is working in the “production area”. |
| Flow of Events | 1. User hits “play button” on audio player, playing music as generated by the pre-trained neural network from the seed. |
| Postconditions(s) | 1. Music begins playing correctly from the audio player. |

|  |  |
| --- | --- |
| **Use Case: GenerateNewSeed** | |
| Use Case ID | UC\_09 |
| Actor | User |
| Precondition(s) | 1. The user has opened the web page or app. 2. The user has selected a genre and is working in the “production area”. |
| Flow of Events | 1. User will hit button that generates a new random seed for the production of music in their selected genre. |
| Postconditions(s) | 1. A new random seed has been generated and is displayed in place of the old one. |

## Requirement Traceability Matrix

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 | UC10 | UC11 | UC12 |
| FR1 |  |  |  |  |  |  |  |  |  |  |  |  |
| FR2 |  |  |  |  |  |  |  |  |  |  |  |  |
| FR3 |  |  |  |  |  |  |  |  |  |  |  |  |
| FR4 |  |  |  |  |  |  |  |  |  |  |  |  |
| FR5 |  |  |  |  |  |  |  |  |  |  |  |  |
| FR6 |  |  |  |  |  |  |  |  |  |  |  |  |
| FR7 |  |  |  |  |  |  |  |  |  |  |  |  |

# **Design**

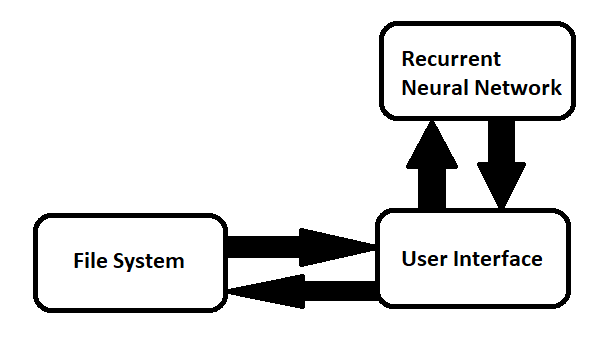
## Summary of Changes in Project Design

The only significant change in the project design since Fall of 2017 is the replacement of the generator and discriminator networks with a single RNN for music generation. This may only be the case for the initial implementation, so it is possible the project will eventually make use of adversarial networks again. For now, the switch to the single RNN is reflected in the updated design.

## High-Level and Medium-Level Design

Architectural Pattern

|  |  |
| --- | --- |
| Name | Repository |
| Description | Users are able to select sample inputs from the database which are then used by the neural network to train and produce high quality output. This output is stored in a local folder and then able to be extracted by the user through the user interface. See diagram below for reference. |
| Example | An IDE with several modules which all have access to a central repository of design information and can use this information to generate new data. |
| When Used | This is a suitable pattern for systems where one component generates data and another uses it. |
| Advantages | This pattern allows the system to propagate the necessary information to all relevant components. |
| Disadvantages | The system is reduced to a state where issues with the file system or neural network can cause the entire system to fail. |

****

## File System

The file system is the user’s computer and all applicable files they might have. These files are used as the samples for the neural network to generate music for the user. Files used as samples for the neural network must be MIDI files which have an interface with the neural network providing the capability to learn and create new songs.

## User Interface

The user interface provides a means for the user to select a group of songs to be used as input for the artificial intelligence along with other preferences such as song length. The songs that are generated by the artificial intelligence will also be available through the user interface.

## Recurrent Neural Network

The neural network will identify patterns within the samples provided by the user’s file system and use this information to train and learn how to create music with the characteristics of the samples. After sufficient training, the network is able to begin producing music.

## Class Diagram

## 

## User Interface

|  |
| --- |
| User Interface |
| AddSamples()  RemoveSample()  Generate()  SelectSong()  PlaySong()  PauseSong()  Download() |

Description: Provides a means for the user to interact with the artificial intelligence.

AddSamples(): Adds songs selected by the user to the sample that will be used for the artificial intelligence and alerts the system to begin generation.

RemoveSample(): Removes a sample selected by the user.

Generate(): Invokes the artificial intelligence to begin generation.

SelectSong(): Allows the user to select a song to preview.

PlaySong(): Allows the user to play the song currently selected.

PauseSong(): Allows the user to pause the song currently playing.

Download(): Allows the user to download the generated song.

## File System

|  |
| --- |
| File System |
| AvailableSongs  Input  Output |
| getInput()  getOutput() |

Description: User’s file system which contains songs available for sampling.

AvailableSongs: All files that are valid on the user’s file system.

Input: A collection of songs which have been determined to be used as input for the artificial intelligence.

Output: The song which is generated by the artificial intelligence after machine learning is complete.

getInput(): Method to extract the files designated as input from the file system.

getOutput(): Method to extract the file created by the artificial intelligence.

## Recurrent Neural Network

|  |
| --- |
| Recurrent Neural Network |
| TrainingVariables  LossFunction  Optimizer |
| Run() |

Description: Neural network which takes sampled data from the file system and then engages in machine learning to produce a new song.

TrainingVariables: Variables that are utilized to improve the performance of the neural network.

LossFunction: A metric which measures how much the neural network can improve by tuning the training variables.

Optimizer: A method which adjusts the training variables according to the loss function.

Run(): Method which runs the neural network and generates a song.

## MIDI Interface

|  |
| --- |
| MIDI Interface |
| ParseMIDI()  CreateMIDI() |

Description: Interface which allows the neural networks to interact with MIDI files.

ParseMIDI(): Method which parses MIDI files so that they can be manipulated and analyzed by the neural networks.

CreateMIDI(): Method which allows the neural network to create a MIDI file.

## MIDI Files

|  |
| --- |
| MIDI Files |
| HeaderChunk  TrackChunk |
| GetHeader()  GetTrack() |

Description: Files which contain song data in MIDI format.

HeaderChunk: Chunk which contains meta information about the MIDI file.

TrackChunk: Chunk which contains information about each track (instrument) present in the MIDI file and records of notes that are played.

GetHeader(): Retrieves the header chunk.

GetTrack(): Retrieves the specified track chunk.

## Nodes

|  |
| --- |
| Nodes |
| Input  Output |
| ActivationFunction() |

Description: Nodes emulate the function of neurons in the human brain and together they can emulate the functions of the human brain as a neural network.

Input: Data received by the node to apply transformations on before sending it as output to another node.

Output: Input data which has been transformed and passed along to another node in the neural network.

ActivationFunction(): Function which transforms the input of a node and propagates it throughout the neural network.

## Header Chunk

|  |
| --- |
| Header Chunk |
| Identifier  Length  Format  NumTracks  Division |
| GetIdentifier()  GetLength()  GetFormat()  GetNumTracks()  GetDivision() |

Description: The first chunk to appear in a MIDI file. Contains information about the overall MIDI file such as size and identification.

Identifier: Code used to identify the file as a MIDI file and header chunk.

Length: The length of the header chunk.

Format: Format of the MIDI file. MIDI files may have single tracks, multiple tracks, or be a collection of single track MIDI files.

NumTracks: Specifies the number of track chunks which follow the header chunk.

Division: Specifies how the MIDI file divides time.

GetIdentifier(): Retrieves the identification of the MIDI file.

GetLength(): Retrieves the length of the header chunk.

GetFormat(): Retrieves the format of the MIDI file.

GetNumTracks(): Retrieves the number of track chunks present in the MIDI file.

GetDivision(): Retrieves the unit of time division.

## Track Chunk

|  |
| --- |
| Track Chunk |
| Identifier  Length  TrackEvents |
| GetIdentifier()  GetLength()  GetTrackEvents() |

Description: Chunks which follow the header chunk. Each track chunk represents an instrument.

Identifier: Code used to identify the chunk as a track chunk.

Length: size of the track chunk in bytes.

TrackEvents: A sequence of track events.

GetIdentifier(): Retrieves the identification for the track chunk.

GetLength(): Retrieves the size of the track chunk in bytes.

GetTrackEvents(): Retrieves the track events present in the chunk.

## Track Event

|  |
| --- |
| Track Event |
| ElapsedTime  Event |
| GetElapsedTime()  GetEvent() |

Description: Contains information about the track (instrument).

ElapsedTime: The time that has elapsed since the last track event.

Event: Details on what the event is. This event can be one of three types: MIDI Event, Meta Event, or System Exclusive Event.

GetElapsedTime(): Retrieves the time that has elapsed since the last track event.

GetEvent(): Retrieves the event that has occurred on the track.

## MIDI Event

|  |
| --- |
| MIDI Event |
| ChannelStatus  Note  Velocity |
| GetChannelStatus()  GetNote()  GetVelocity() |

Description: Events which control the instrument on the track.

ChannelStatus: Details on the channel the track is associated with such as whether the instrument is currently playing or not.

Note: Details on the note pertinent to this event.

Velocity: Controls aspects of the event occurring such as volume, filters, and mapping multiple samples to one note.

GetChannelStatus(): Retrieves information on the status of the channel associated with the track.

GetNote(): Retrieves the note being played in the event.

GetVelocity(): Retrieves the velocity magnitude for the event.

## Meta Event

|  |
| --- |
| Meta Event |
| Type  Length  Event |
| GetType()  GetLength()  GetEvent() |

Description: Non-MIDI events which do not contain information about aspects such as notes and velocity.

Type: Specifies the type of meta event. Examples of such events include tempo setting, key signature, and end of track

Length: Specifies the length of the meta event as a variable length value

Event: Contains detailed information about the event.

GetType(): Retrieves the type of meta event that has occurred.

GetLength(): Retrieves the length of the meta event.

GetEvent(): Retrieves details on the meta event that has occurred.

## System Exclusive Event

|  |
| --- |
| System Exclusive Event |
| Length  Event |
| GetLength()  GetEvent() |

Description: System exclusive events are used define message formats for synthesizers and primary uses include patch dumping and loading.

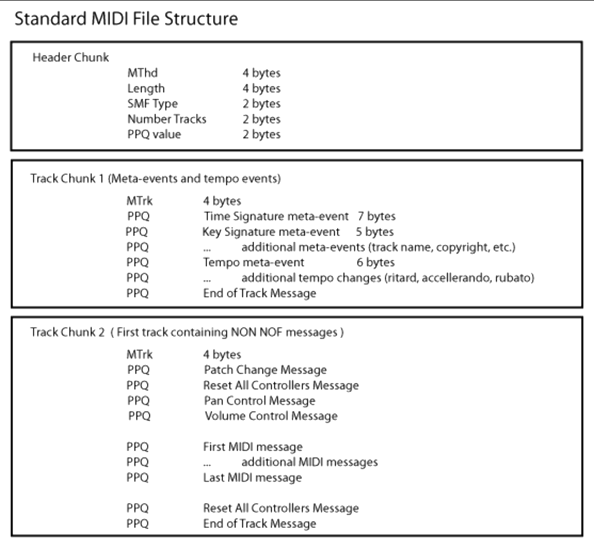
Length: The length of the system event.

Event: Data that is sent as a MIDI event to allow the use of external instruments.

GetLength(): Retrieves the length of the system event.

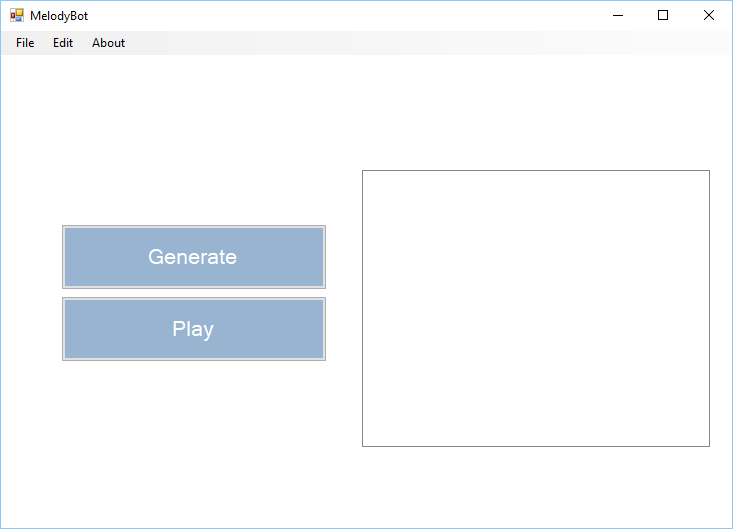
GetEvent(): Retrieves details on the system event that has occurred.

## Main Data Structure: MIDI Files

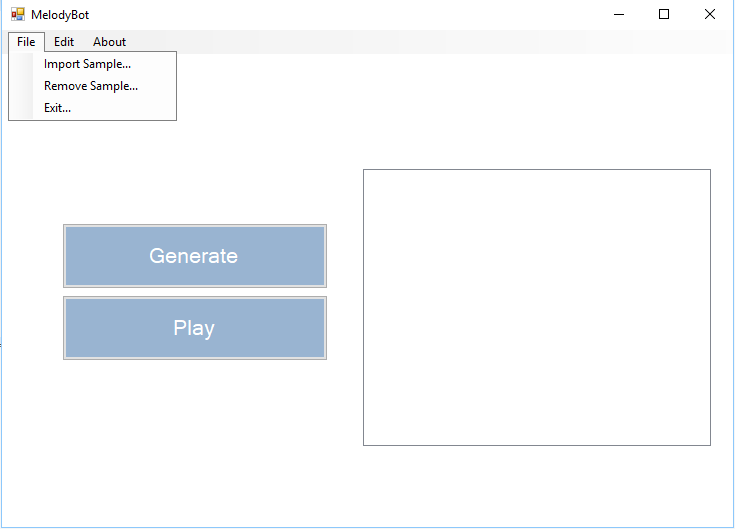


MIDI files provide a standard in which music can be represented electronically. These files consists of a header chunk and a variable amount of track chunks. The header chunk contains metadata about the file such as identifiers, size of the chunk, format, number of tracks, and how time should be divided. The track chunks represent each instrument present in the music. Each track chunk consists of an identifier, the size of the chunk, and track events. Track events contain information about the elapsed time since the previous event, MIDI events such as playing notes, and some non-MIDI events. A MIDI file has a record of every note that is played throughout the duration of a musical piece. MelodyBot will parse the MIDI files of sampled songs and use them to aid in the creation of new MIDI files.

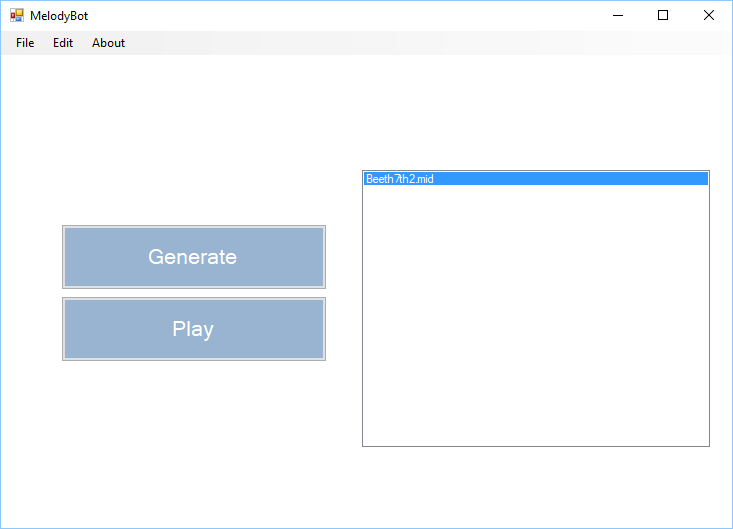
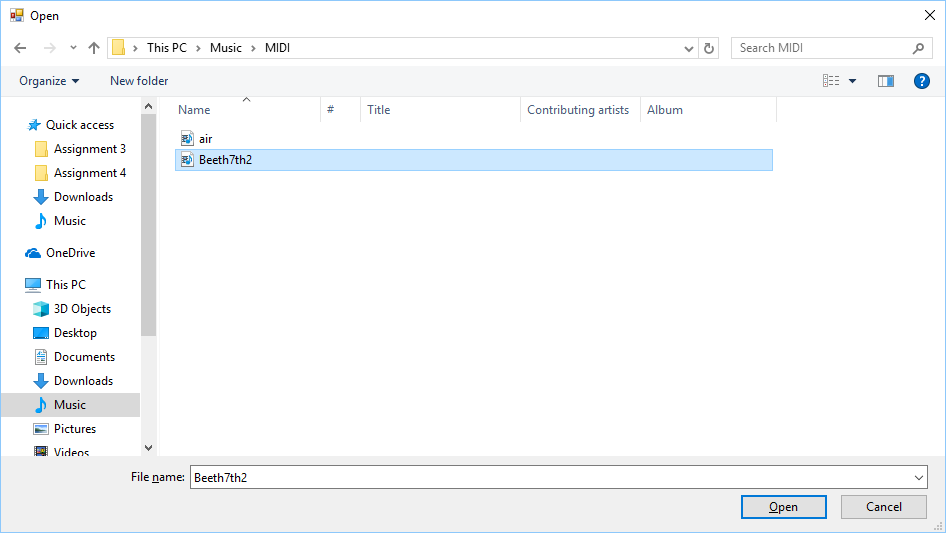
## User Interface Design



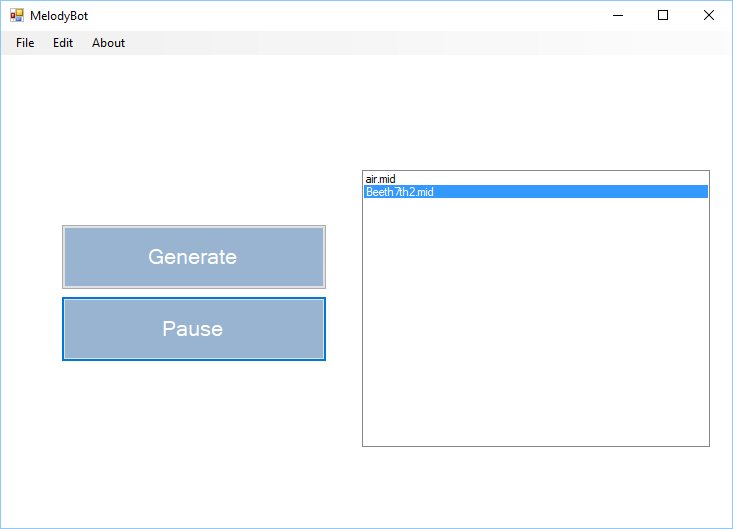
This is the default view of the user interface. From here a user is able to access the menu items available, play files that have been added to the sample list, and begin song generation.



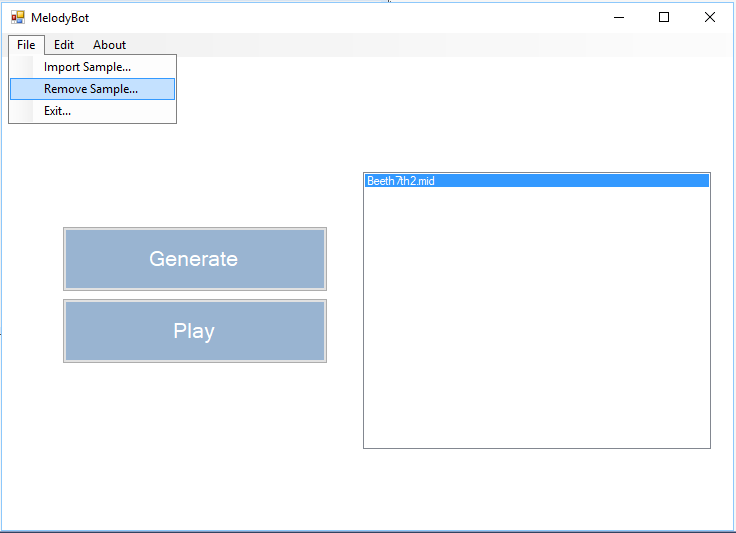
The file menu bar allows the user to import files into their sample list, remove them, and exit the program.



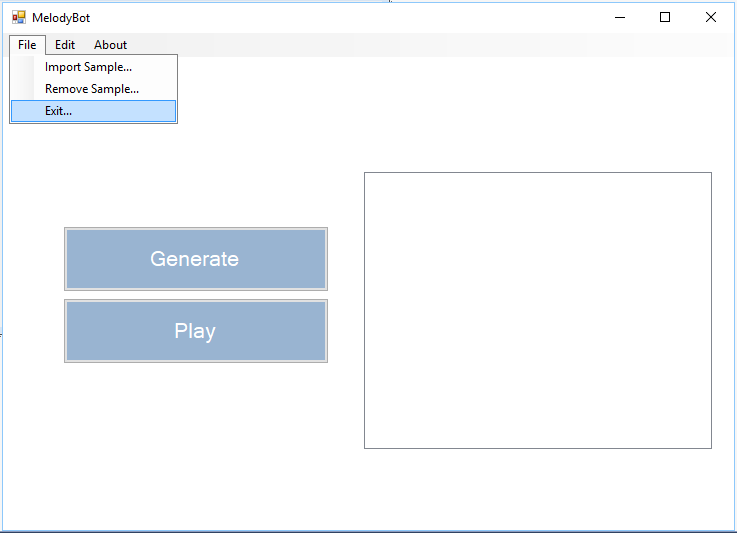
The import sample option will prompt the user to select a file on their computer to add to the sample list.



MelodyBot has a built-in media player which will let users listen to their samples before they are used for generation.



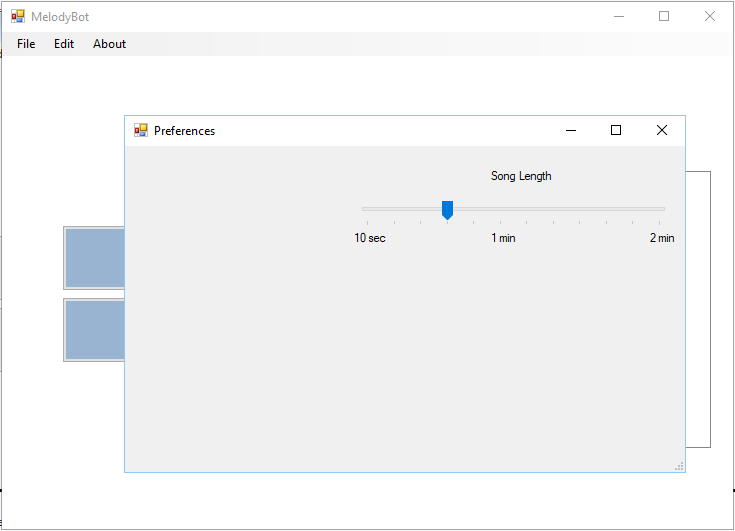
The essential ability to remove files from a sample list is also provided.



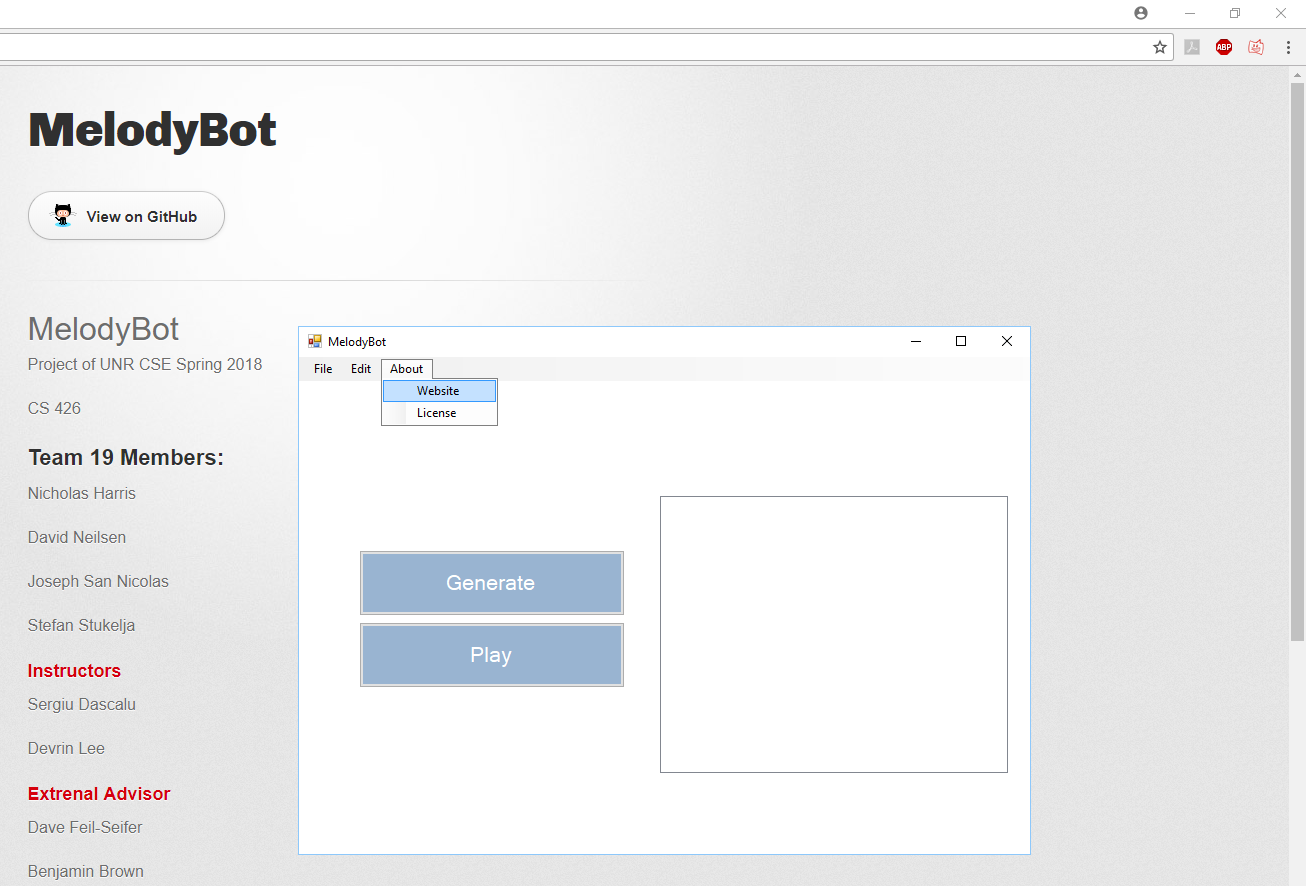
Pressing the exit button will close out of the application.

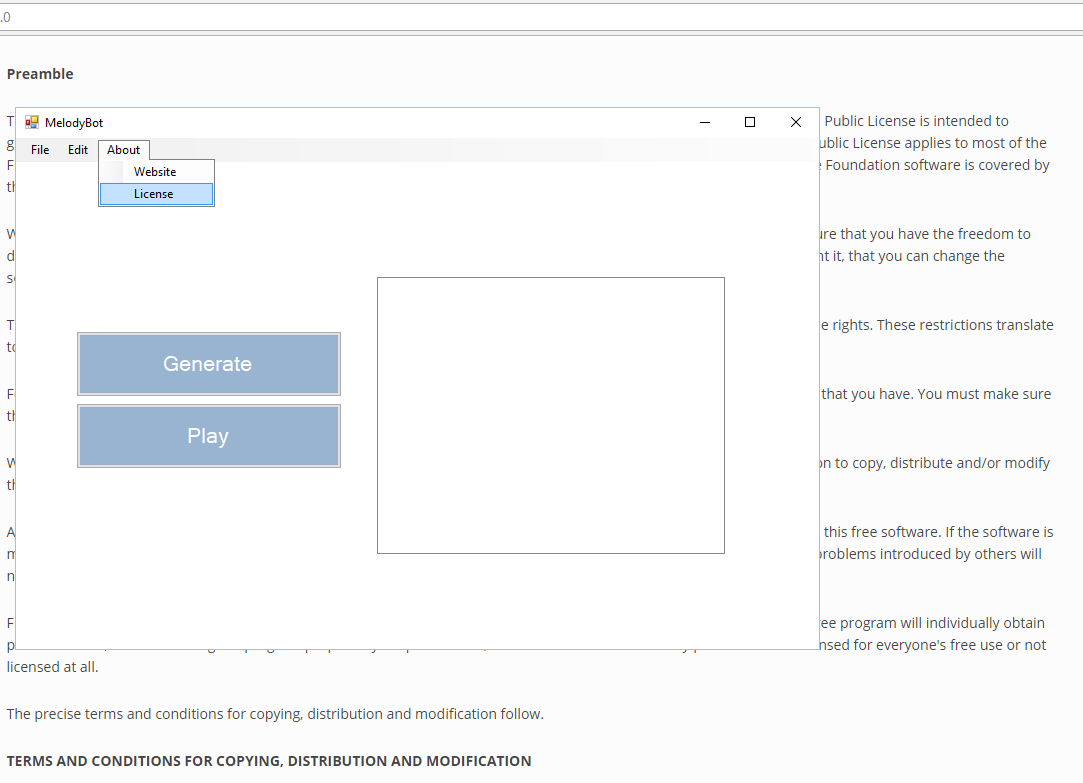


The edit menu allows users to tweak settings for the samples provided to the artificial intelligence as well as influence how the artificial intelligence will generate music.

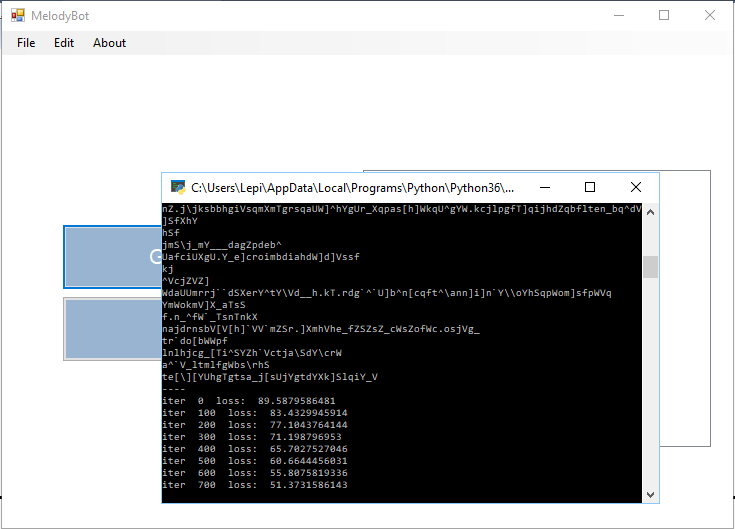


Preferences define characteristics of the music that will be generated by the artificial intelligence. The customization available will increase with the flexibility of the artificial intelligence.





The about menu includes information about MelodyBot such as the project website and licensing information.



The generate button invokes the artificial intelligence which generates music.

## Glossary of Terms

|  |  |
| --- | --- |
| MelodyBot | The product. An artificial intelligence which can sample music of a selected genre and generate new, original music for the user. |
| Artificial Intelligence | Human-like intelligence demonstrated by machines. |
| Knowledge Base | The storage and representation of complex information in a computer system. |
| Machine Learning | Algorithms which can learn and predict based off of data available to it. |
| Neural Networks | Computing systems based off animal brains which can emulate brain functions such as learning. |
| Generative Adversarial Network | Artificial intelligence algorithms used for machine learning. Consists of two neural networks which create a cycle of learning, one network (generator) generates content while the other (detective) discriminates it. |
| Zero-Sum Game | A framework where each player’s gains and losses balance out to a zero sum. This theoretical game model is used in Generative Adversarial Networks to improve the two constituent neural networks. |
| Recurrent Neural Network | A type of neural network which can process arbitrary sequences of input which is useful for tasks such as handwriting, speech recognition, and music generation. |
| MIDI | Standard which allows music to be saved, copied, and read on other computer systems. |
| Seed | Seed refers to the random input used by the neural networks for the generation of new music. |
| Python | An interpreted programming language which has many uses in artificial intelligence. |
| Android Studio | An integrated development environment which aids in the creation of Android mobile apps. |
| Sample | A MIDI file which is imported by the user and used for the machine learning of the neural network. |
| Genre | A classification of songs based on patterns and conventions built over time. |
| TensorFlow | An open-source software library used for machine learning. |
| Pre-Trained Neural Network | A neural network which has already undergone many sessions of learning and is able to produce higher quality output. |
| NumPy | A Python library providing support for high-level mathematical functions on large arrays and matrices. |
| Web Application | Software made to be accessed on the internet in a website format. |
| CSV | File format that MIDI files are converted into to interface with the artificial intelligence. |
| Tempo | The speed at which a piece of music is played. |
| MIDI Interface | An interface which can parse/write MIDI files which will be processed/generated by the neural networks, respectively. |
| Header Chunk | The first major portion of a MIDI file. The header chunk contains information about the file such as identification, size, and the number of tracks present within the file. |
| Track Chunk | Track chunks follow the header chunk and represent instruments within the music. Track chunks contain identification, length, and a sequence of track events. |
| Track Event | Track events represent the state of an instrument. |
| MIDI Event | MIDI events send messages to control the instrument such as which note to play, for how long, and at what velocity. |

## Engineering Standards/Technologies

1.Numpy- is the primary package for scientific computing in Python. It performs matrix computation within the neural network.

2. Visual Basic- programming environment and GUI used to create the user interface for Melody bot

3. Artificial Neural Networks- Computing structure modeled off of biological neural networks which utilizes a collection of connected nodes that can transfer information to each other. Combined they process information and remember patterns.

4. Generative Adversarial Network- Specific type of neural network in which two specialized neural networks work in tandem to create a cycle of learning, one network generates content while the other discriminates it.

5. C++- general-purpose, object oriented programming language. Used for the MIDI conversion script.

6. Python- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It is used entirely for the structure and computation involved in the neural network.

Standards

1. Code refactoring- process of restructuring the code without changes its external behavior. Used frequently in the development of the neural network.
2. Iterative programming- Software development structure in which the project is split into smaller development cycles know as iterations in which each iterations gets closer to the goal. Used frequently overall in this project as new features are slowly added over time.

## List of References

Problem Domain Book: “Virtual Music - Computer Synthesis of Musical Style” by David Cope. This is a useful resource for computer generation of music in general. In particular it describes capturing the musical style of particular artists.

Reference Articles:

* <https://arxiv.org/pdf/1409.2329.pdf>
  + This article goes over regularization techniques for recurrent neural networks. The main part that the team is using from this article is utilizing the RNN for language modeling. Much like a language, music has to have certain notes in order for it to be a song in a certain key, i.e. C, G, etc.
* <https://arxiv.org/pdf/1308.0850.pdf>
  + This article goes over using recurrent neural network in generating sequences. The team found this article to be very helpful as music could be described as a complex sequence. The article also includes code snippets of other projects that utilize RNNs in sequences.
* <http://www.isca-speech.org/archive/archive_papers/interspeech_2010/i10_1045.pdf>
  + This article focuses specifically on using recurrent neural networks for language modeling. This article provides a more in depth discussion about language modeling as well as some example experiments that use RNNs. The team may uses the examples as a comparison to the experiment.

Project Related Websites:

* <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
  + This blog post provides a vast amount of useful information regarding recurrent neural networks, including visual examples of the networks learning and code examples of some functionality of a recurrent neural network. In addition, it explains the theory behind RNNs in great detail, making it an excellent resource for beginners.
* <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/>
  + This blog post provides information in implementing a recurrent neural network with python, numpy, and theano. It includes code samples for RNN training in Theano and Numpy as well text generation. In addition, it also gives a general overview on the generation of a basic RNN.
* [http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks](http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/)
  + This blog post discusses on how to use RNNs for music generation. It gives a brief overview on RNNs as well as neural network training. It also discusses other projects that utilize RNN in music generation. It also provides music samples that the author’s RNN has produced.

## Contribution of Team Members

**Time Log**

This is a record of each team member’s time spent on each section of this report:

* **Nicholas Harris**
  + **Updated Use Case Modeling:** 1 hr
  + **Updated Requirement Traceability Matrix:** ½ hr
  + **Time Log:** ½ hr
  + **Paper Revisions:** ½ hr
* **David Neilsen**
  + **Engineering Standards/Technologies:** 1 hr
  + **Revising paper:** ½ hr
* **Joseph San Nicolas**
  + **Summary of Changes in Specification, Recent Project Changes:** 1 hr
  + **Updated Technical Requirements:** 1 hr
  + **Updated References:** ½ hr
* **Stefan Stukelja**
  + **Updated high- and medium-level design:** 1½ hr
  + **Updated User Interface Design:** 1 hr
  + **Updated Glossary of Terms:** ½ hr