The Algorithmic Composer

CSCE470 Final Report

Andrew Mason

5/2/2014

1. Abstract

The Algorithmic Composer is a program that generates music using training and evolutionary systems. The application is also a fully functional MIDI player. The clients for this application are Dr. Mock and myself, we are the only affiliated parties.

2. The Problem

Music is complex and time consuming to create. With The Algorithmic Composer music can be created in a much faster way. Music is essential in games and movies, creating an algorithm to generate music would save time and money. Musical theory is a huge field of study and is quite a hand full for those of us who don't understand the rules of music. My application is made to ease the process by gathering rules from songs already in existence, removing the burden from the user.

3. Project Requirements

3.1 Functional Specifications

1. A file menu for handling the loading and saving of files, and the genetic algorithm.
2. Controls for MIDI playback, play all, stop, shuffle, previous, and next buttons.
3. A list of MIDI files loaded that can be reordered and songs can be removed/added.
4. Progress bars to show wait times and current song position.

3.2 System Specifications

The application will be built in java and will be able to run on many machines and including web platforms. Because java is interpreted it will be easily accessible on many computers. The application will require that the user computer have a 1GB of memory and a pretty decent CPU.

4. Prototype Development

Because so much research was involved with this application I felt that prototyping would be the best development process. I broke sub-components of the total system in to their own stories to help individualize them. This was a pretty smart move I felt the transition to newer versions went smoother because of it. When beginning a new prototype I could focus on the sub-components that really needed attention, instead of the entire application. There were things that never needed re-factoring after version 1.

5. The Development Process

When first starting the project I began researching the MIDI file format. I spent 10-15 hours researching the file type before coding. This was an extremely helpful move, the code written was very solid and needed little to no re-factoring In the first prototype the GUI was a simple terminal I could enter commands into. There was feature creep, I wanted to listen to the songs in the training set in the application to verify that they were good songs to use. The main focus of the version 1 prototype was to effectively design the back-end while paying little attention to the front. Things went smoothly until I made it to the genetic algorithm. Eager to make music I rushed out the code in the GA. The code I spent 6 hours writing took me almost 12 hours to debug. I should have been patient and tested first, I put more work on myself than needed.

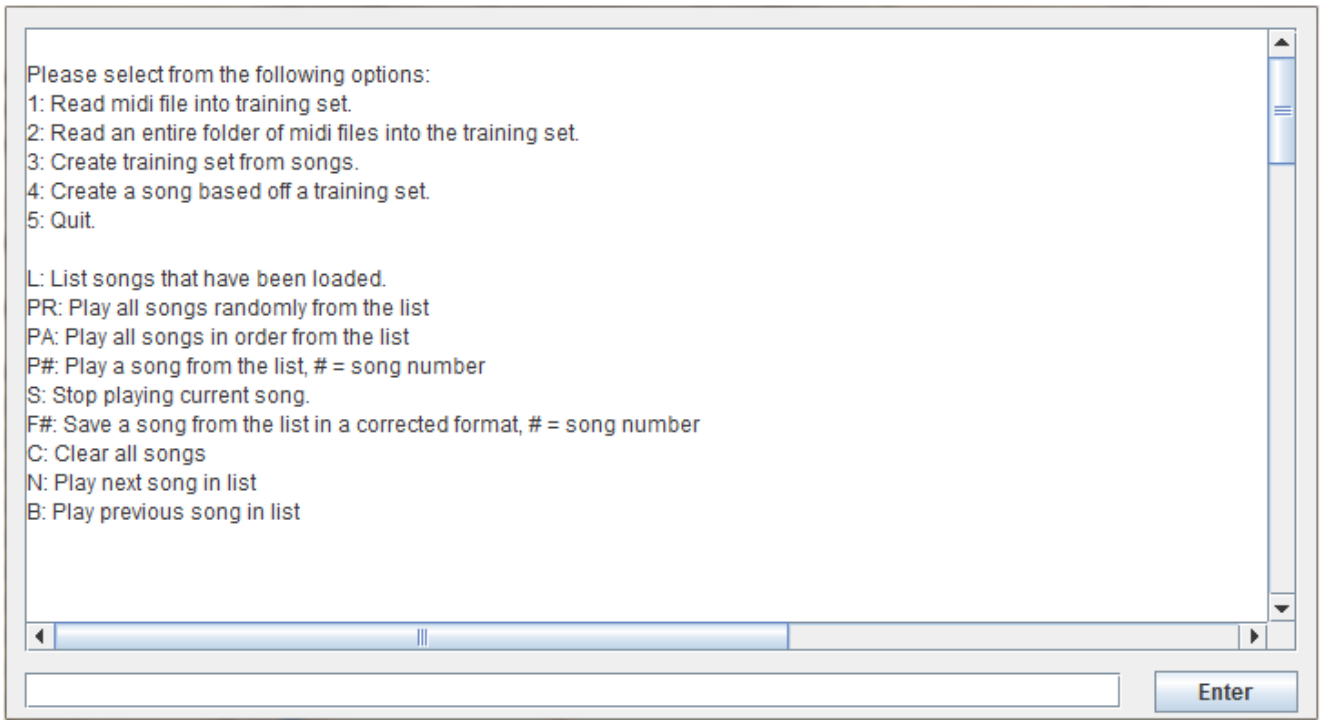
The second version is where I started paying more mind to the user interface. Prototyping was a really good approach because the functionality I added to the first version helped me design the user interface in the second. At this point I mainly focused on the GUI and the GA.

6. The Design

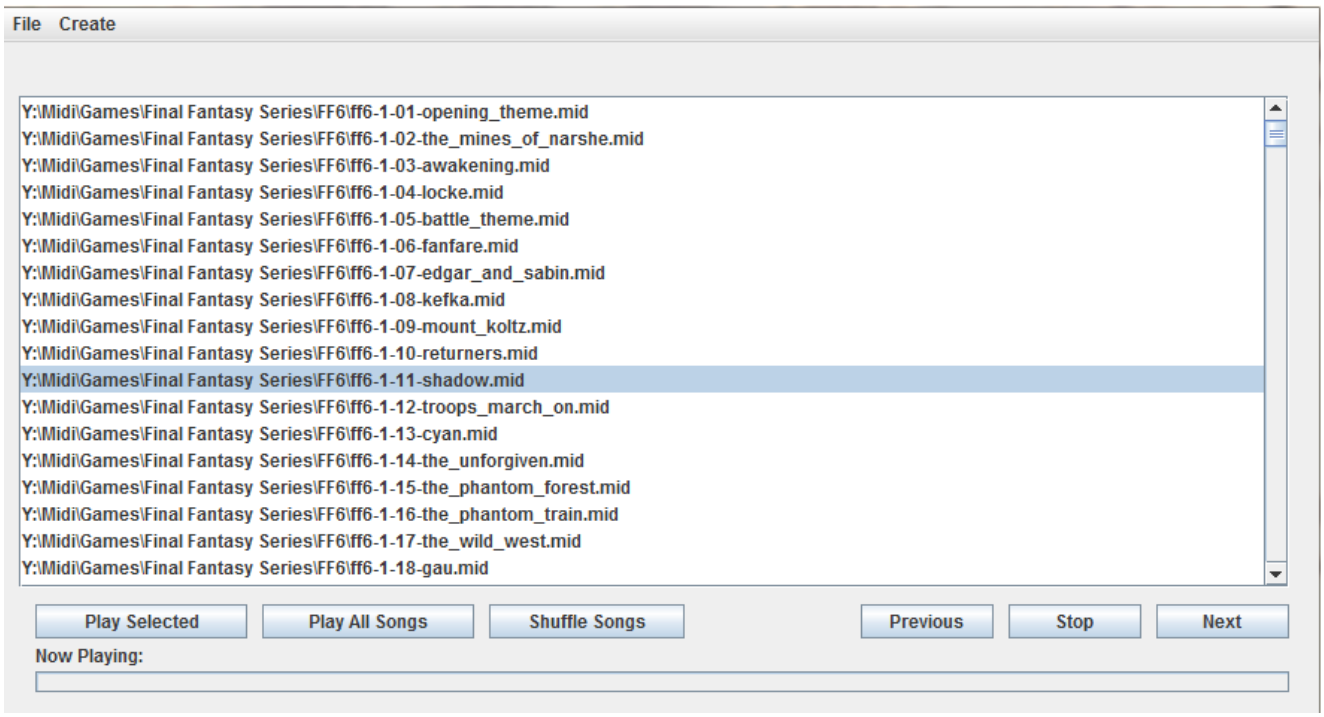
Because the application was developed using the prototyping process I wanted the back-end and front-end to be pretty well separated. I implemented a model view controller to help separate the GUI from the back-end. The view sends user input to the controller which handles the desired functionality for the application. This allowed a really smooth transition between the version 1 and version 2 prototype interfaces.

As stated previously, the version 1 interface was a simple terminal that encapsulated the functionality needed for the back-end. The version 1 commands reflected the functionality of the back-end. I did not implement the commands first I simply created the functions in the back-end and then added commands to utilize them.

Here is the version 1 interface:



Here is the version 2 interface:



Throughout the development life-cycle I tested first on functions and classes I thought would be prone to error. For methods that I knew would be shared between multiple classes I used static methods in a utility class. This made it so I did not have to do extensive passing of a class to all the classes that needed it. The utility class has methods like sort array, search array, and copy array. The utility class also handled things like printing hexadecimal characters and array values to console.

Another smart move was giving recognizable identifiers to unrecognizable MIDI commands. Located in my definitions class are static variables that are used between many classes. This is also where debugging options are set, if a debug flag is set to true, the console will print out a lot of information during run-time.

Saving and loading training data is done by using special characters to separate numeric values. All of the data classes can be exported to a string and then take that same string as an argument in their constructors. Training data held on to things like note harmonies, note movements, tempo, time division, and much more. When training data is loaded into the genetic algorithm it tries to reproduce the movements and harmonies in the training set.

7. Analysis, Results, and Discussion

The genetic algorithm never made anything too amazing. Though it did present a noticeable structure to songs that were generated after much re-factoring In version 1 mutations were chosen probabilistically, this was not a good move as the mutations were being applied to notes that didn't need them. Also in version 1 I used a measure class that would hold on to notes. This created a bug I didn't find until way later where notes that fell on the cusp of measures would never be played. Also the training data could be revised, I grabbed pretty good information from the songs but with a bit more understanding on musical theory it could have been improved.

The MIDI reader was an amazing success and completely paved the way for my Open CV application. MIDI can be a bit tedious to work with, I could not find a way to create a listener to listen to MIDI commands, such as note on, note off. By reading the commands directly from the loaded sequence I could store the note on and note off data. Once the data is stored in the song class I had access to when every note in a song should play. I saved enough data in the song class to create a sequence of the same song in my own fashion. The note class even holds on to durations, where as MIDI uses a note on/off combo to tell when a note stops.

During song generation a lot of movement and jump mutations are made but not so many duration mutations. The durations are determined from what durations play in a song, but the mutation only occurs if the duration was not in the training data. In retrospect it would have been better to use the actual rhythm data to compare duration movements. Then evaluate the song based on what durations come after others, just like the note movement data.

8. Conclusion and Lessons Learned

My research and testing for implementing a MIDI system helped significantly. I should have not gotten impatient when creating the genetic algorithm. If I had stuck to researching and testing like I did the MIDI reader I probably would have found the songs to be much more appealing. The genetic algorithm had a lot of unknowns when creating, I thought I had a good approach in mind but I had to find a better approach through testing the application. I was so eager to push out a song, I rushed the GA and in turn the songs generated sounded rushed. This is definitely a field that I would love to study more. Now that I have a suitable way of handling MIDI I could spend more time focusing on the GA and training data.

User Manual

File menu:

Load MIDI Files – Loads a directory or a single MIDI file.
Save Training Data – Saves training data from the songs loaded.
Save Selected Song – Saves a song from the play-list.
Delete Select Song – Deletes a song from the play-list.
Save Play-list – Saves the file-paths of songs in the play-list.
Load Play-list – Loads file-paths from a text file.

Create File Menu:

Create Song Using Data – Loads a training data file and generates a song using the genetic algorithm.
Create Song From Selected – Creates a song based off of a song from the play-list.
Stop Generating – Stops the genetic algorithm and returns the best song produced.

The Main Window:

Play Selected – Plays the selected song in the play-list.
Play All Songs – Plays the play-list's songs from start to finish.
Shuffle Songs – Shuffles the songs and then plays all from start to finish.
Previous – Plays the song before the current song playing.
Stop – Stop playing the current song playing.
Next – Plays the song after the current song playing.
Progress Bar – Clicking on the progress bar updates the songs current t tick position.
List – Songs can be reordered with a click and drag.
Delete – Songs selected can be deleted by pressing ctrl + d or delete.

Genetic Algorithm Dialog:

Population Size – Determines the number of individuals in the population.
Number of Generations – Determines the generations individuals will go through before the GA stops.
Mutation Chance – Determines the chance of mutation. Reproduction will always have a 0.1 chance. Crossover is 0.9 – the mutation chance.

Code Review Changes

Medium: Needs more commenting before methods.
Fix: Added commenting before methods.

Low: Make Random once in class and reuse.
Fix: Made Random once.

High: typo in comment.
Fix: Fixed typo.

Low: Duplicate method exit points.
Fix: Made 1 exit point in crossover method.

Medium: Don't change loop index.
Fix: Fix pending.

Low: Magic numbers.
Fixed: Gave identifiers to numbers.

Low: Duration split keeps track of numbers
Fix: Re-factored defect does not exist.

Low: long sequences.
Fix: Didn't fix, re-factor pending.

Low: this.ticklength
Fix: Did not fix, good for practice but would leave already existing code inconsistent(will use this from now on though).

Low: Merge code with brackets to remove extra lines
Fix: Did not fix, would leave existing code inconsistent.