MooScore Project Report

Ben Young, Rahul Prabhu

Introduction

Writing sheet music by hand is a very laborious task, so most music writing is now done with a computer program. These programs allow the user to edit the sheet, playback their work using synthesized instruments and finally export the finished project to a PDF or other printable document. These programs allow users to implement the complete set of extensive music notation and create distributable final products. This complexity means however that these programs have steep learning curves, and some have steep price tags or limited functionality for free versions.

Literature Review/Background Study

There are several prominent music writing programs currently available, including Musescore, Finale, and Sibelius. These programs are heavy-weight, commercial products designed mainly for professionals and experienced writers. The goal for this project is to create a lightweight simple to open and use program that will allow for editing in basic music notation, playback, and saving/loading their work. We will call this program MooScore, as a completely original homage to our experience as UC Davis students.

Methodology

We will be writing our program with the Java Swing framework, which is a component-based GUI library for the Java programming language. The Swing framework will allow us to quickly iterate on our design and make the program cross-platform compatible. Our first step in designing our app was creating a layout sketch, and deciding which features we wanted to implement. We are musicians as well as programmers, so it was easy for us to understand how music notation should function and which features would be essential and which are optional.

After creating a sketch for the layout, we designed the program's UML diagram which would describe what components we would need to implement for our program. Creating each component from the bottom of the dependency hierarchy upwards allows us to test our progam as it becomes more complex, and identify missing/necessary features that will need to be implemented.

Implementation Details

Mouse interaction:
To enable dragging symbols around the screen with the mouse, a moues handler is added to the GUI. This event handler checks if a component has been clicked on, and if so selects it. It also uses mouse dragging events to compute and draw a rectangle from the start point to the end point of a user's mouse click to enable the selection of multiple components at the same time. Selected components are shown in a highlighted color to help the user navigate the interface.

Playback:
Audio playback for our program is implemented using Java's MIDI library, which contains built in functions and objects that allow us to translate a sequence of notes from our GUI into a series of MIDI events, known as a track within the MIDI library. Notes duration is determined by their symbol, and their pitch is determined by their vertical position within their row. These tracks can be sent to a MIDI sequencer for playback, and are played at the user defined tempo. The MIDI library is powerful and may allow us to have additional voices, and choosing the instrument for playback.

Music Symbol Images:
To display the large number of different musical symbols inside our program, we use a large PNG file containing all of the symbols we need layed out in a grid. Each symbol's image is extracted from the master image and is contained within an enum. This enum also contains information about the symbols height, width, preferred drawing scale, and the duration of the note if applicable. This is used to translate the GUI components into a MIDI track for playback. This system allows us to have minimal files within the source as well as a consistent visual style for the entire program. It would also potentially allow for several different symbol fonts to be selected from the user, such as a jazz font.

Text Input:
For user text input (titles, subtitles, composer, tempo), Jlabels and JTextFields were used. The GUI uses absolute positioning to place components onto the panel, so setting bounds for width, height, x, and y positions along with the current window height and width were used to place these components onto the screen dynamically.

ToolBar / MenuBar:
These components were extended the library components JToolBar and JMenuBar. By extending from these components, we were able to grab the default look of these components and then add the extra buttons that were specific to our project. Most of the buttons employed JButton, but for the notes specifically, the JButtons were created with Icons containing the desired resized note images

Testing and Evaluation

Week 1:

During the initial week of planning and design, we decided to explore note placement onto the staff. We decided to pursue an approach that takes images of music notes and place them where the user clicks. Upon figuring out how to program event listeners that look for a mouse click and place an image, we determined that the centers of the images are not where the base of the note is, and that we could either create images that either fulfill this requirement or place images that are offset from where the user clicks to make it appear.

Week 2:

During Week 2, we refined the features that we had implemented last week. Since we had prototyped the ability to add/move notes and generate the staff / measure lines the previous week, we worked on improving these features. Now, the staff lines are generated dynamically according to the location of notes: notes on the bottommost staff will generate a new one below if the user needs to add more notes. Notes within the GUI are sorted based on their vertical position, simplified to which row of the staff they belong to, as well as their X position on the screen. This will allow us to playback the notes from top to bottom, left to right. The measure tracking system was also improved so that when all the notes in a measure are added up, it equals 4 beats. The position of notes are adjusted to align with the measure bar locations and ensure no invalid measure state exists. Last week, the toolbar functionality was not implemented in the backend, so now clicking a note symbol in the toolbar sets the active note that the user wants to place. Visually, the title, subtile, tempo, and composer information was added to the GUI for the users to customize. Upon customization, this data would get saved to variables for later use in the MIDI player and for file saving.

User Manual

Upon opening the app, users are able to see a representation of a blank music sheet. Users will be able to immediately edit the music sheet with the preset notation attributes. Other than adding music, the user will be able to type in the title, subtitle, and author of the music. The preset attributes will be that the clef is treble clef, the key signature is C (no sharps or flats), the time signature is 4/4, and users will initially be able to place quarter notes. Users will also be able to enter information about the music: a title, subtitle, tempo, and composer. There are two sections above the notation sheet where users will be able to do, the menu bar and the component bar.

The Menu Bar

The menu bar will perform extensive functions that are beyond the scope of the tool bar. For now, it will include three buttons: file, edit, and help. Upon clicking the file button, users will be prompted to either save or load their music. When saving music, the application will take the

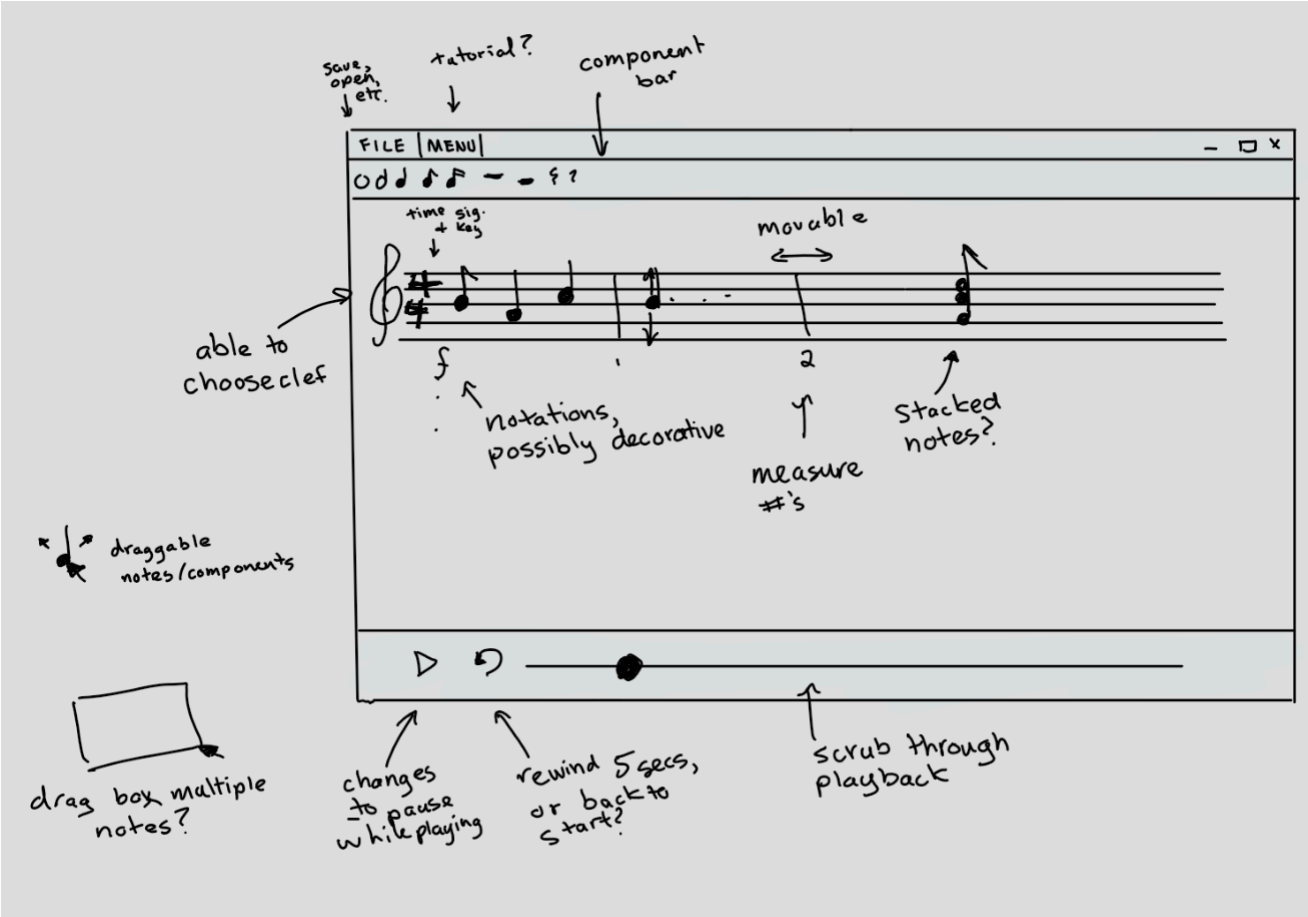
current music sheet and save the metadata into a JSON file for the user to store on their file system. Conversely, the load option will allow the user to take a previously stored JSON file and load it into the application. The edit button will for now include the ability to select one or more measures of music and perform operations to them, such as shifting or clearing. The help button will create a pop-up that will display the contents of the user manual to provide any clarifications about the functionality.

The Tool Bar

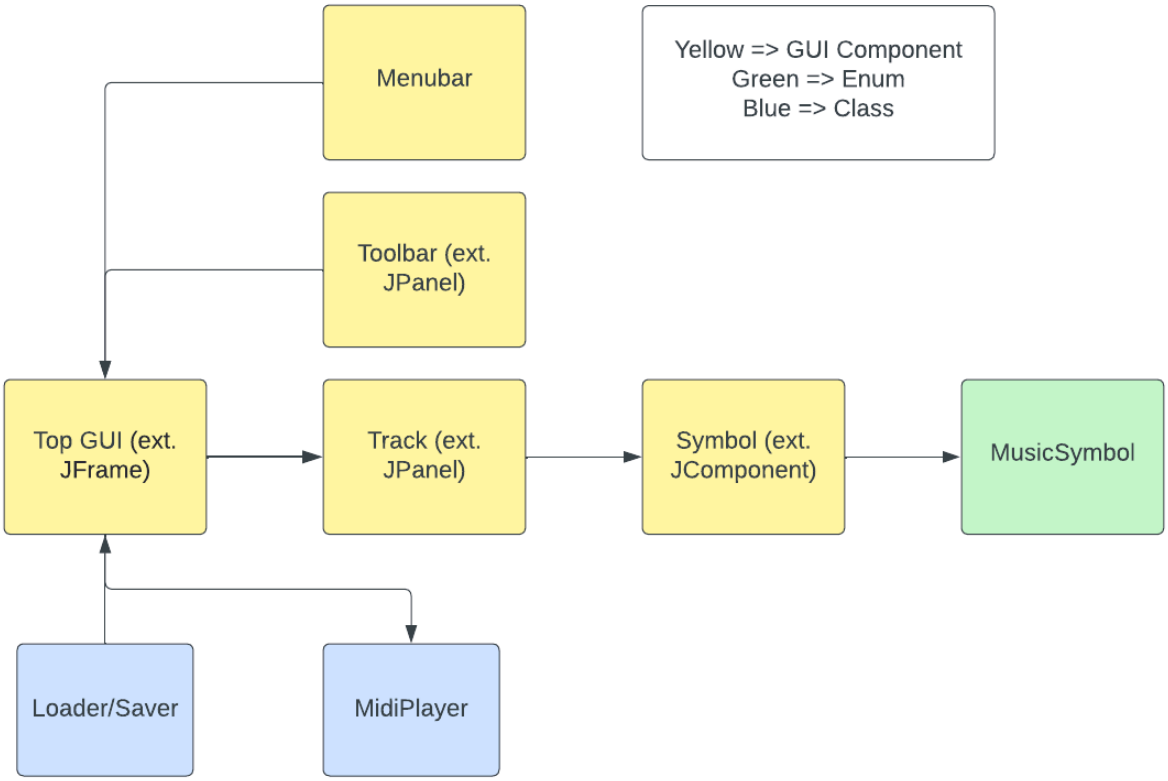
The tool bar will provide all the music note options and functions. These will include choosing note and rests (sixteenth, eighth, quarter, half, whole). After choosing this base note, it will remain as selected until the user changes to another length. Users will also be able to enable a note as dotted (meaning that the note will have its length increased by half). There will also be options to select the key signature, time signature, clef, tempo, and if any accidentals can be added to notes.

Design Manual

Layout Sketch:



UML Diagram:



Appendix

Chat GPT Logs

- Chat GPT Log - Ben Young
- Chat GPT Log - Rahul Prabhu