Chapter Project Report

Introduction: The Simple Music Notation Editor (SMNE) project was created with the goal of providing an easy to use tool for people to make, share, and edit music scores. Our scope was creating a program in which users can place notes and press a button in order to play a simple chord based on the musical score they have created with the note, with additional features as development allows. Our project will create a lightweight music score editor which can be difficult to find in the sea of heavily monetized and difficult to use musical score programs.

Literature Review/Background Study: There have been various other music programs in the past with similar goals as our project. One is flat.io, which we tested to evaluate its strengths and weaknesses when thinking about how to implement our own project. We found that some parts of the UI were awkward and non intuitive such as the play button being in a random corner. There were other aspects that we deemed as well designed such as having rests that would change into notes when a note is placed, and the UI was functional but quite clunky.

Methodology: We started by analyzing other available tools as previously stated. Then we expanded on this by looking at the specifications from the client and seeing what may be possible versus what may not be possible given the time constraints for development. We mainly used Visual Studio Code as our workspace, and worked as a duo to code the project with the help of ChatGPT as our assistant. We also used version control via GitHub and would share changes to each other, while also mainly talking together in live time while working on the specifications for more fluid communication.

Implementation Details: We looked at the starter Java Swing code to get an idea of how to create a similar project. Then we created a framework for the buttons to be mapped to functions much like the buttons in the starter code. The phases of development went similarly to creating a box with measures, then adding the ability to place notes, then having the notes play, while fixing any issues that came up in the process.

We ran into many coding challenges along the way, starting with errors in scaling the window as the notes would shift along the window. We fixed the issue by scaling the window dynamically based on the monitor size. Another challenge we faced was limiting the nodes to only be placed in the proper locations, which we solved by coding the logic necessary to limit placement into the appropriate places on the measures.

Testing and Evaluation: We iterated by testing each change in code to see if any different behavior will occur, and evaluated it by putting it through various test cases to make sure we would not miss anything in the manual review. We often found areas for improvement via our testing, which we would improve on. The software itself ended up robust and reliable, with no instability or crashing.

Results and Discussion: We had good success at achieving the initial achievements as we completed a robust program able to place and play a variety of musical scores. We focused on creating an attractive but clean UI which is a stark contrast to the cluttered UI of the other score creation programs. Future improvements would include the addition of more features while keeping the UI minimalistic as we currently have. Some possible features we left out were things like a giant keyboard UI for placing notes, as we decided that clicking on the score to create the notes felt more genuine. We went for a more modern approach by not having dozens of useless buttons as it was a key detriment we saw in the other musical score applications.

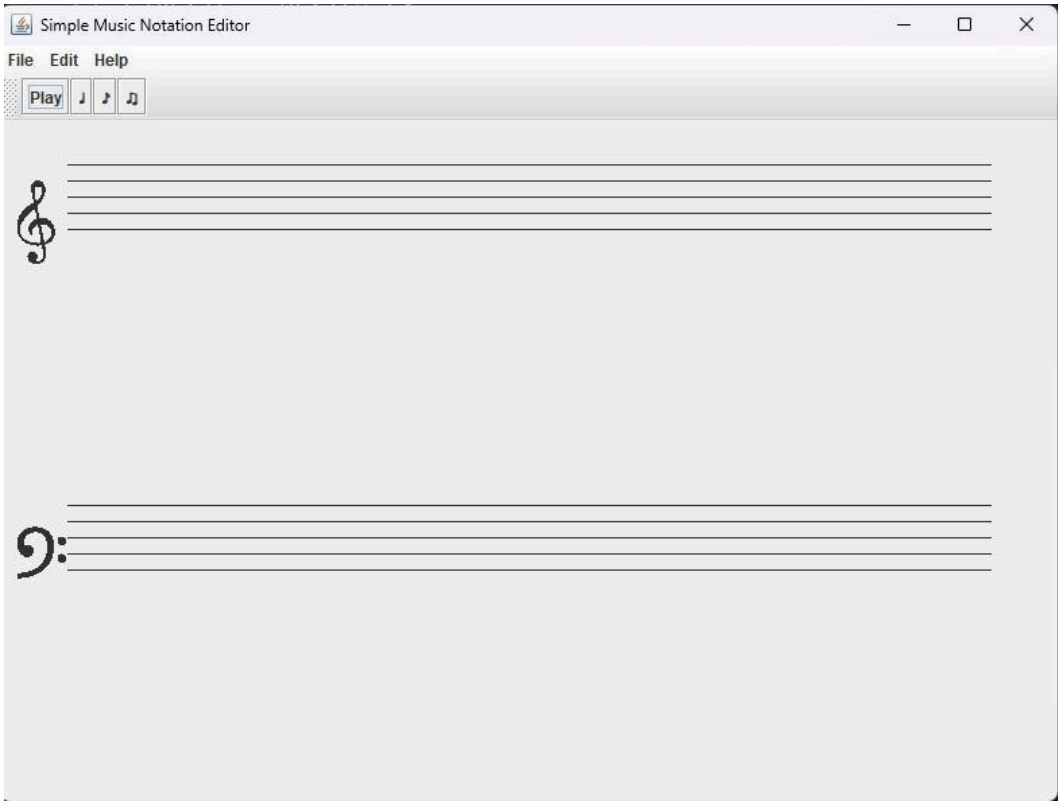
Conclusion:

We achieved the creation of a lightweight and simple Java swing app for the creation and playback of musical scores, which was our main goal. We learned about many unexpected problems that can come up during development and how to deal with them appropriately.

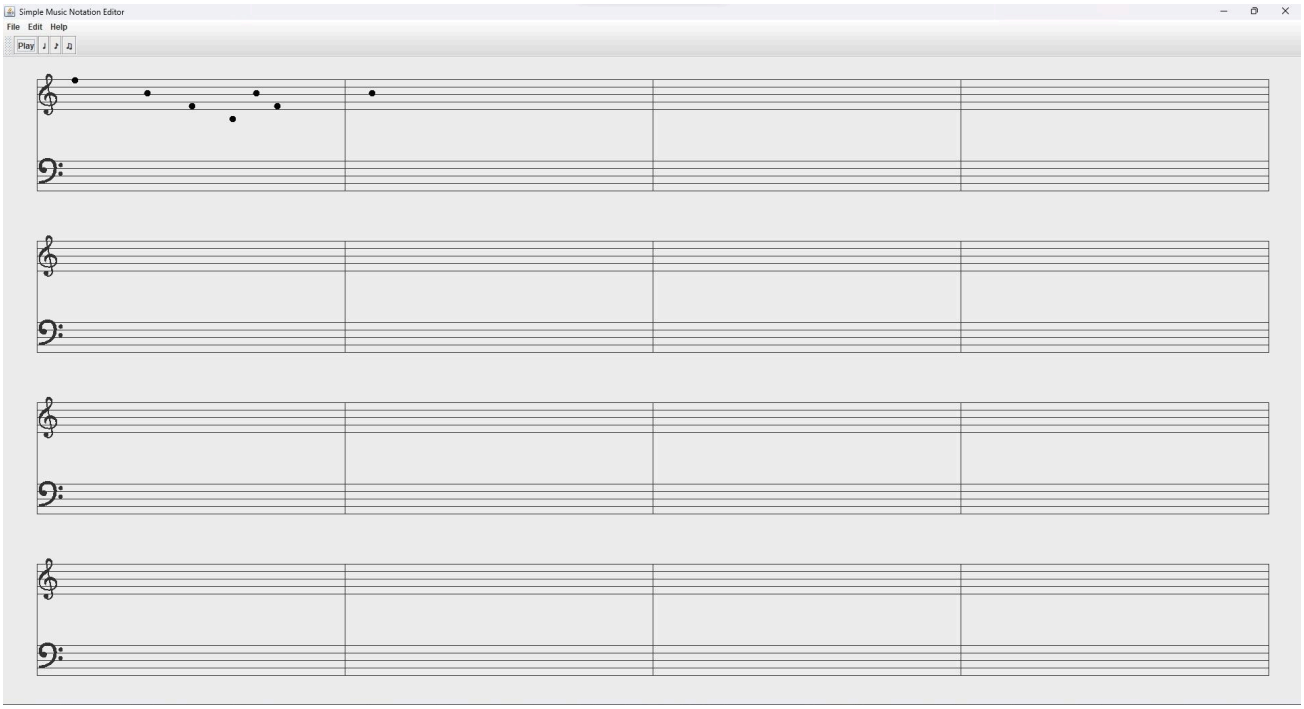
References and Appendices:

Flat.io was one of our references for investigation, as well as the course material provided from ECS160.

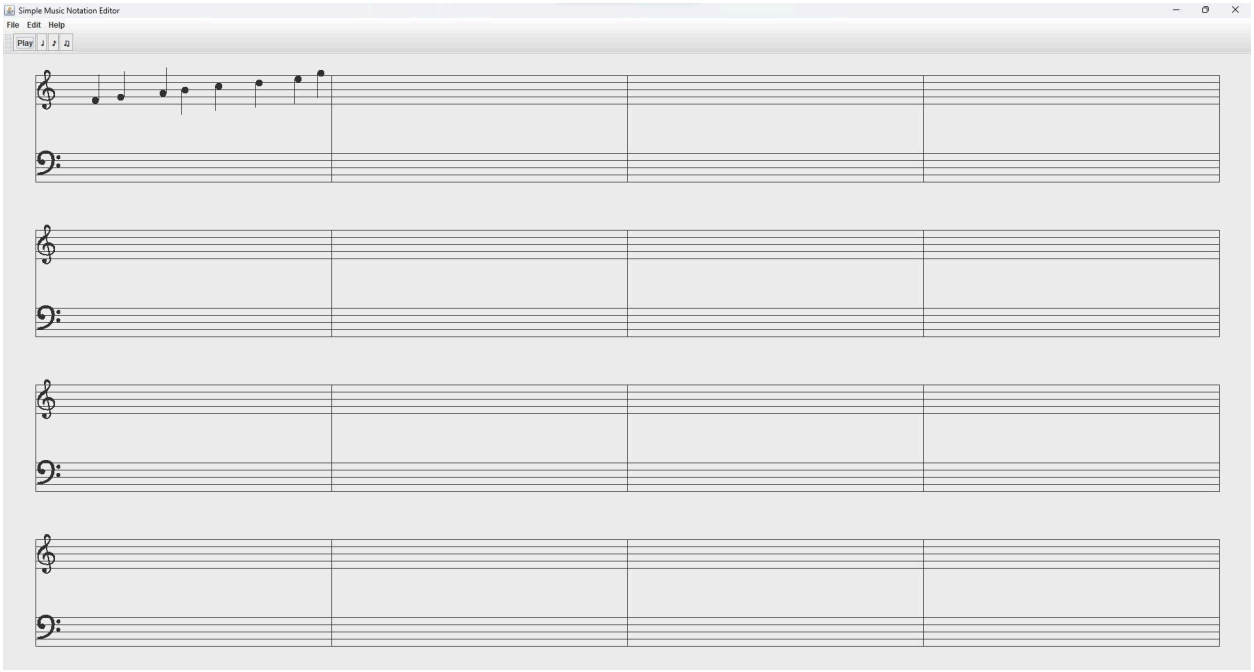
Week 1 UI progress.



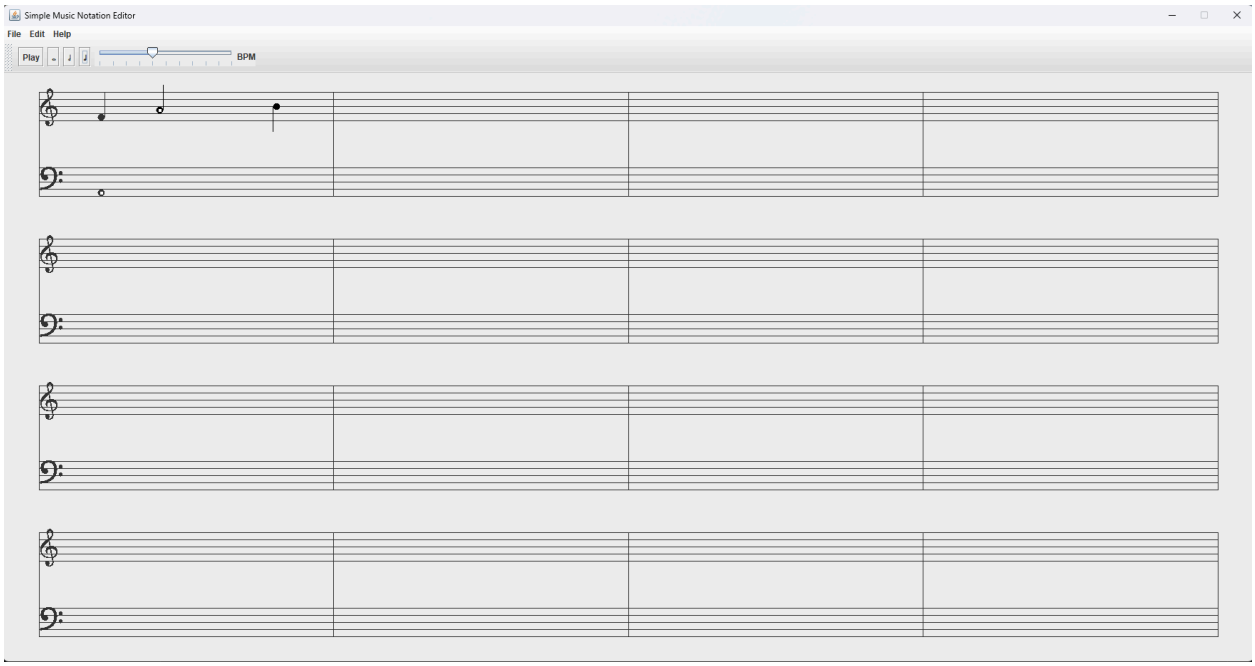
Week 2 UI update + basic functionality



Week 3 Note functionality



Week 4 Playback functionality



Chapter User Manual

Introduction

Welcome to the Simple Music Notation Editor User Manual! This guide is intended to help you effectively utilize the software to create and edit musical scores. Whether you're a music student or educator, this manual will walk you through the features and functionalities of the application.

Installation Instructions

To install the Simple Music Notation Editor, follow these steps:

1. Install the latest version of Java.
2. Download the application from the GitHub repository.
3. Run the executable.

====

User Interface Overview

Upon launching the Simple Music Notation Editor, you will encounter the following components:

- **Musical Staff Display**: Represents the musical staff where notes are placed and edited.
- **Toolbar**: Menu that allows users to to select different note durations before placement. Also allows users to change the time signature and key for the composition.
- **Play Button**: Initiates playback of the composed music.

Adding and Editing Notes

- Click on the staff to add notes.
- Click the note again to delete.
-

Playback

- Click the “Play” button on the top to play the composed melody with synthesized sounds.
- Use slider to change BPM.

Toolbar

- Use the toolbar to select from different note durations like whole, half, and quarter notes.

Troubleshooting

If you encounter any issues while using the Simple Music Notation Editor, consider the following troubleshooting steps:

- Ensure that your system meets the minimum requirements for running the application.
- Check for any error messages displayed by the application.
- Verify that Java is installed and up to date on your computer.

Frequently Asked Questions (FAQ)

Q: Can I save my compositions in the Simple Music Notation Editor?

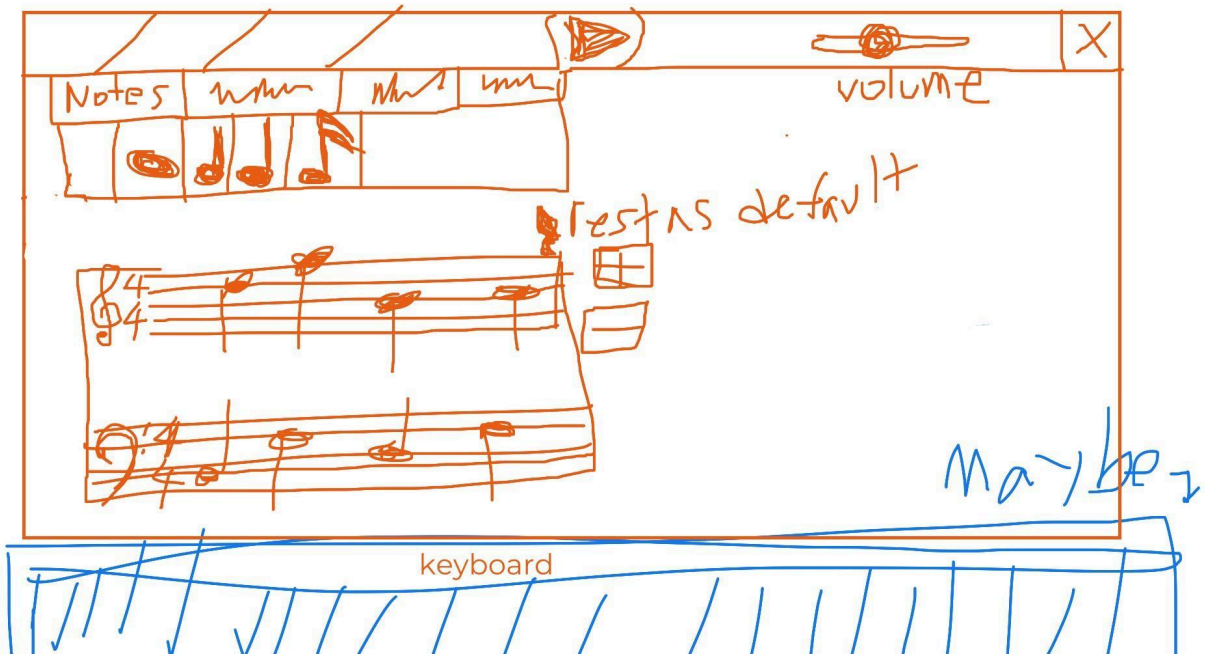
A: Currently, the application does not support saving compositions or exporting audio files.

Q: How can I change the instrument sound?

A: The application currently only uses a default synthesized sound for playback.

Chapter Software Design

Basic Sketch of User Interface



User Stories:

- As a musician I want the ability to drag notes to change pitches so that I can visualize and edit compositions easily.
    - Add a feature to drag notes to desired pitch and playback that note.
- As a musician I want to be able to replay key sections of my composition so that I can ensure I find the perfect sequence to complete my piece.
    - Add a “play” button so that the composition’s notes can be turned into a melody.
    - Add a looping feature so that key parts of the composition can be focused on.
- As a musician I want to have the freedom to create any composition so that my music is unique and interesting.
    - Add a toolbar to change the duration of the notes.

    - Add a feature to change time signatures and the key of the composition so users have more options to create.

The class structure of the Music Notation Editor application is designed to facilitate the creation, manipulation, and playback of musical notes on a staff. It follows a Model-View-Controller (MVC) architecture, which promotes separation of concerns and modularity. Let's discuss the key classes and their roles within the application:

Class Structure

Model Classes:

1. Note:
    - Represents a musical note with properties such as pitch, duration, and position. Contains methods for setting and retrieving note attributes.
2. Staff:
    - Manages a collection of notes and provides methods for adding, removing, and modifying notes.
    - Responsible for maintaining the arrangement and layout of notes on the musical staff.
3. Synthesizer:
    - Handles the playback functionality by synthesizing instrument sounds based on the composed music.
    - Converts musical notes into audio signals for playback.

View Classes:

1. MusicalStaffPanel:
    - Displays the musical staff on the user interface.
    - Handles user interactions such as adding, moving, and removing notes on the staff.
    - Observes changes in the Staff model and updates the display accordingly.
2. Toolbar: Contains controls for selecting note durations and initiating playback. Provides user-friendly access to various functionalities of the application.

Controller Classes:

1. NoteController:
    - Mediates interactions between the View and Model components.
    - Listens for user inputs from the View and updates the Model accordingly.
    - Notifies the View of changes in the Model, ensuring synchronization between the user interface and underlying data.

Design Considerations:

1. Separation of Concerns:
    - Each class is responsible for a specific aspect of the application, promoting maintainability and extensibility.
    - The Model classes encapsulate the data and business logic related to musical notation.
    - The View classes handle the presentation and user interaction aspects of the application.
    - The Controller classes manage the flow of data and user actions between the Model and View components.
2. Encapsulation:
    - Classes are designed with well-defined interfaces and encapsulate their internal implementation details.
    - This allows for easy modification and enhancement of individual components without affecting the overall system.
3. Flexibility and Scalability:
    - The class structure is designed to accommodate future enhancements and extensions to the application.
    - It allows for the addition of new features, such as support for different musical instruments or advanced notation symbols, with minimal impact on existing code.
4. Abstraction and Reusability:
    - The use of design patterns such as MVC promotes abstraction and reusability of components.
    - Classes are designed to be modular and reusable in other parts of the application or in future projects.

Appendix

ChatGPT Logs for Ryan Tan and Noel Lee (Shared ChatGPT account):

- Week 1
    - **gpt0221.pdf** Asked ChatGPT to create a user guide manual based on the prompt given. I then modified it to add features we would like to implement that would be useful.
    - **gpt0222.pdf** Used ChatGPT to generate UI code for this application. Generated the unicodes but really did not want to place the clefs on top of the staff so still working on that part.
- Week 2
    - **gpt0229.pdf** Asked ChatGPT to generate bar measures. I took this code and changed it to match how a staff should look. I then asked GPT to give me code so I can add notes to the staff on mouse click. The proper logic is not implemented yet but notes appear on the staff and are generated to the closest staff line.
- Week 3
    - **gpt0312.html** Asked GPT for help with placing notes. Ended up having to code it myself using a treemap to map the staff y coordinates to pitches. Worked on having the quarter notes drawn instead of using the unicode. Have the stem direction of notes face upwards or downwards depending on where they are placed. Notes are given a pitch where they are placed. Working on playback functionality.
- Week 4
    - **gpt0321.html** GPT helped with using the MIDI synthesizer to play notes. Had GPT make a slider to change BPM.