

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.

Literature Review/Background Study: #todo

Methodology: Describes the development methodologies and tools used in the project. #todo

Implementation Details: We began by analyzing the starter Java Swing code to get an idea of how to create a similar project. We created buttons to be mapped to functions much like the buttons in the starter code.

Provides an overview of how the project was implemented, including key phases of development, technologies used, and significant coding challenges and solutions.

Testing and Evaluation: We iterated by testing each change in code to see if any different behavior will occur, and *created thorough test cases to make sure we would not miss anything in the manual review.

Discusses the testing strategies employed, test cases, bugs found, and the overall performance and reliability of the software. Note that for school projects, these are not often formal.

Results and Discussion: #todo

Presents the outcomes of the project, analyzes its success in meeting the initial objectives, and discusses potential improvements or future work. A key element of this section is to outline the feature choices that you made for your project and why, as well as some that you decided not to implement and your reasons for leaving that for future work.

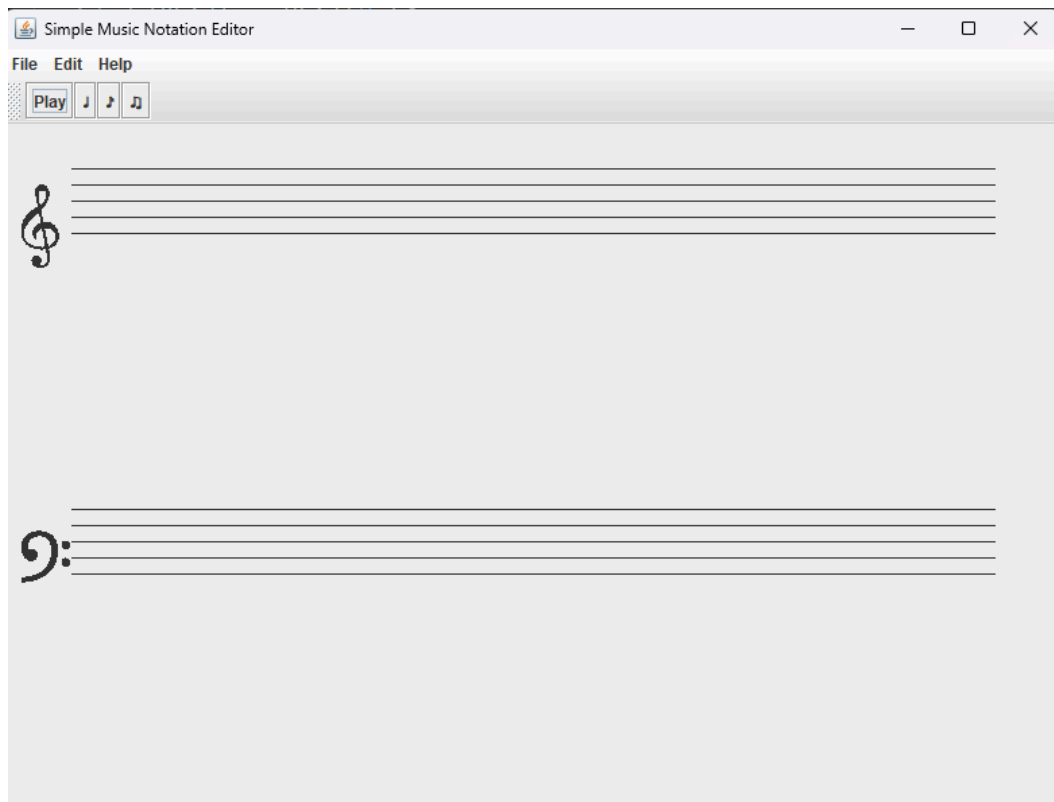
Conclusion: #todo

Summarizes the project's achievements and learnings.

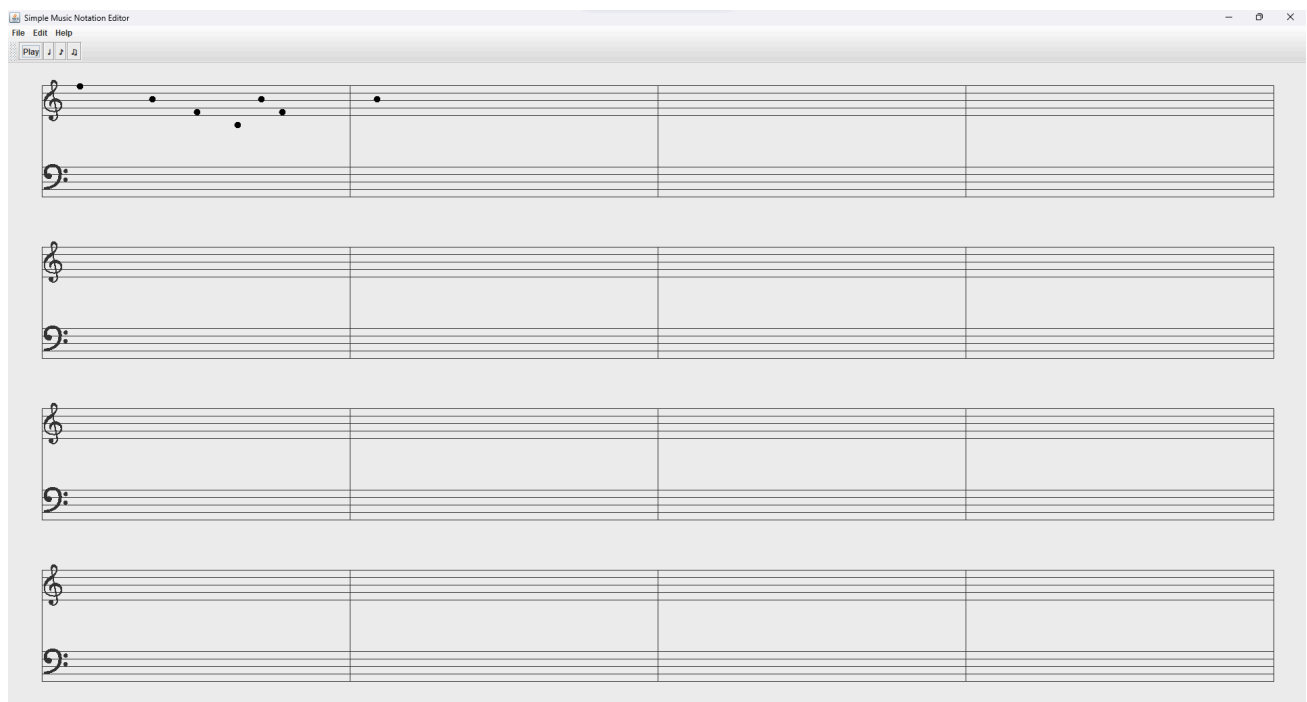
References and Appendices: #todo

Includes citations of sources referenced in the report and any supplementary material such as code listings, additional data, or user manuals.

Week 1 UI progress.



Week 2 UI update + basic functionality



Week 3 Note functionality

Simple Music Notation Editor

File Edit Help

Play 1 2 3 4

The screenshot displays the Simple Music Notation Editor application. The interface includes a title bar with the text 'Simple Music Notation Editor' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', and 'Help'. A toolbar contains a 'Play' button and four numbered buttons (1, 2, 3, 4). The main workspace features four systems of musical staves. Each system consists of a treble clef staff and a bass clef staff. The first system is populated with musical notation: the treble staff contains a sequence of notes (quarter, eighth, quarter, eighth, quarter, eighth, quarter, eighth) and the bass staff is empty. The remaining three systems are empty.

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 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.
- Drag existing notes to change their pitch or position.
- Click the note again to playback.
- To delete a note highlight the note and use the backspace key.

Playback

- Click the “Play” button on the top to play the composed melody with synthesized sounds.
- To enable looping, highlight the section of the staff with the start and end markers.

Toolbar

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

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?

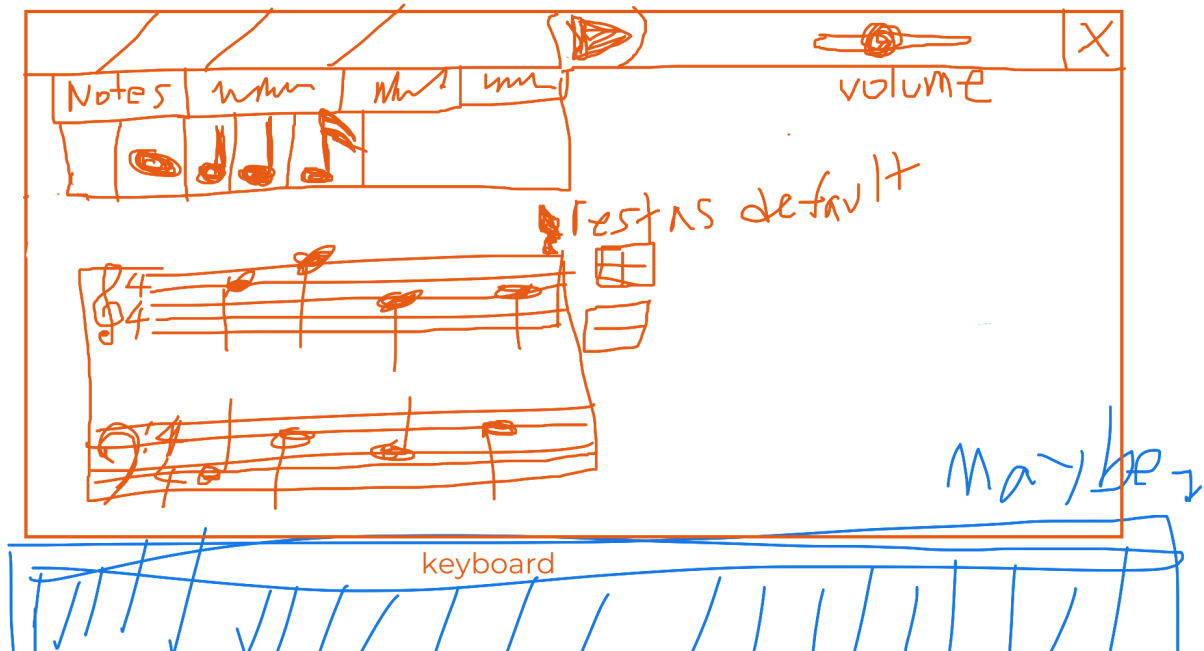
A: Currently, the application does not support saving compositions or exporting audio files.

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:

- Week 1
 -  ryan0221.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.
 -  ryan0222.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
 -  ryan0229.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
 - ryan0312.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.

ChatGPT Logs for Noel Lee:

None as of moment