MEscore

Jack Pulsfort

05/09/2023

Abstract

MEscore aims to provide a bridge between the world of music and software development by means of a music notation software built and run using Python.
The overall objective for the project is to create a functional basic music notation software with an appealing and easy to use GUI. In order to achieve this I will have the ability to add notes onto a traditional grand staff, change the tempo of the composition, edit or delete notes as the user is working, and play the composition.
The first step was to make a command line based prototype that allowed the user to type in notes one at a time and then type “play” to hear their composition. This used a simple list of strings to represent each note and the Beep function from the winsound [1] module to play the notes using a sine wave. Next I developed a GUI that relied on the functionality of the prototype but used buttons to add notes to the composition. I used CustomTKInter [2] to build the GUI components and the final version is written to run almost completely independently from my original prototype, having most of the program functions be contained within the GUI itself. I have used the mingus and fluidsynth modules to allow the program to store notes, chords, bars, and tracks so the user can play their compositions using a piano sound [3][4].

Keywords
GUI, mingus [3], Python, fluidsynth [4], note, composition

Table of Contents

List of Tables and Figures

Introduction and Background

Problem

The need for composers to have a way to compose music using a computer rather than needing to write it out by hand.

Objectives

Allow users to input notes of varying length (whole note, half note, quarter note, etc) using a visually appealing and practical GUI onto a traditional music staff to create their own unique compositions.

Users

Composers, music students, or anyone who desires to write music using technology.

Background

This project is intended to be used by someone who at minimum knows how to read a musical score. An understanding of the basic concepts of music theory, mainly the general rules of basic musical composition, would be a great tool for the user to have but is not required in order to use MEscore. Music notation software has been created several times through softwares such as Musescore, Noteflight, or Sibelius. There are plenty of other music notation softwares available, these are just a few examples that are very popular. The strengths of these projects come from having a large dev team behind them to enable faster implementation of new designs or features. However having a large dev team and a global network of users does limit the amount that user feedback is taken into account when implementing these features.

Value

MEscore is a much smaller scale project than the commercially available music notation softwares and therefore I will be able to take feedback directly from the users and testers regarding features that they would want to see added or changed which promotes better user interaction with the software myself as the developer.

Scope

MEscore will aim to address some of the more specific problems users of other music notation softwares may have. For example, the software Musescore is unable to play some of the articulations that the user inserts onto the page such as a scoop (going from a lower pitch and "sliding" up to the pitch that is written after the scoop) and many composers who do not have access to a group of instrumentalists to play their music for them will be unable to hear how those articulations sound.

Features

The user should be able to add any of the 12 music notes and their variations/equivalences (such as Eb is equal to D#) This is done by selecting a note length and then clicking a position on the staff. Then the proper note length will be added to the list. This note will then be placed on the staff where the user clicks and will be able to play the proper pitch for that specific note value.

A music staff to "hold" the notes where the user will click to insert notes. It will display the time signature, key signature, and the notes that the user inserts. When the user presses the play button, the program will start at the beginning of the composition and play what the user has written.

The ability to change the key signature using a key signature button. When clicked, the user will be able to select from a list of the standard 15 key signatures. This will then change the key signature to display the correct sharps or flats for the selected key signature. For example, if the user selects the key of Bb, the signature would display a flat symbol on the line for B and for E (the key of Bb is Bb and Eb)

The ability to change the note length by selecting from 6 different note lengths, whole note, dotted half note, half note, dotted quarter note, quarter note, and eighth note. The user can then add notes of any pitch to the staff and it will display the correct length. For example if the user clicks the half note button and then clicks the line for B (on the bottom bass clef staff) the program should display a quarter note on the line for B in the staff and play that note when the user presses play.

The ability to select an accidental for the note (sharp, flat, or natural) using 3 different buttons for sharp, flat, or natural. By default the accidental is set to natural and to change it, the user would click one of the accidental buttons before inserting a note to change which accidental is associated with that note. For example, the user could click quarter note and sharp, and then click the line for F (in the bass clef staff) to place an F# quarter note on the page.

The ability for the user to edit notes within their composition without having to delete the note or restart their composition. This feature is implemented using left and right arrow buttons to select the measure containing the note that needs to be changed. Once the measure has been selected, the arrow buttons are used to navigate between the notes in that measure. The user will then be able to change this selected note to a different note of the same length.

Check each measure (referred to as a Bar in the mingus.containers module) as the user adds notes to ensure that the note being added can fit into the measure [3]. This is done by having each measure start out empty by default and as the user adds notes onto the page, check the length of the note the user wishes to add and ensure it is less than or equal to the amount of space remaining in the measure. For example, in a 4/4 time signature, if the user inputs a quarter note into an empty measure and then tries to add a whole note to that same measure, the program will display an error message stating that the note being added is too long. This message will disappear once the user adds a note that will fit within the measure.

The ability to play the composition and have the correct pitches play. This is achieved using the mingus note and note containers framework [3]. With this framework, notes are stored as a Note object which can be edited and placed into a bar. This bar can then be added to a track or composition which can be played on a variety of synthesized instruments using the fluidsynth [4] library and an instrument library file. Each note the user places has a corresponding hertz value and a Note object is constructed from that information using the Note.from_hertz function. This note is then added to the current Bar which, once full, is added to a global Track object that is used to play the composition. When the user clicks play, the play_Track function from the fluidsynth [4] module is used on the track to be able to play each specific note (at the correct pitch) for the length of time the user had specified for that note.

Design, Development, and Test

Design

MEscore is a desktop application written entirely using Python.
To facilitate the creation of a GUI, the storage of music notes, and the playback of music notes, the following external modules were used:

The CustomTKInter [2] module was used to create the GUI. The GUI consists of buttons for note length and accidentals, a textbox to enter a new tempo, a button to change to the new tempo, a textbox to enter a new octave, buttons to enable the editing of notes in the composition, a section to display the composition with notes in a text-based format, and a staff used to insert notes into the composition.

The mingus and fluidsynth modules are used to store the composition as a collection of Note objects as well as enabling the playback feature of the program [3][4]. Given an instrument library to use, the mingus module uses the play_note() function from fluidsynth to play the given note on the given instrument in the library (by default this is a piano) [3][4].

Development

Risks

The first and most obvious risk is my inexperience with building a GUI in Python. I have never built a GUI in Python before and therefore I anticipated some problems arising when trying to link the GUI to the underlying code that runs the application.

Similarly, due to my inexperience working with the mingus [3] module resulted in many issues with my project and eventually required some parts (such as how notes are formatted) to be rewritten.

Minimizing Risk

After researching a few different libraries to create a GUI in Python, CustomTKInter [2] was chosen as it is very simple, easy to understand, and easy to implement, all of which helped reduce the risk associated with my lack of Python GUI experience.

The risk with using mingus mainly came from the prospect of needing to rewrite the way my program handles user input in order to fit mingus's specifications [3]. To mitigate this I used built in functionality

provided by the mingus module to change my input format (purely strings) to one that works for mingus (Note objects) [3]. This helped me avoid rewriting some parts of the program while also making it easier to update the program in the future.

Test

Table 1 below illustrates the methods used to test each individual feature outlined in the Features section on pages 6-7

Feature Being Tested	Method of Testing
Input of notes into the composition	As the user adds a note, print to the command line the parameters of that note (Note Length, Hertz value, and Note Name) and the current bar before and after adding the note. This was tested using all different combinations of note, length, and accidental and any inaccuracies were recorded and fixed.
Editing notes in the composition	Print the current bar as the user presses the left arrow and right arrow buttons. Once the user presses the Select Bar button, when the arrow buttons are clicked, print the current note in the selected bar to check for accuracy. This was performed for a variety of cases including edge cases such as the user attempting to edit an empty composition. Any inaccuracies were fixed as they were found.
The playback feature	This feature was tested using a variety of regular cases (such as Hot Cross Buns to test for note playback accuracy) as well as edge cases such as an empty composition or a composition with an unusually large amount of notes. A third party application (Musescore) was used to test the accuracy of the playback feature. A test case (such as Hot Cross Buns) was deemed as passing if the playback from MEscore produced the same result as the

	playback when the same composition was entered into Musescore.
Changing octaves	To ensure the octave selection box works properly, an A natural was added to the composition at each octave 1-6 which would print out the corresponding Hertz value for an A natural at each of those octaves. These values were then checked against the expected Hertz values of an A natural at each octave 1-6.
Changing tempo	This was tested by creating a composition at the default tempo of 120 bpm and then changing the tempo first to 240 bpm where the length of each note would sound half as long as before if the tempo change worked properly. This method was then repeated with different tempos to ensure that the note lengths changed by the correct amount each time.
Removing notes from the composition	This feature was tested by adding many notes to a composition and then deleting them 1 by 1 and testing the play function to ensure that the composition still sounded correct and that the proper note was indeed removed. This was then repeated with different amounts of notes being removed each time before playing the composition.

Table 1

Results

Table 2 below illustrates the results of the project organized by feature. It outlines whether the feature was implemented and if so which objectives were met.

Planned Feature	Implemented or Not Implemented	Objectives Met
Add notes to the composition	Implemented	The user is able to add all 12 notes (including equivalences as described in the Features section) into the composition without error. All objectives for this feature were met
Music staff	Implemented	The program has a music staff that the user can interact with to insert the appropriate notes. Most objectives for this feature have been met however the music staff does not actually display the notes that are entered and has therefore failed to meet that objective. It is purely used for input and that functionality meets all objectives.
Change key signature	Not Implemented	None of the objectives as given in the Features section were met as this feature was not implemented for this version of the program.
Change note length	Implemented	The user is able to use a set of radio buttons to select from the 6 different note lengths listed in the Features section. This feature is fully functional and has met all objectives.
Change accidentals	Implemented	Users can select from 3 different accidentals, namely flat, natural, or sharp, using a set of radio buttons before adding a new note to the composition (or whilst editing a note that is already in the composition). The new added note will then have the proper accidental associated with it. This feature is fully functional and has

		met all objectives.
Editing notes	Implemented	The user can use the arrow buttons to select a measure and then a note within said measure which can then be changed to any other note of the same length. This feature met all objectives and is fully functional.
Ensuring measure validity	Implemented	This feature is performed behind the scenes and displays an error message any time the user tries to add a note that will not fit in the current measure. The note is not added to the composition and the message remains on the screen until the user changes the note length to one that will fit in the measure and then adds a note of that length. This feature met all objectives and is fully functional
Playback of the composition	Implemented	The playback is accomplished by using mingus tracks to store the notes and fluidsynth in combination with an instrument library to play the notes on that track using the instruments in the library [3][4]. The feature is fully operational and has met all outlined objectives.

Table 2

The final result of MEscore is a music notation app built entirely using Python and Python modules. The program consists of a user-friendly interface that allows the user to select an accidental of either flat, natural, or sharp, and click the lines and spaces of a traditional grand staff and have those notes be added to the composition. The user is able to see the notes appear (in a string format) as they add to their composition. There is a textbox to change the tempo as well as buttons for editing notes and a button for removing the last note from the composition. Once the user is happy with their work they can click the play button to hear a synthesized playback of their composition.

A few notable issues came up over the course of the project which needed to be addressed for the program to function properly. First among these was an issue with the displaying of notes when a note was deleted from the composition. Initially, there were cases where the note would disappear from the display but still be present in the composition itself. This issue was quickly resolved by changing the way the display is calculated so that it was always directly tied to the list of notes in the composition itself. The second major issue arose while implementing the edit notes feature. The initial implementation of the editing feature changed the track variable from a Track object to a List and therefore after the user edited a note everything appeared to function until the user tried to make any other changes (adding/deleting/editing a note) to their composition at which point the program became non-functional and had to be restarted. This issue was quickly resolved by simply converting the track back to a Track object any time the user makes an edit to the composition. The last major issue came from making the transition from a string and list based system to store notes to the Notes Container system provided by the mingus module [3]. This transition resulted in many of the features of the program breaking and having to be partially rewritten or added onto for the program to resume normal functionality.

Conclusions and Future Work

MEscore addresses the modern day issue of composers requiring a way to write and hear their musical compositions without needing to have access to a piano or an orchestra to hear their work. To solve this issue, I have created a musical notation desktop app using Python to enable composers to create digital compositions that they can add to, edit, and listen to as they see fit. To support the features of this application, I used the mingus and fluidsynth modules which allowed for the storage of Note objects as well as the playback feature [3][4]. I also used the CustomTKInter module to create a GUI for the user to easily interact with the application. The result of this project is a robust and self-contained music notation application with a simple and user friendly GUI built entirely in Python.

After completing this version of the project I gained a much better understanding of the Python programming language as well as the software engineering lifecycle as a whole. My initial predictions of what I would be able to accomplish in a three month timespan were very optimistic and this taught me a valuable lesson about project planning, management, and most importantly scope definition. Especially in the beginning of the project I had extremely high ambitions for the project and subsequently had unrealistic expectations for myself which this project has helped me immensely with finding my own limitations. While not everything that I had originally hoped for, the resulting version of my project is a very well built standalone application that in the end still met almost all of my initial expectations and is something that I can be very proud of.

The current version of the project would not be the best option for a composer looking to quickly record their ideas as there is currently no way to save or export a given composition. However I think MEscore in the current version could serve as a great resource to teach children about the basics of music and provide some form of interactivity to keep them engaged and help inspire a love for music in their young minds. This would especially be useful for schools or programs that do not necessarily have access to a piano or instruments for the children to be able to hear the notes.

Future improvements for MEscore would begin by completing the implementation of all unfinished features that were intended to be completed for this project. After that the next step would be to edit the way the grand staff functions so that it can actually display the notes as the user enters them which would enable the program to function at a level that is much closer to many of the alternatives that are currently commercially available. Once that is complete the next step would be to add both Save and Load/Open capabilities to the application so that composers can save their work and pick back up again where they left off. Finally, the project would continue to evolve as new musical features such as accidentals, dynamics, rests, repeats, etc. are added.

References

[1] Anon. 2001. Winsound - sound-playing interface for Windows. (2001). Retrieved May 8, 2023 from https://docs.python.org/3/library/winsound.html

[2] Tom Schimansky. Tomschimansky/customtkinter: A modern and customizable python UI-library based on Tkinter. Retrieved May 8, 2023 from https://github.com/TomSchimansky/CustomTkinter

[3] Bart Spaans. Mingus¶. Retrieved May 8, 2023 from https://bspaans.github.io/python-mingus/index.html

[4] Tom Moebert, Jean-Jacques Ceresa, and Marcus Weseloh. 2020. Documentation. (2020). Retrieved May 8, 2023 from https://www.fluidsynth.org/