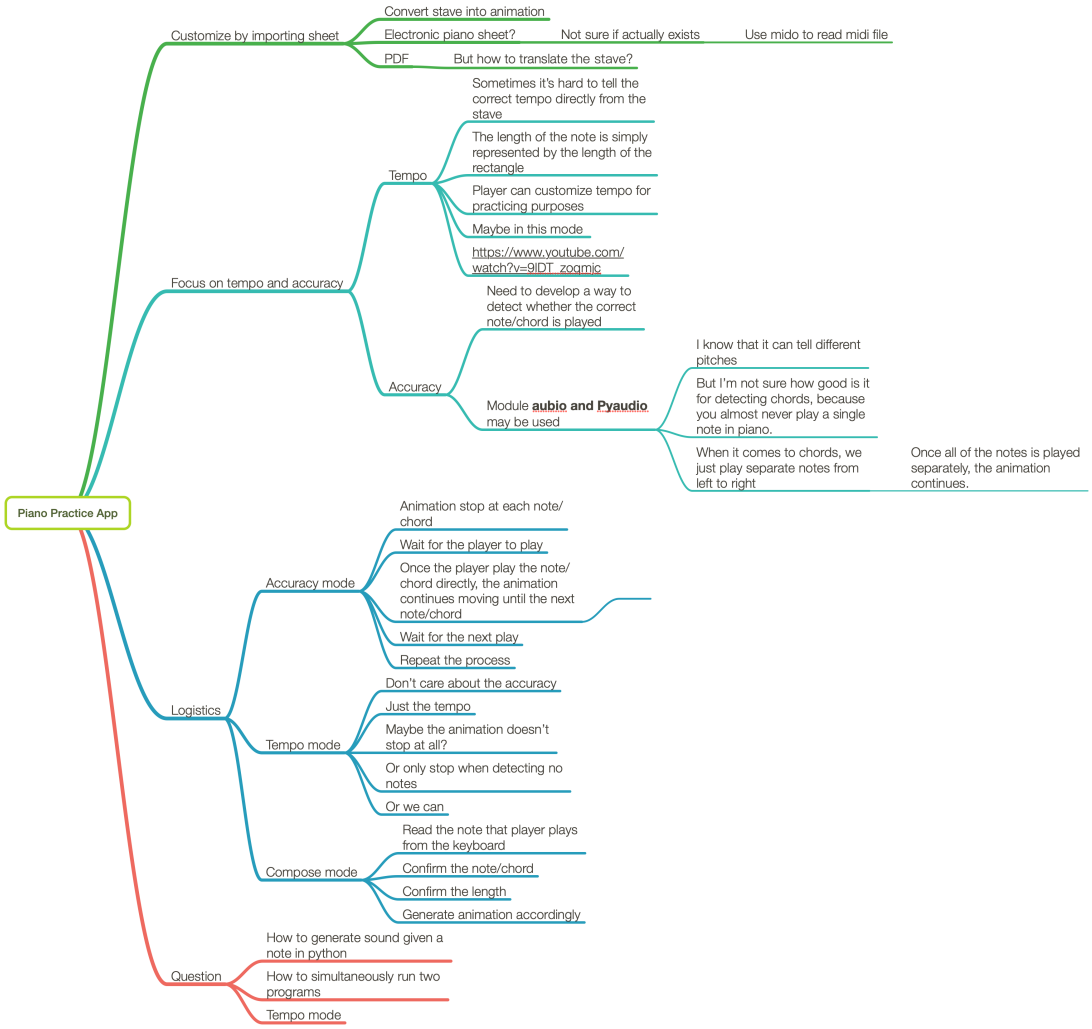
CMU 15-112 TERM PROJECT



Project Proposal

Prepared for: Anyone who will be seeing this

Prepared by: Ziyao Luo (Vincent), a student from F19 15-112

November 19, 2019

PROJECT DESCRIPTION

Introduction:

This project is named **Piano Tile** (for now). It is a practice app for recreational piano players to learn to play any customize script without the actual physical stave. Users can simply download any MIDI[1] file from the internet as import. As a result, this app will generate corresponding piano tile animation for users to follow and practice.

More In-Depth:

Ideally, the app will ultimately include three modes: listening, practicing, and composing mode.

Listening mode is where users simply listen to the file they import along with the generated animation. Users can get a general sense what the melody looks like.

For practicing mode, this app will mainly focus on accuracy (to help player find the correct note/chord to play) and the tempo (to ensure that the player is playing at the right pace).

(TBD) In composing mode, player use their instrument (preferably a well-tuned piano) to create any music they want, and the program generates corresponding animation and MIDI file as export.

COMPETITIVE ANALYSIS

#1: Tile Animation Generated Using External Devices

Refer to this video: https://www.youtube.com/watch?v=XUzwdBQDzxw. In this video, it looks like the player connected some external device onto his piano. The device detect each key he plays and record the length of keypress to generate the entire animation.

Although the animation is what this app aims to generate, it is only an animation without any interactive elements. On the other hand, the focus of this app is more on the interaction with the user and guide the users to play correctly, which the animation in the video fails to achieve.

[1] Musical Instrument Digital Interface: https://en.wikipedia.org/wiki/MIDI

#2: Musescore App:

Refer to this link: https://musescore.com/user/11989916/scores/5549583/piano-tutorial. Musescore is an app everyone to both compose and practice music. Users can upload their own script as well as MIDI file onto this website. The website seems to have preset algorithms to generate corresponding animations for every piece of music. However, it still does not interactive with players and does not help them to practice.

Overall, the advantage of this app that this proposal presents is that it is not just an animation generator. More importantly, it is really an app that help users to practice playing piano in an interactive way. In other words, it not only includes the features of the existing products, but develops further upon it. Also, this app does not require any external devices except for the computer that will be running it.

STRUCTURAL PLAN

File Reading And Initialization

Each class below will be written and tested in a separate file before finalization.

A. Object sheet
   1. Use module mido to read MIDI file
   2. Extract and organize useful information as preparation for generating the animation

B. Object keyboard
   1. Draw a keyboard according to a given height and width
   2. Store the positions of every single key so as to match the falling tiles

C. Object stream
   1. Use pyaudio and aubio to open up a stream for sound recording
   2. Write the recording part as a method of this class
   3. Call this method in the animation framework whenever needed to detect the target note.

Animation

A. Listening mode
    1. Animation keeps going non-stop
    2. Whenever a tile hits the key, play the corresponding note(s)
    3. Problem is that it will probably be a trouble to run the animation while playing the audio only using cmu_112_graphics.
    4. Pygame might be applied after MVP is reached?

B. Practicing mode (MVP target)
    1. Accuracy
        a. Stop the animation whenever a tile or multiple tiles hit the keyboard
        b. Need a method to detect collision here
        c. Apply the record method of the stream class to detect target note
        d. Animation continues when the note is detected
        e. Detecting multiple notes at the same time (chords) might be tricky to solve.
    2. Tempo
        a. This will probably looks more like listening mode. It's just there is no sound playing. Instead, a beat buzzer or some kind of indication for tempo might be applied so that the player will be able to know that whether they're on the right pace.
        b. We don't care about accuracy here.
        c. Problem is that since the animation is generated according to the file imported in real-time, so the tempo of the animation will not reflect the actual tempo. Testing shows that there will be approximately 3 seconds of delay per minute.

C. Composing mode
    1. Use stream class to detect note being played
    2. Confirm note
    3. Confirm length
    4. Generate file and animation

ALGORITHMIC PLAN

Generate Animation Corresponding To Specific File

Instead of importing a picture of the script of analyzing PDF. A MIDI file seems to be more omnipotent. A slight difference in the actual script will result in a huge difference in reality. However, the format of the MIDI file is relatively neat and convenient: it simply tells you that at what time which notes should be played.

So, we can easily extract and organize these information for our animation. In this project, a MIDI file is processed so that all of the keys played in a piece of music is put in to a list of commands. Each command is a tuple of 3 elements: (key, press/release, time to wait).

To go through the command, we simply loop over the list

Dealing With Chord (Multiple Notes To Generate At Once)

If we are to process only one command at time, there will be a problem when generating a chord. It looks like the multiple notes being played is not generated at the same time, because python needs time to do everything (get a new command from the list, execute the command, draw the actual tile). These little bits of time adds up to a significant amount.

One way to improve this is to execute multiple commands (command that will be executed at the same time) at once. This can be achieve by using a while loop when extracting commands out of the command list. Stop extracting when the command is executed at the next time spot.

Running Stream Class Without Freezing The Animation

In accuracy practice mode, this won't be a problem because the program opens up the stream when the animation is paused, so these two functions doesn't need to be run simultaneously.

However, in listening mode and tempo practice mode, we need to play/record sound continuously while the animation is moving. One way to solve this is to put the stream in timerFired and only play/record for a short amount of time each time it's called. This would work but both the animation and the stream will slow down and thus will be weird.

TIMELINE PLAN

TP0 & TP1

- Finish testing all of the modules
- Finish writing all of the class in this version of the proposal (11/19)
- Be able to put the class together to generate customize animation

TP2

- Finish writing accuracy practice mode
- Start to work on tempo practice and listening mode
- Start to work on improving UX
- Hopefully can reach the MVP

TP3

- Try to implement all three modes, which is almost impossible
- So instead, improve UX on the existing modes.

VERSION CONTROL PLAN

Back up TP folder to google drive on a daily basis. Keep all of the versions. Also update separate files that contains different parts of the project.

MODULE LIST (OUTSIDE 112)

- Mido
- Pyaudio
- Aubio
- Numpy

TP2 UPDATE

Status Update

As written in the previous part of the document, by the deadline of TP2, I should Finish writing accuracy practice mode and start to work on tempo practice and listening mode. It turns out that the practice mode has been pretty much finished by now, with all of the features that I can think of up to this point installed already.

However, it turns out that in order to reach the ideal effect of the tempo practice and listening mode according to the original proposal. Extra modules may, such as threading or pygame, will have to be implemented, which I am not allow to do before hitting the MVP. Therefore, instead of trying to install these two modes to perfection, we decided to implement compose mode first, which seems more practical and easy to make significant progress using existing modules.

About Practice Mode

Up to this point, in practice mode, the user can import any midi file that they desire, and the program will be able to generate animation accordingly.

When a tile hit the keyboard in the screen, the user should play the corresponding key in order to pass that tile. When it is a chord, the user can just play the keys of the chord one by one separately (order doesn't matter) in order to pass it. This feature works pretty accurate, but there's a slight delay because it requires time to open up stream every time.

Also, the keyboard will show the wrong key that the user plays in red color, so that the user will have an idea where to move next. It will also shows that correct key played in cyan.

It will also asks if the player is stuck either after the player doesn't play a key in 5 seconds or plays too many wrong keys and still not reach the correct ones. This is implemented in part to prevent the program from stocked in the while loop.

Non-stop mode is implemented. In this mode, the tile will not stop when hitting the keyboard and the stream will not be activated. But user can still stop the animation by pressing 'Space.' This mode is designed for the user to use after reaching a certain proficiency or just want to get the big idea of how the piece look like.

About Compose Mode

Up to this point. Users are able to compose their own pieces in this mode using their own instrument. Here's the process: user first plays a key, after which the program will ask "if this is all of the keys you

want to play at once.” After confirmation, the player plays the key for a certain length, and this length will be recorded by the program, and the program will generate the animation in real-time accordingly. After finish recording one note, the user should press ‘d’ for detach so as to move on to the next one.

To add empty note, simply press ‘Space’ for whatever length you want.

With all the information, the program will organize it into a list of tuples (the same format as we used in practice mode for generating animation) so that the users will be able to play back their work.

Also, the player will be able to save their work in the program (not as a file) and play their work back using the practice mode.

However, there are plenty of features that haven’t been implemented yet. So, the next step for this mode is to add more editor feature such as rewind, redo, delete…

Problems

With only the modules in the module list above, there is a trade of that we have to face in this project. Consistency and speed: if we put the audio sampling program directly into timerFired, this would allows the programs to continue collecting sound without stopping the animation. However, both programs will be slowed down significantly.

Otherwise, if we only open up the audio sampling program when needed (when a tile hits the key), both programs will run at the speed that we desire, but it requires time to open up and close the stream every time. So, by doing this, we lose consistency.

Like in compose mode, the tiles generated is much shorter than it suppose to be in practice mode. This is resulting from the lost of speed.

Plan

Maybe threading will be introduce after hitting the MVP to solve the problem above…

Just keep collecting sound but only use it when needed in the animation frame work.

TP3 UPDATE

Help Modes Added:

To access all three of the help modes, press “h.”

- General Help Mode (attaches a introduction to the app and what each mode does)
- Practice Help Mode (explain what non-stop mode is and provide instructions on what each key does)
- Compose Help Mode (explain play-back mode and provide instructions on what each key does)

Non-Stop Mode:

Non-stop mode is a sub-mode within the practice mode. User can access or quit this mode by pressing “n” in the practice mode.

Non-stop mode can be used either when you are pretty proficient in playing the piece and want to use the animation to calibrate your tempo, or if you simply want to get a broad sense how the piece looks like.

Play-Back Mode:

Play-back mode is a sub-mode within the compose mode. User can access or quit this mode by pressing “p” in the compose mode.

Play-back mode allows you to check your own progress without switching modes, it plays back user's creation up to that point so that you can see what have you created so far before continuing, so that you can have a clear idea of the whole thing.

User Experience And Bug Fixes:

- Practice mode
  - Bug fixed:
    - Fix the bug that the program crashes when the user enter some file that is not in the directory or didn’t enter anything. Instead of crashing, the app prints error message or return to splash screen.
    - Fix the bug that the animation crashes sometimes by copying the list to revised before the for-loop.
    - Fix the bug that the animation generated does not reflect the script after pausing by taking the time of the pause into account.

- Feature added:
  - Add the feature of player's performance assessment by generating a plot showing the graph of time spent on each key vs. the relative position of the key.
  - Add the key counter along with which the user may find which key in specific they need to practice more.

◉ Compose mode

- Bug fixed:
  - Fix the bug that the animation generated is inconsistent with the commands generated (i.e when playing back the work in practice mode, it would be a different animation than the one that was created in the compose mode).
- Features added:
  - Add the play-back mode, as described above
  - Improve the "detach" method. Before, the user can only detach the key as the sequence they added the key. Now, they can select whichever key they want to detach.
  - Add the "undo" key. By storing the "previous state," user can undo their mistakes made during composing the piece. For example, it can delete the wrong key added by the player, or clear the length of the key when it is assigned with a wrong length.
  - Add the on-screen instruction. This is different from the help mode, where users get different instruction of what they should do and what each key stands for at different point of composing. This is a step-by-step super-help that will hopefully guide the first-time users familiarized with the app.
  - Add the feature of file saving. Now the user is able to save their work into an txt-file and use it next time when they start the app. Before, it is only stored temporarily in the app and is cleared as soon as the player close the app. To import the file saved in practice mode, simply enter "xxx.txt"
  - Add an on-screen timer.

User Interface

- A nice splash screen is added with an infinite moving piano background picture, which is soooo classy and cooooool!!!! The algorithm took me a while to figure out
- All texts are written in a well-chosen font and located at a reasonable position on the screen.
- Different types of texts have different fonts and formats so that the user will know.

Things To Improve (From User-Study-Athon):

- Non-stop mode should play corresponding note concurrently with the animation
  - Like mentioned before, this would only be solved by using pygame or multiprocessing
  - I tried both of these options but don't have time for either of these
  - Let's just save it for the future
- Should generate a real stave (an actual image of a piano sheet) in the compose mode
  - This is definitely a good idea, but, once again, an actual piano sheet is way more complex than it might look like.
  - But still, it's a good idea and can be put into future schedule
  - Maybe consider generating a MIDI file first
- Include more profession musical features
  - Professionally means complexity
  - Once it becomes more professional, the target user will change accordingly.