Loop Station Progress Report

Mia Jane La Rocca
Computer Science Exercises

1 Project Introduction

For my project I will create a music loop station in Python 3. The loop station will have a user specified number of separate looping tracks. The user can record in a track by first selecting which synthetic instrument to use, the beats per minute (bpm) and the number of beats in the loop (bpl). Then the user hits record and performs the part of the loop using the computer keyboard as a MIDI input device to select which notes the synth will play. If the user has selected the quantize button, the notes will be quantized to the nearest quarter of a beat. The user can choose different loop lengths for each looping track to create more complicated polyrhythms and can enable and disable each track individually as well as set the volume for each track. This project uses FluidSynth the pyfluidsynth library to convert MIDI notes into sounds and PyQT5 for creating the GUI. Additionally numpy and time are used to manage the note scheduling.

2 UI Design and Description of Features

2.1 UI Design

A basic mockup of the GUI is shown in Figure 1. The GUI consists of a control panel which has yet to be designed which will include the controls for loading and saving the tracks. Then there is a set of Looper GUIs which is shown in more detail in Figure 2 which control each individual looping track. The user can specify how many loopers they want on launch. The final element of the GUI is the keyboard visualizer which will show the users which key on their computer corresponds to each key on the keyboard (see the current progress figure, Figure 3, for more details). The keyboard visualizer will also highlight which keys are currently pressed.

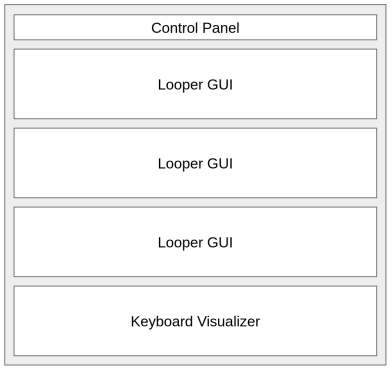

Figure 1: GUI Mockup

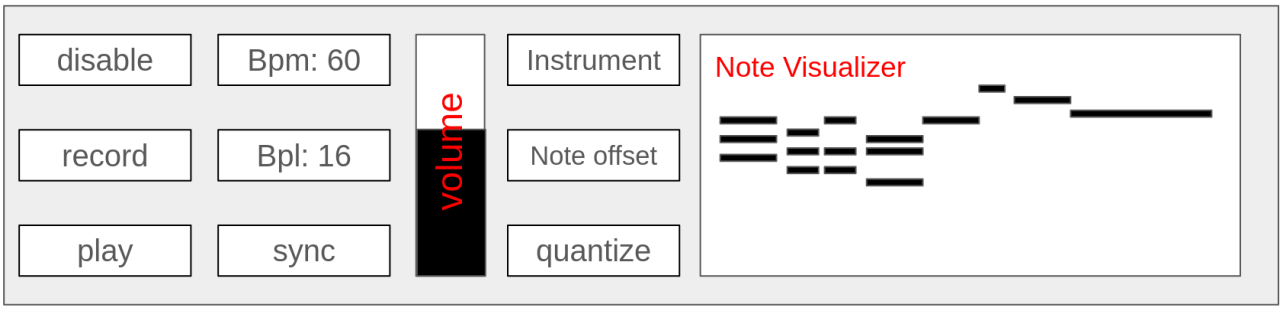

Figure 2: Looper GUI Mockup

2.1.1 Looper GUI Button Descriptions

Feature	Description
Disable	Switches looper into DISABLE mode
Record	Switches looper into RECORD mode. This clears the track and allows the user to add new notes.
Play	Switches looper into PLAY mode. This plays the current recording
BPM	The user sets the beats per minute (bpm) of the looping track. The beats are used for quantizing.
BPL	Sets the total number of beats in the loop
Sync	This is used to synchronize the current track with another one such that it has the same loop start time
Volume	Sets the MIDI velocity of the notes in the loop
Instrument	Selects the fluidsynth instrument to use
Note Offset	Sets the pitch offset of the track. Because the computer keyboard does not have as many keys as an actual keyboard the user needs to be able to set the offset to increase the instrument range
Quantize	Whether to quantize notes so that they are perfectly on rhythm. Quantizes notes to the nearest 16th note
Note Visualizer	A display which shows the MIDI notes in the loop. Time flows from left to right and pitch goes from top to bottom.

3 Back End Architecture

The app is built using PyQT. The classes used are described in Table 3. The MainWindow spawns LooperGUIs and LooperSequencers for each looping track. Each looper shares the same clock. The LooperGUI sets changes the settings of the LooperSequencer. The instrument, volume and note offset settings are passed to the Synth and BPM, BPL, synchronization and track enable settings are sent to the Clock. In the main window, a PyQT event handler is used to capture key press events and transmit them to LooperSequencers. If the LooperSequencer is

in RECORD mode, the LooperSequencer will command the Synth to turn on that note and ask the clock for the current beat. When the key is released LooperSequencer will command the Synth to turn off that note and ask the clock for the current beat. The note on beat and note off beat are then recorded in the looper's note schedule. If the LooperSequencer is switched into either DISABLE or PLAY mode, the new note schedule is sent to the Clock. The Clock will continuously play the notes for each track in PLAY mode, sending note on and note off commands to the Synth on the correct beats.

Name	**Function**
MainWindow	This is the main widget used to make the GUI
LooperGUI	This is the PyQT widget used to create the GUI for each of the four looping tracks
Synth	A wrapper class to fluidsynth.Synth to add more functionality
LooperSequencer	The back end of the looper which keeps track of the current looper sequence and plays the notes if the looper is enabled
Clock	Used to synchronize the four looping tracks and used to schedule note on and note off events
Controller	Contains can clear all tracks, can load and save past tracks, and can reset the start times of all loops

Table 1: Main Classes Used

3.1 Timing and Synchronization

The timing for each track is recorded in terms of beats rather than seconds. This allows the user to speed up and slow down a track. It also allows the notes to be quantized to specific beats to improve rhythmic accuracy of the music. When a track is enabled or disabled, the track start time is set to the current time and the current note is set to zero. The Clock's on_update() function is called by the MainWindow's on_update() function which exists in its own QThread. The on_update() function carries out the relevant commands as described in Algorithm 1.

4 Implementation Progress

4.1 Completed Work

4.1.1 UI

Although I originally proposed using Kivy for the GUI, as I was implementing I decided to switch to using PyQT5. This would allow the dropdown menus for the instrument selection to look much nicer as well as giving access to custom QSignals. I have created the UI for each looper aside from the Note Visualizer. I have also implemented the Keyboard Visualizer, showing which keys are currently pressed by highlighting them in red as shown in Figure 3.

Algorithm 1 Clock on_update()

for each *track* **do**
  $t_{current} \leftarrow$ current time
  if *track* has schedule and is enabled **then**
    $current\_beat \leftarrow \frac{1}{60}(t_{current} - track.t_{start}) * beats\_per\_minute \% beats\_per\_loop$
    if $current\_beat < track.previous\_beat$ **then** ▷ checking whether end of loop reached
      while $current\_command < track.schedule.n\_commands$ **do**
        command $track.schedule[current\_command]$ to synth
        $current\_command \leftarrow current\_command + 1$
      end while
      $current\_command \leftarrow 0$
    end if
    while $track.schedule[current\_command].beat < current\_beat$ **do**
      command $track.schedule[current\_command]$ to synth
      $current\_command \leftarrow current\_command + 1$
    end while
    $track.previous\_beat \leftarrow current\_beat$
  end if
end for

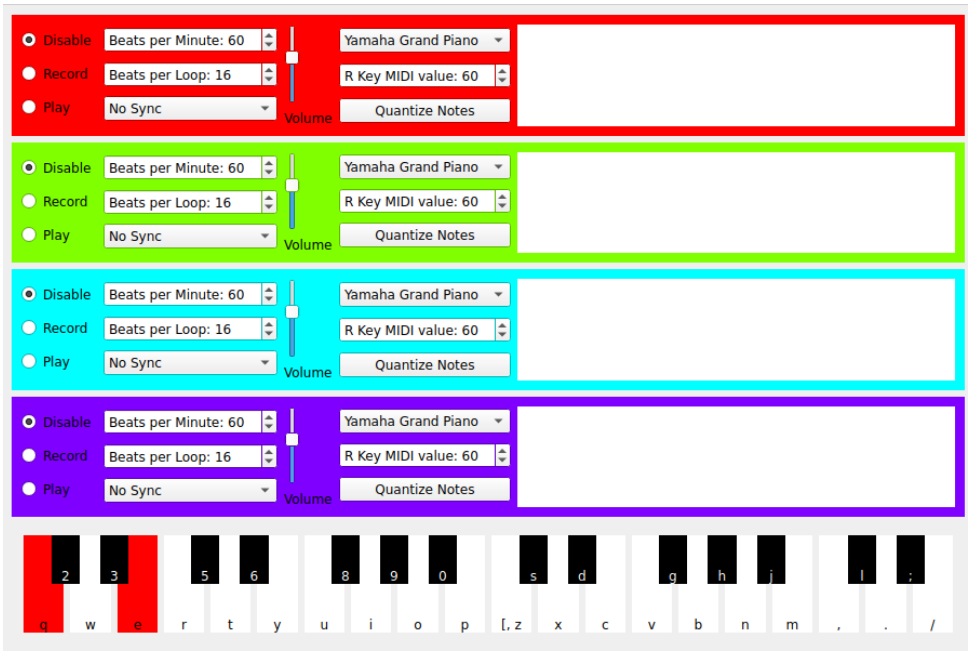

Figure 3: Current status of the GUI (taken while q and e keys are pressed)

4.1.2 Back End

The Synth is completed to handle commands from the Clock and Looper using the settings set with the LooperGuis. The Clock is currently implemented and able to play notes from a hardcoded note schedule.

4.2 Future Work

4.2.1 Minimum Viable Product

I still have to implement the RECORD mode for the loopers to create note schedules. I also need to add the visualization for the note sequences in the LooperGUIs. Additionally, the control panel for saving and loading tracks needs to be developed. I also need to write documentation.

4.2.2 Potential Additional Features

- Allowing the user to change how much to quantize by to allow for triplets or swung notes
- Allowing the user to decide whether to clear the track before recording on top
- Allowing the user to add/delete tracks through the GUI (currently the number of tracks is set on launch)
- Some way to have different note volumes within the same track (this is difficult as I do not have a velocity sensitive MIDI controller)