Paper Title

Mia Jane La Rocca
Keio University

Abstract
This project is 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. The user can choose to synchronize tracks with other tracks and save their tracks to a yaml file to be loaded in a future session. The code and install instructions for this project is posted in: https://github.com/miajl/loop_station.

Contents

1 Introduction

Looping is a technique in music production music is recorded and played back during a performance. This can be used by a single musician to build up a backing track they will record over. The first dedicated looping device, called the Paradis LOOP delay, was released in 1992 [2]. This device allowed musicians control the loops with buttons or footswitches while playing on their isntruments [2]. Since then many looping plugins have been created for Digital Audio Workstations (DAWs), the software used by musicians to create music, such as Looper in Ableton Live [1] or Nabla Looper [5] for REAPER [3]. These tools all require the DAW software which can be quite expensive and difficult to use. For this project, I wanted to create a simple standalone tool that can be used to play around with different beats and musical ideas instantaneously.

2 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. Each of the buttons are described in Table 1. 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. The keyboard visualizer will also highlight which keys are currently pressed.

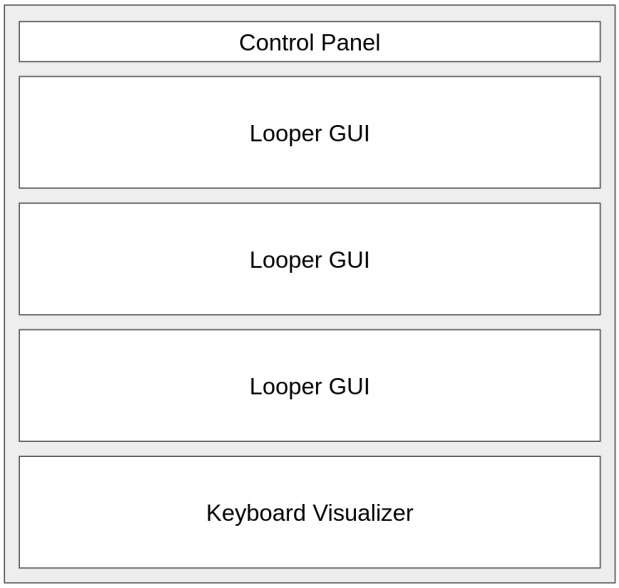

Figure 1: GUI Mockup

3 Implementation

The implementation includes approximately 1000 lines of code written by me in Python 3.7.17. The libraries PyQt5 [4], PyFluidSynth [7], and PyYAML [6] were used in addition to python's standard libraries.

3.1 Back End Architecture

The app is built using PyQT. A simplified block diagram is shown in Figure 3. The MainWindow spawns LooperGUIs, Loopers and Synths 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 beats per minute, beats per loop, 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 Loopers. If the Looper is in RECORD mode, the Looper will command the Synth to turn on that note and ask the clock for the current beat. When the key is released Looper 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 Looper 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.

3.2 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 by changing the beats per minute. It also allows the notes to be quantized to specific beats to improve rhythmic accuracy of the music. When a track is enabled the current note is set to zero and the time is reset to zero if it was enabled from DISABLE mode. Addionally if the metronome is en-

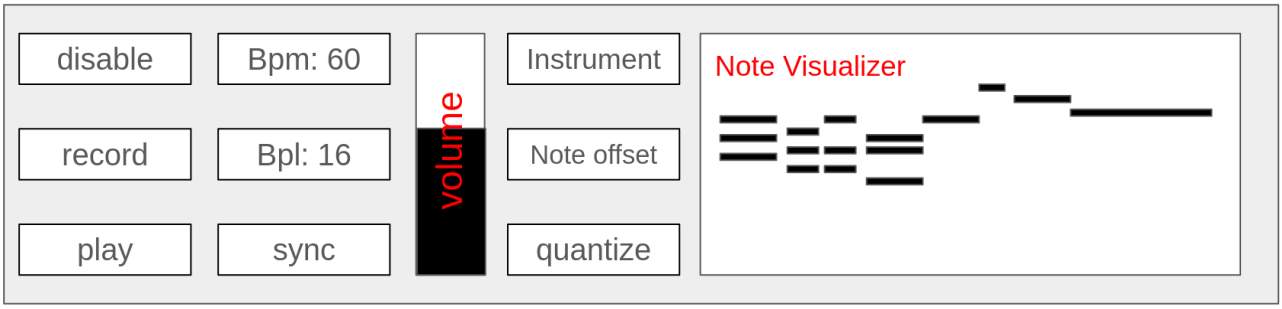

Figure 2: Looper GUI Mockup

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 bottom to top.

Table 1: Looper GUI Button Descriptions

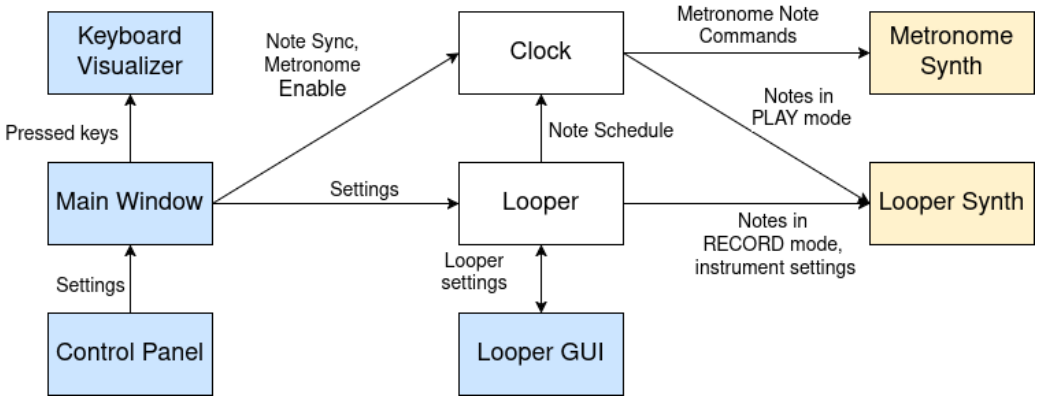

Figure 3: Simplified Block Diagram

abled, the clock will play the metronome when a track is in RECORD mode. 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.

3.3 GUI Implementation

The final GUI is shown in Figure 4.

3.3.1 Main Window

The main window is created using a QVBoxLayout containing the Control Panel GUI, the looper GUIs and the Piano GUI.

3.3.2 Control Panel

The Control Panel consists of the save button, load button and sync button. The save button opens a file dialogue box where the user can specify a filename to save their current tracks. The load button opens a dialogue box where the user can pick a file to load from. The sync button is a QPushButton which causes the track offsets for all the tracks to be set to the current value. This synchronizes their start times. The Use Metronome button tells the Clock whether to use the metronome or not.

3.3.3 Looper GUI

The Looper GUI is built off the main QHBoxLayout. Inside the main QHBoxLayout is the QVBoxLayout for the mode buttons, the QVBoxLayout for the beats per minute, beats per loop, and sync selectors, the volume slider, the QVBoxLayout for the instrument selectors, and the Note Visualizer. All of these buttons are connected to their back end implementations in the Looper.

3.3.4 Note Visualizer

The note visualizer displays the current notes as QRect objects as well as a cursor indicating what time in the loop it currently is as a QLine. The NoteVisualizer is repainted every on_update while the track is in RECORD or PLAY mode and updated once when the track is switched to DISABLE mode to turn off the cursor and change the color of the notes to gray.

3.3.5 Piano GUI

The Piano GUI is created by a QStackedLayout which contains the white key widget and black key widget. The White key widget has a QHBoxLayout consisting of the key widgets for the white keys. the black key widget has a QHBoxLayout consisting of the key widgets for the black keys. The QHBoxLayout is placed in a QVBoxLayout with stretch 1 and extra stretch 1 so the black keys are shorter than the white keys. All the key widgets include the name of the key on the keyboard to press and the keys will turn red when pressed.

4 User Testing

I conducted informal user testing among my friends on this project. In general, they were able to use it with some assistance. Some of the features they suggested adding include modifications to the synchronizing system. Before user testing, syncing to a reference track would just sync it at that moment, but subsequent changes to the reference track would not change the track that was synced to it. Additonally there was no way to sync the start times of the tracks without modifying the bpm and bpl of the tracks. To address this, I created the Sync All Tracks button to reset the start times of all tracks and modified

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**
        if $track.schedule[current\_command]$ is a note off command **then**
          command $track.schedule[current\_command]$ to synth
        end if
        $current\_command \leftarrow current\_command + 1$
      end while
      $current\_command \leftarrow 0$
    end if
    Initialize *on_notes* as empty list
    while $track.schedule[current\_command].beat < current\_beat$ **do**
      if $track.schedule[current\_command]$ is a note off command **then**
        command $track.schedule[current\_command]$ to synth
        $current\_command \leftarrow current\_command + 1$
      else
        $on\_notes.append(track.schedule[current\_command])$
      end if
      $current\_command \leftarrow current\_command + 1$
    end while
    for *note* in *on_notes* **do**
      command *note* to synth
    end for
    $track.previous\_beat \leftarrow current\_beat$
  end if
end for
if Using metronome and a track is recording **then**
  $metronome\_time \leftarrow t_{current} - recording\_track.t_{start}$
  $metronome\_beat \leftarrow metronome\_time * \frac{1}{60} * beats\_per\_minute$
  if $metronome\_beat - previous\_metronome\_beat \geq 1$ **then**
    Send metronome note on
    $previous\_metronome\_beat \leftarrow int(metronome\_beat)$
    $metronome\_note\_on \leftarrow True$
  end if
  if $metronome\_beat - previous\_metronome\_beat \geq 0.2$ and $metronome\_note\_on$ **then**
    Send metronome note off
    $metronome\_note\_on \leftarrow False$
  end if
end if

Figure 4: Final GUI

the syncing behavior such that the dependent track would stay synced to the target track until the user unsyncs it or modifies the dependent track on their own. They also wanted the notes to show during RECORD and DISABLE mode and not just PLAY mode. Finally they suggested I add a metronome so it is easier for them to know when the beats are.

5 Conclusion

This project creates a fun environment for informal and easy music experimentation. This project has a wealth of features which allows the user to come up with complicated musical ideas and play around with them. There are still some areas for improvement though. For example currently in RECORD mode, a note is only displayed in the NoteVisualizer after the key is released. This could be fixed by having the note rectangle display but with the end time set to the current time. Also while in RECORD mode if the time loops back around, existing notes do not get played. Finally a very useful feature would be to add the ability for the user to record the loops and export them as audio files. Additionally, there is an issue when the user uses the keyboard to type a number into a spinbox, for example to set the bpm, even after pressing enter, the user has to click away in order for their key presses to be registered as notes. More work could also be done to user-proof the software such as preventing the user from loading a file in the wrong format.

References

[1] Ableton. *Ableton Live*. https://www.ableton.com/en/live/. 2025.

[2] Matthias Grob. *Growth due to limitations*. 2009. URL: http://www.livelooping.org/history_concepts/theory/growth-along-the-limitations-of-the-tools/.

[3] Cockos Incorporated. *REAPER*. https://www.reaper.fm/. 2025.

[4] Riverbank Computing Limited. *PyQt5*. https://pypi.org/project/PyQt5/. 2024.

[5] NablaTools. *Nabla*. https://github.com/NablaTools/Nabla. 2023.

[6] Kirill Simonov. *PyYAML*. https://github.com/yaml/pyyaml. 2024.

[7] Nathan Whitehead. *pyFluidSynth*. https://github.com/nwhitehead/pyfluidsynth/. 2024.