Assignment 4

Overview

In Lecture, we introduced a GM (General Midi) Synthesizer called FluidSynth to give us a larger palette of instrumental sounds than sine waves. We also laid the foundation for generating rhythms by creating Clock, Conductor, and Scheduler classes. We tested the results by writing a Metronome class. In this assignment, we'll flesh out the Conductor and Scheduler functionalities and create a musical device more interesting than a metronome.

Part 1: Changing Tempo

Add the following to Conductor's functionality:

```
class Conductor(object):
   ...
   def set_bpm(self, bpm):
      pass

   def get_bpm(self):
      pass
```

Test your function calls by making two keys in *on_key_down* that raise and lower the BPM. You should hear the Metronome get faster and slower. Show the current BPM on the Label widget. Also note that when you raise and lower the metronome values, you can do so either additively or multiplicatively. Which is better? Why? Provide your answers in your README file.

Part 2: Remove

The Scheduler class works, but is missing a critical element: removing a command. It is often necessary to remove a command after it has been posted, but before it executes. This is most often used to stop or pause a system mid-stream. Write the function:

```
class Scheduler(object):
   ...
   def remove(self, command):
      pass
```

Make sure to return the *command* back to the caller of *post_at_tick*. In other words:

```
class Scheduler(object):
   ...
   def post_at_tick(self, tick, func, arg = None):
```

should be modified to return the command.

Now that you have a remove function, add a way of stopping that incessant metronome. Create this:

```
class Metronome(object):
   ...
   def stop(self):
      pass
```

And test it in *on_key_down*.
Remember to not leave a note-on hanging when you stop the Metronome. Temporarily change the instrument program to a sustaining instrument to help you test this.

Part 3: Arpeggios

One common technique for automated rhythmic/melodic generation is the Arpeggiator. Create a new class (in the same style as Metronome) that does arpeggiation of a given set of notes. The real-time controls of the arpeggiator will let you:

- Start and stop the arpeggiator's note production
- Change the notes being played
- Change the pulse / rhythmic value of the notes
- Change the direction of the notes to be either up, down, or up/down.

For example, if notes are: [60, 64, 67, 72], pulse is 1/8th note, and direction is down, your arpeggiator would produce a steady the stream of 1/8th notes: 72, 67, 64, 60, 72, 67, 64, 60, etc...

Start with the template class provided in the code directory. You will need to add member variables and create the appropriate helper functions.

Test your class by hooking up to *on_key_down* as before. Test all aspects of the class interface and make sure that everything works and sounds right when you change values in real time (including, for example, setting a different number of notes and making sure you don't get index-out-of-range crashes). You should also keep the Metronome running as you test the Arpeggiator. This is to make sure that your Arpeggiator plays in time to the beat appropriately. When you change to a different rhythmic pulse, make sure that the arpeggiator notes are aligned to that rhythmic grid.

Part 4: Make some mappings

In this part, you will create / compose a mini performance system to create a short 30-60 second piece.

- Create two arpeggiator instances – one for a "bass line" and one for a "lead line" (using instruments that you like from the GM sound bank).
- Come up with some number of note chords that you can apply to the arpeggiators. These can be regular triadic chords, but of course, they don't have to be.
- Decide what rhythmic pulses you would like the arpeggiators to use. Hint – the bass line should probably use slower moving rhythms than the lead line.

- Hook up key mappings so that you can control your arpeggiators.
- Perform on your performance system and make a video recording. Upload to Youtube or attach the video file (as in Assignments 1 and 2).
- Provide a README file that explains your mappings.

Finally...

For this assignment, please do not upload the 147MB soundfont bank since I have a copy. This will make uploading/downloading faster.

Add a sentence on collaboration if you collaborated beyond lab time. Please have good comments in your code. Please make sure that every part of this assignment is testable. When submitting your solution, submit a zip file that has all the necessary files (except for the soundfont bank). Upload your zip file to Stellar / Homework.