Jerome Andaya (jandaya@bu.edu)

Ryan Walsh (rmwalshy@bu.edu)

CS 591 S1

Launchpad-Lite

For our final project, we decided we wanted to create our own "launchpad" in Python. Given a simple UI and a keyboard, a user is able to design certain sounds based on pre-loaded signals we have supplied. With nine different keys to choose from, a user is able to record music based on the keys they press and for how long. Additionally, users are able to apply filters to modify their sounds and have audio playback all within our program. We chose a project like this because of the things that were taught in this course. From the basics of learning how to make our very own sine waves to applying advanced filters to modify frequency and time, we felt like we had a large tool set to work with. However, what we thought was missing was the complete ability to actually design, create, and playback the signals we have been learning about. For this reason, we took some of the most important concepts we learned about and applied them to a fully functioning launchpad that any person could play, regardless of their knowledge in computational audio.

Starting on the outside and working our way in, we will start with the UI of our program. Since we had been working in Python all semester long and were comfortable modify signals in that environment, we decided to continue with Python and use the library PyGame. PyGame allows for a quick and easy setup to provide an easy to use UI for almost any application. With PyGame's built in classes like Sound and Sprite, we are able to store sound files with ease and create a UI in no time.

We use PyGame’s built in Sprite class to create the UI. This involves a simple rectangle class containing an image of our choice. Here, we have a simplistic UI with the Launchpad body being the same image as our buttons, but this can easily be changed, given higher resolution images. The sprite classes are also extensible in the way that the images can be changed during an event like a button press. This would allow us to give feedback to the user while they play. Right now our feedback is displayed through the terminal.

We are also using PyGame’s event handlers to record key presses like a launchpad does. For example, to record any one of our nine sounds that are mapped to the keys Q, W, E, A, S, D, Z, X, and C, all that is needed of the user is to press any button for however long they desire. These keystrokes will be recorded by our program and sent to the back-end for the actual synthesis, but more on that later. To actually record a sound, the “1” key must be pressed. This will tell our program to begin listening to the keystrokes as mentioned above. Once the user is done hitting their desired keys, pressing “2” will stop playback. Hitting “3” will then play back whatever sound was just produced in the project directory. Below is a table with all the current key mappings:

Key	**Function**
Q, W, E, A, S, D, Z, X, C	Beat mappings
1	Start recording
2	Stop recording
3	Playback recording
I/O	Pitch modification on/off
=/-	+.25/-.25 pitch factor

K/L	ASR filter on/off
P	Forces a pitch recalculation with the current pitch scale

But how are the sounds played? They are played by a PyGame Sound class. We store these Sounds in a list. The list essentially acts as pointers to classes, where each class has a unique sound. We dereference our “pointers” and invoke the .play() method to play the sounds on the keydown event. We set loop to -1 to set the tone playing forever, but the play call is stopped once we release the key.

What is great about PyGame is the simplicity in setting up it up. With minimal lines of code we were able to have a fully functioning launchpad in no time. Another great feature of this project is that it is very scalable. By adding more key mappings, we were able to add other additional features like ASR filters and pitch modification. These extra features made this project really feel like more than just simple sine waves being played together. As we learned in class, simple algorithms like these help bring life to a sound that may have sounded a little too computer generated at first.

Moving on to the back-end of this project, this is where the true audio processing is done. Receiving a list of keystroke events from the front-end, this is where they are all made sense of. Given a key pressed, we know when it was pressed, how long is was held for, when it was let go, and what sine wave it is mapped to. With these four crucial pieces of information, we have enough to build our sound. For each unique key pressed, we construct a signal of the desired duration. Since we know what key was actually pressed, we know what the desired frequency will be as well. With a frequency, a duration, and default relative amplitude of 1.0 and a phase of zero, we have a given beat. When a user presses many keys, all we do is merge the synthesised beats into one whole signal. While constructing these beats, we can also apply an ASR filter to them if it is toggled on by the user when the recording begins. This makes for a more realistic sound instead of a very abrupt start and end to a sine wave. Once the whole signal is processed, the program can also modify the entire pitch and time according to the user's liking.

What is exciting about this project is the scalability of the application. While at it's core this program does everything we want it do to, there is no end to what could be done to improve it. For example, by mapping more keys that toggle different filters on and off, we could include tens of more filters to allow for the user to express exactly what they want in their sound. Another big feature we would have really liked to improve is the UI. While it is there to simply give the user a sense of what a launchpad should look like, having it be interactive would greatly increase the program's overall look and feel. Similarly, giving the user even more variety in what they could do by allowing them to map buttons themselves would bring great flexibility to our launchpad.

One other big improvement, which may be Python's downfall, is incorporating threading into the project. While simple recordings can be constructed quite easily, applying timepitch modifications really slows down the program. Because of this, the program becomes unresponsive while it takes time to process. If we devised this in a different language, incorporating true multi-threading would be quite easy and bring a better overall feel to the project. The drawback would be the difficulty in creating a user interface, as well as dealing with events from keypresses. With that said, we think Launchpad-Lite brings real life to the course and shows our understanding of the key concepts we learned about. It is our hope to continue on with this project outside of class and to continue adding new features to it.

Outline:

- Overview of the project
- Pygames
  - Controls: Two states. Recording and Not Recording.
    - 1: Record
    - 2: Stop Recording
    - 3: Playback
    - QWERADSZXC: Sounds
    - - : Pitch_scale -= 0.25
    - = : Pitch_scale += 0.25 (didnt want user to use shift to + scale)
    - I: pitch mod on // P: pitch mod off
    - P: process current signal with pitchModification
    - K: ASR on // L: ASR off

        - Buttons are pygame sprite classes
            - Has option to change image on button press (future stuff)
            - Could have better images
        - Sounds are pygame.mixer.Sound classes
            - Stored in a list of Sounds ((Library))
            - Played by ``` Library[0].play(loop=-1) ```
                - Invokes Sound class .play(...) method.
                - Loop = -1 // refers to infinitely looping (until you stop pressing it)
        - UI
- Backend
    - Combining beats
    - Applying filters
- Diagrams (show controls)
- Improvements to project