SI515 Assignment 1
Katy Madier

AUDIO LOOPER

Introduction

The Audio Looper is a python program that records audio individual 5-second audio clips then overlays them together in a composited wav file. This system was inspired by loop station pedals typically used by musicians to loop together different phrases on an instrument.

The System

The recording and playback functions are initiated using acoustic tracking through a small microphone attached to the side of the drum. The microphone picks up sound frequencies and analyzes each .15 second for a specific range of frequencies through the attached computer.

To record an audio file, a user knocks loudly on the drum. To playback the composited file, a user taps lightly on the drum. The knock and tap frequencies are identified by the program using their distinctive frequencies. Once identified, the program runs the record or play function every time a knock or a tap is heard.

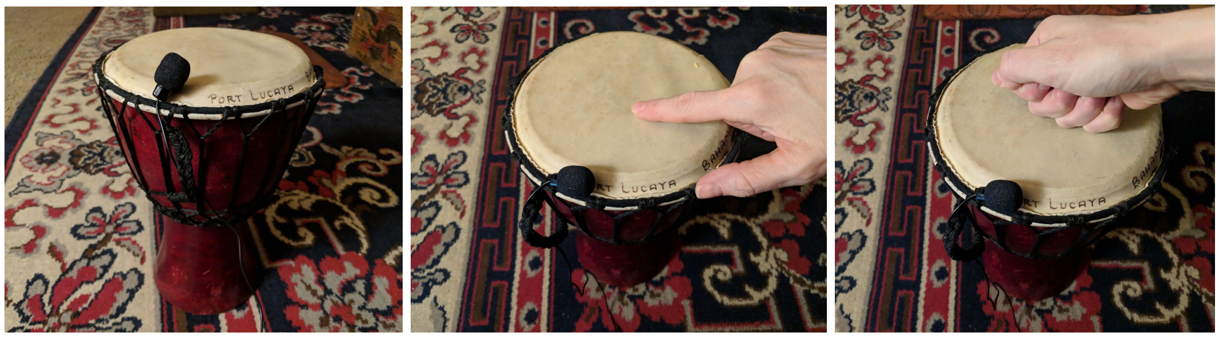
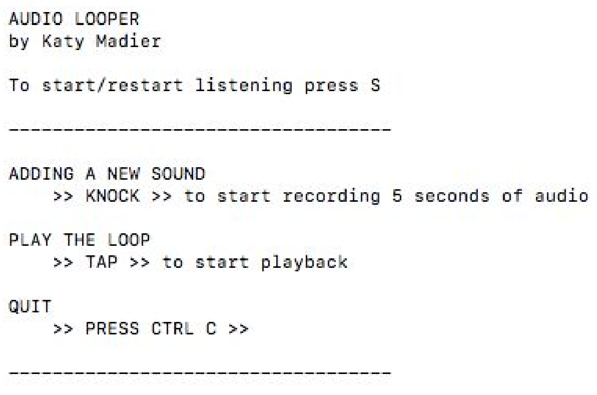


Figure 1. Left: The Audio Looper drum and microphone (computer not shown). Center: tap gesture. Right: knock gesture

Implementation

To implement this program, I used pyaudio and pydub as the main libraries to handle the audio stream. The program is initialized with a main function that holds the program open for input.

In the main function, an open audio listener function is activated that listens to any sound coming in from the microphone. This function reads the audio stream and analyzes the frequency data to find the knocks and taps. In this program, the knocks are identified by any frequency over 3000hz. The taps are identified by any frequency between 900hz and 1100hz.


```
AUDIO LOOPER
by Katy Madier

To start/restart listening press S

-----------------------------------

ADDING A NEW SOUND
    >> KNOCK >> to start recording 5 seconds of audio

PLAY THE LOOP
    >> TAP >> to start playback

QUIT
    >> PRESS CTRL C >>

-----------------------------------
```

Figure 2. Left: The Audio Looper interface menu.

If a knock is identified, the loop will break and the audio record function will begin. In the record function, a new file name is made based on the list of files present in the audio folder. The recording runs for 5 seconds, the new audio is applied to the existing loop file, and then the new composited mix is played back.

If a tap is identified, the composited mix is played back. To end the program, a user can type CTRL C at any time.

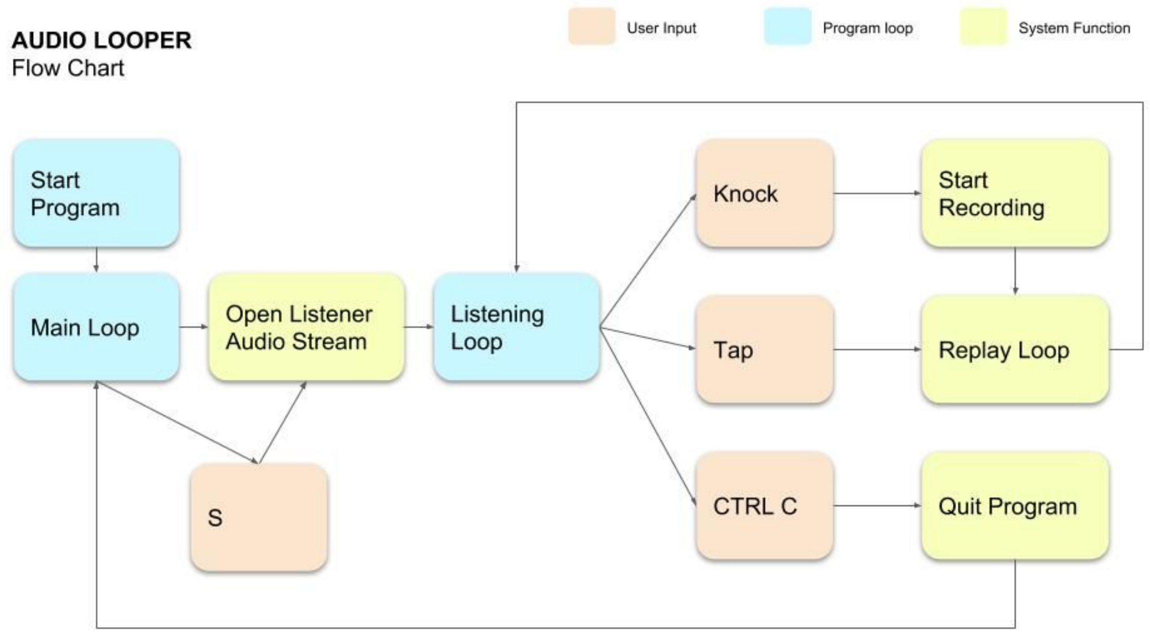


Figure 3. System flow chart showing user input, program loops, and system functions.

Challenges

Differences in frequency were not intuitive

It was difficult for me to distinguish between frequencies of many interactions (blowing, scraping, snapping). The data provided by these sounds were all very similar and inconsistent. Sometimes the data would return 800hz, while other times it would return as 2800hz. It was difficult to get consistent results. I found that hard knocks, loud claps, and light taps would return the same results consistently. I don't know enough about sound to understand why yet. Maybe I was not finding the peak correctly.

Adding too many loops sound bad

The looper starts to sound tiny and strange after too many audio files get overlayed by the pydub function. It would be great to have a way to minimize this sound through a filter.

The program doesn't always stop with CTRL C

The program sometimes gets stuck in a loop and will not escape with the key interrupt function. CTRL Z usually fixes the problem, but sometimes CTRL C works just fine. I also don't know why this is happening.

Unimplemented Features

Initially, the concept of the system included many more features than what was implemented in this prototype. Some features that were never completed were **BLOW** to remove the last recording, **QUICK SCRAPE** to speed up the playback of the mix, and **SLOW SCRAPE** to slow down the playback of the mix. Because of issues with capturing the scrape and blow frequencies, I decided to use more staccato style sounds as input. Additionally, each audio interaction's intended use was to start and stop the recording and playback. I could not figure out how to interrupt the audio stream to start and stop those functions, so I changed their scope to a limited timeframe.