AIST 2010 Project Report

Name: Cheng Ka Pui & Lam Puy Yin

Sid: 1155125534 & 1155126240

AIST 2010 Project Report

Abstract

This project, ”Beat Shouter!” is a music rhythm game featuring an input sound control function. In the game, the user is going to hit beats on the screen along the music when the beats arrive at the hitting range. The sequences of the beats will be generated according to the rhythm of the music. The method of hitting the beats is making a short sound input such as a clap, a short shout within a second. The goal of the game is to hit the beats accurately along the rhythm.

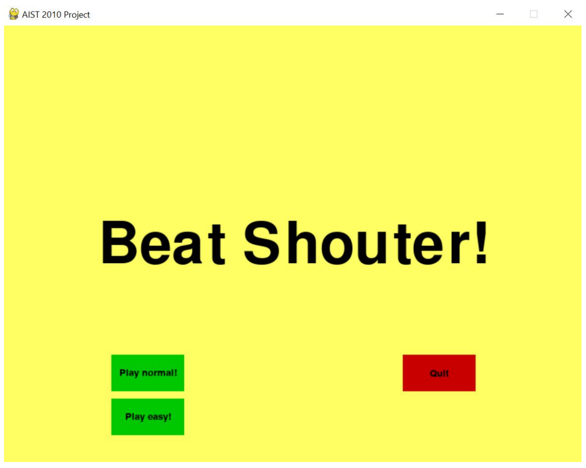


Figure 1a: Main Menu of the game

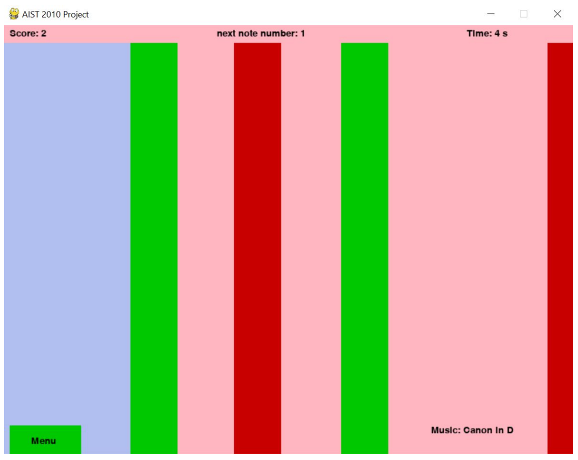


Figure 1b: Game display

Background

In the course, we have learned about music information retrieval. In this project, we will use music information retrieval programming to find the music feature, rhythm, of certain music with the aid of Librosa which we learned from labs and lectures. After attaining the music features, they will be used in the rhythm game. The rhythm will be used for generating beats and the tempo will be the speed of the beats. Also, we are going to analyze the sound input instantly and find features in the instant sound in our sound control function.

Our project is inspired by two games. The first one is YASUHATI[1]. In this game, the player controls the movement of the character by instant voice input. The fascinating idea of using voice to control has given us the insight to introduce a game mechanic with sound input control function.

The second game is Taiko no Tatsujin[2], it is a famous rhythm game series. In the game, there is a sequence of beats that the player needs to hit the beats on time to get scores. In our project, we would like to implement a rhythm game like this.

Figure 2: YASUHATI [1]   Figure 3: Taiko no Tatsujin: Drum 'n' Fun! [3]

Methodologies

Programming language and libraries

In this project, we use Python as our programming language as we want to use the music information retrieval libraries in python. Here is the list of the main libraries we used in our project.

1. librosa 0.8.0 [4]

   Librosa is used for finding music features in the music such as onset detection and beat tracking.

2. PyAudio 0.2.11 [5]

   PyAudio is used for recording audio from the user microphone as the input.

3. pygame 2.0.0 [6]

   pygame is used for implementing the interface of the project, such as the beats on the screen, and handling interactive activities with the user.

AIST 2010 Project Report

Name: Cheng Ka Pui & Lam Puy Yin

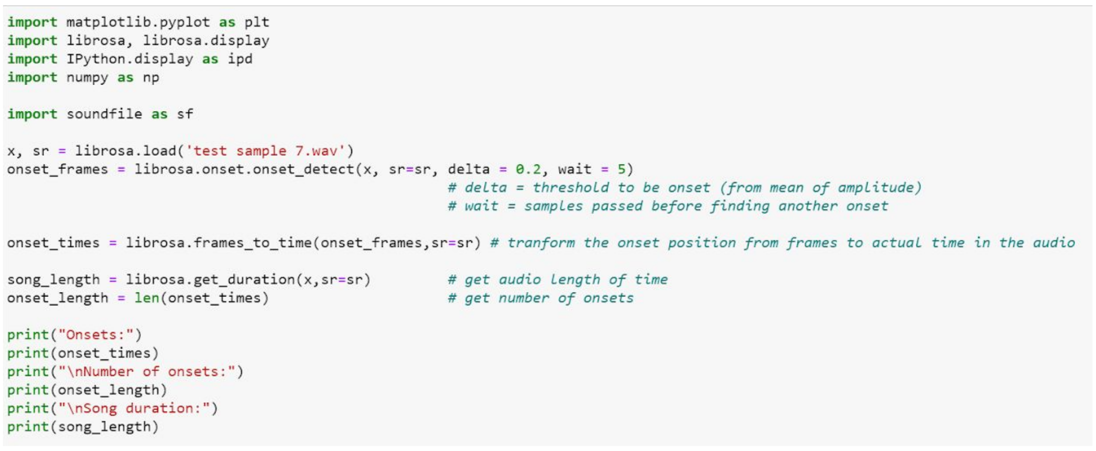
Sid: 1155125534 & 1155126240

Project work

1. Rhythm Detection

   Since we are implementing a rhythm game, it is important to get audio mechanics from the python library. And this is why we used onset detection.

   To put it simply, an onset is the very beginning of a sound, which is where we detect the initial impact of the sound impulse, or also known as the attack of the envelope. We also adjusted the threshold of the minimum energy input (delta) to be recognised as an attack, along with the minimum activation time (wait) before finding another onset again. And thus, by using librosa.onset.onset_detect(), the function will detect any spectral energy peaks in the audio and return an array containing the frame of those peaks. Then we can change the frames into time format and also find the number of onsets (using length() function) in order to correctly implement the music beats during the game (both the number and position of the beats).



```
import matplotlib.pyplot as plt
import librosa, librosa.display
import IPython.display as ipd
import numpy as np

import soundfile as sf

x, sr = librosa.load('test sample 7.wav')
onset_frames = librosa.onset.onset_detect(x, sr=sr, delta = 0.2, wait = 5)
                                                    # delta = threshold to be onset (from mean of amplitude)
                                                    # wait = samples passed before finding another onset

onset_times = librosa.frames_to_time(onset_frames,sr=sr) # tranform the onset position from frames to actual time in the audio

song_length = librosa.get_duration(x,sr=sr)         # get audio length of time
onset_length = len(onset_times)                     # get number of onsets

print("Onsets:")
print(onset_times)
print("\nNumber of onsets:")
print(onset_length)
print("\nSong duration:")
print(song_length)
```

Figure 4: A snippet of the Onset Detection Function

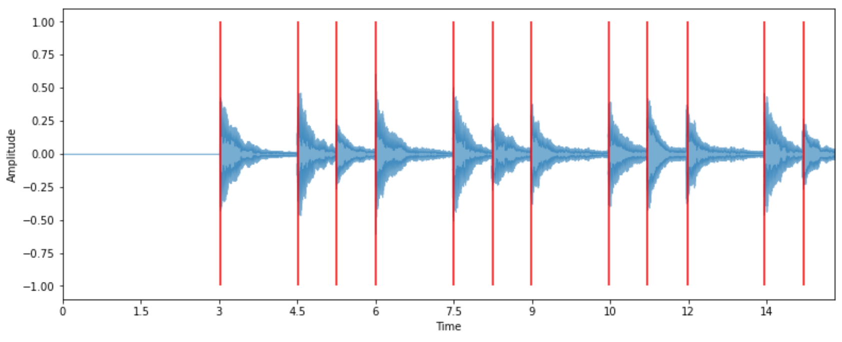


Figure 5: The onset of the music

Extended: Beat Tracking

Previously we have discussed the possibility of using a beat detection function like librosa.beat.beat_track() to calculate the beat of the music and find the hit frames for the game. However, the result returned is a uniform beat with respect to the tempo of the song, which makes melody change or energy change in between the beats undetected. This could affect the complexity of the rhythm game, since a constant tempo of beats will lack variety, which would be an unfavourable experience to the player.

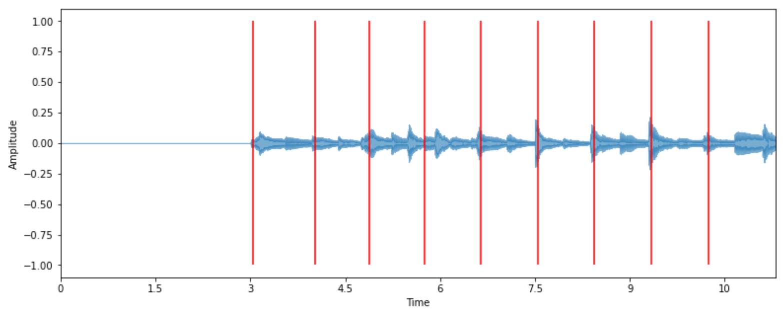
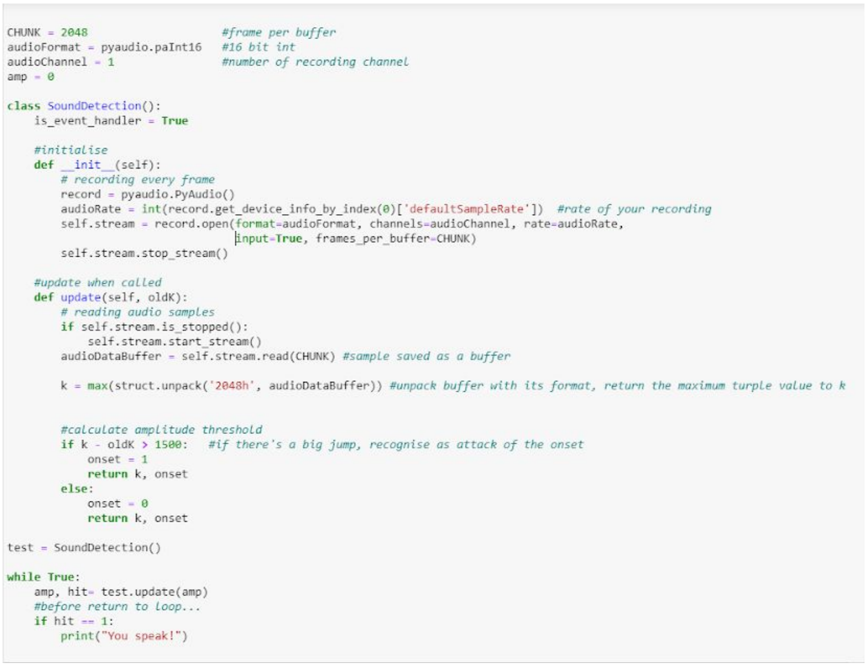


Figure 6: The beat of the music in a fixed tempo

2. <u>Input recording</u>

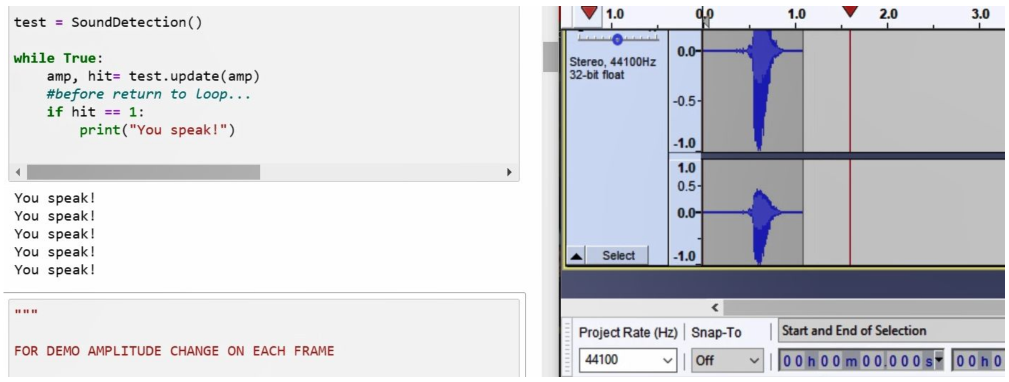
Amplitude Level Measuring

For the input audio with pyaudio. We can measure the energy of the audio input through the audio bytes of that instance, which is also known as frame per buffer. By reading [self.stream.read()] and unpacking [struct.unpack()] the buffer frame to a variable (k), the maximum energy collected (using max() function) from the audio can then be used for checking if the user has spoken at the moment, which enables us to implement the hit interaction mechanics in the game.



```python
CHUNK = 2048                #frame per buffer
audioFormat = pyaudio.paInt16   #16 bit int
audioChannel = 1            #number of recording channel
amp = 0

class SoundDetection():
    is_event_handler = True

    #initialise
    def __init__(self):
        # recording every frame
        record = pyaudio.PyAudio()
        audioRate = int(record.get_device_info_by_index(0)['defaultSampleRate'])  #rate of your recording
        self.stream = record.open(format=audioFormat, channels=audioChannel, rate=audioRate,
                                  input=True, frames_per_buffer=CHUNK)
        self.stream.stop_stream()

    #update when called
    def update(self, oldK):
        # reading audio samples
        if self.stream.is_stopped():
            self.stream.start_stream()
        audioDataBuffer = self.stream.read(CHUNK) #sample saved as a buffer

        k = max(struct.unpack('2048h', audioDataBuffer)) #unpack buffer with its format, return the maximum turple value to k


        #calculate amplitude threshold
        if k - oldK > 1500:   #if there's a big jump, recognise as attack of the onset
            onset = 1
            return k, onset
        else:
            onset = 0
            return k, onset

test = SoundDetection()

while True:
    amp, hit= test.update(amp)
    #before return to loop...
    if hit == 1:
        print("You speak!")
```

Figure 7: A snippet of the Onset Detection Function



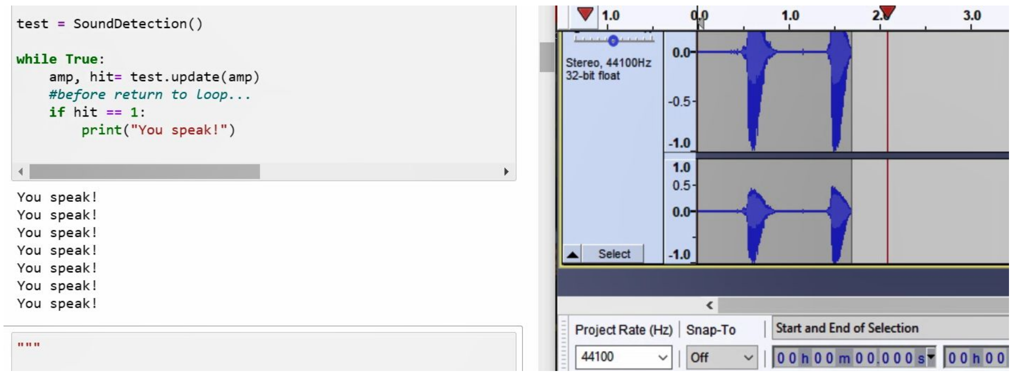
```python
test = SoundDetection()

while True:
    amp, hit= test.update(amp)
    #before return to loop...
    if hit == 1:
        print("You speak!")
```

```
You speak!
You speak!
You speak!
You speak!
You speak!
```

```
"""

FOR DEMO AMPLITUDE CHANGE ON EACH FRAME
```

Project Rate (Hz): 44100 | Snap-To: Off | Start and End of Selection: 00 h 00 m 00.000 s

Figure 8a: Results returned when recording for sound detection (part 1)



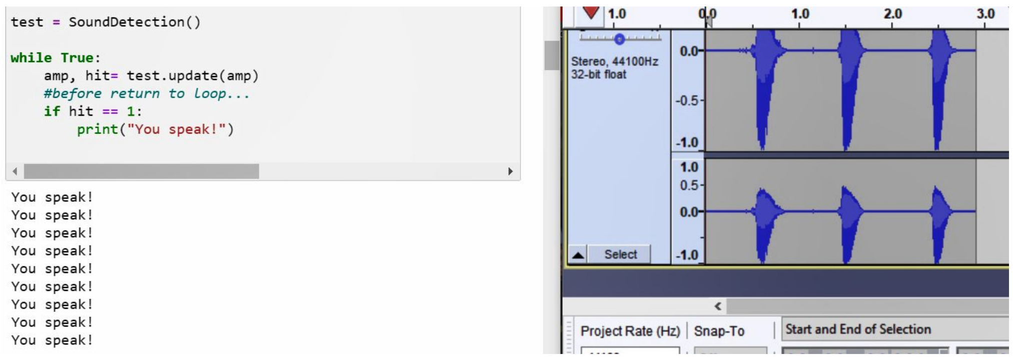
```python
test = SoundDetection()

while True:
    amp, hit= test.update(amp)
    #before return to loop...
    if hit == 1:
        print("You speak!")
```

```
You speak!
You speak!
You speak!
You speak!
You speak!
You speak!
You speak!
```

```
"""
```

Project Rate (Hz): 44100 | Snap-To: Off | Start and End of Selection: 00 h 00 m 00.000 s

Figure 8b: Results returned when recording for sound detection (part 2)



```
test = SoundDetection()

while True:
    amp, hit= test.update(amp)
    #before return to loop...
    if hit == 1:
        print("You speak!")
```

```
You speak!
You speak!
You speak!
You speak!
You speak!
You speak!
You speak!
You speak!
You speak!
```

Figure 8c: Results returned when recording for sound detection (part 3)

Extended: Continuous background Onset detection

We initially proposed the idea of detecting the onset of the user's input by recording in a short period of time. Yet, we have found that this function could cause the game to halt until the recording time is finished. This is a critical problem as a rhythm game, any lag or delay will directly affect the user experience.

3. <u>Interface</u>

Beat generating

The rhythm of the music will be generated as a sequence of rectangles. We will draw all the rectangle sequences on the right of the window where the player cannot see. When the game is started, the whole sequence will move from right to left. Then, we can see the rectangles, the beats are coming from right to left.

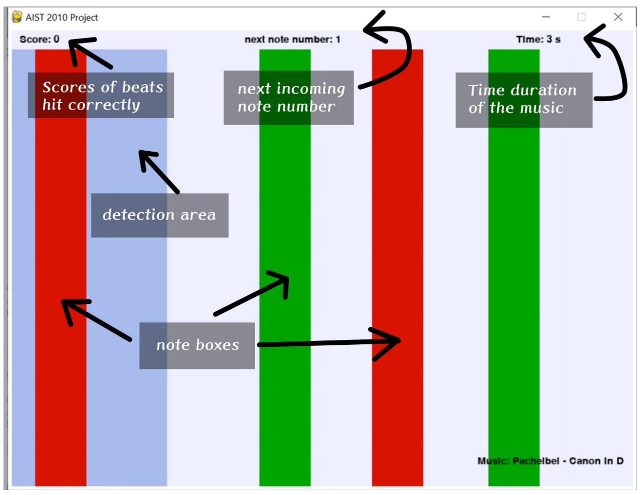

AIST 2010 Project
Score: 0
next note number: 1
Time: 3 s
Scores of beats hit correctly
next incoming note number
Time duration of the music
detection area
note boxes
Music: Pachelbel - Canon In D

Figure 9: Game interface's main elements

4. Interaction

When the player enters the game, the player will see beats coming from right to left. The player needs to “hit” the beats when the beats arrive in the scoring area.

The program will automatically call the audio recording function at each frame in order to find a valid hit, which is determined by whether the time of audio input is within the onset time period (between a period of 0.2 seconds before/after the onset time). And when a valid hit is detected, the score will be updated by 1. Moreover, the detection box will give out responses depending on the user's action. For example, the detection area will remain blue in colour if there’s no audio input, and it will change to light blue if the recording function detects an input. And if there’s a valid hit detected (i.e. audio input is detected when a note collides with the detection area), it will turn to yellow to indicate a correct hit.

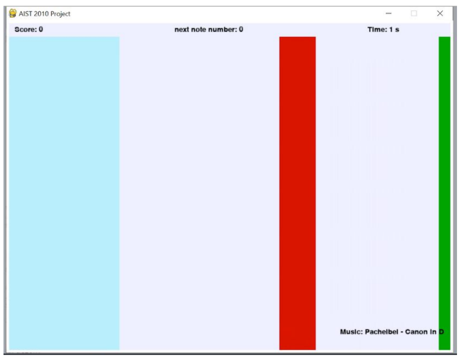


Figure 10a: When audio input detected

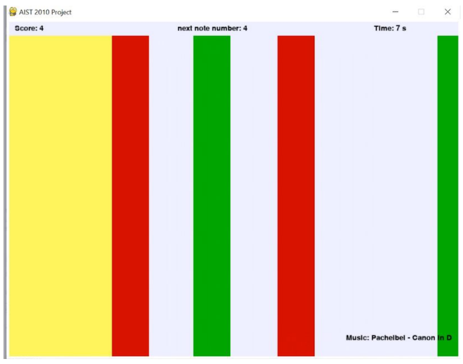


Figure 10b: When getting a valid hit

Conclusion

In conclusion, with the use of onset detection, input spatial energy detection, and game interface from pyaudio, librosa and pygame functions, we have successfully implemented a simple music rhythm game with an audio input as the control.

However, our journey of making this project is not easy, during the project, we have encountered and overcome many technical issues and learned a lot from the problem such as the difference of onset and beats and music terminologies. Furthermore, there are also a few limitations in our project we would like to point out. First, we need to find a suitable music track for the project as different melodies will affect the ways for attaining the music features. For example, melodies played by music instruments with low energy intensity such as violin and flute may not be detected since they can be easily covered by musical

instruments with high energy intensity like percussion.Therefore, we only use monophonic music for music source now.
Also, when we integrate multiple libraries together, some of our written functions do not work properly such as our original microphone input onset detection function. Choosing proper libraries and checking how to implement them is therefore important before the actual work. Nevertheless, we also learned to embrace failure and learn from failed programming parts, and keep trying. Sometimes failure teaches us more than success.

Labour distribution

1. Rhythm Detection: Cheng Ka Pui
2. Amplitude Detection: Lam Puy Yin
3. Interface: Cheng Ka Pui
4. Interaction: Lam Puy Yin

References

[1] "YASUHATI," Freem!, [Online]. Available: https://www.freem.ne.jp/win/game/13993. [Accessed 21 12 2020].

[2] "太鼓の達人シリーズ公式サイト ドンだーページ," BANDAI NAMCO Entertainment Inc., [Online]. Available: https://taiko-ch.net/. [Accessed 21 12 2020].

[3] "太鼓の達人 Nintendo Switchば～じょん！ ダウンロード版," Nintendo, [Online]. Available: https://store-jp.nintendo.com/list/software/70010000001508.html. [Accessed 21 12 2020].

[4] B. McFee, "librosa 0.8.0," [Online]. Available: https://pypi.org/project/librosa/. [Accessed 21 12 2020].

[5] H. Pham, "PyAudio 0.2.11," [Online]. Available: https://pypi.org/project/PyAudio/. [Accessed 21 12 2020].

[6] "pygame 2.0.0," [Online]. Available: https://pypi.org/project/pygame/ . [Accessed 21 12 2020].