Acoustic Control for the Super Mario Game

Lei Zhang

February 2019

1 Abstract

In this report, I presented a system that uses audio input as control signals to play the Super Mario Game. The system can detect three types of signals: tapping, scratching, and the pronunciation of "ssss".

2 Introduction

Microphones are the most common sensor found in consumer electronics today. Sound in our physical world provides a rich information source on knowing users' activities. In this project, I utilize the built-in microphone of a laptop to capture and process users' signals to control the movements of the character in the Super Mario Game. There are three types of signals that can be detected: tapping, scratching and the pronunciation of "ssss". Tapping on the desk triggers the character to jump. Scratching on the desk causes the character to move forward. And Pronouncing "ssss" controls the character to move backward. If no control signal is found, the game character will stand still.

3 User Interface

Since the only sensor we need is the built-in microphone, which we believe is more ubiquitous, the setup of this interface only requires a laptop to play the game and a desk to interact with. Figure 1 shows the physical setup of the system.

4 System Design

Figure 2 shows the diagram of this system. Since we need to monitor the user's audio input in real-time in order to detect the control signals, we open a new thread for real-time streaming and audio processing while running the game in the main thread. In Thread 1, we keep the audio stream running and within each loop we take a trunk (1024 frames) of audio input data and process it for signal detection. Each question within Thread 1 in Figure 2 represents a signal detector, which we will elaborate in 4.1.

Figure 1: User Interface.

I did not implement the whole Super Mario Game, which should be credited to this open-source project[1]. However, I did hack into the code and modify the key bindings in the game mechanism in order to trigger the game character's actions with self-defined events, which we will elaborate in 4.2.

4.1 Signal Detector

For tap/knock detection, we first convert the trunk of audio input data into a buffer with values, which represent the amplitude or how loud the signal is. Then we simply take the peak (the max value in the buffer) and compare it with a pre-defined threshold. If the peak amplitude exceeds the threshold, then a "tap" event is detected. Since the threshold is pre-defined, it actually depends on the material of the desk the user is tapping on. During several trials we find that the amplitude of a tap event is significantly higher than the other two events. It is also higher than people's voice of talking at a normal volume, which is because the user's taps are usually much closer to the microphone of the computer. After the trials we determine the threshold to be 0.5, which turns out to be fairly robust on different desk.

For scratch detection, we use the frequency feature of sounds, since the friction of two different materials can provide very unique frequency, or pitch. In this system we utilize the pitch detection function from the *aubio*[2] library, which returns the estimated pitch in *Hz* and the corresponding confidence, scaling from 0 to 1. Note that the frequency can vary dramatically depending on the material that the user scratches on, the

[1] https://github.com/justinmeister/Mario-Level-1
[2] https://aubio.org/

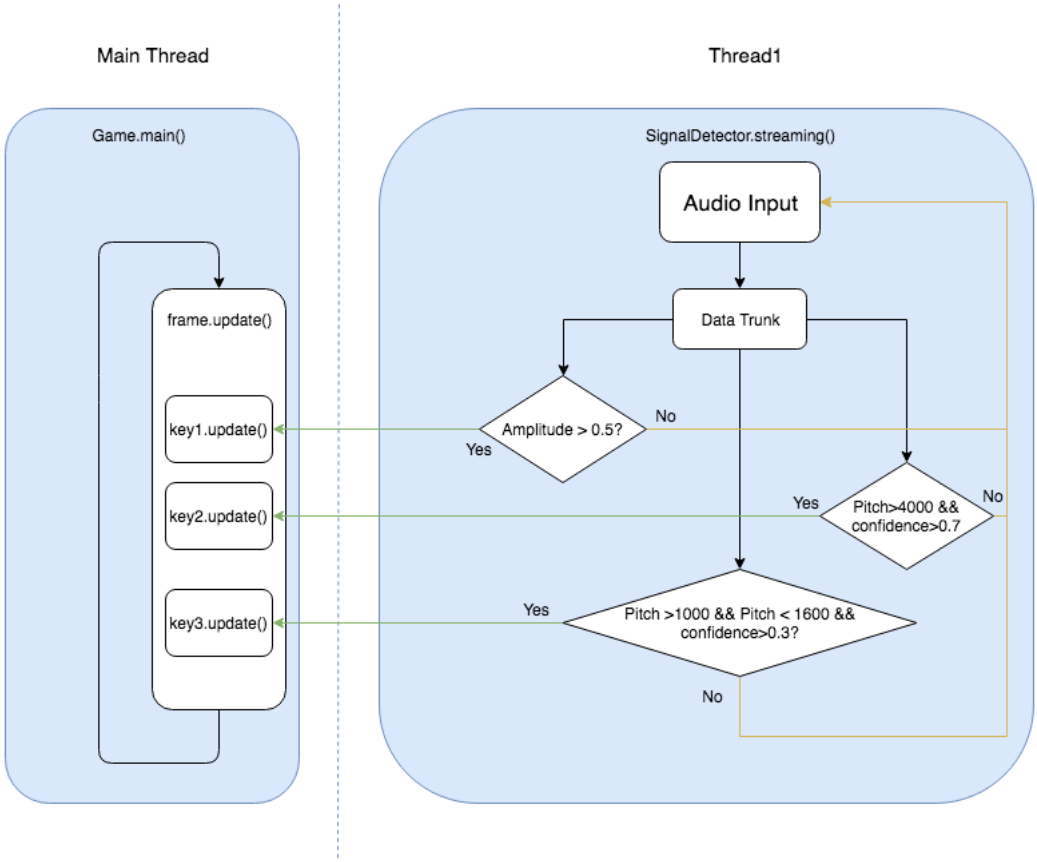

Figure 2: System Diagram.

strength and the speed of the scratches. After several trials on an office desk made of wood, we set the range for a scratch signal to be 1000 Hz - 1600 Hz and the confidence threshold to be 0.3. We set it to be above 1000 Hz in order to filter out the ambient noise and below 1600 Hz to filter out the higher frequency sounds. This frequency range turns out to be less robust comparing to tap detection since we found that even scratching on different desks that are made of wood, the frequencies were very different.

For detecting the pronunciation of “ssss”, we use the same technique as scratch detection. A threshold of 4000 Hz and 0.7 confidence turns out to be more robust than scratch detection, which may be because that people pronounce it very similarly.

4.2 Game Input and Multi-threading

In the original game users use the ”right arrow” and ”left arrow” on the keyboard to control the character moving forward and backward, respectively. The jumping action of the character is triggered by the “A” key on the keyboard. After modification, each

acoustic signal from Thread 1 will emit a “KEY_DOWN” event to the event queue in the game. I also wrote the module for handling such events. Moreover, since all the three acoustic signals are not able to emit “stop” events, which means that we can only detect the happening of such signals but not the ending of such signals, we set up a time window for each acoustic signal. If the same signal is not detected within the time window, we will emit a “KEY_UP” event, which means the stop of the signal.

Multi-threading is another challenge in this project. As in Figure 2, we run a separate thread for audio streaming, processing in order to simultaneously run the game and detect the signal. However, we do not separate audio streaming and processing into two different threads, even if the processing time may cause latency for user input. This is because after several tests we found that the tap detection takes 0.15ms on average and the pitch detection takes 14ms on average, which is acceptable in terms of the latency of game input.

5 Results and Discussion

In this report, I presented a system that utilizes acoustic signals as user inputs for controlling the character in the Super Mario Game. Generally this game works as expected. It is both novel and interesting for the Super Mario Game. However, there are also several limitations in the project. First, the thresholds are pre-defined, thus not robust for many situations. For example, when I let other people try out this game, we found that the detection of tapping and the pronunciation of “ssss” is satisfying but the detection of scratching is not. This is because people scratch on the desk differently in terms of strength and speed. Therefore, setting customized thresholds for different users would be ideal. Second, using thresholds to filter the amplitude and frequency features of acoustic signals is not advanced enough. Since many of our actions in the physical world can provide very distinctive features in terms of frequency, it is possible to utilize advanced machine learning methods to extract such features and detect them. Finally, we find that using acoustic signals for character controls in the Super Mario Game is less adaptive than using keyboard controls. This may be because there is no clear “button up” events for such signals. For future improvements, we can do the input bindings better in order to provide a more natural interaction with the game. I believe that using the built-in microphone on the laptop is more challenging than using an external microphone, but my hope is that this technique can be applied universally on everyone’s laptop.