Megalovania Piano

Report By:

Hana Younis 52-25989 P-23

Habiba Essam 52-0627 P-23

This report encompasses playing out the first three seconds of the song “Megalovania-Undertale’, attached in the link below.

The procedure was carried out by first finding the notes for the song. After carefully selecting the most appropriate natural and sharp notes, our song’s notes occur in the third, fourth and fifth octave.

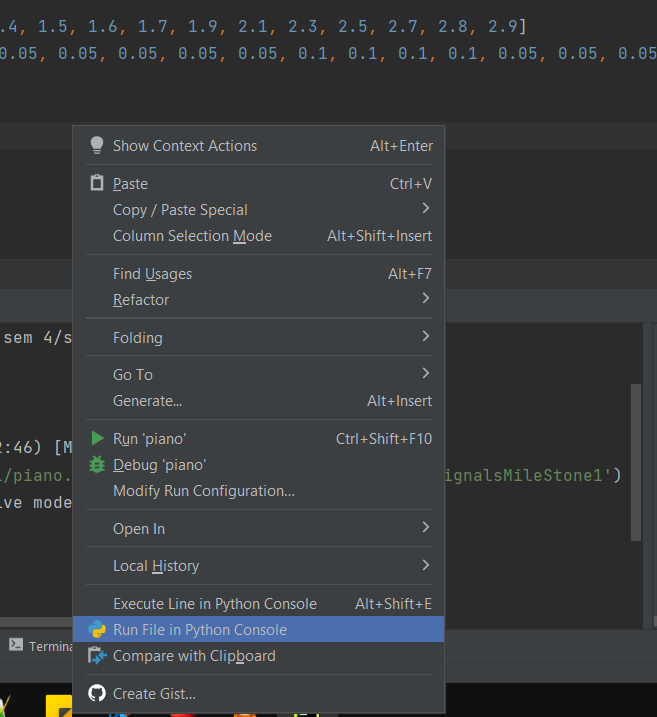
After the note selection, we found their matching frequencies. This was then put into an array (n). The array (ti) represents the starting time for each corresponding note in array (n). The array (Ti) represents how long each note lasts. This was done by multiple trial and errors, slowing down and speeding up the song, and manipulating the (Ti) for every note. This took the bulk of the time, as pauses were also accounted for.

Finally, a loop was used to represent the summation of every signal that makes up every note, where then the outcome is displayed as a plot and the song is played.

Some problems we faced was getting the song to play as (Spyder) wouldn’t play the song even with the necessary libraries installed, so (Pycharm) was opted for. Despite that, it could only play while running the file in the python console. In addition, we faced an issue with the overlapping frequencies, resulting in a deep and rather eerie song in the beginning. Last but not least, the bulk of the work was put in actually using our ears to get the harmony correct with varying speeds.

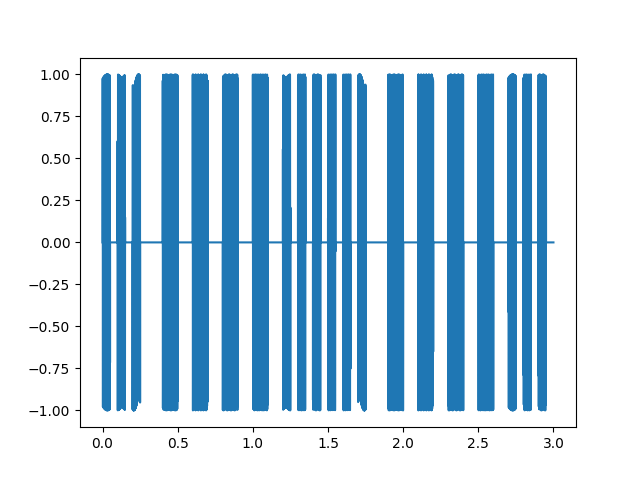
In conclusion, the program written plays a song using each note represented as a signal, and summing it into (x) to provide a continuous signal that plays the tune corresponding to the signals summed according to their timing and position in graph. The plot is attached below.

PS We ran the python file in Pycharm, right clicked and chose Run File in Python Console



Undertale - Megalovania:

https://m.youtube.com/watch?v=wDgQdr8ZkTw

Plot:

**Milestone 2:**

Picking up where we left off, we created (x\_f) which is the frequency domain signal of (x), using (f) which is the horizontal axis, and plotted it. Then (fn) was used to generate an array of 2 elements consisting of random integers from 0 to 512 which are the frequencies used to create the noise. The noise was then created with (noiseToAdd) which used the (fn) elements and sin functions, the noise was then added to (x) resulting in (xn) which is the signal with noise. The frequency domain signal of (xn) is (xn\_f), both signals are then plotted using subplot. In order to get the noise frequencies, we subtracted (x\_f) from (xn\_f) to get (noiseToCancel). A for loop was then used to get an array of 2 elements called (freq) which consists of the frequencies causing the noise, which was found by detecting the frequencies at which the peaks are much higher than the range of the amplitudes for the rest of the frequencies. The noise is cancelled in (xfiltered) by using both rounded frequencies found in (freq), (xfiltered\_f) which is the frequency domain signal of (xfiltered), was found and both signals were plotted using subplots, and finally (xfiltered) is played. See plots attached below:

Plot: