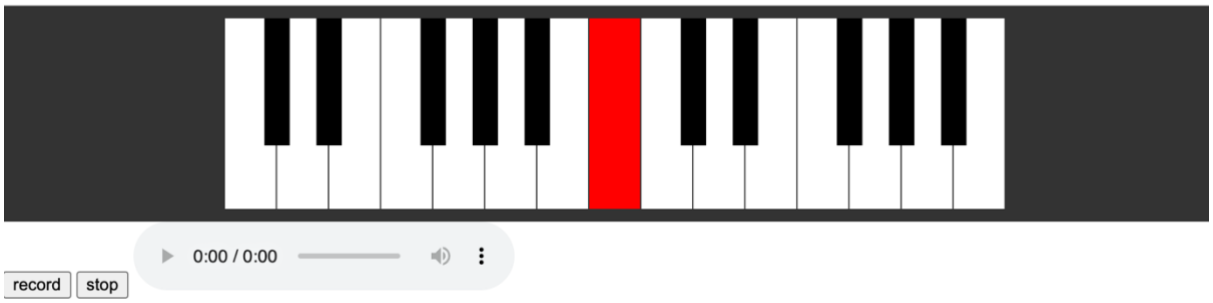
Sound Design Project report: Virtual piano

Piano



Project demo:
https://youtu.be/fXCRXaiCHqs

1) Description of the fictional scenario for which you did the sound design.

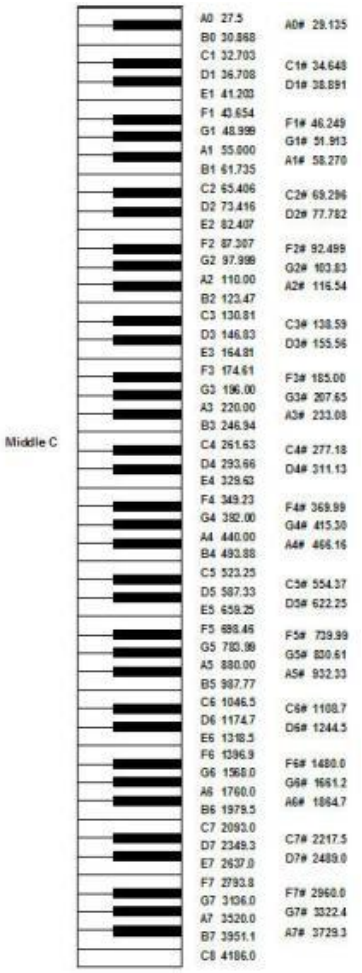
The objective of my sound design project was to use Web Audio API to build a virtual piano application. Selected piano note samples and the necessary audio recording were required for this project.

The sound design team for this project would consist of a sound designer, audio engineer, programmer, and UI/UX designer. Selecting and producing sounds for the virtual piano, such as specific notes and synthesised sounds, would be the responsibility of the sound designer. Any required piano samples for the virtual piano would be recorded and mixed by the audio engineer. Utilizing the Web audio API, the programmer would implement the virtual piano and create the necessary tools for users to record, play, and save notes. The user interface designer would be responsible for the css of the piano.

2) Theoretical concepts

The oscillator node of the Web Audio API was used to synthesise the notes of the virtual piano. In order to show the user, the differences between a synthesised piano note and a real one, I also used pre-recorded sounds of the piano notes that I was synthesising.

I investigated waveform selection while synthesising the piano notes. I decided to use the oscillator node to generate sine waveforms after researching and experimenting with various waveforms. Sine waveforms are frequently used in piano synthesis due to their simplicity and harmonic purity. The other waveforms did produce sounds as well, but they were other kinds of piano sounds that I was not looking for.



When researching I decided to find the specific frequencies the piano notes that I was intending to make (the middle C octave). After setting the oscillators nodes frequency variable to these certain frequencies I was able to generate the desired synthesised piano notes I was looking for.

Synthesised Piano notes:

```
// Function to Play Piano Sound
function playSound(note) {
    if (currentOscillator) {
        currentOscillator.stop();
    }

    const oscillator = audioContext.createOscillator();
    oscillator.frequency.value = noteFrequencies[note] || 0;
    oscillator.connect(audioContext.destination);
    oscillator.start();
    oscillator.stop(audioContext.currentTime + 2);
    currentOscillator = oscillator;
    currentOscillator.connect(myNoiseGain)
}
```

```
// Define Frequencies for C and D
const noteFrequencies = {
    'c': 261.63,
    'cSHARP': 277.18,
    'd': 293.66,
    'dSHARP': 311.13,
    'e': 329.63,
    'f': 349.23,
    'fSHARP': 369.99,
    'g': 392.00,
    'gSHARP': 415.30,
    'a': 440.00,
    'aSHARP': 466.16,
    'b': 493.88,
};
```

3) **Analytical Process**

Steps

I started my research by looking at the Web audio API and basic theories of audio synthesis. To better understand the ideas involved, particularly the oscillator node, I read the web audio API documentation and looked through earlier lab activities.

To replicate these noises in a virtual piano model, I then examined the features of a real piano. Studying various piano sound components, such as the harmonics and overtones, and figuring out how to synthesise those components using the oscillator node in the Web Audio Api.

Then, to get the desired sound and responsiveness and choose the right node graph for the model, I experimented with various node configurations. Before I started coding, I analysed other video game and film adaptations of virtual pianos to better understand how other sound designers went about creating a virtual piano and the methods they employed.

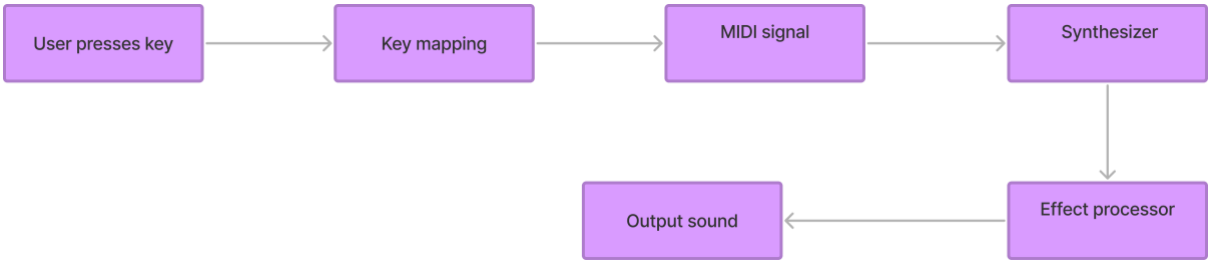
At the beginning of my prototyping phase I gave every note a unique function to play synthesised noises. Later I realised that there was a much more reliable way to do this and that this made my code difficult to debug due to it making the code unnecessarily long. Instead, I had one function for a note being played and saved all the desired frequencies in a 2D array and passed the specific frequencies into this function.

Some sources I consulted during my analytical process:

https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

https://mediapronerdzone.wordpress.com/2010/11/17/fundamental-frequency-chart-indispensible-eq-tool/

4) **Provide a detailed description of the signal chain behind your model and what decisions you took to arrive at the final version. Use a block diagram. Refer back to your analysis while describing components of the signal chain.**



User input - The user's keystrokes on the virtual piano are recorded in the first rectangle of the signal chain. This block determines which key was pressed and transmits that information to the following block.

Key mapping - The second block links the user's selected key to the appropriate piano note. The users' input is converted by this block into a digital signal that can be processed before being synthesised.

MIDI signal - The third block of the signal chain transforms the note information from the key mapping block into a signal that the synthesiser can understand.

Synthesizer - The sound of the virtual piano is produced by the fourth block. The synthesiser generates a digital audio output from the MIDI signal it receives as input. The synthesiser oversees sound quality.

Effect processor - This block enhances the audio by using a variety of digital signal processing techniques. For instance, the length of time the note is played.

Output sound - In the final block, the digital audio signal is changed into an analogue signal that can be played through speakers or headphones. This block provides the virtual piano's final sound.

I used an oscillator node sine wave with specific frequencies for each note, as I mentioned in my analysis. As a result, when a signal is sent from the MIDI signal, the synthesiser applies certain frequencies to the oscillator node. The user is able to record their plays using the GAIN node output that I implemented.

I used a similar strategy for the recorded piano notes, using an event listener to create an audio buffer when the note was pressed. A new audioBufferSourceNode is created with the createBufferSource() when the button is pressed. The previously created buffer is then filled with the decoded recorded sound. This AudioBufferSourceNode is then connected to the gain node and audiocontext.destination.

```
160   const cRsound = "piano_middle_C.mp3"
161   fetch(cRsound)
162   .then(response => response.arrayBuffer())
163   .then(buffer=> audioContext.decodeAudioData(buffer))
164   .then(audioBuffer => {
165
166     const cRsoundPiano = document.getElementById('cR');
167
168     cR.addEventListener('click', () => {
169       const source = audioContext.createBufferSource();
170       source.buffer = audioBuffer;
171       source.connect(audioContext.destination)
172       source.connect(myNoiseGain)
173
174       source.start(0);
175     });
176
177   });
```

5) **Describe the parameters in your sound model, how they are used and the effect of changing their values.**

The main parameter that affects the note played is the sound's frequency. The frequency values for each note is defined by the noteFrequencies object; if the pitch of each note needs to be changed, this can be done by altering the frequency values in the object.

The audio buffer parameter is used to store the audio information from the sound file being played for the recorded noise. This parameter would be modified to alter the note being played.

The gainNode that regulates the sound's volume is called myNoiseGain; altering this value would alter the sound's overall volume.

The start() method's parameter, 0, specifies the time at which the audio source should begin playing. If this parameter is changed, the piano will start playing at a different time.

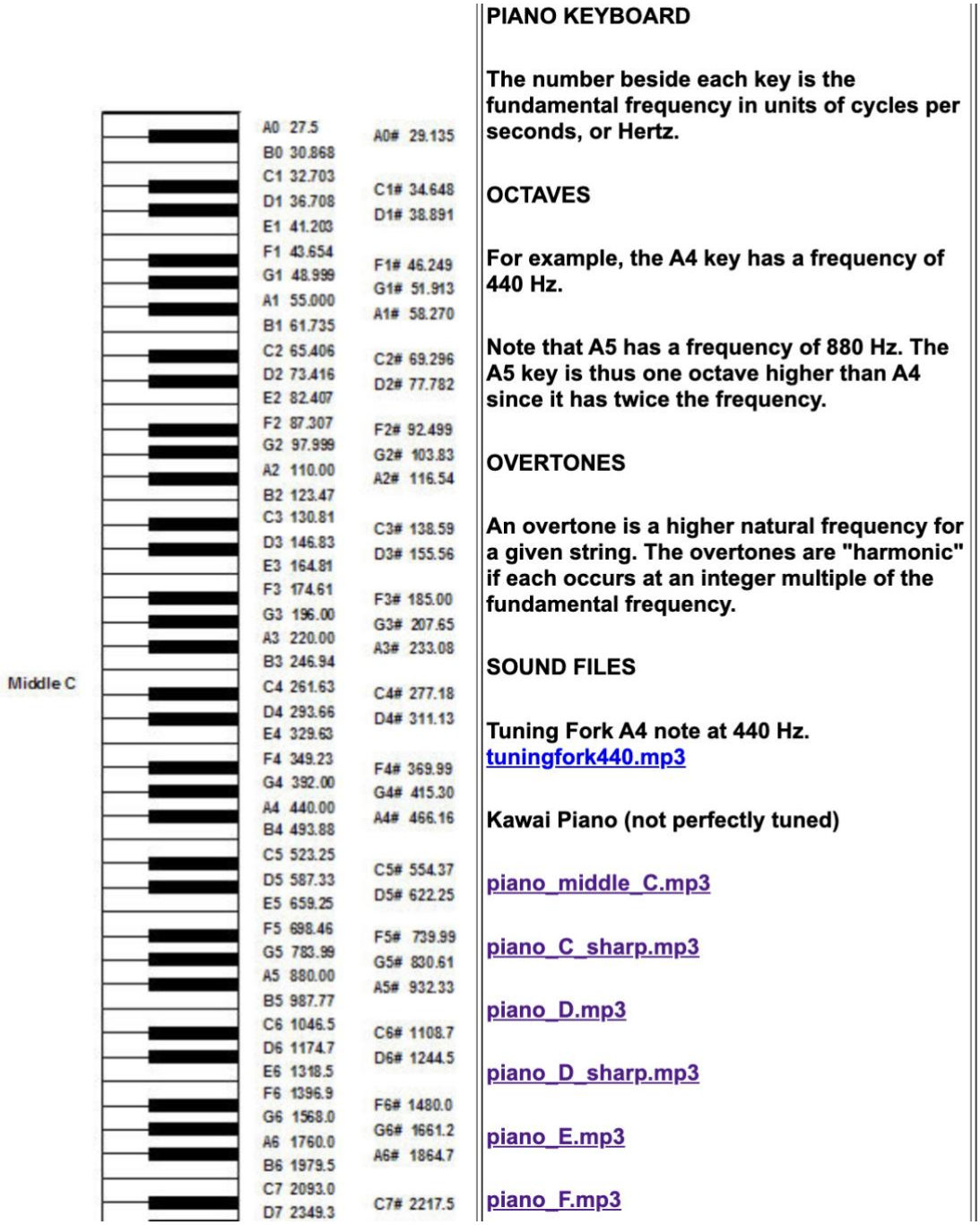
The stop() method parameter specifies the time the sound stops playing using "audioContext.currentTime + 2". The time is currently set to 2 seconds; changing this value would alter how long the notes are played.

The frequency of the oscillator that is being created is determined by the note parameter in my playSound function. The oscillator frequency is specifically set to correspond to the note being played.

The recorder variable changes the output node parameter to a different node, which alters what audio is being recorded, to record audio.

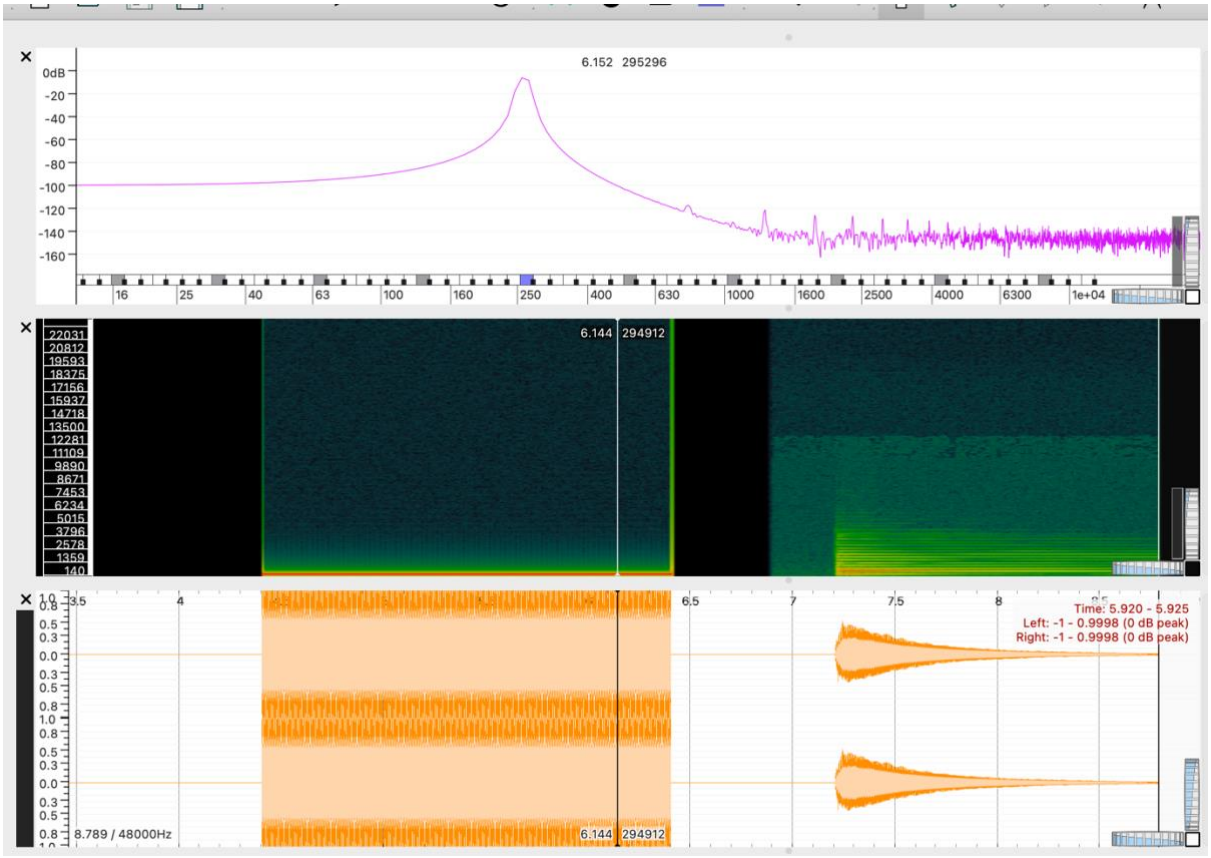
6) **Describe the recorded sounds you used, how they were recorded and how they fit into the sound design piece.**

For my virtual piano's recorded sounds. I downloaded From http://www.vibrationdata.com/piano.htm, piano note recordings. These sounds were captured by placing a microphone close to the piano's strings, which accurately captured the instrument's natural sounds. I then downloaded and imported these sounds to the corresponding notes in my virtual piano.

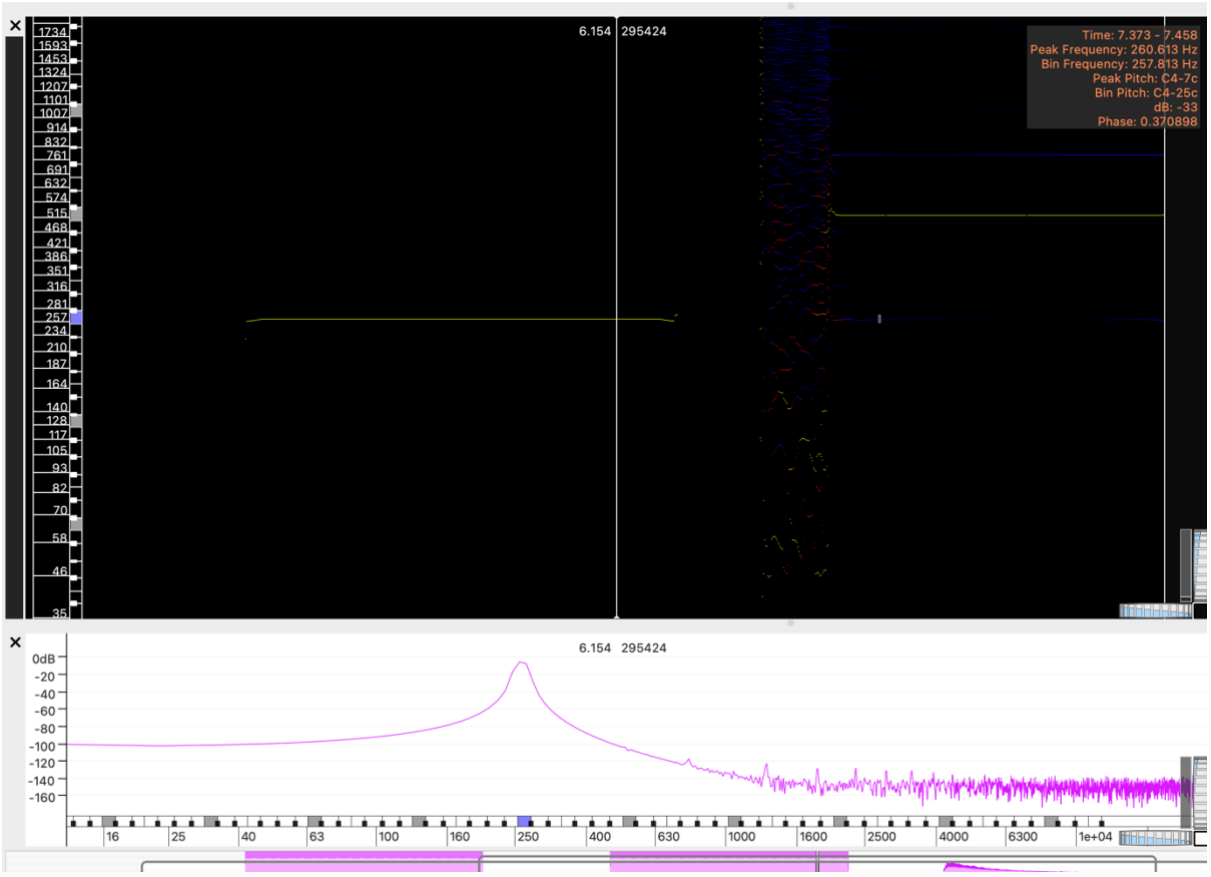


The user of my virtual piano can compare these recorded sounds. It enables users to contrast synthetic piano sounds with those from a real piano. The associated recorded sound is played back when the note is clicked, giving the user a realistic playing experience. I was able to create a sound comparison that was more accurate and thorough thanks to the use of recorded piano sounds.

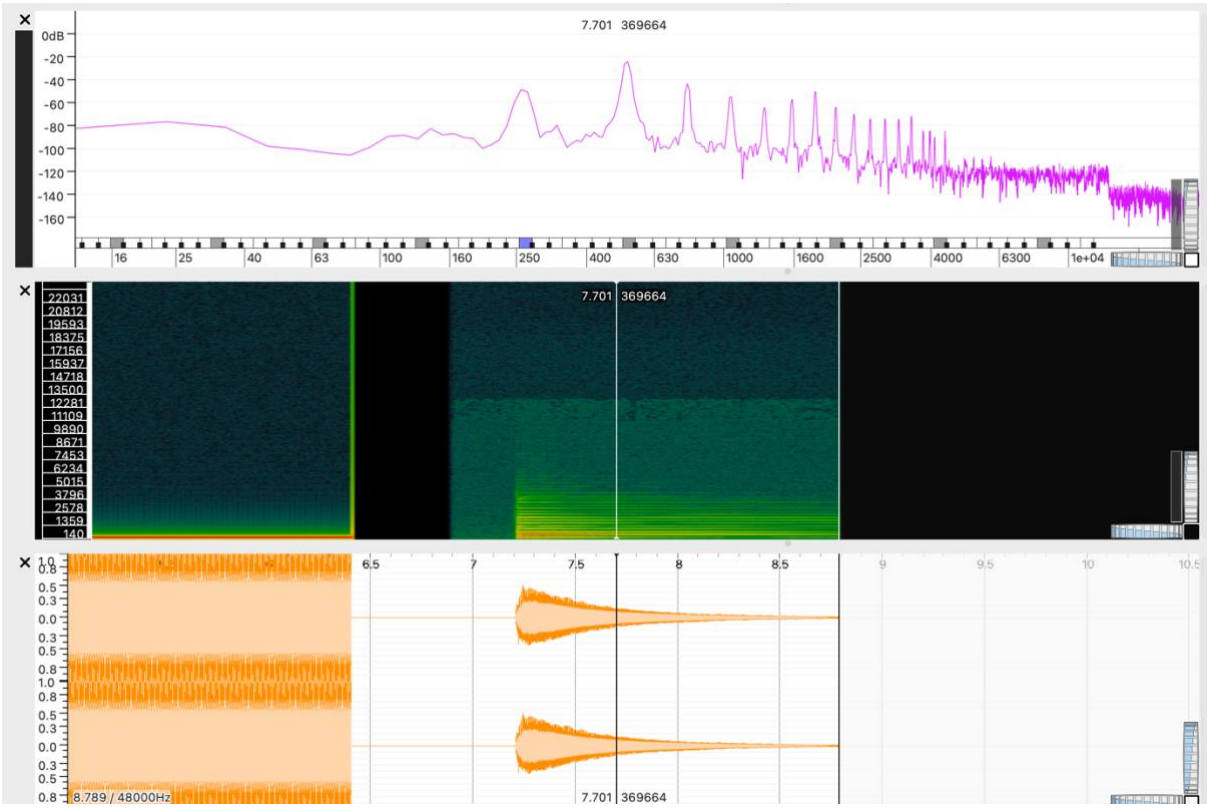
7) Analyse the sound that you've generated using a tool like Sonic Visualiser.



Here is the synthesised C4 note with the Real C4 note of a piano. On the synthesised piano note we can see there is a single frequency peak at 261.63 Hz. The peak is relatively narrow, which indicated how the sound is pure and free of harmonics. We can also see there is minimal noise and distortion.



In comparison to the real C4 piano note I downloaded



Here, we can see that the spectrogram contains a complex, rich, and multifaceted harmonic content that gives the piano its distinctive sound. On this spectrum, there are numerous peaks that correspond to the various overtone frequencies. Unlike the smooth sine wave produced by the Oscillator node, the waveform exhibits an initial attack followed by a gradual decay.

Comparing the two, we can see that the synthesised C4 note has a single peak at 261.63 Hz and a simple, pure waveform with no overtones. Comparing the two, a real c4 piano note has a more complex and dynamic sound than a sine wave that has been synthesised at the same frequency.

8) **Give a short evaluation of how well you think you implemented the model and how it could be improved or extended in future work.**

Overall, I think I implemented my virtual piano model successfully and I was able to create a range of sounds that could be played by clicking the notes. However, there are some areas that I believe I could improve on in future work.

One area that I believe could be improved is the organisation of the code for the recorded piano notes. I have repeated the lines for the recorded piano note's function which resulted in making the code a lot more lines than it needed to be and this would not be convenient if I wanted to extend the piano and introduce more octaves.

Furthermore, another area for improvement could be the implementation of other virtual instruments. While I was able to synthesise notes for the piano, I did not implement any for the drums and only used recorded noises. It would be interesting to experiment with the sound synthesis of different instruments and comparing them to real sounds.

In conclusion while my current model is functional and produces a wide range of sounds that the user can experiment with, there is definitely room for improvement and further exploration in future work.