Introduction

Cycles Per Second (CPS) has been created as a single page application, however the GUI design consists of five different 'pages' (**Appendix 1)**. All pages are imbedded within the same html file and are either hidden or shown depending on the users input and the functionality of each navigation button. The application has three different game modes such as; Tone, Noise and Samples, where each game mode requires the user to guess which frequency is either produced or boosted.

GUI Design and DSP Algorithms

The GUI design of the application took a minimalistic approach by using a monochromatic colour scheme. All game types use the 'frequency-select' panel, where the users guess of the frequency is chosen. The frequency-select panel had been divided into three sections, being the header, the body and the footer, where each section had been divided further into sub sections (Figure 1). The header, divided in three sub-sections are used to display information such as the game name, game type, the question number and a prompt saying "Correct!" or "Wrong!" in the centre. The body of the frequency-select panel consists of a slider which is used by to select the frequency. The body had further been divided into 'frequency bins' and is displayed behind the input slider, to provide a visual aid of where the frequencies lie. The thumb of the slider has a moving value to further indicate which frequency the user will select. Finally, the footer provides the majority of the controls in each game type, divided into three sub-sections; the first section holds the playback controls, the right section holds the filter and menu button and the centre section holds the 5 buttons (see appendices), the listening options, the submit, next and finish buttons. The remaining pages had also been divided into sub-sections similar to the 'frequency-select' panel to group buttons appropriately (Appendix I – J).

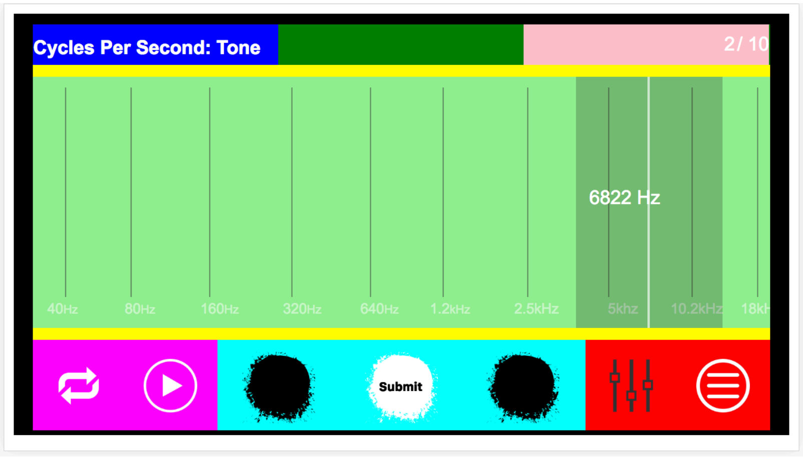
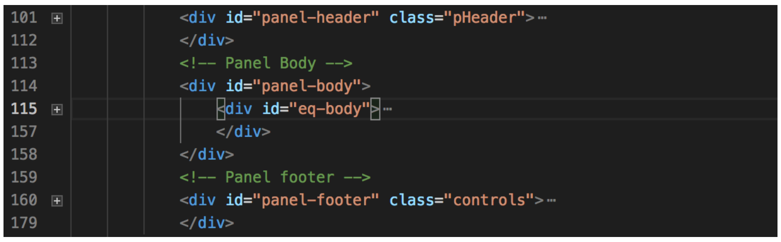


Figure 1 Separating the GUI with DIV's to have an organised single page application

The DPS algorithms used for CPS involved creating oscillators and buffers to load generated noise and audio samples. The noise and the audio samples are connected to a filter, which is used to boost frequencies across the human frequency range. A random number between the range of 20 and 20 000 is generated to determine the frequency value of the oscillator and the filter frequencies. Furthermore, the gain of the audio is controlled in-order to prevent pops and click from starting and stopping the sound. The samples used for the application have all been normalised in Ableton 9 and bounced at a level of -15dB to allow head room for the filter gain boost.

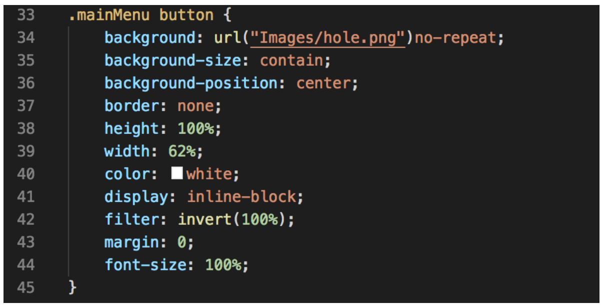
GUI and DSP algorithms Implementation

As mentioned above, the implementation of the GUI began by dividing panels into multiple sections within the HTML (Figure 2). The sections are divided by using the '<div>' objects, where a main wrapper is used to contain all sub divisions called "panel-inactive". Each '<div>' has a unique identified in order to target the styles from the CSS file and functionality from the JS files which are linked in the header of the HTML file. Furthermore, some '<div>' objects have a second identifier called a 'class' which have



```
<div id="panel-header" class="pHeader">⋯
</div>
<!-- Panel Body -->
<div id="panel-body">
    <div id="eq-body">⋯
    </div>
</div>
<!-- Panel footer -->
<div id="panel-footer" class="controls">⋯
</div>
```

Figure 2 Panels are divided into <div>'s

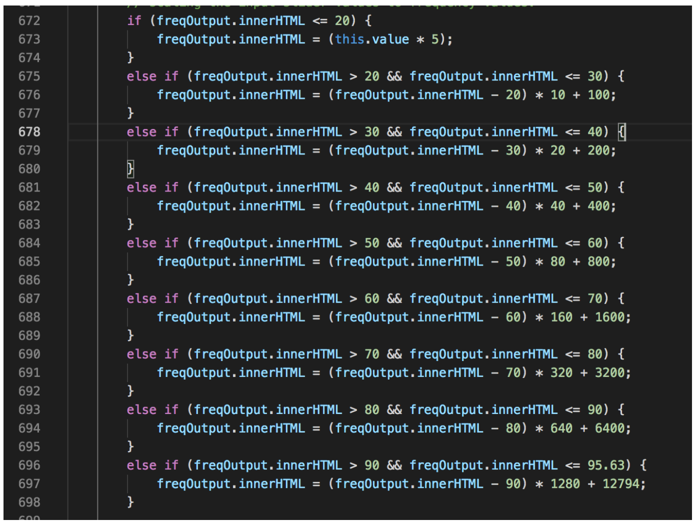


```css
.mainMenu button {
    background: url("Images/hole.png")no-repeat;
    background-size: contain;
    background-position: center;
    border: none;
    height: 100%;
    width: 62%;
    color: white;
    display: inline-block;
    filter: invert(100%);
    margin: 0;
    font-size: 100%;
}
```

Figure 3 Styling multiple buttons within a class

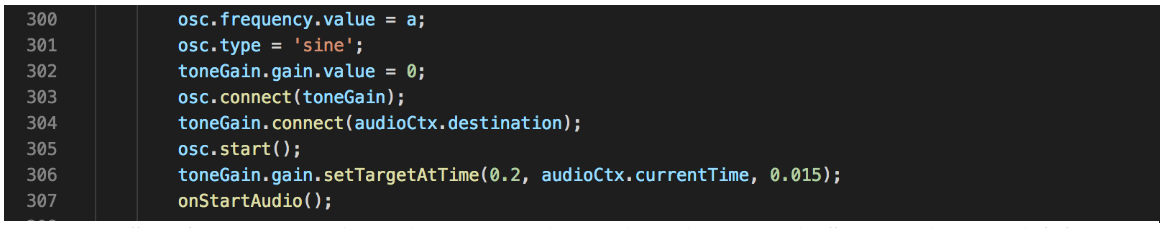
occasionally been used to style multiple objects within the divisions such as buttons, paragraph inputs and even sub divisions (Figure 3). The GUI implementation of the frequency slider required multiple steps in-order to be accurate and aesthetically pleasing. The slider is created by using an <input type='range'> object, however the values of the slider are linear and are not ideal for a frequency selector and therefore a system was developed to scale the values appropriately. The values of the input slider are ranged from 4 to 95.63 in order to achieve precise values between 20 and 20 000. Values 4 to 20 are scaled between 20 and 100 Hz, and every increment of 10 on the input slider doubles the scaled value the previous frequency e.g. Input 20 to 30 = 101 to 200Hz, input 30 to 40 = 201 to 400hz, input 40 to 50 = 401 to 800Hz etc. (figure 4).

Cycles Per Second takes advantage of the MCAD library and Web Audio API to achieve its goals. An audio context is used to control the creation of audio nodes and the execution of the audio processing and decoding (Mills, 2018). The audio context allows the possibility to create a variety of digital signal processing tools, however CPS only makes use of the oscillators, buffers, filter and gain nodes to execute the application. The tone generator creates an oscillator specified as a sinusoidal wave, and is connected to a gain node which in turn is connected to the audio context for playback (Figure 5). The frequency of the oscillator is determined by a random math generator using numbers from 0 to 100, where numbers 0 to 65 are scared between 20 and 2000, numbers 66 to 90 are scaled up between 2000 and 10 000, and numbers 90 to 100 are scaled up between 10 000 and 20 000 (Figure 6). The numbers are scaled as such, to favour the lower end of the frequency



```javascript
if (freqOutput.innerHTML <= 20) {
    freqOutput.innerHTML = (this.value * 5);
}
else if (freqOutput.innerHTML > 20 && freqOutput.innerHTML <= 30) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 20) * 10 + 100;
}
else if (freqOutput.innerHTML > 30 && freqOutput.innerHTML <= 40) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 30) * 20 + 200;
}
else if (freqOutput.innerHTML > 40 && freqOutput.innerHTML <= 50) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 40) * 40 + 400;
}
else if (freqOutput.innerHTML > 50 && freqOutput.innerHTML <= 60) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 50) * 80 + 800;
}
else if (freqOutput.innerHTML > 60 && freqOutput.innerHTML <= 70) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 60) * 160 + 1600;
}
else if (freqOutput.innerHTML > 70 && freqOutput.innerHTML <= 80) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 70) * 320 + 3200;
}
else if (freqOutput.innerHTML > 80 && freqOutput.innerHTML <= 90) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 80) * 640 + 6400;
}
else if (freqOutput.innerHTML > 90 && freqOutput.innerHTML <= 95.63) {
    freqOutput.innerHTML = (freqOutput.innerHTML - 90) * 1280 + 12794;
}
```

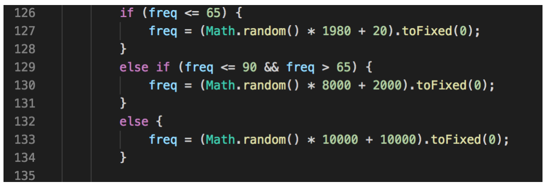
Figure 4 Scaling the range input appropriately to resemble a frequency spectrum



```javascript
osc.frequency.value = a;
osc.type = 'sine';
toneGain.gain.value = 0;
osc.connect(toneGain);
toneGain.connect(audioCtx.destination);
osc.start();
toneGain.gain.setTargetAtTime(0.2, audioCtx.currentTime, 0.015);
onStartAudio();
```

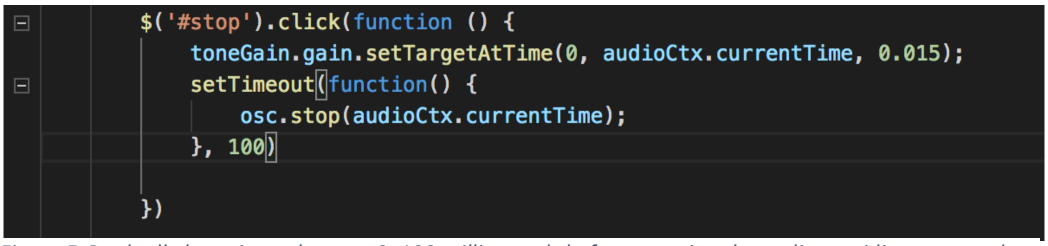
Figure 5 Oscillator being connected to audio context and gain adjusted to 0.2 gradually to avoid pops and clicks

spectrum as the higher range can accentuate ear fatigue (TMA, 2018). The noise in generated by filling a 2-channel audio buffer with random value between -1 and 1, where it is connected to a gain node, a filter and ultimately the audio context. The samples are loaded into a buffer when called by the user and are similarly connected to a gain node, the filter node and the audio context. The values of the biquad filter, used by the noise and sample modes, is set when the app is loaded. The filter type used is a 'peaking' filter, allowing frequencies inside the range to get boosted or attenuated leaving the outside frequencies unchanged. The quality factor (Q) of the filter is set to 1 and is unchanged, the gain of the filter is either 0 (off) or 10 (on) and the frequency of the filter is changed by using the same randomly generated number used for the tone generator. The gain nodes for all game types are used to avoid pops and clicks when the audio is either started or stopped, by increasing or attenuating the gain exponentially in a given time before starting or stopping the audio (Figure 5 & 7).



```javascript
if (freq <= 65) {
    freq = (Math.random() * 1980 + 20).toFixed(0);
}
else if (freq <= 90 && freq > 65) {
    freq = (Math.random() * 8000 + 2000).toFixed(0);
}
else {
    freq = (Math.random() * 10000 + 10000).toFixed(0);
}
```

Figure 6 Generating a random frequency between 20 and 20 000 Hz



```
$('#stop').click(function () {
    toneGain.gain.setTargetAtTime(0, audioCtx.currentTime, 0.015);
    setTimeout(function() {
        osc.stop(audioCtx.currentTime);
    }, 100)

})
```

Figure 5 Gradually lowering volume to 0, 100 milliseconds before stopping the audio, avoiding pops and clicks.

User Testing

Once the application was built, a combination of user testing strategies had been applied in order to eliminate any bugs or bad user experience. The app strategies used were as follows (Forbes, 2016):

- Learn to listen; providing the user with a task and not equip any other information, watch and listen, then fix any spotted problems.
- Eliminate you bias; do not compare how the user is interacting with the application to the 'right way' of using it, the user's way is always the right way.
- Test with real users; testing the application with people who are the target market.

These strategies were applied simultaneously by building the application and installing on the phones of three participants with different phone models and carried out multiple times after each build. The first couple of builds identified major issues with the GUI, where the frequency sliders would surpass the screen size and only frequencies 0 to 150 Hz could be selected. Minor bugs, such buttons being deformed on different screen sizes, had also been picked up and adjusted. The user testing required 8 different builds to be installed on the participants phones in order to enhance the user experience.

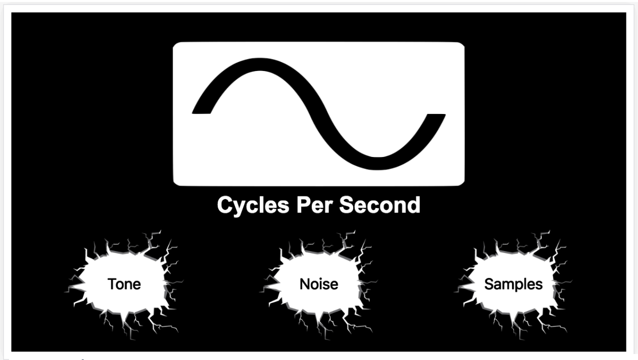
Evaluation and Conclusion

The application is in good working order, where the user is able to listen to the audio and try to determine which frequency is being played or boosted by the filter. The application does not include all the features proposed initially such as; filter gain selection, filter Q selection, user ability to load own samples, a complex point system and the game type 'Intervals'. The dropped features were done due to prioritising the optimisation of important features for the app to run smoothly. The application may lag slightly when showing the sample 'page' the first time it is opened, a minor but noticeable weakness of the app. Future improvements to the application are to fix remaining bugs mentioned, as well as include a portrait mode view. Furthermore, adding the features which had been dropped would greatly improve the application.

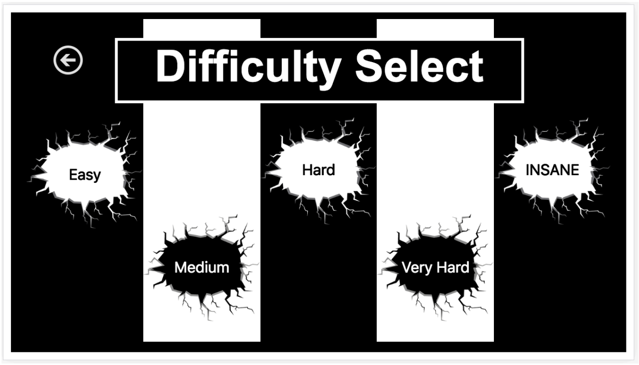
References:

- Forbes, 2016. *Eight Strategies For Effective User Testing* [online]. Forbes. Available from: https://www.forbes.com/sites/forbestechcouncil/2016/09/27/eight-strategies-for-effective-user-testing/ [Accessed 20 Jan 2019].
- Mills, C., 2018. *AudioContext* [online]. MDN Web Docs. Available from: https://developer.mozilla.org/en-US/docs/Web/API/AudioContext [Accessed 19 Jan 2019].
- Shea, C., 2018. *BiquadFilterNode* [online]. MDN Web Docs. Available from: https://developer.mozilla.org/en-US/docs/Web/API/BiquadFilterNode [Accessed 20 Jan 2019].
- TMA., 2018. *Audio Spectrum Explained - Teach Me Audio* [online]. Teach Me Audio. Available from: https://www.teachmeaudio.com/mixing/techniques/audio-spectrum/ [Accessed 20 Jan 2019].

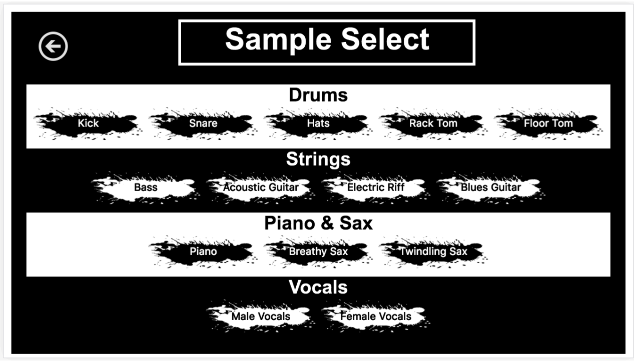
Appendices



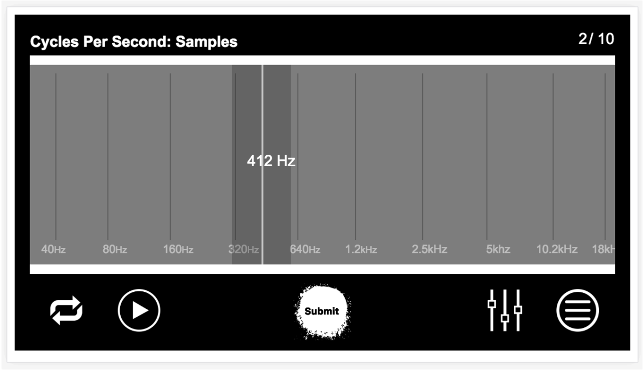
Appendix A



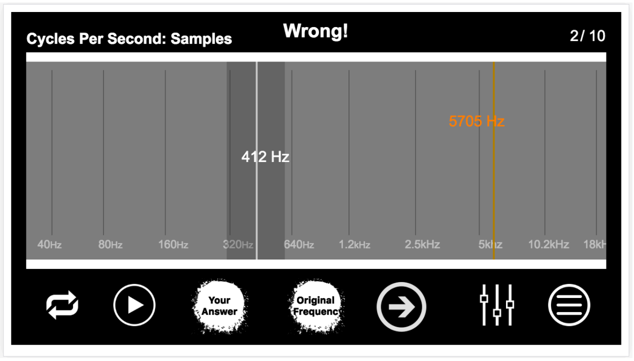
Appendix B



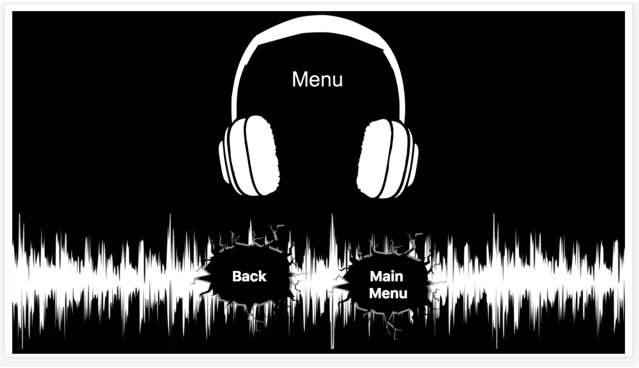
Appendix C



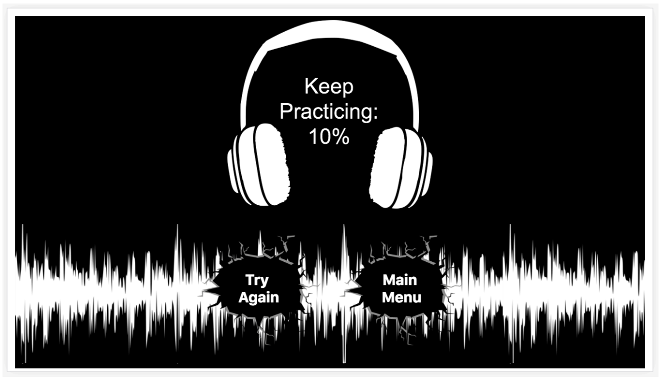
Appendix D



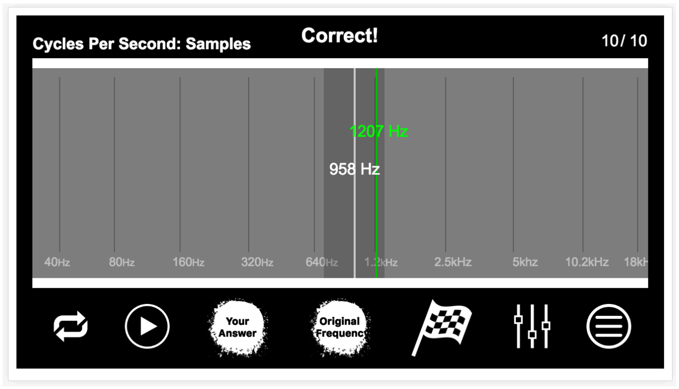
Appendix A



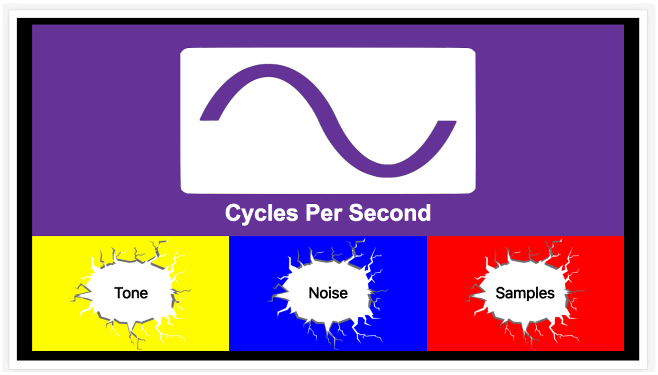
Appendix B



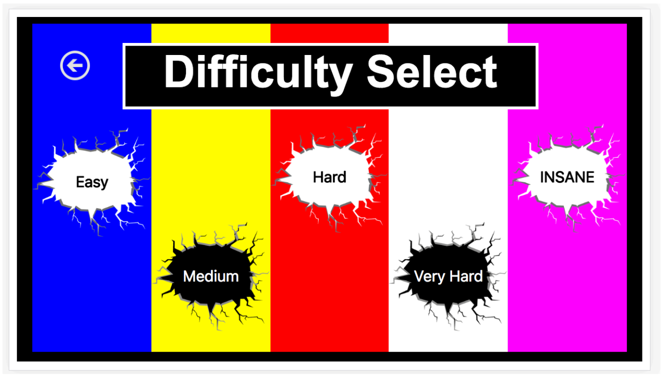
Appendix C



Appendix D



Appendix E



Appendix J