

Conor Dever

AudioWorkout Mobile Application

28th of January 2020

0.0 Introduction

Outlined in this report is the development of the Cordova mobile application, AudioWorkout. An application designed for the purposes of musical education. Detailed are the high-level design decisions behind the Graphical User Interface (GUI), Digital Digital Signal Processing (DSP) algorithms, as well as the low-level code implementation of both. Furthermore, user testing, usability and improvements for future iterations will also be discussed.

1.0 Feature Design

1.1 High-level descriptions and diagrams of GUI design

The GUI design of this application follows the simplicity principles outlined by Chee Kit Yee et al. in their 2012 conference paper, *GUI Design Based on Cognitive Psychology: Theoretical, Empirical and Practical Approaches*:

“if the users does not spend much time in learning and understanding how the GUI functions, that information system is considered to have a very effective GUI”

Simplicity is the key to effective GUI design. If there is a steep learning curve within the GUI, user engagement will suffer as a result. The GUI needs to be intuitive and easily understood. Another key step is making the GUI look great. “Beautiful interface[s] help people understand and interact with content while never competing with it”. Familiarity is another critical element, “well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect” (Apple 2020).

With these factors in mind: simplicity, familiarity and beautiful design, the first step in the GUI design was choosing the primary application colour. Teal was chosen as the primary colour, it benefits from calming qualities of blue colours and feelings associated with this colour described as “open communication and clarity of thought” (Canva 2020). An online palette generator was used to generate the secondary supporting colours of white `#fff`, grey `#333` and dark grey `#111`.

FontAwesome (2020) was selected to provide CSS icons throughout the application, to evoke feelings of familiarity and to ease the learning curve within the application. FontAwesome is used on 6 million websites throughout the web (SimilarTech 2020), including on major sites like samsung.com (2020). The GUI was designed to be easily understood and easily navigable via the familiar icons that encourage user interaction, such as the menu icons, which then update to a close window icon once the user has interacted with it.

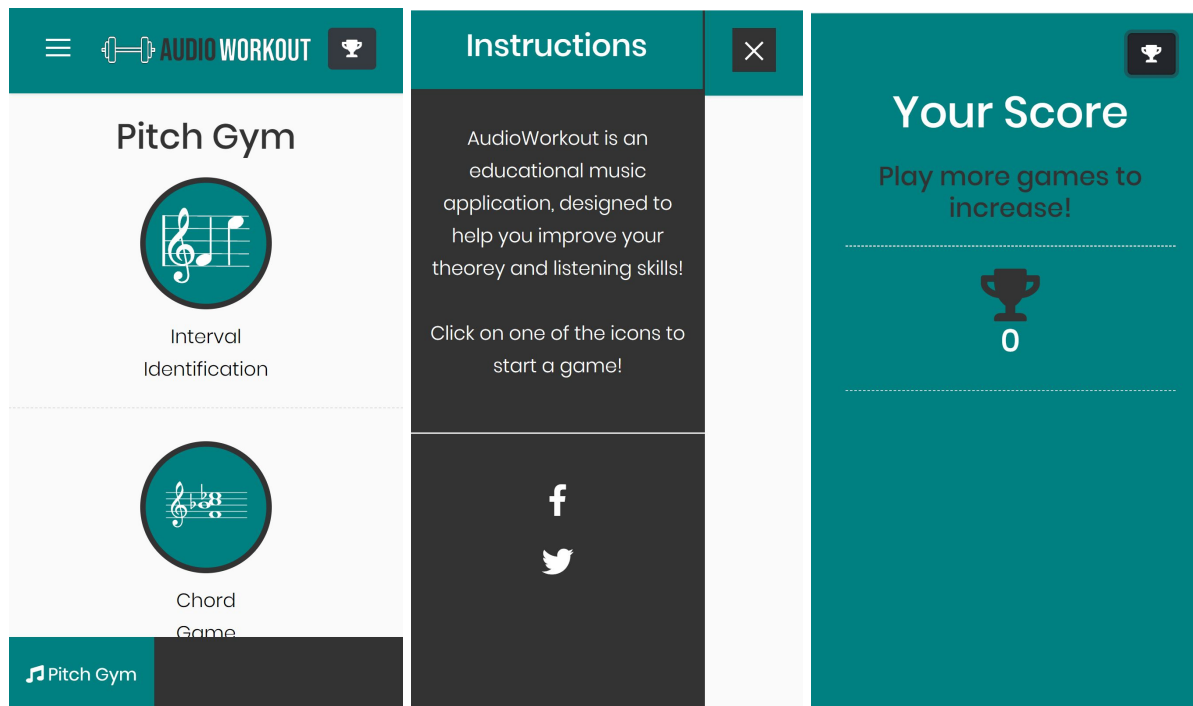


Figure 1: Primary section

Figure 2: Instructions Menu

Figure 3: The Score Menu

This simple design philosophy is evident from the homepage, where the user is not overwhelmed by the main functionality of the application. Instead they are greeted by a clean and crisp interface. The GUI of this application is split into two sections, with additional supporting menus available in each section. The primary section (**Figure 1**) is supported by an instructions menu accessed by a menu icon (**Figure 2**) and a score menu accessible via a trophy icon (**Figure 3**). The second section, is the quiz element of the application, which contains the main functionality of the application (**Figure 4**). It is supported by pause menu (**Figure 5**) accessed via the menu icon within the quiz, it also appears as the game over menu upon completion of the quiz (**Figure 6**).

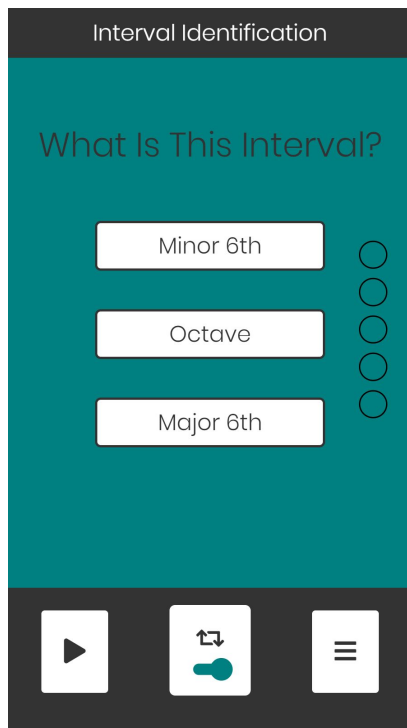


Figure 4: Quiz section

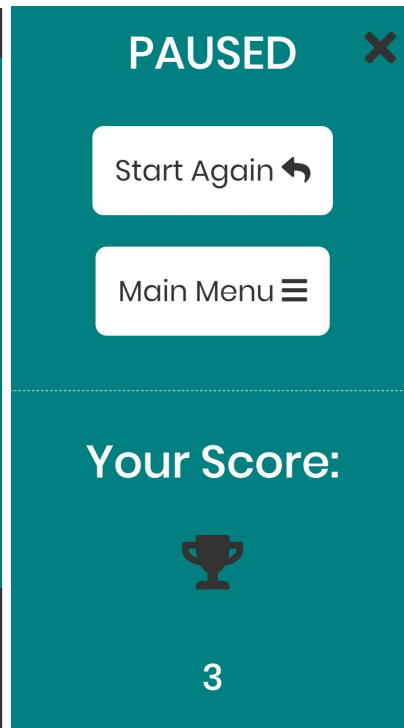


Figure 5: Pause Menu

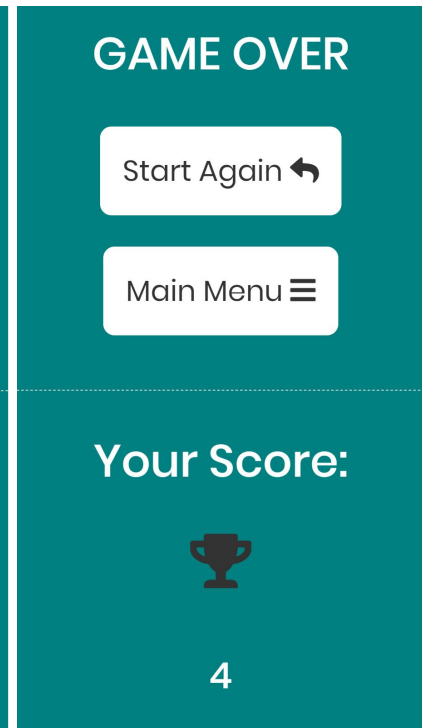


Figure 6: Game Over Menu

1.2 High-level description of DSP algorithms

Tone.js (2020) was selected as the primary framework behind the handling of audio within the application. Tone.js allows for intuitive musical abstractions built on top of the Web Audio API. It handles event scheduling in a much more simplified way in comparison to the Web Audio API (Wilson 2013), allowing for musical timing intervals instead of millisecond values and includes pre-built synthesisers, as well as polyphony, which were all highly desired features utilised throughout this application.

The deployment of the DSP algorithms can be split into two sections, generation and playability. The ethos behind this application is musical education, as such it was not desired that the user would be able to guess the correct answers by predicting the outcome of the algorithm generating the sounds the user interacts with. Therefore all of the DSP algorithms are generated using random numbers where possible. When a user selects from one of three quizzes: Interval Identification; Chord Game; and Frequency Accuracy. The program matches the title `<h1>` text of the selected game and generates the appropriate question. The same method is used to play the audio, the program matches the appropriate play function to the quiz title, as the handling of audio is slightly different in each context.

2.0 Feature Implementation

2.1 GUI implementation

Bootstrap (2020) was instrumental in the layout of the home section of the application. It is a HTML, CSS and JS library that provides prebuilt responsive, mobile-first layout options. It provides containers and rows that size appropriately for mobile or desktop. Its use can be seen in this example providing pre-built rows:

```
<div class="singleRow">
    <div class="iconContainer">
        <div class="iconBorder">
            <div class="iconPitchCircle">
                <div id="intervalIcon">
                    
                    <div class="overlay"></div>
                </div>
            </div>
        </div>
    </div>
    <p>Interval<br>
    Identification</p>
</div>
<div class="line"></div>
```

Custom CSS was used to beautify elements and add conformity where possible. In the example above, an image is selected to serve as the interval icon, CSS (**Appendix A**) was used to detail these images with a circular border (**Figure 7**) and teal highlight as they open. Completely transforming the original image in the process in a way that makes the icons uniform.



Figure 7: Transformation of input image (left) to icon used on homepage (right) via CSS

jQuery (2020) was used to add extra javascript functionality to the application. The majority of events within the application are handled by jQuery, in the form of click events. These click events show or hide the appropriate content via id tags, such as the example below which hides the main page `#pitchGym` and shows the `#quizGame` for Interval Identification, as you can see this is not done through simple `.show()` and `.hide()` functions. Instead it is doing using jQuery animations, that time fades/events appropriately to deliver a higher UX.

```
$("#intervalIcon").click(function () {  
    $('#pitchGym').fadeToggle(500);  
    quizTitle.innerHTML = "Interval Identification";  
    resetQuiz();  
    $('#quizGame').slideDown(1000);  
    $('#controlMenu').slideDown(1800);  
});
```

Following on from this example, again common to the theme of GUI simplicity and conformity, is the generation of the quiz. The three different games or quizzes are the same HTML page with the appropriate HTML elements update via javascript to display different quizzes, through the use of the `populateQuiz()` and `renderQuestion()` functions (**Appendix B**). The quiz page allows features a progress bar, that colours in green for a correctly answered questions, and red for an incorrectly answered questions (**Figure 8**).

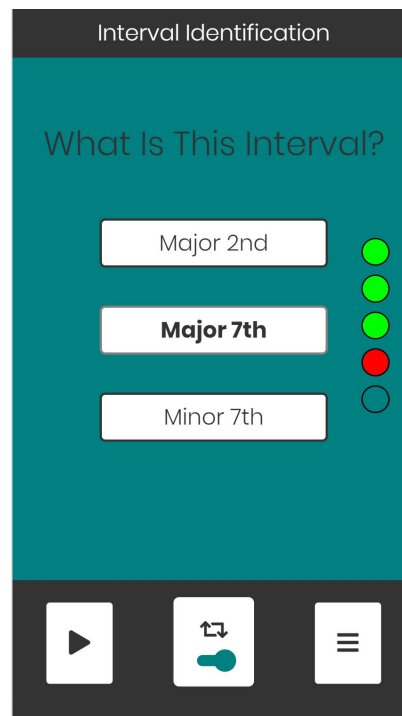


Figure 8: Quiz Progress Indicator

2.2 DSP algorithms implementation

The DSP implementation begins with the initialisation of three Tone.js elements utilised within the application. These elements are synthesisers and are called on to play intervals, chords and frequencies generated by the functions discussed below.

```
//-----|Declare Tone.js Elements|-----//
//-----//

// Monophonic synth to play intervals
var monoSynth = new Tone.Synth().toMaster();

// Polyphone synth to play chords
var polySynth = new Tone.PolySynth(3, Tone.Synth, {
  "volume": 0,
  "oscillator": {},
  "portamento": 0.005
}).toMaster();

// Sine
var sine = new Tone.Oscillator(freq, "sine").toMaster();
```

Each quiz is generated randomly, this generation starts with the `genRanFreq()` function which generates a random frequency using the `Math.random` function. This is done a chance basis, however we want to minimise the use of frequencies that are hard to hear on a traditional phone speaker. Following a mixing guide outlined by Waves (White 2018), and through testing on a OnePlus 5T the phone is able to replicate frequencies from 150 Hz to 10 kHz without issue. There is therefore an 80% chance that the frequency generated will be within this range. Less desirable (hard for the user to listen to) frequencies such as the ones above 10kHz where given a 15% chance of occurring, while frequencies sub-150 Hz only have a 5% chance of occurring. This function also generates two additional frequencies, these two additional frequencies are generated within a range of +/- 1000 of the first frequency to populate the quiz with two incorrect answers While loops have been implemented to ensure the additionally generated frequencies are not the same as the first frequency. This function serves the Frequency Accuracy section of the application, a different generation function is utilised within the two other quizzes.

```
//-----| GENERATE RANDOM FREQUENCY FUNCTION |-----//
// This function generates a random frequency every times it's called
// The generation of that frequency is done a chance basis to select
// frequencies that can be heard through a phone's speaker
function genRanFreq() {
```

```

//Generating a random number between 1 and 100 (Percentage%)
freq = (Math.random() * 100).toFixed(1);

// 80% chance that the frequency generated is between 150 Hz
// and 10 kHz
if (freq <= 80) {
    freq = Math.floor(Math.random() * (10000 - 150 + 1)) + 150;
}
// 15% chance its between 10 kHz and 20 kHz
else if (freq <= 95 && freq > 80) {
    freq = Math.floor(Math.random() * (20000 - 10000 + 1)) + 10000;
}
// 5% chance its below 150 Hz (not easily replicated on a phone
speaker)
else {
    freq = Math.floor(Math.random() * (150 - 0 + 1)) + 0;
}

// This generates a second random frequency used as a possible
outcome within the quiz element
// It generates a number +/- 1000 Hz of the random frequency
generated above
freq2 = Math.floor(Math.random() * ((freq + 1000) - (freq - 1000)
+ 1)) + (freq - 1000);

// Loops around until the two random frequencies aren't equal to
each other
while (freq2 === freq || freq2 <0)
{
    freq2 = Math.floor(Math.random() * ((freq + 1000) - (freq
- 1000) + 1)) + (freq - 1000);
}

// This generates a third random frequency used as a possible
outcome within the quiz element
// It generates a number +/- 1000 Hz of the random frequency
generated above
freq3 = Math.floor(Math.random() * ((freq + 1000) - (freq - 1000)
+ 1)) + (freq - 1000);

```



```

    // Loops around until the two random frequencies aren't equal to
    each other
    while (freq3 === freq2 || freq3 < 0)
    {
        freq2 = Math.floor(Math.random() * ((freq + 1000) - (freq
- 1000) + 1)) + (freq - 1000);
    }

}

```

The `genRanNote()` function serves as the initial generation method behind the Interval Identification and Chord Game quizzes. It operates in a similar way to `genRanFreq()` however, the output it generates is from 0-127 representing the MIDI note scale.

```

//-----| GENERATE RANDOM MIDI NOTE |-----//
// This function generates a random midi note every times it's called
// The generation of that frequency is done a chance basis to select
// frequencies that can be heard through a phone's speaker
function genRanNote() {

    //Generating a random number between 1 and 100 (Percentage%)
    midiNote = (Math.random() * 100).toFixed(1);

    // 97% chance that the frequency generated is between 36 (C2) and 71
(B4)
    if (midiNote <= 100) {
        midiNote = Math.floor(Math.random() * (71 - 36 + 1)) + 36;;
    }
    // 2% chance it's above 71 (B4), high registers people are
unfamiliar with
    else if (midiNote <= 97 && midiNote > 99) {
        midiNote = Math.floor(Math.random() * (127 - 72 + 1)) + 72;;
    }
    // 1% chance it's below C2, bass clef not easily heard on phone
speaker
    else {
        midiNote = Math.floor(Math.random() * (35 - 0 + 1)) + 0;
    }
}

```

The `midiNote` variable updated via the `genRanNote()` function is then used as the input to the `genRanInterval(int)` function. This function then generates a random musical interval based on this `midiNote` by selecting a music interval from a `semiToneArray[]` and then applies it to `midiNote2`. The function then updates a `intervalType` variable with the name of the musical interval from the `intervalTypeArray[]`. Much like the `genRanFreq()` function, `genRanInterval()` generates two incorrect musical intervals that are used to populate the quiz with incorrect answers.

```
//-----| GENERATE RANDOM INTERVAL |-----//
// This function generates a musical interval every times it's called
// The generation of the interval is performed from a section of arrays
// The variable argument is the midiNote variable populated by
genRanNote()
function genRanInterval(int) {
    // An array of all the semi tones, which added to the midiNote will
    produce the interval
    var semiToneArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];
    // An array of all the interval names
    var intervalTypeArray = ["Minor 2nd", "Major 2nd", "Minor 3rd",
    "Major Third", "Perfect 4th", "Augmented 4th", "Perfect 5th", "Minor 6th",
    "Major 6th", "Minor 7th", "Major 7th", "Octave"]
    // Creates a random variable that generates a step from 0 - 11 (12
    possible outcomes, same length as the arrays above)
    var ranStep = Math.floor(Math.random() * (11 - 0 + 1)) + 0;
    // int = midiNote - input of function
    // midiNote2 is the midiNote + the selection from the semitone
    array
    midiNote2 = int + semiToneArray[ranStep];
    // the intervalType variable is used to populate answers for the
    quiz
    intervalType = intervalTypeArray[ranStep];
    // generates a second intervalType for the quiz
    intervalType2 = intervalTypeArray[Math.floor(Math.random() *
    intervalTypeArray.length)];
    // Generate Random Answer again if intervalTypes are the same
    while (intervalType2 === intervalType) {
        intervalType2 = intervalTypeArray[Math.floor(Math.random() *
        intervalTypeArray.length)];
    }
}
```

```

    // Generate Random Interval Answers for quiz
    intervalType3 = intervalTypeArray[Math.floor(Math.random() *
intervalTypeArray.length)];
    // Generate Random Answer again if intervalTypes are the same
    while (intervalType3 === intervalType2 || intervalType3 ===
intervalType) {
        intervalType3 = intervalTypeArray[Math.floor(Math.random() *
intervalTypeArray.length)];
    }
}

```

The generation of chords by the `genRanChords(int)` function is very similar to the method described above. Each of the generation functions are followed by a function that plays what was generated, these are: `playInterval()`, `playChord()` and `playFreq()`. Each differ slightly as they handle different tone.js elements.

Below is the `playRanInterval()` function, it receives `midiNote`, and `midiNote2` as its input. Its first step is to ensure the polySynth is disconnected as the two synthesisers can interfere with each other if they are both simultaneously connected to the master output. Unlike the WebAudio API, Tone.js conveniently contains a transport, which can be started, stopped and resumed by the program. However, with UX in mind it was decided that the interval should loop if the loop toggle is set to true, so that the user doesn't continually need to press the play button. Therefore the interval itself is looped by default within the `Tone.Transport.scheduleRepeat`, the MIDI notes are placed within a note array and the function plays each of them for a duration of '4n' or a quarter note. The loop then repeats every '1m' or one measure, and only exits the loop if the loop toggle is set to false.

```

//-----| PLAY INTERVAL FUNCTION |-----//
// This function plays a musical interval every time it's called using
Tone.js
// It also controls the functionality of the transport buttons within
the
// interval quiz section of the application
// input variables are midiNote and midiNote2
function playInterval(int, int2) {
    // Disconnects the tone.js chord synth
    polySynth.disconnect();
    // Connects the tone.js interval synth
    monoSynth.toMaster();
    // Convert Midi Note To Frequency
    var note1 = Nexus.mtof(int);

```

```

var note2 = Nexus.mtof(int2);
// Create Array of Notes
var notes = [note1, note2];
// Index Counter For Loop
var index = 0;
// Create a Tone.js repeat function (used to loop sound)
// Repeats after one measure "1m" (one bar)
Tone.Transport.scheduleRepeat(function (time) {
    repeat(time);
}, "1m");
// Create a Tone.js repeat function (used to loop sound)
// Repeats after one measure "1m" (one bar)
function repeat(time) {
    // %modulo around the whole notes array
    var note = notes[index % notes.length];
    // Play each note for a 1/4 note "4n"
    monoSynth.triggerAttackRelease(note, "4n", time);
    // advance loop index
    index++;
    // break out of the function if the Nexus UI loop toggle is off
    // Only happens after the second note is played, so whole
interval is heard
    if (loopToggle.state === false && index === 2) {
        // Stop the tone.js transport and change style of
play/pausebuttons
        stopAudio();
    }
}
// start playback
Tone.Transport.start();
// Change state of play/pause
startAudio();
// If pause button is clicked stop the audio
$("#pauseBtn").click(function () {
    // stop playback t/o 500ms
    setTimeout(stopAudio(), 500);
});

// If the loopToggle is turned to off then pause after the second
note is heard
loopToggle.on('change', function (v) {

```

```

        if (loopToggle.state === false && index === 2) {
            // stop playback t/o 500ms
            setTimeout(stopAudio(), 500);
        }
    });
}

```

These play functions are supported by `startAudio()` and `stopAudio()` these functions update the CSS styling of the play and pause button. `StopAudio()` contains an extra function, ensuring that all the different synth and oscillator elements are stopped immediately when it's called.

```

//-----| START AUDIO FUNCTION |-----//
// This function changes the CSS style properties of the play/pause
// buttons within the quiz section of the application
function startAudio() {
    // Hide play button
    playBtn.style = "display: none;";
    // show pause button
    pauseBtn.style = "display: unset;";
}

//-----| STOP AUDIO FUNCTION |-----//
// This function changes the CSS style properties of the play/pause
// buttons within the quiz section of the application and also stops
// the transport and oscillators
function stopAudio() {
    // show play button
    playBtn.style = "display: unset;";
    // hide pause button
    pauseBtn.style = "display: none;";
    // stop Tone.js transport
    Tone.Transport.stop();
    // stop the sine oscillator
    sine.stop(1);
}

```

To summarize the DSP is implemented in by a primary generation function, a secondary function and then is played by a selective play function represented by the table below.

Name	Generation	2nd Generation	Played By
Interval Identification	genRanNote()	genRanInterval()	playInterval()
Chord Game	genRanNote()	genRanChord()	playChord()
Frequency Accuracy	genRanFreq()	N/A	playFreq()

3.0 User Testing

3.1 Determining the usability of entire app

Mobile applications have become so ubiquitous in our everyday lives, with 204 billion downloads from Google Play Store and Apple's App Store in 2019 alone (Clement 2019). With this metric in mind we can start to understand the competition that face this application. Google's Play Store as of December 2019, lists 2.9 million unique mobile applications (Statistica 2019). Studying the store's most successful applications reveals they are "perceive[d] by users as being easy to learn, user-friendly and less time-consuming when completing tasks".

Therefore we must ensure that our application meets a strict usability guidelines, as "usability and user experience is a key element in creating successful high-quality applications" (Mifsud 2016). Testing for usability can be conducted in a number of ways, but due to limitations on the testing resources of this project, lab testing was the primary method used for this application. This testing method is performed by testing the application on a range of low power devices, to determine their usability across a range of devices (Liang et al 2011). Android Studio (2020) features an emulator that allows us to emulate a range of low powered Android phones, such as the Samsung J3. From these tests we were able to conclude that the application would function across a wide range of Android devices.

3.2 Eliciting any bugs/bad user experience

There have been some recurring issues that revealed themselves in initial testing of the single page application through Google Chrome's developer mode. These included freezing, and delayed responses to button clicks. The issue could be resolved by reloading the page. However these issues didn't reappear once the application was wrapped with Cordova and deployed as an SDK on a One Plus 5T. There is still an issue with the chord game where the chord played will contain a fragment of a higher or lower pitch than the chord being played, almost like a voice of the synth has gotten stuck on. Measures were taken to disconnect synthesisers so they wouldn't interfere with each other. However, it was ultimately found that other users have been having

issues with the same elements of the polysynth object listed on Tone.js' GitHub account (Github 2019), this bug may have reappeared in their code.

4.0 Critical Evaluation and Conclusions

4.1 Strengths/weaknesses of your complete app

The main strength of this application is its randomly generated aspect. The user won't be bored by a one dimensional set quiz, instead they are dynamically generated on the fly and different each time the users plays them. Another strength is the clean and consistent design present throughout the application.

Ultimately the biggest weakness of this application is it's lack of gaming aspects, such as levels and objectives. A system where the user has to complete a daily challenge, or can level up by playing more quizzes would contribute to greater user engagement. The present scoring system is lacking this aspect.

4.2 Any improvements you would make for future iterations


In future iterations we would like to develop on the number of quizzes available to the user and to develop the initially proposed Rhythm and Theory Gyms. Ultimately those were left out of this iteration due to time constraints on the project. We would also like to add a tiered system with levels and objects, like the ones present within the application that inspired AudioWorkout, SoundGym (2020). Thus far the application has solely been tested and deployed on android, in the next iteration it is desired that we would deploy and test an iOS version of the application.

References

Android, 2020. *Run apps on the Android Emulator | Android Developers* [online]. Android Developers. Available from: <https://developer.android.com/studio/run/emulator> [Accessed 30 Jan 2020].

Apple, 2020. *Themes - iOS - Human Interface Guidelines - Apple Developer* [online]. Developer.apple.com. Available from: <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/> [Accessed 22 Jan 2020].

Bootstrap, 2020. *Bootstrap* [online]. Getbootstrap.com. Available from: <https://getbootstrap.com/> [Accessed 29 Jan 2020].



Canva, 2020. *Teal Meaning, Combinations and Hex Code - Canva Colors* [online]. Canva's Design Wiki. Available from: <https://www.canva.com/colors/color-meanings/teal/> [Accessed 15 Jan 2020].

Clement, J., 2019. *Topic: Mobile app usage* [online]. www.statista.com. Available from: <https://www.statista.com/topics/1002/mobile-app-usage/> [Accessed 30 Jan 2020].

FontAwesome, 2020. *Font Awesome* [online]. [Fontawesome.com](https://fontawesome.com/). Available from: <https://fontawesome.com/> [Accessed 15 Jan 2020].

GitHub, 2019. *Polysynth Voice Inconsistency · Issue #426 · Tonejs/Tone.js* [online]. GitHub. Available from: <https://github.com/Tonejs/Tone.js/issues/426> [Accessed 18 Jan 2020].

jQuery, 2020. *jQuery* [online]. [Jquery.com](https://jquery.com). Available from: <https://jquery.com/> [Accessed 15 Jan 2020].

Liang, H., Song, H., Fu, Y., Cai, X. and Zhang, Z., 2011. *A remote usability testing platform for mobile phones*. In: 2011 IEEE International Conference on Computer Science and Automation Engineering [online]. Shanghai: IEEE, pp. 312-316. Available from: https://ieeexplore.ieee.org/document/5952477?tp=&arnumber=5952477&url=http:%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs_all.jsp%3Farnumber%3D5952477 [Accessed 18 Jan 2020].

Mifsud, J., 2016. *Usability Testing Of Mobile Applications: A Step-By-Step Guide* [online]. Usabilitygeek.com. Available from: <https://usabilitygeek.com/usability-testing-mobile-applications/> [Accessed 18 Jan 2020].

Samsung, 2020. *Mobile | TV | Home Electronics | Home Appliances | Samsung US* [online]. Samsung Electronics America. Available from: <https://www.samsung.com/us/> [Accessed 18 Jan 2020].

SimilarTech, 2020. *Font Awesome Market Share and Web Usage Statistics* [online]. SimilarTech. Available from: <https://www.similartech.com/technologies/font-awesome> [Accessed 15 Jan 2020].

SoundGym, 2020. *SoundGym | Audio Ear Training Online* [online]. SoundGym. Available from: <https://www.soundgym.co/> [Accessed 15 Jan 2020].

Statista, 2019. *Google Play Store: number of apps 2019 | Statista* [online]. Statista. Available from: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/> [Accessed 15 Jan 2020].

Tone.js, 2020. *Tone.js* [online]. Tonejs.github.io. Available from: <https://tonejs.github.io/> [Accessed 15 Jan 2020].

White, M., 2018. 8 Tips for Great Mixes on Small Speakers | Waves [online]. waves.com. Available from: <https://www.waves.com/tips-for-great-mixes-on-small-speakers> [Accessed 15 Jan 2020].

Wilson, C., 2013. A Tale of Two Clocks - Scheduling Web Audio with Precision - HTML5 Rocks [online]. HTML5 Rocks - A resource for open web HTML5 developers. Available from: <https://www.html5rocks.com/en/tutorials/audio/scheduling/> [Accessed 15 Jan 2020].

Yee, C., Ling, C., Yee, W. and Zainon, W., 2012. GUI design based on cognitive psychology: Theoretical, empirical and practical approaches. In: 8th International Conference on Computing Technology and Information Management (NCM and ICNIT) [online]. Seoul: IEEE, pp. 836-841. Available from: <https://ieeexplore.ieee.org/abstract/document/6268617> [Accessed 15 Jan 2020].

Bibliography

Pluralsight, 2020. *Introduction to the Web Audio API* [online]. Pluralsight.com. Available from: <https://www.pluralsight.com/courses/web-audio-api-introduction#> [Accessed 30 Jan 2020].

Appendix

Appendix A - Icon CSS

```
/* -----  
  ICONS  
----- */  
.pitchIcon {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
  width: 80px;  
}  
  
.iconContainer {  
  position: relative;  
  font-size: 1.5em;  
  text-align: center;
```

```

    height: 80px;
    border-radius: 50%;
}

.iconContainer:hover .overlay {
    opacity: .7;
    border-radius: 50%;
}

.overlay {
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
    height: 100%;
    width: 100%;
    opacity: 0;
    transition: .5s ease;
    background-color: teal;
    border-radius: 50%;
}

.iconBorder {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 120px;
    height: 120px;
    background: #333;
    border-radius: 50%;
}

.iconBorder:hover {
    -webkit-border-radius: 60%;
    -moz-border-radius: 60%;
    border-radius: 60%;
    -webkit-box-shadow: 0px 0px 10px 0px rgba(0,128,128 ,1 );
    -moz-box-shadow: 0px 0px 60% 0px rgba(0,128,128 ,1 );
    box-shadow: 0px 0px 60% 0px rgba(0,128,128 ,1 );
}

```

```

}

.iconBorder:hover .singleRow p {
    text-shadow: 0 0 32px black;
}

.iconPitchCircle {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 110px;
    height: 110px;
    background: teal;
    border-radius: 50%;
}

```

Appendix B - HTML Quiz & Javascript Quiz

HTML

```

<!--QUIZ SECTION -->
<div id="quizGame">
    <div class="quizWrapper">
        <h1 id="quizTitle"></h1>
        <div id="qContainer">
            <div id="questionsLayout">
                <div id="question"></div>
                <div id="choices">
                    <div id="progress"></div>
                    <div class="choice" id="A"></div>
                    <div class="choice" id="B"></div>
                    <div class="choice" id="C"></div>
                </div>
            </div>
            <div id="scoreContainer" style="display: none"></div>
        </div>
        <div id="controlMenu">
            <div class="buttonColumn">
                <button id="playBtn"><i class="fas

```

```

fa-play"></i></button>
    </div>
    <div class="buttonColumn">
        <button id="pauseBtn" hide><i class="fas
fa-pause"></i></button>
    </div>
    <div class="buttonColumn">
        <div id="loopBtn"><i class="fas fa-retweet"></i>
        <div id="loopToggle"></div>
    </div>
    <div class="buttonColumn">
        <button id="menuBtn"><i class="fas
fa-bars"></i></button>
    </div>
</div>
</div>
</div>

```

JAVASCRIPT

```

//-----|POPULATE QUIZ FUNCTION|-----//
// This function generates questions for all three quizzes
// The beauty of this function is that the generation occurs
// Randomly, a new question is generated randomly every time the
// function is called. The input variable is runningQuestion
function populateQuiz(int) {

    // Create a step variable
    var quizChoice = 1;
    // Generate a number between 1 and 3
    quizChoice = Math.floor(Math.random() * (3 - 1 + 1) + 1);

    // If the user has selected interval quiz
    if (quizTitle.innerHTML === "Interval Identification") {

        // Generate a random midi note
        genRanNote();
        // Generate a random interval based on this midi note
        genRanInterval(midiNote);
    }
}

```

```

// In a multiple choice quiz there are three possible outcomes
// This function randomly selects one of three outcomes
// Therefore the user can't select answers based on probability
if (quizChoice == 1) {
    questions[int] = {
        question: "What Is This Interval?",
        choiceA: intervalType,
        choiceB: intervalType2,
        choiceC: intervalType3,
        correct: "A"
    }
}

if (quizChoice == 2) {
    questions[int] = {
        question: "What Is This Interval?",
        choiceA: intervalType2,
        choiceB: intervalType,
        choiceC: intervalType3,
        correct: "B"
    }
}

else if (quizChoice == 3) {
    questions[int] = {
        question: "What Is This Interval?",
        choiceA: intervalType2,
        choiceB: intervalType3,
        choiceC: intervalType,
        correct: "C"
    }
}
}

// If the user has selected Chord Game
if (quizTitle.innerHTML === "Chord Game") {

    // Generate a random Midi Note
    genRanNote();
}

```

```

// Generate a random chord based on the Midi Note
genRanChord(midiNote);

// In a multiple choice quiz there are three possible outcomes
// This function randomly selects one of three outcomes
// Therefore the user can't select answers based on probability
if (quizChoice == 1) {
    questions[int] = {
        question: "What Is This Chord?",
        choiceA: chordType,
        choiceB: chordType2,
        choiceC: chordType3,
        correct: "A"
    }
}

if (quizChoice == 2) {
    questions[int] = {
        question: "What Is This Chord?",
        choiceA: chordType2,
        choiceB: chordType,
        choiceC: chordType3,
        correct: "B"
    }
} else if (quizChoice == 3) {
    questions[int] = {
        question: "What Is This Chord?",
        choiceA: chordType2,
        choiceB: chordType3,
        choiceC: chordType,
        correct: "C"
    }
}
}

// If the user has selected Frequency Accuracy
if (quizTitle.innerHTML === "Frequency Accuracy") {
    // Generate random frequency
    genRanFreq();
}

```

```

// In a multiple choice quiz there are three possible outcomes
// This function randomly selects one of three outcomes
// Therefore the user can't select answers based on probability
if (quizChoice == 1) {
    questions[int] = {
        question: "What Is This Frequency",
        choiceA: freq,
        choiceB: freq2,
        choiceC: freq3,
        correct: "A"
    }
}

if (quizChoice == 2) {
    questions[int] = {
        question: "What Is This Frequency",
        choiceA: freq2,
        choiceB: freq,
        choiceC: freq3,
        correct: "B"
    }
} else if (quizChoice == 3) {
    questions[int] = {
        question: "What Is This Frequency",
        choiceA: freq2,
        choiceB: freq3,
        choiceC: freq,
        correct: "C"
    }
}
}

//-----|RESET QUIZ FUNCTION|-----//
// This function resets the variables used in the quiz
function resetQuiz() {
    progress.innerHTML = "";
    runningQuestion = 0;
    score = 0;
    // Calls function to start quiz

```

```

    startQuiz();
}

//-----|RENDER QUESTION FUNCTION|-----//
// This function renders the quiz questions listed in populateQuiz()
// It updates the html elements of the quiz page
function renderQuestion() {

    // Calls the function to populate the quiz with a question
    populateQuiz(runningQuestion);

    // Create variable for question index
    let q = questions[runningQuestion];

    // Generate question title
    question.innerHTML = "<p>" + q.question + "</p>";
    // Generate choice A
    choiceA.innerHTML = q.choiceA;
    // Generate choice B
    choiceB.innerHTML = q.choiceB;
    // Generate choice C
    choiceC.innerHTML = q.choiceC;
}

//-----|START QUIZ FUNCTION|-----//
// This function calls all the varies quiz functions
// It also changes the html element to display the questions
function startQuiz() {
    renderQuestion();
    questionsLayout.style.display = "block";
    renderProgress();
}

// render progress
function renderProgress() {
    for (let qIndex = 0; qIndex <= lastQuestion; qIndex++) {
        progress.innerHTML += "<div class='prog' id=" + qIndex +
"></div>";
    }
}

```



```

// checkAnswer

function checkAnswer(answer) {
    stopAudio();
    if (answer == questions[runningQuestion].correct) {
        // answer is correct
        userScore++;
        // change progress color to green
        answerIsCorrect();
    } else {
        // answer is wrong
        // change progress color to red
        answerIsWrong();
    }
    if (runningQuestion < lastQuestion) {
        runningQuestion++;
        renderQuestion();
    } else {
        // end the quiz and show the score
        scoreRender();
    }
}

// answer is correct
function answerIsCorrect() {
    document.getElementById(runningQuestion).style.backgroundColor =
"#0f0";
}

// answer is Wrong
function answerIsWrong() {
    document.getElementById(runningQuestion).style.backgroundColor =
"#f00";
}

// score render
function scoreRender() {

    // calculate the amount of question percent answered by the user

```

```
const scorePerCent = Math.round(100 * score / questions.length);
menuTitle.innerHTML = "GAME OVER";
closeBtn.style = "display: none;";
displayUserScore.innerHTML = userScore;
$('.quizMenuWrap').slideDown(1000);
$('#quizGame').fadeToggle(200);

}

$("#A").click(function () {
    checkAnswer('A');
    stopAudio();
});
$("#B").click(function () {
    checkAnswer('B');
    stopAudio();
});
$("#C").click(function () {
    checkAnswer('C');
    stopAudio();
});
```