A New User Interface for Collaborative Remote Music Performance

Major Qualifying Project

Advisor:

PROFESSOR CHARLES ROBERTS

Written By:

BENJAMIN KLAIMAN

A Major Qualifying Project
WORCESTER POLYTECHNIC INSTITUTE

Submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science.

OCTOBER 2021 - MAY 2022

Abstract

In a live coding performance, (also known as an “algorave”, derived from the words “algorithm” and “rave”), performers create music and live visuals by programming them on stage, and projecting the resulting source code for audience members to follow along. This MQP seeks to improve upon the visual presentation of the algorave held in WPI’s Higgins Hall in February 2019. Specifically, this paper presents a more intuitive way to visualize algorave-generated music, while still retaining the algorithmic complexity these events are known for. Upon completion of background research and exploring audiovisual esoteric programming languages, the idea for a web-based solution emerged. This new software, designed to be run in a web browser, will allow users to collaboratively create music using a streamlined user interface and see their music visualized in real time.

TABLE OF CONTENTS

List of Figures

Executive Summary

This MQP presents a concept for software designed to be run in a web browser, where users can collaboratively create music together using a streamlined user interface and see their music visualized in real time.

In a live coding performance, (also known as an “algorave”, derived from the words “algorithm” and “rave”), performers create music and live visuals by programming them onstage in front of an audience and projecting the resulting source code for audience members to follow along. This MQP seeks to streamline these key components of algoraves and thus drastically reduce the barrier of entry for participation and understanding of what makes an algorave work.

Users participate in creating music in a group known as a Session as either a Conductor, or one of up to three Performers. This Session exists for a limited time before it ends, and the Conductor can choose whether to restart the Session with the existing Performers or disband the Session and start over.

Performers can choose from a selection of musical notes defined by the Conductor as well as a brief period of silence, known as a “rest”. Conductors can control various parameters of their Session, which in turn affects the potential output of the Performer(s) in the Session. In addition, the musical output of all Performers is visualized in a dedicated space in the center of the screen, using various shapes and colors to represent each Performer as well as each specific musical note, respectively. If a user chooses to create a Session, that user automatically becomes the Conductor of that Session. Likewise, if a user chooses to join an existing Session, that user will automatically become a Performer for that Session.

Each Session can only have one Conductor and between one and three Performers. Conductors and Performers have different user interfaces to reflect their roles in a Session. As shown above, any Performers in a Session have access to a Note Palette where Performers can input a sequence of musical notes and rests, the output of which is shown in the Note Grid, displayed from left to right, in the order the Performer selected. Performers also have access to a panel where they can control basic audio effects that are applied to the next note selected. When a Performer is satisfied with their created pattern, they can click the green Send Pattern button to send it into the wider mix, where it will be looped until a new musical pattern replaces it. The Conductor of a Session can control the tempo, or speed, of all Performers’ playback in beats per minute, or BPM. The selection of musical notes Performers can choose from are also chosen by the Conductor, in the form of a dropdown selector with different musical keys. The keys provide different Note

Palettes for all Performers in the Session. All changes made by a Conductor are applied to their Session in real time.

Multiple backend and frontend web technologies were used to create a mockup to determine project feasibility, for reasons explained in Project Constraints.

- Google Firebase is used to store and retrieve metadata for each Session created, including user-submitted screen names, musical key, tempo, as well as whether each member is either a Performer or Conductor.
- P5.js is used to create the background visualizations.
- Socket.IO is used to facilitate the exchange of musical data sent by Performers.
- Audio library Tone.js is used to generate audio for certain musical instruments where regular audio files are not used, and to programmatically adjust the pitch of a limited number of audio samples.
- The Audio Effects panel for the Performer(s) and Music Settings panel for the Conductor are created in Tweakpane.
- All other elements of the UI, including the Note Palette, Note Grid, song information displays, and buttons are created in pure HTML and CSS.

CHAPTER 1

INTRODUCTION

The Computer Science (CS) undergraduate program at Worcester Polytechnic Institute excels at providing its students with the technical experience necessary for a career in software programming and engineering. As of writing WPI also offers a major in Humanities and Arts for those who are more creatively-minded. Despite, or perhaps because of, the vast difference in teaching and learning styles needed to approach each major, students who are interested in one major for a career but learn best using the methodologies of the other generally find it slightly more difficult to progress through WPI than most. During my junior year while I looked for a Major Qualifying Project, I managed to find myself in this exact predicament, having already taken over two years' worth of CS classes and discovering that I learn best by not simply doing, but creating.

It was this desire to bridge code and art that led me to discover the work of Professor Charles Roberts. I first learned about Professor Roberts through a live coding performance, known as an algorave, held at WPI as part of a separate MQP. An algorave is a musical event where the music is entirely powered by programming, and the code that drives the music is publicly displayed for all attendees to see [1]. I went to this performance, and it was a unique experience for me, but as I left I couldn't help but think that while the showy effects of the code powering the music proved to be interesting visuals, there were relatively few indicators about what parts of the code controlled what parts of the music. Most importantly, I did not think the overlap of code and music would be as apparent to people with no coding experience, because under the assumption that in an event marketed to the general public (as I recall this event was), not all attendees are guaranteed to know how to program.

Because of this, when the time came to choose an MQP, I opted to continue the work paved by projects such as this algorave and take a different direction to better show the connection

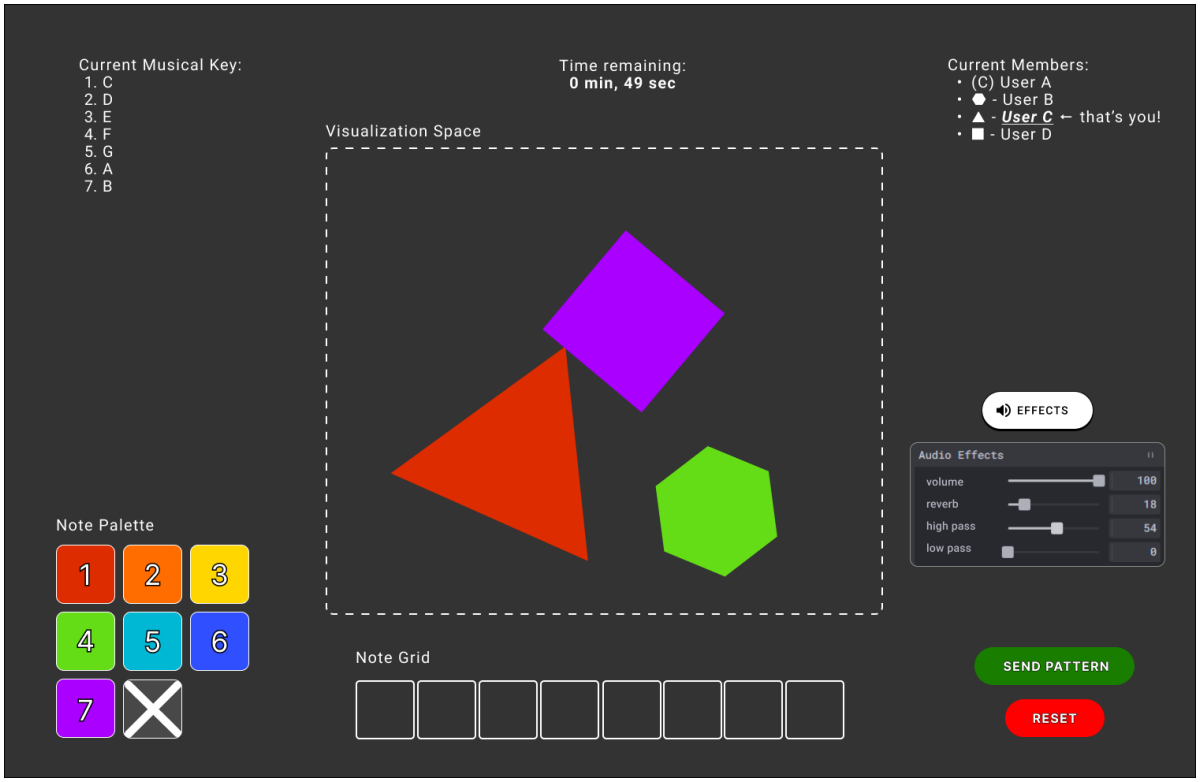

Figure 1.1: *A screenshot of one part of the end result.*

between code and music. One of my first tasks of this MQP was to tackle the question, "How does one make a performance of code, visuals and music more accessible to an audience that may not know how to code?" To me, the answer seemed direct: remove the code entirely, or more succinctly, abstract the code down to a mapping of colors and shapes, which itself serves as a useful starting point for the visualization aspect of algoraves. This effort resulted in a mockup prototype, shown in Figure 1.1, for software that allows users to collaboratively create music through a web-based application.

CHAPTER 2

BACKGROUND

Research in previous examples of networked musical performance and audiovisual programming languages was necessary in order to obtain a better understanding of the technology that powers the interactions between code, music, and visuals. Networked musical performance, or NMP, refers to the practice of people simultaneously collaborating on a singular piece of music from multiple different locations, connected by sharing audio streams with all involved parties [2]. In order to facilitate this exchange of data, there is an emphasis on incurring as little audio latency as possible, in order for all audio streams to synchronize appropriately [2]. In addition, the rise of web audio technologies such as the Web Audio and Web MIDI APIs, and other APIs such as WebSockets, have allowed for not only simple playback of audio in a web browser, but also a direct way to send musical data between web browsers and/or servers over HTTP [3]. There exists a significant body of previous work in the field of NMPs and the computer languages created to drive them, therefore for the sake of clarity, individual projects are separated into those which focus on the networking aspect of performance and those which focus on the computer language aspect of performance.

2.1 Computer Networked Systems and Performances

2.1.1 The Blind Lemon Music Gallery NMP

One of the earliest displays of networked music performance took place at the Blind Lemon Music Gallery in Berkeley, California on July 3rd, 1978. Here, three programmers created individual programs for three KIM microprocessors that interacted with each other while simultaneously mixing their own output together into audible sound for a live audience [4]. The first microprocessor, taking input from the third microprocessor, generated a musical tone from a waveform,

pitch and timbre dictated by a randomly generated string of numbers, as shown in Figure 2.1 [4]. The second processor, in contrast, used a fixed two-dimensional terrain map to generate a sequence of musical notes based on a specific location on this terrain, specified by a predetermined mathematical function as well as the velocity of movement across this terrain, as shown in Figure 2.2 [4]. Lastly, the third microprocessor utilized a 29 tone-per-octave scale, shown in Figure 2.3, to randomly generate more pleasant-sounding musical sequences based on whether two musical notes that comprise a sequence satisfy a specific set of mathematical ratios or are both related to a third tone in the sequence [4]. This produces the effect of, according to one observer, "usually [seeming] right on the edge of breaking out into a recognizable tune" [4]. As shown in Figure 2.4, to produce music, each microprocessor fed its output into both the other microprocessors as well as the mixer connected to the amplifier for the audience to hear.

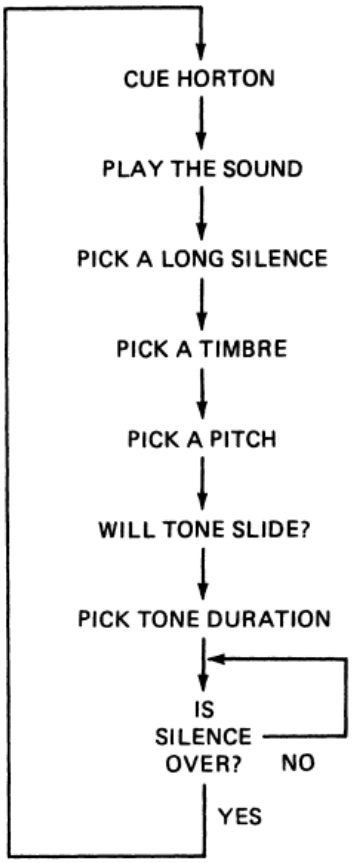

Figure 2.1: *The flowchart of the first microprocessor's program.*

2.1.2 Sensus Smart Guitar

A more interactive use case of NMPs comes from MIND Music Labs' Sensus Smart Guitar. The system Sensus employs expands upon the functionality of a playable guitar with electronic

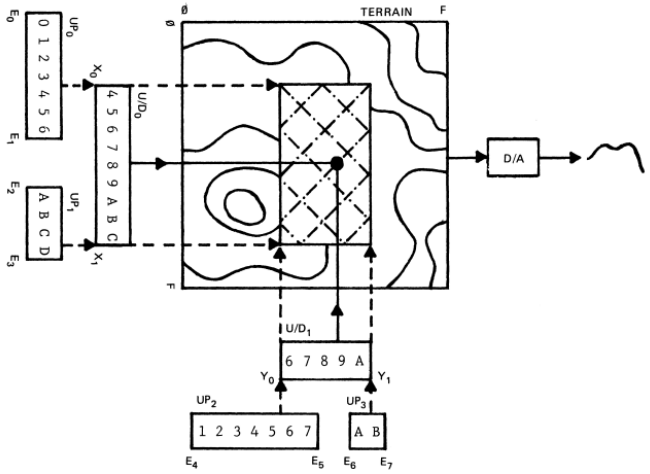

Figure 2.2: *A visualization of the second microprocessor's program.*

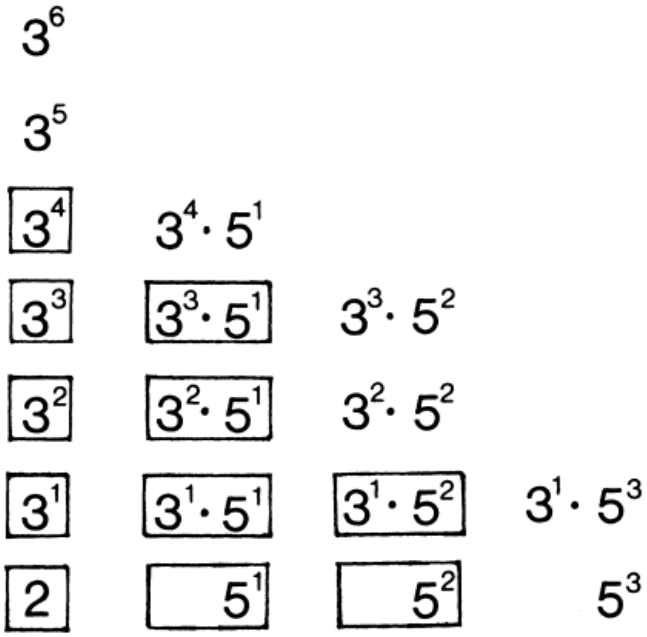

Figure 2.3: *The musical scale the third microprocessor's program uses to generate pleasant-sounding patterns.*

enhancements to enhance the playing experience, and wireless connectivity with smart devices, the cloud, and virtual reality (VR) headsets [5]. With its nine different sensors and inertial measurement unit, this "augmented instrument" is capable of tracking the guitar player's motions, such as moving the guitar up or down, or from front to back, to respond in a variety of different ways, including but not limited to, remotely controlling visualizations displayed on an external display using data gathered from the sensors, as shown in Figure 2.5 [5].

A smartphone app exists for both iOS- and Android-based devices that allows users to

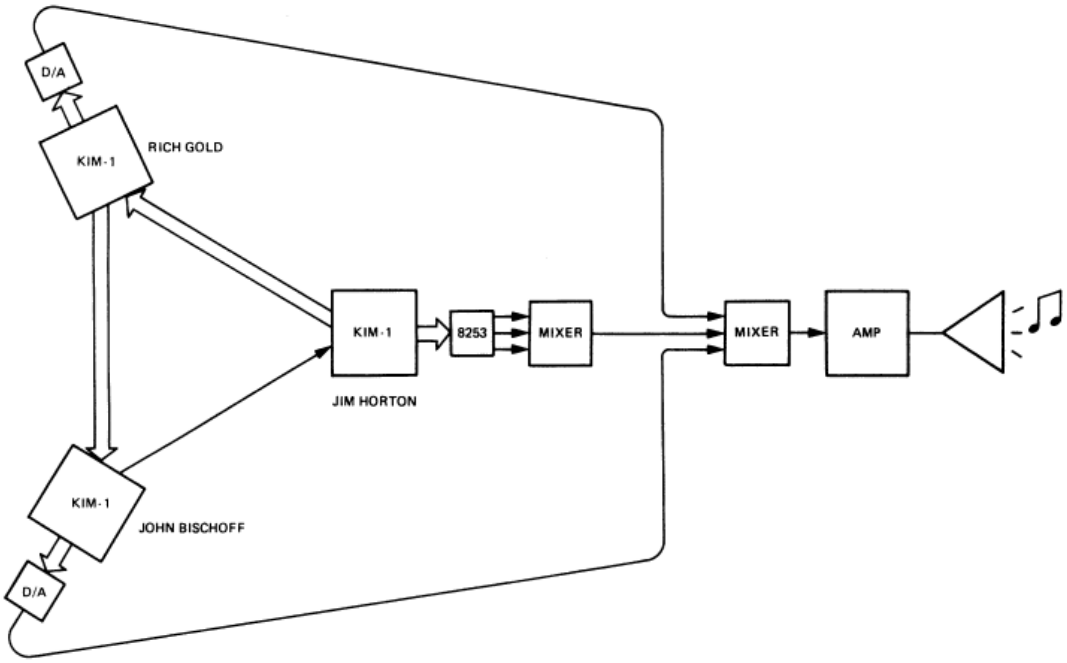

Figure 2.4: *The basic flow of information in the musical performance. Notably, all three microprocessors, labeled "KIM-1", send information between each other as well as to the final output.*

"perform" with the Sensus player by means of wirelessly transmitting MIDI or OSC data to the Sensus itself. These data are processed using an onboard digital audio workstation (DAW) and then mixed back into the sound Sensus emits, allowing the audience to, in effect, adjust the sound of the performance as it is being played [5]. A more experimental use case for this technology involves its compability with VR technology, where data gathered from the Sensus's sensors can influence not only the soundscape an end listener hears, but also an entire virtual environment to complement it, as shown in Figure 2.6 [5].

Figure 2.5: *A demonstration of Sensus in action during its first live public performance at Slush Music 2016.*

Figure 2.6: *A demonstration of the Sensus's data influencing both the audio and visuals delivered through an Oculus Rift VR headset.*

2.1.3 Public Sound Objects

Public Sound Objects (PSOs) is a musical instrument accessible via the World Wide Web from anywhere in the world [6]. PSOs is designed to provide a visual representation of music that is accessible to the general public without previous musical knowledge [6]. In this setup, multiple PSOs clients, via an interactive user interface showin in Figure 2.7, communicate with the main PSOs server through the transfer of specialized data packets in the form of a ball bouncing around a large box [6]. Each client is assigned a specific musical instrument. Each wall in the box contains an individually-controlled parameter that affects the sound produced when that specific wall is hit. In addition, the ball inside continuously moves at a constant rate and changes direction once it bounces off one of the walls [6]. The size, position, and velocity, of the ball, as well as the trail left by the ball when moving, additionally affect the resulting audio. Upon touching a wall, a musical tone is played to all clients. [6]. As shown in Figure 2.8, it is possible to showcase the output of multiple PSOs clients side by side as an art installation visualizing the process of how its music is produced [6].

2.2 Computer Languages

2.2.1 Livecodelab and Livecodelang

Livecodelang, used in its environment Livecodelab and shown in Figure 2.9, is a language focused on making both music and graphics [7]. This software, designed to run in a browser, has a very simple and quick language design conducive to live coding and effectively serves its purposes. However, the visuals and music appear too disconnected from each other. In addition, the visual

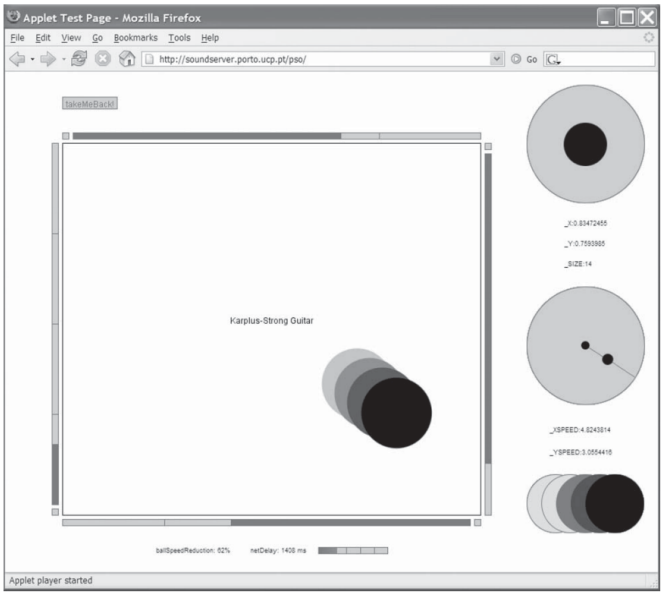

Figure 2.7: *The PSOs client controller interface.*

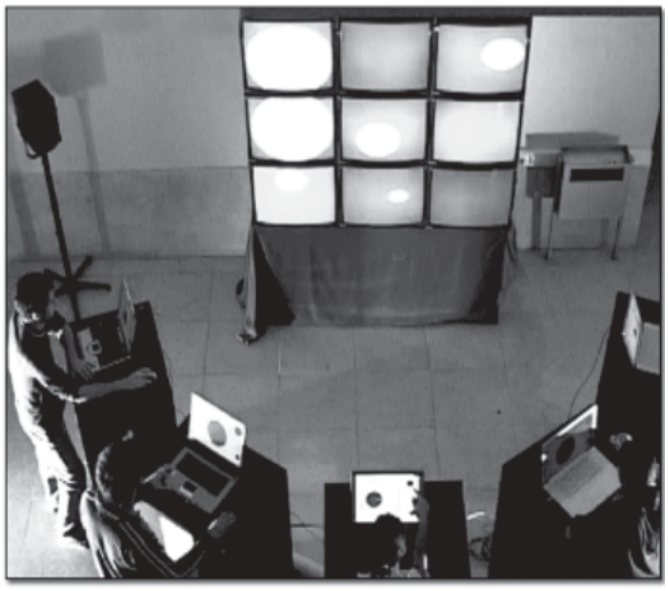

Figure 2.8: *An installation of PSOs at Porto School of the Arts in October 2014.*

component of the code is slightly lackluster; the program's design focuses more on functionality and ease of use rather than the live coding itself.

2.2.2 Gibber

Gibber, shown in Figure 2.10, is a creative coding environment for audiovisual performance and composition [8]. This environment contains features for audio synthesis and musical sequencing, 2D drawing, 3D scene construction and manipulation, and live-coded shaders. The most inter-

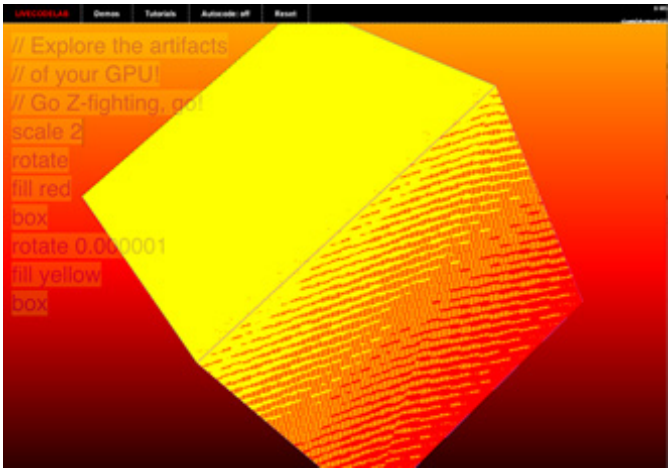

Figure 2.9: *A screenshot of Livecodelab.*

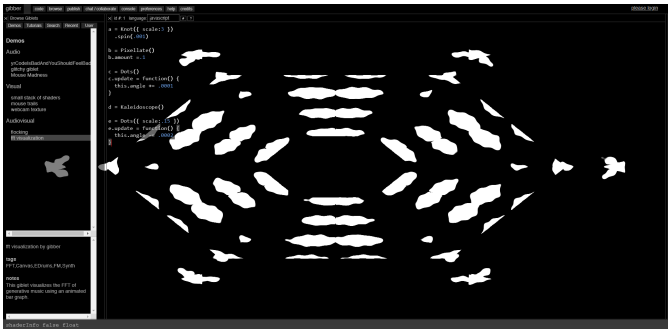

Figure 2.10: *A screenshot of one possibility for Gibber's visuals.*

esting features are the ability to connect audio with sound production, the variety of options it has while doing so, and the feature of creating a visual display for melodies. Gibber additionally provides the ability to run separate lines of code individually, showing the immediate results of the executing code, a useful feature in making algoraves more accessible to non-programmers [9]. As shown in Figure 2.11, while code is executing, various parts of the running code change appearance in relation to the state of audio playback in real time [10].

2.2.3 Orca

Orca, shown in Figure 2.12, is an esoteric programming language designed to quickly create procedural sequencers, in which every letter of the alphabet is an operation [11]. Lowercase letters operate on bangs, a type of digital signal that triggers a specific event, while uppercase letters operate each frame. Orca does not generate audio on its own, instead sending MIDI messages which can be received by other applications and devices. Its primary advantages are its simplicity in terms of how quickly a user can start making sounds but large amounts of depth that could be created with its functions. These functions range from switch registers to complex music generators where one variable change could set off a chain reaction, creating a whole new

Figure 2.11: *A screenshot of Gibber code mid-performance. Of note are the bordered boxes which show musical tones and sequences played, as well as assorted random number generator output displayed in between the dark gray /* and */ symbols.*

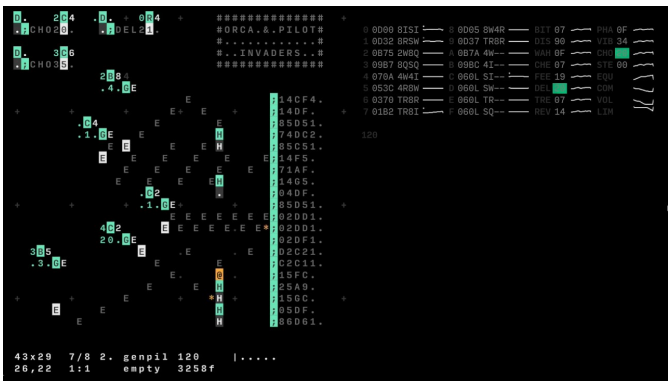

Figure 2.12: *A screenshot of Orca.*

direction in a song. Because of how simple functions are used, Orca often scales quite rapidly. This setup can either create a unique and interesting visual display or a mess of alphanumeric symbols.

2.2.4 Reactable

The Reactable, shown in Figure 2.13, is an electronic musical instrument with a tabletop tangible user interface. It is also available in app form under the name ROTOR, which has a menu of functions users can choose from rather than the block table format. This application uses its simplicity and visual display better than other previous examples. Each function when connected to the center node creates strong, direct visual feedback in that connection. Reactable also has a more tangible artistic feel, and does not require a complex method of input; users physically

Figure 2.13: *A picture of Reactable in action.*

rotate each function to change its parameters. However, Reactable lacks the esoteric appeal other languages can provide and it is also more limited in the type of music that can be created with it.

2.2.5 Tone.js

Tone.js is a JavaScript framework that allows users to create interactive music within existing web browsers [12]. This framework exposes various parts of the Web Audio API for users to control via JavaScript including musical timing, audio buses, several software-based synthesizers, and the ability for users to import their own audio samples for playback and/or manipulation by Tone.js. Musical scores, or sequences of predefined musical notes and effects represented on a linear timeline, and state information of various Tone.js components are described and modifiable with JavaScript Object Notation (JSON).

2.2.6 Soundworks

Another JavaScript web audio framework exists in the form of Soundworks. Unlike Tone.js, Soundworks follows a client/server architecture [13]. This allows the server, written in Node.js, to communicate with not only web browsers but also any device capable of running a Node.js client, such as a Raspberry Pi, for example, as shown in Figure 2.14 [13]. Soundworks has been used in live musical performances such as Garth Paine's *Future Perfect*, shown in Figure 2.16, and Jean-Luc Hervé's *Biotope*, shown in Figure 2.15 [13].

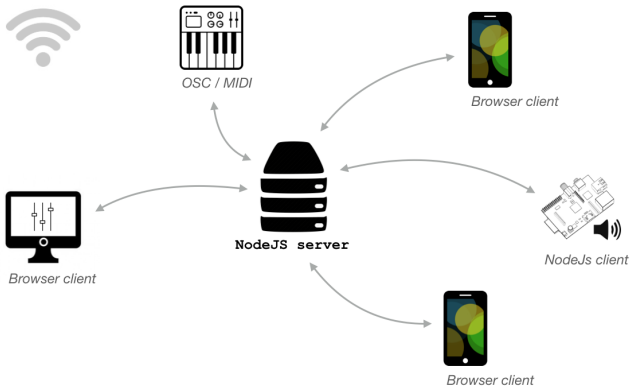

Figure 2.14: *An overview of the architecture of Soundworks.*

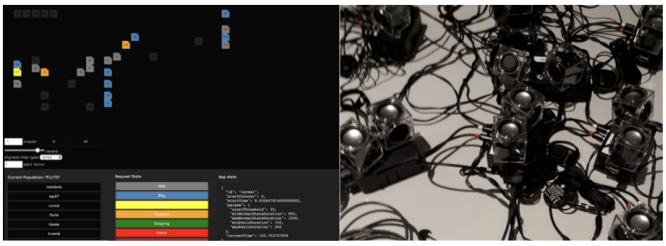

Figure 2.15: *On the left, screenshot of the centralized controller. On the right, photography of the musical agents, running on Raspberry Pi microcomputers, created for the Biotope installation composed by Jean-Luc Hervé'.*

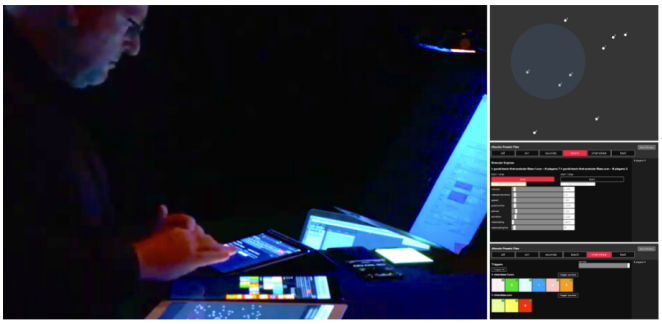

Figure 2.16: *On the left, artist Garth Paine performing Future Perfect. On the right, screenshots of the different interfaces used during composition and performance.*

Chapter 3

Implementation

In an effort to make the performance of code, visuals and music more accessible to an audience that may not know how to code, the most direct solution was remove traditional code entirely. To this end, an approach was taken to abstract the code down to a mapping of colors and shapes, in order to better communicate the interactions between "code" and music. The process of music creation is both iterative and collaborative in order to make the creation of music more accessible to end users with varying degrees of programming or musical experience [14].

3.1 Application Flow

Users participate in creating music in a group known as a Band as either a Conductor or Performer, to participate in a Session. This Session exists for a limited time before it ends, and the Conductor can choose whether to restart the Session with the existing Performers or disband the Session and start over. Performers can choose from a selection of musical notes defined by the Conductor as well as a brief period of silence, known as a "rest". Conductors can control various parameters of their Session, which in turn affects the potential musical output of any Performer(s) in the Session. In addition, the musical output of all Performers is visualized in a dedicated space in the center of the screen, using various shapes and colors to represent each Performer as well as each specific musical note, respectively. If a user chooses to create a Session, that user automatically becomes the Conductor of that Session. Likewise, if a user chooses to join an existing Session, that user will automatically become a Performer for that Session.

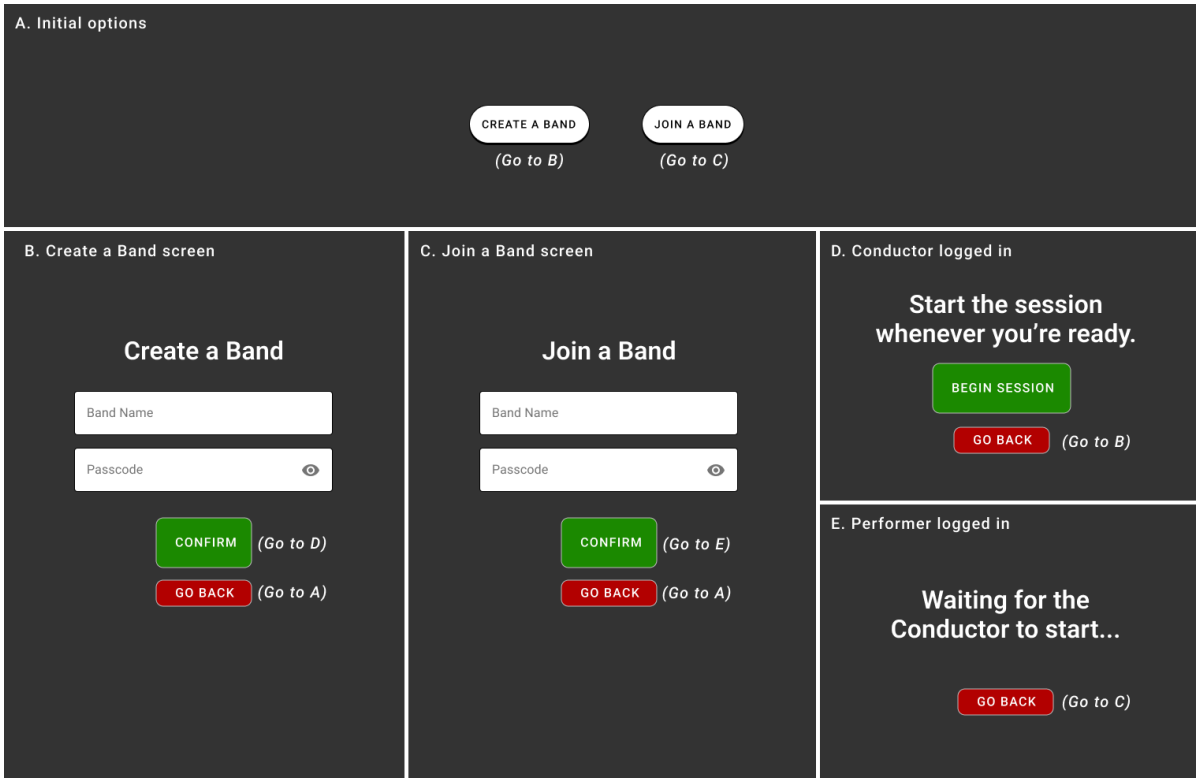

Figure 3.1: *The flow of user interaction for creating or joining a Session.*

3.1.1 Initial Screen

Figure 3.1 depicts the flow of user interactions in either creating or joining a Band. The application will present the user with two options, to either create a Band or join an existing Band (A). If the user chooses to create a Band, they will be directed to a form where the user can fill in a Band name, in addition to a passcode required for other members to join (B). On this same form an option exists to return to the previous screen should a user change their mind. The process for joining an existing Band is very similar, only this time the user must input the name of the Band they wish to join as well as its passcode (C). Upon confirmation of the Band the user wishes to either create or join, they will be presented with the next step, which varies depending on whether they are creating or joining a Band. If a Band has been created (i.e. logged in as a Conductor), the user (now the Conductor) will be provided with the ability to start a Session upon selecting the large 'Begin Session' button or return to the Create a Band screen (D). On the other hand if a Band has been joined, the user (now a Performer) instead must either wait for the Conductor of the Band to start a Session or return to the Join a Band screen (E).

3.1.1.1 User Interface Details/Design

The user interface as shown in Figure 3.1 is not representative of a singular screenshot, rather it is a collage of the different stages in the initial Session setup process. The overall design attempts

to clearly communicate what elements need to be interacted with and how. Across the overall design, text input fields consistently have distinctively square corners with gray background text indicating what information the user is expected to enter. Likewise, text buttons consistently have comparatively rounder corners, indicating that a user can click or tap them in order to produce a desired result. In particular, the Passcode text field has the traditional eye icon, indicating that it can be interacted with to reveal the plaintext of a password otherwise obscured with asterisks. This allows the user to ensure they have entered their desired password correctly. All buttons labeled 'Go Back' are colored a distinct shade of red in order to contrast with their counterpart buttons that allow progression to their respective next phase of the application. These buttons, in contrast, are colored green and significantly larger than the 'Go Back' buttons in order to appear equally contrasting to all users regardless of colorblindness. At the end of each branch of the user flow, bold white text is presented against a dark background to concisely convey what the user is expected to do in order to advance to the next stage of the application, whether that is starting the Session (if the user is a Conductor) or waiting for a Conductor to start the Session (if the user is a Performer).

3.2 Performers and Conductors

Each Session can only have one Conductor and between one and three Performers. Conductors and Performers have different user interfaces to reflect their roles in a Session. As shown in Figure 3.2, any Performers in a Session have access to a Note Palette where Performers can input a sequence of musical notes and rests, the output of which is shown in the Note Grid, displayed from left to right, in the order the Performer selected.

Performers also have access to a panel where they can control basic audio effects that are applied to the next note selected. When a Performer is satisfied with their created pattern, they can click the green Send Pattern button to send it into the wider mix, where it will be looped until a new musical pattern replaces it. Performers are randomly assigned one of three predefined musical instruments, two of which are software synthesizers native to Tone.js, the sound library used, and one of which is a standard piano sampled from a digital audio workstation (DAW). The Conductor of a Session can control the tempo, or speed, of all Performers' playback in beats per minute, or BPM. The selection of musical notes Performers can choose from are also chosen by the Conductor, in the form of a dropdown selector with different musical keys, which provide different Note Palettes for all Performers in the Session. All changes made by a Conductor are applied to their Session in real time.

3.2.1 Performers' User Interface

The user interface for Performers in a Session, as shown in Figure 3.2, consists of multiple elements surrounding the large visualization space in the center of the screen. The top left corner

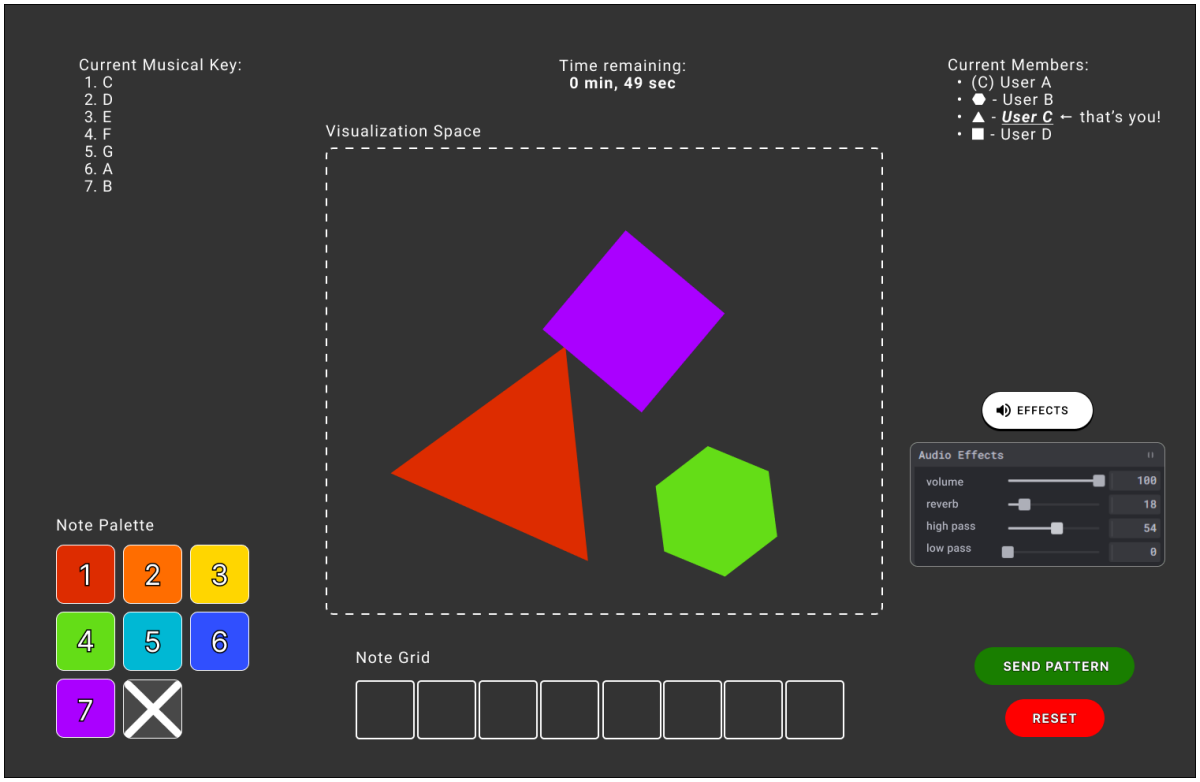

Figure 3.2: *The user interface for a Performer.*

of the screen lists the specific selection of musical notes that all Performers in the Session can choose from in the Note Palette, as well as which musical notes correspond to which number. The top center of the screen displays the remaining time for the current Session before it ends and the Conductor has the option to restart the Session or not. The top right of the screen lists all participants in the session as they entered their username in the setup phase of the application. The bottom right corner of the screen contains the Note Palette, which allows users to input a sequence of musical notes and rest periods much like they would enter a code on a numerical keypad. The user's input is then updated on the Note Grid to the left of the Note Palette in the form of a linear sequence displayed from left to right. To the right of the Note Grid, located in the bottom right corner, there are two options for the user. They can send the musical pattern displayed in the Note Grid into the wider musical mix, emptying the user's current Note Grid, where it will be looped until a new pattern is sent to replace it. Or, they can just reset the Note Grid to an empty state in case a user makes a mistake or simply does not like their current pattern. Additionally, above these two buttons there is a panel, which can be toggled with the "Effects" button, that allows the user to choose from a set of effects which are applied to the next musical note the user selects. The user can choose to adjust the volume, reverb, and high and low passes of a certain musical note. All of these are adjustable from a scale of 0 to 100, where 0 indicates the lowest possible value and 100 indicates the highest. For example, a volume level

of 0 will mean the note is not audible, while a volume level of 100 indicates the loudest volume possible.

3.2.1.1 User Interface Details/Design

For the Current Musical Key display in the upper left corner, the available notes are listed in an ordered list from one to seven, directly mapping the musical notes listed to the individual buttons in the Note Palette below it. Where the time remaining in a Session is displayed, the time itself has been bolded for emphasis as it slowly ticks down. In the list of current members of a Session, the text representing the current user is underlined and bolded, as well as the text "that's you!" beside it to emphasize that this refers to the current user, regardless whether they are the Conductor or a Performer. In front of each username is a symbol representing their role in the Session. If the user is the Conductor, a letter 'C' wrapped in parentheses will be displayed in front of their name. If the user is a Performer, a filled-in polygon will display in front of their name, with the number of sides corresponding to which instrument they have been randomly asssigned as well as what shape they are represented with in the visualization space. The Note Palette consists of eight square-shaped buttons that correspond to the seven musical notes available to a given Performer, and one button with a white cross symbol indicating a period of silence where no musical notes are played, The bottom-right corner of the Palette is empty in order to retain an overall square shape so the other buttons line up with each other horizontally and vertically. The individual buttons of the Note Palette have rounded corners to indicate they can be clicked or tapped to interact with them. These buttons each have one of seven different colors to differentiate between musical notes, which have been selected using Google's Material Design Color Tool in order to ensure maximum clarity. These colors are, in order:

- #DD2C00 (red)
- #FF6D00 (orange)
- #FFD600 (yellow)
- #64DD17 (light green)
- #00B8D4 (light blue)
- #304FFE (dark blue)
- #AA00FF (purple)

To account for colorblindness, the Note Palette buttons also have a large number with a bolded outline in their centers that numerically indicate which musical note out of all available musical notes a particular button corresponds to. The Note Grid, to the right of the Note Palette, consists of eight horizontally-arranged squares with less rounded corners, indicating that they

cannot be interacted with directly, but are still functional nonetheless. As the user selects buttons in the Note Palette, the leftmost empty square will be filled with the symbol corresponding to the Note Palette button pressed. If a button representing either of the seven available musical notes is pressed, the specific square in the Note Grid will be filled with a circle whose color represents the specific musical note out of the seven available that the user selected. The size of said circle is determined by the volume parameter at the time the Note Palette button is pressed; the lower the volume, the smaller the circle, and vice-versa. If the button representing a period of silence (the white 'X') is pressed, the square in the Note Grid will instead be filled with the same white 'X' symbol in the Note Palette button. The three buttons labeled "Send Pattern", "Reset", and "Effects" have rounded corners, indicating they can be pressed and that pressing them will result in the indicated action taking place. The "Send Pattern" and "Reset" buttons, while being colored contrasting green and red respectively, have widths corresponding to the length of the text inside in order to make them distinguishable separately from their colors. In addition, the "Effects" button has an icon of a speaker inside it, indicating that this button controls whether the audio effects panel is visible or not. The effects panel itself is the standard Tweakpane UI, in order to appear more user-friendly than similar, more programmer-oriented UI configuration libraries such as Dat.gui.

3.2.2 Conductor's User Interface

The user interface for the Conductor of a Session, as shown in Figure 3.3, is a variation of the Performer's user interface. The current musical key, time remaining, and current members list as well as the Visualization Space in the center remain unchanged from the Performer's user interface. The sole difference is in the bottom-left corner of the screen, which contains the Music Settings, a panel which can be toggled by interacting with the "Settings" button with a musical note, similarly to the "Effects" button that a Performer would otherwise see. This panel contains the ability to control the tempo, or speed, of the music being played back in beats per minute (BPM) by means of a slider, which can be set to any value between 60 and 300 BPM, inclusive. In addition, there exists an option to change the selection of musical notes the performers can choose from by means of a dropdown list containing multiple predefined musical keys, all of which correspond to a specific combination of musical notes in order to keep the resulting music in tune.

3.2.2.1 User Interface Details/Design

The design of the Conductor's user interface, as shown in Figure 3.3, is nearly identical to that of the Performers' user interface, with the exception of the Music Settings button. This button has a musical note within it to indicate that pressing it will reveal options for directly controlling the music.

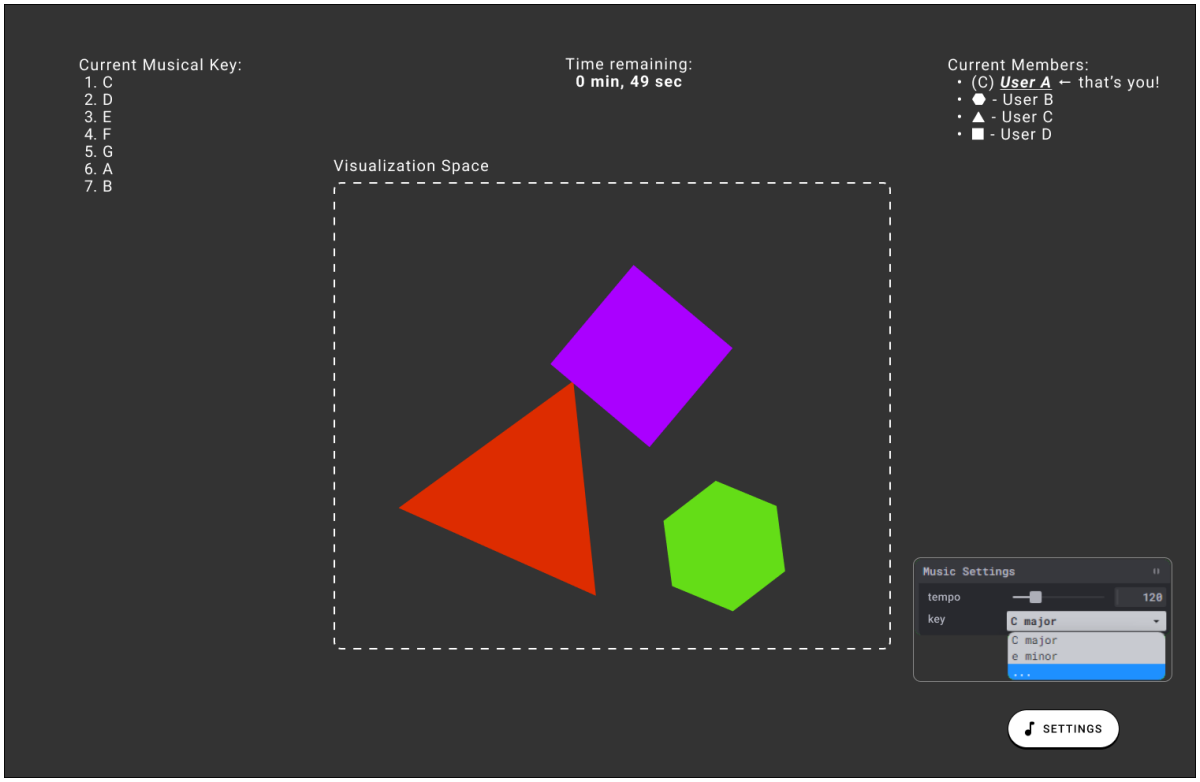

Figure 3.3: *The user interface for the Conductor.*

3.2.3 Visualization Space

The Visualization Space is a large area in the center of the user interface visible to all participants in the Session. Whenever a note is played back, a regular polygon with a predetermined size, shape, and color appears in this space before shrinking and disappearing to make room for more shapes. The individual shapes themselves, a regular triangle, square, and hexagon, are randomly assigned to an individual user to unique identify them in the Visualization Space. The color directly corresponds to the specific musical note being played back, and the initial size directly corresponds to the volume of a specific musical note being played at that particular moment in time. Where the shape appears in the Visualization Space is randomly chosen from any possible coordinate in the space. This serves no technical purpose, only to intentionally add some variety to the area.

3.3 Technology Stack

Multiple backend and frontend web technologies have been employed in creating a mockup to determine project feasibility, for reasons explained in Chapter 4. Google Firebase is used to store and retrieve metadata for each Session created, including user-submitted screen names, musical key, tempo, as well as whether each member is either a Performer or Conductor.

Creative coding library P5.js, itself more accessible to artists and creative thinkers by design [15], is used to create the background visualizations. An external JavaScript library was used as opposed to HTML Canvas with the desire to expedite the creative process, particularly when it comes to animating and rendering complex shapes. Having prior experience with its older relative Processing, aside from a few minor alterations I could switch from Processing's Java-based environment to the JavaScript-based environment of p5.js with relative ease. Another factor that led to p5.js becoming the framework of choice is its extensive documentation and sample code. The sample code provided proved so useful that my code used to render regular polygons in the final prototype started with p5's own sample code, with my own additions to fit the project on top of it. The Audio Effects panel for the Performer(s) and Music Settings panel for the Conductor are created in Tweakpane. Dat.gui was originally used for this purpose since I had experience with this framework in a previous class, but in the middle of development the decision was made to switch to Tweakpane because it appears significantly more user-friendly than Dat.gui's minimalist, programmer-oriented user interface and experience. All other elements of the UI, including the Note Palette, Note Grid, song information displays, and buttons and created in HTML and CSS, purely out of a desire to experiment.

Audio library Tone.js is used to generate audio for certain musical instruments where regular audio files are not used, and to programmatically adjust the pitch of a limited number of audio samples. The idea to use Tone.js came from Professor Charles Roberts who, having prior experience with web-based creative coding environments such as Gibber, recommended it to me for its extensive capabilities compared to its ease of use. All three musical instruments used to produce the sounds are created or altered by Tone.js in some way. The two software-based synthesizers come directly from Tone.js. For the piano, I recorded three samples in one octave of FL Studio Mobile's standard grand piano, and had Tone.js pitch-correct each sample to "fill in" any of the nine remaining missing musical notes required, saving valuable storage space.

Firebase is used to store metadata for each Session: the Band name, the members of each Session, who is assigned which instrument, and the Session length. This information is accessed and updated with a custom backend API written in JavaScript. The Pusher library is used to facilitate the exchange of musical data sent by Performers.

CHAPTER 4

CONCLUSION AND FUTURE WORK

When I showcased my poster to the wider WPI community containing the UI prototypes, the reception was largely positive, with particular praise for how the conceptual use of colors and regular polygons to individually represent musical notes, as well as animation to clearly show when a particular instrument is being played back, a major part of my original vision for the project. The use of assigning colors to numbers to account for potential color-blind users also received positive attention. Any constructive criticism primarily focused on the presentation remaining in prototype form. Multiple factors have led to demonstrating this MQP as a series of UI prototypes rather than a complete codebase as a hopeful compromise. An early contributor to the project tasked with backend coding pulled out of the project shortly before development began. Another contributor worked on the backend for one academic term, but has ceased primary development after the end of that term. In addition, due to my primary development work starting alongside the COVID-19 pandemic, as well as my relative inexperience with coding in JavaScript, solo development has been severely hindered. Code has been written that demonstrates much of the functionality depicted in these UI prototypes, but due to these reasons could not make it into the final MQP.

That being said, several opportunities remain to expand upon this project in the future. As the project exists now, there is no framework for any sort of percussion or drum part alongside the main Performers. One possible implementation of this feature would be to leave the control over the percussion to the Conductor of a Session. Alongside controlling the musical key and tempo, the Conductor could also choose from a variety of predesigned drum patterns in the form of a dropdown menu that plays at the assigned tempo, which is also designed to loop seamlessly with the other Performers in a Session. In addition, another possibility is to allow Performers to choose from a wider selection of musical instruments than the standard three instruments

provided. This could be achieved through a dropdown menu presented to Performers while they are preparing to join a Session.

ACKNOWLEDGEMENTS

The author would like to thank David Martindale for helping design the initial direction of the project and is credited as a co-writer on the proposal paper. The author would also like to thank Kaamil Lokhandwala for writing the backend code using Google Firebase as well as an API for interacting with it.

BIBLIOGRAPHY

[1] N. Collins and A. McLean, "Algorave: Live performance of algorithmic electronic dance music," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. London, United Kingdom: Goldsmiths, University of London, Jun. 2014, pp. 355–358. [Online]. Available: http://www.nime.org/proceedings/2014/nime2014_426.pdf

[2] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, "An overview on networked music performance technologies," *IEEE Access*, vol. 4, pp. 8823–8843, 2016.

[3] L. Wyse and S. Subramanian, "The viability of the web browser as a computer music platform," *Computer Music Journal*, vol. 37, no. 4, pp. 10–23, 2013.

[4] J. Bischoff, R. Gold, and J. Horton, "Music for an interactive network of microcomputers," *Computer Music Journal*, vol. 2, no. 3, pp. 24–29, 1978.

[5] L. Turchet, M. Benincaso, and C. Fischione, "Examples of use cases with smart instruments," in *Proceedings of the 12th International Audio Mostly Conference on Augmented and Participatory Sound and Music Experiences*, ser. AM '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3123514.3123553

[6] A. Barbosa, "Public sound objects: a shared environment for networked music practice on the web," pp. 233–242, 2005.

[7] D. D. Casa and G. John, "Livecodelab 2.0 and its language livecodelang," in *FARM '14*, 2014.

[8] C. Roberts, M. Wright, J. Kuchera-Morin, and T. Höllerer, "Gibber: Abstractions for creative multimedia programming," in *Proceedings of the 22nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 67–76. [Online]. Available: https://doi.org/10.1145/2647868.2654949

[9] C. Roberts, M. Wright, and J. Kuchera-Morin, "Beyond editing: Extended interaction with textual code fragments," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, E. Berdahl and J. Allison, Eds. Baton Rouge, Louisiana, USA: Louisiana State University, May 2015, pp. 126–131. [Online]. Available: http://www.nime.org/proceedings/2015/nime2015_310.pdf

[10] C. Roberts, “Code as information and code as spectacle,” *International Journal of Performance Arts and Digital Media*, vol. 12, no. 2, pp. 201–206, 2016. [Online]. Available: https://doi.org/10.1080/14794713.2016.1227602

[11] M. Messina, L. Aliel, C. M. Gómez Mejía, M. C. Filho, and M. T. Soares de Melo, “Live Coding on Orca, the Geopolitics of the English Language and the Limits of Creative Semantic Anchoring: A Preliminary Hypothesis,” in *Proceedings of the 11th Workshop on Ubiquitous Music (UbiMus 2021)*, ser. Proceedings of the 11th Workshop on Ubiquitous Music (UbiMus 2021). Matosinhos, Portugal: g-ubimus, Sep. 2021, pp. 57–61. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03368436

[12] Y. Mann, “Interactive music with tone.js,” in *Proceedings of the International Web Audio Conference*, ser. WAC ’15, S. Goldszmidt, N. Schnell, V. Saiz, and B. Matuszewski, Eds. Paris, France: IRCAM, January 2015.

[13] B. Matuszewski, “Soundworks - A Framework for Networked Music Systems on the Web - State of Affairs and New Developments,” in *Proceedings of the Web Audio Conference (WAC) 2019*, Trondheim, Norway, Dec. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02387783

[14] E. Dr. Manara Miletto, M. Soares Pimenta, F. Bouchet, J.-P. Sansonnet, and D. Keller, “Principles for music creation by novices in networked music environments,” *Journal of New Music Research*, vol. 40, no. 3, pp. 205–216, 2011. [Online]. Available: https://doi.org/10.1080/09298215.2011.603832

[15] B. Gross, H. Bohnacker, J. Laub, and C. Lazzeroni, *Generative Design: Visualize, Program, and Create with JavaScript in p5.js*. Princeton Architectural Press, 2018.