Audio Synthesis in Juce™ with C++

Kealan O'Callaghan

B.Sc.(Hons) in Software Development

April 23, 2022

Final Year Project

Advised by: Gerard Harrison

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract This project is an Audio Synthesizer with C++ using the JUCE framework. The output of this project will be a VST3 (.dll) file, also known as a 'plugin' which will be executable in any digital audio station, such as FL Studio or Ableton Live. The programme will have midi input and output capability meaning that the user will have the ability to play their own notes and melodies using whatever sound they create with the plugin. The synth will be conducted of 2 oscillators with the option of each being a Saw, Sine or Square wave. It will have have FM frequency and depth knobs in the oscillator. There will also be an Amp envelope, mod envelope and a filter module all with amendable values to tweak and shape the sound. I will be exploring in great depth the different corners of audio synthesis and engineering in relation to software to gain a greater understanding of the technology behind it and to create a useful dynamic instrument. The package which will contain the Synth vst will be hosted on an informative website that will be hosted on a server which will allow anybody to visit the website, learn about the Synth, what differentiates it from other Vst's and will guide them through installation of the package so they can try it first hand.

Authors Kealan O'Callaghan

Chapter 1

Introduction

1.1 Background Context

This project was undertaken with the aim of building a multi-functional, single oscillator 'soft' synthesizer plugin capable of being ran on all digital audio workstations (DAW) using the JUCE framework with C++.

Bringing music and technology together garnered much interest in the last century across the world. It takes the form of interactive experiences, light- and music shows, or creative software. Ever since technology is accessible at our fingertips through computers and mobile phones, research, innovation and new material in the field has grown exponentially. With the advent of powerful and cost-effective processors and Graphics Processing Units (GPU) and availability of enormous amounts of data, people can create music that would have, in previous years taken thousands of euros worth of equipment to make. Music itself has taken a massive shift in the last few decades into the digital world. What once would have taken a master composer or sound engineer weeks to create, can now be created and modified by an interested teenager in their bedroom on a PC within the space of a few hours. In the field of electronic music, the large companies are constantly trying to come up with new and innovative ways to make music production more simplified to the average user while also trying to improve existing technologies for the more adept to create new and exciting music with software - but the core concepts behind all of these pieces of music software are the same, just implemented differently to provide musicians with new forms of workflow in the hope of creating something that will encourage musicians across all genres.

In this project, the Synthesizer will contain all basic elements that are core to making versatile and dynamic sounds.

In a typical Synthesizer, there is a few core features that can be found across almost all implementations which we will go through in further depth to get a better understanding of how the Synthesizer actually works using code:

- **Oscillators:** Oscillators are the part of the Synthesizer which creates sound. They can be thought of as a throat, it produces an electronic signal which creates a vibration that oscillates over and back just like vocal chords, which is the basis of the sound. The oscillator can have various shaped waveforms which all produce their very own distinctive sound.

- **Filters:** A filter is set to literally filter out certain frequencies of sound. For instance, a low-pass filter allows low frequencies to pass through while blocking higher frequencies. A high-pass filter does the opposite. Filters can be set for anywhere along the audio spectrum, blocking one set of frequencies while allowing others to pass through. In many ways, a filter is just an extreme variant of an equalizer (EQ) effect.

- **LFO:** A low-frequency oscillator (LFO) generates electronic signals on the low end of human hearing, typically around 20 Hz. These devices are useful for producing rhythmic effects like tremolo, which rapidly varies the volume of a sound wave.

- **ADSR:** Volume can also be manipulated by envelope filters, which control the attack, decay, sustain, and release of an audio signal.
  Attack: The attack is the amount of time it takes for an audio signal to reach peak volume. **Decay:** Decay is the amount of time it takes for the signal to go from its peak volume to a lower level known as its sustain volume. **Sustain:** Sustain is the volume level the sound remains at until the key is released. **Release:** The release is the amount of time it takes for the sound to go from its sustain volume to absolute silence.

In today's popular music, synthesizers dominate the instrumental palette. Thanks to versatile keyboard synthesizers from brands such as Roland, Korg, Yamaha and others, electronic musical instruments are often found on stage at major concerts, as they are able to produce sounds that were once reserved for traditional instruments such as guitar and drums.

At the same time, a large number of software libraries from companies such as Ableton, Native Instruments, EastWest, Fruity Loops, Logic, etc.

have turned personal computers into synthesizers for both the studio and the stage.

Genres with particularly heavy synths include Top 40, house, club, EDM, hip hop, and trance. In addition, genres that are usually associated with traditional musical instruments, such as rock, country, and RB, now also use synthesizers extensively. Even classical and jazz artists have entered the synthesizer game, although these genres are still dominated by acoustic instruments.

Whether it is an 88-key chord synthesizer, a 61-key monophonic synthesizer, or a computer equipped with a soft synthesizer, new musical instruments continue to emerge to further integrate electronic sounds into the public consciousness. The power of synthetic audio has helped define the music of the 21st century, and it remains a key element in countless genres.

1.2 Objectives

The aim of this project is to create a piece of software which covers all of the basic elements of a 'soft' synth and allow the user to be able to intuitively manipulate the sound to their preference while also learning how waveforms and digital audio works underneath the front end. The user will be able to make sounds which span across various genres.

The objectives of this project can be summarised as follows:

- Build a plugin which can be ran on any digital audio workstation as a VST3 file.
- Have a program with midi in and out capabilities.
- Utilise the JUCE framework to maximum effect to create a powerful synth.
- Create a synth with various filters and envelope modulation capabilities.
- Test the application to gain a better understanding of what users would want from the application and to assess how well the developed features work.
- Gain user feedback that could contribute to future development for the application.

- Learn how to program with C++ and work within a comprehensive development environment.
- Create a website to give info on the program and host it, giving a visitor the option to download and test the plugin themselves.
- Create the host website with Rest/JS.
- Become adept at programming by one's self and learn new skills for future projects.

These objectives provided a wide scope of activities from JUCE audio software development to working with RESTJS and web development. This type of variety in a project would allow for the developer to improve their skills across a wider number of areas within development rather than just focusing on one aspect. This meant that the project also provided a beneficial experience for the developer.

The following components of the project were considered to be key milestones:

- Allow the plugin to be ran across all platforms and OS's.
- Create a versatile and intuitive synthesizer.
- Host the software on a website with info on the program and audio synthesis.

1.3 Achievements

This section details achievements and benefits of the project.

1.3.1 Tangible Achievements

This project was a success in terms of addressing the objectives that were established. A soft Synthesizer was built and was capable of performing all of the music creation that was envisioned in the planning of the software. The application was also hosted on a website (kealanssynth.com). This allows anyone to visit the website from a browser, read about the program and audio synthesis as well as download the plugin provided they have a DAW to run it in.

1.3.2 Experience in a Development Environment

The project succeeded in delivering a real development experience to the developer. Prior to commencement of this project the developer for this project had limited experience of applying their knowledge of C++ programming to any noteworthy or tangible applications and had never been educated in any audio software modules. This project not only provided the developer with an opportunity to develop with a new language, but to develop an application that utilises several different technologies including C++, Javascript, RESTJS, JUCE, Visual Studio and Ableton. This is an achievement that could only have been accomplished within a complete development environment. Educational simulations of development do not compare to the real issues that are present in both the setting up and use of a multi-dimensional programming landscape.

1.3.3 Development process

This project presented an opportunity to gain insight into each stage of the development process. Design was an ongoing process throughout development, having multiple components that needed to work together allowed the developer to improve their design skills as things needed to be developed with future work in mind. Implementation of the work enabled the developer to work on problem solving skills as well as gain hands on experience of new technologies.

1.4 Overview of Dissertation

This dissertation covers the following in each chapter:

Chapter 1 - Introduction: This chapter provides some background as to why this project was undertaken and the relevance as to why this project needed to be done. The projects objectives are outlined as well as the perceived achievements.

Chapter 2 - Methodology: This chapter details the processes and the rationale behind the development of the JUCE plugin.

Chapter 3 - Technology Review: This chapter takes a look at the technology used in the full process of the project from start to finish.

Chapter 4 - System Design: This chapter provides a detailed explanation of the overall system architecture and how the project functions as it does.

Chapter 5 - System Evaluation: This chapter evaluates the project against the objectives stated earlier in the introduction.

Chapter 6 - Conclusion: This chapter summarises the achievements of the project and evaluates the outcomes of the development. This chapter also highlights future work for the project that didn't fall within the scope of this project.

Chapter 2

Methodology

This chapter aims to provide some context as to how development of the application was conducted. The processes that were followed and the general principles that enabled progressive development will be detailed here.

As a lot of learning was being done at the same time as development there were many surprises along the way and many implementations did not work as intended. Therefore as can be seen in Figure 1 the software development cycle consisted of gaining requirements, designing a solution, implementing the solution and testing it. This was an iterative cycle that was repeated incrementally for each requirement in order to build a robust solution.

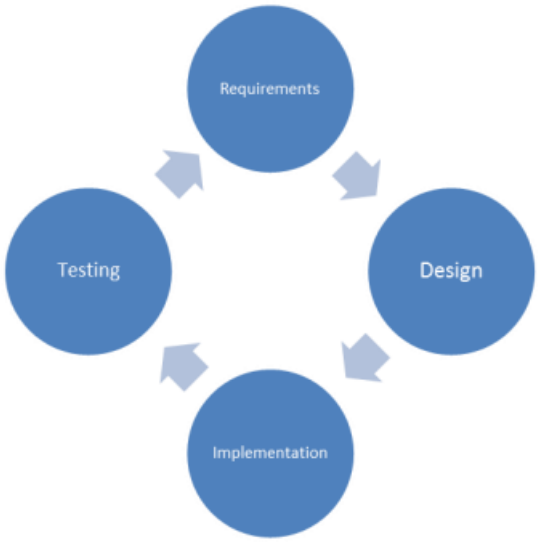

Fig. 1. Development cycle throughout the development phase

This development style created a set of dynamic sub-requirements that were constantly changing due to priority and the manifestation of novel ideas. Having to constantly evaluate the necessity of certain sub-requirements and features within the application kept development interesting and avoided getting hung-up on any one particular element.

When planning for the project and the different elements it would include, extensive planning and research was done in order to decide which features would be most vital for a synthesizer with the primary focus of simplifying the core features to a user without much background in audio synthesis and music. An incremental and iterative hybrid development style was used for developing the application. The main focus of the project was to get four main components of the synthesizer working together in conjunction with each other (oscillator, filter, ADSR modulation) then to add features piece by piece to provide a well-rounded prototype of a soft synthesizer. This development style led to multiple proto-type applications that would perform a dedicated function. The implementation of these proto-type applications were then brought together to build a functional synthesizer plugin.

The website was then developed over the course of approximately 2 weeks. This section of the project fell very much into the camp of agile development that focused on the iterative nature of software development. Many of the features included in the final website were not requirements initially, they were features that were added on after realizing that a website to download and install the software would be optimal for distribution and demonstration of the project rather than just a Github repository.

In terms of testing, the process was slightly cumbersome. Every time I though i had a piece of the functionality completed, a full build of the project was necessary which provided me with a file to be imported into my DAW (Ableton). This meant that the testing approach was very much black box, as I was testing the program as an end user and not through the code itself.

Github was also used extensively throughout the development of this program. As the size of the file to be pushed up after every development session was so large, Github large file storage had to be used in order to store all of the files in the repository, which added more complications to the development process.

Extensive research had to be done before the commencement of this project. Research had to be done in relation to the JUCE framework, the history of

audio synthesis, DSP (Digital signal processing) as well as all of the other frameworks which could have been used to create the audio plugin. After extensive research, the decision was made that JUCE would be the best fit to the need of the project as it is the most 'bare bones' of all the audio plugin frameworks with the least amount of boiler plate code.

Chapter 3

Technology Review

This chapter details information regarding the existing JUCE plugin and how it operates, as well as existing solutions for soft Synthesizers and how they work in a landscape of what already exists. Understanding how it fits together will make it easier to see where the need for this synthesizer plugin comes from. a

3.1 Background

The Synthesizer in discussion is a single oscillator instrument capable of being played and ran in any digital audio workstation [1]. The benefit of this synthesizer is the effectiveness of its components when compared to other synths that are made by the leading audio companies.

Soft synthesizers play a huge role in the production of all music produced in the last 20 years. However, their usability and robustness remains ignored in literature. Soft synths tend to have extremely complex user interfaces (UI's), and generally warp the usability of more mainstream software. Instead, their UI's generally try to replicate those of physical synthesizers. As a result, soft synthesizers can be difficult for new users to get a grasp on as they may have never had the opportunity to interact with a physical synthesizer [2].

> "Early computers could not do real-time performance – they took many seconds to compute a single second of sound. But today, real-time performance is possible; and practical real-time programming is a big part of this book. Thus, laptops can join with chamber groups and orchestras and thereby add rich new timbres to the already beautiful timbres of acoustic instruments.
>
> What now is the musical challenge of the future? I believe it is the limits in our understanding of the human brain; and specifically knowing what sound waves, sound patterns, timbres and sequences that humans recognize as beautiful and meaningful music – and why!" Max Matthews, 2006 [3]

3.2 VST

This synthesizer is a VST3 plugin, which is the latest installment of audio plugin technology. The VST standard was coined and developed by Steinberg in 1996, which revolutionized music production. VST3 is the name given to the dev kit that most developers utilize when creating audio plugins, although there are other options. VST3 is supported by almost all DAW's across all operating systems, which made it the obvious choice for this Synthesizer.

The VST3 adds several useful new features compared to VST2. Although the VST3 standard has been around since 2008, it is only now starting to overthrow VST2 as the most used kit for creating audio plugins [4]. VST3 brought many welcomed features, such as:

- **More efficient processing:** VST3 programs are designed that they are only active when an audio signal is present. This means that CPU resources cannot be wasted when there is no signal present. In VST2 programs, the processor would stay active regardless of an audio signal being present or not, which caused a lot of unnecessary processing power being used. This makes VST3 much more efficient and increases the number of plugins you can have on one channel without overloading your DAW or computer.

- **Enhanced MIDI handling:** VST3 plugins can now provide a dedicated event handler bus, which allows for a wide variety of control and modulation messages beyond traditional simple MIDI messages. In fact, support isn't only limited to the MIDI protocol, and other future control methods may utilize these functions. Advanced control of MIDI at a note level is now supported. For example, a particular event like a pitch bend can be associated with a specific note with a unique note ID, so that the modulation is applied to only that note, even in a polyphonic context like playing a chord. This is a huge improvement from the traditional midi capabilities of VST2 and allows for much more in depth customization of sounds [5].

- **Support for multiple MIDI I/O:** In VST2, one plugin could only be assigned to a single MIDI IO. With VST3, plugins support unlimited MIDI IO's which can be switched and modulated on the fly. This opens up a lot of possibilities while performing music live and allows for more flexible routing [5].

- **Resizable GUI:** Although a small change, the ability to change the size of a plugin window is extremely useful when working in a limited space DAW and was a much welcomed addition to VST3.

3.3 The JUCE framework

3.3.1 The framework

This Synthesizer was created using the JUCE framework. JUCE is a partially open-source cross-platform C++ application framework, used for the development of desktop and mobile applications. JUCE is used in particular for its GUI and plug-ins libraries. Like many other frameworks, JUCE provides all of the functions that a developer would need in order to create an application, specifically, audio processing and creation plugins. JUCE is very powerful

in the sense that it is capable of manipulating a large list of user-interface elements like graphics, audio, XML and JSON parsing, networking, cryptography, multi-threading, an integrated interpreter that mimics ECMAScript's syntax, and various other commonly used features.

The most notable feature of the JUCE framework is its large set of audio functionality. This is due to the fact that JUCE was originally created with the purpose of being a framework for Tracktion, an audio sequencer, shortly before branching off into a standalone product [6]

Another attractive feature of JUCE is its portability across different operating systems and needs. When a JUCE user builds an audio plugin, a single binary is produced that supports most mainstream plugin formats like VST, VST3, AAX, RTAS and AU using wrapper classes. As all the platform specific code is contained within these wrappers, the user can build VST, VST3, AAX, RTAS and AUs from a single codebase. Targeting the LV2 format is also possible with the use of a fork. In this synthesizer, JUCE made it possible to create a platform accesible plugin from the same codebase.

3.3.2 Projucer

JUCE's 'Projucer' is an IDE which comes installed with the JUCE package. It is a tool for creating and managing JUCE projects. When the files and settings for a JUCE project have been specified, the Projucer automatically generates a collection of 3rd-party project files to allow the project to be compiled natively on each target platform. It can currently generate Xcode projects, Visual Studio projects, Linux Makefiles, Android Ant builds and CodeBlocks projects. As well as providing a way to manage a project's files and settings, it also has a code editor, an integrated GUI editor, wizards for creating new projects and files, and a live coding engine useful for user interface design.

JUCE's 'Projucer' is an IDE which comes installed with the JUCE package. It is a tool for creating and managing JUCE projects. When the specs and settings for a JUCE project have been selected, the Projucer automatically generates a collection of 3rd-party project files to allow the project to be compiled natively on each target platform. It can generate Xcode projects, Visual studio projects, Linux makefiles, Android ant builds and CodeBlocks projects. As well as providing a way to manage a project's files and settings, it also has a code editor, an integrated GUI editor, wizards for creating new projects and files, and a live coding engine useful for user interface design.

As I wanted to develop in Visual studio, I did not need to use the Code editor that is built into the Projucer [6].

3.4 IDE

As mentioned in a previous section, Visual Studio was selected as the development IDE. This was due to a number of factors. Visual studio is the default development environment for JUCE on a windows machine. Setting up Visual studio for JUCE was as simple as specifying in the Projucer that this was the desired platform from the list of options. Once that was chosen, the Projucer provided a button which launched Visual Studio with all the build options and folders already set up for the user so the user could begin development instantly [6].

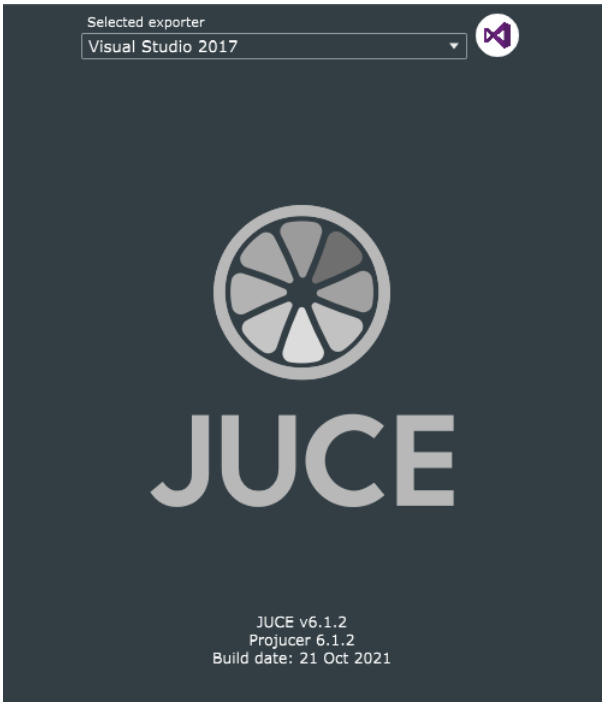

Fig. 2. Projucer interface

3.5 Elected Synthesizer Features

This section covers the core features that were selected for the synthesizer. These tools were not strictly part of the necessities for the synthesizer and were chosen by the developer. Each subsection will briefly mention the role that each component played and how it contributed to the development of the Synthesizer.

3.5.1 Ableton Live 11

Ableton Live 11 (Ableton) is the DAW which was chosen to run and test the Synthesizer after each iteration of development of a feature. Ableton is a widely used digital audio workstation (DAW). The program's name refers to its intended use as a performance instrument for electronic musicians, a way to literally play the studio as an instrument in real time. This practice aligns with Brian Eno's concept of "the studio as compositional tool". Ableton's user base is mainly comprised of electronic dance music producers and hip hop producers, but is also becoming commonplace in academic settings because of its rich and complex work flow. In this proeject, Ableton proved to be extremely useful with its wide variety of options, such as being to modulate the different parameters of the synthesizer and a very effective midi keyboard to play the synth in real time. [7]

3.5.2 Github

A dedicated GitHub repository was set up to store the project. This reduced the risk of losing work done on the project and also gave the project more portability in terms of when and where work could be done on it.

GitHub is Git repository; a Git is a computing term for a VCS (Version Control System) that enables developers to store branches (different versions) of their source code. For this project, GitHub was used as a portable storage solution negating the need for any physical media such as memory sticks or discs. This not only made the project accessible at any time, but protected it from loss or damage. Using GitHub also provided experience in what is a widely used tool within software development or any area where version control is important.

3.6 Elected Synthesizer Features

This sections highlights the different features of the Synthesizer which were chosen to be included in the Synthesizer by the developer. Each subsection will briefly mention the role that each component played and how it contributed to the development of the Synthesizer. The features will always be explained with music theory in mind to highlight their significance.

3.6.1 The Oscillator

In audio synthesis, and oscillator is the part of the Synthesizer which creates sound. It is an electronic/acoustic device used to generate signals with a specific waveform. Oscillators are used as the base for any sound made by an oscillator [8]. Similar to a voice box, the oscillator vibrates causing a waveform to hit the eardrum and pass a specific noise depending on what shape of waveform is selected. In this synthesizer, the user has the option of using:

- **Sine wave:** a sine wave is the most basic and common waveform shape found in synthesis. It has an even and rounded wave shape which produces a smooth, soft sound.
- **Saw wave:** a saw wave is a waveform which is in the shape of a saw tooth. It has one vertical edge and one slanted edge which makes a more harsh, buzzing type of sound.
- **Square wave:** a square wave which is shaped as a repeating wave with rectangular edges. This type of waveform sounds the most electronic and is quite harsh.

[9]

Once the user has chosen their waveform type, they also have the option of using frequency modulation (FM) to shape the waveform further. The user can use two sliders to change the FM frequency and the FM depth. Essentially, FM attempts to apply a specific waveform on top of the existing waveform created by the oscillator, which in turn changes the shape of the original waveform by combining the two, thus creating a new sound. The sound of FM synthesis generally sounds like the noise is wobbling between frequencies depending on how you choose to tweak the two parameters. The fm frequency slider changes the frequency of the new waveform being applied to the original waveform. The fm depth slider changes how deep the waves

on the waveform are and thus changes how drastically the waveform created by the fm modulator affects the original waveform. [10]

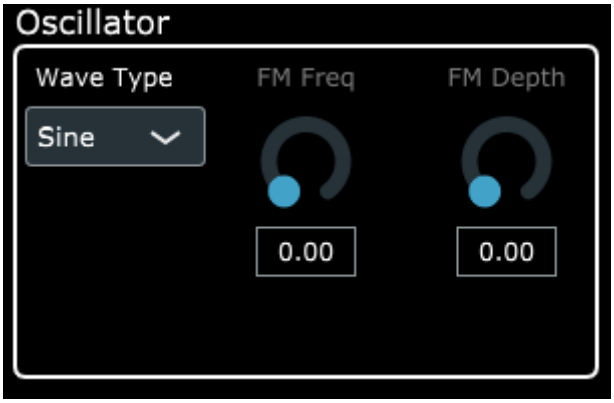

Fig. 3. The Synthesizer Oscillator

3.6.2 The ADSR Component

The ADSR envelope is a section of the synthesizer which controls the ATTACK, DECAY, SUSTAIN and RELEASE of the synthesizers volume envelope. Each of these components is controlled by a slider which can be tweaked depending on what type of sounds you want to make. [11] Each of their purpose is as follows:

- **Attack:** The attack of the volume envelope is how long the sound takes to get to maximum volume. This means that a steep attack curve would make the sound loud and abrupt in a short amount of time while a longer less steep attack curve would make the sound take longer to reach full volume. The attack is always at the beginning of the envelope

- **Decay:** The decay of the volume envelope is how long the sound stays at full volume. A long decay would simulate a sound similar to a violin being slowly strung with a bow while a short decay would simulate a sound similar to a guitar being abruptly plucked with one's finger. The decay is always the second point on a volume envelope.

- **Sustain:** The sustain of the volume envelope is how long the sound takes to drop off. A short sustain means the sound wont be heard for very long when the key is lifted, while a long sustain means the sound will slowly get lower until it reaches the release of the envelope.

- **Release:** Finally, the release is how long the sound takes to reach zero volume, giving the sound an echo or 'reverb' effect, almost like shouting in a large room with good acoustics and being able to hear the noise even after you've finished shouting.

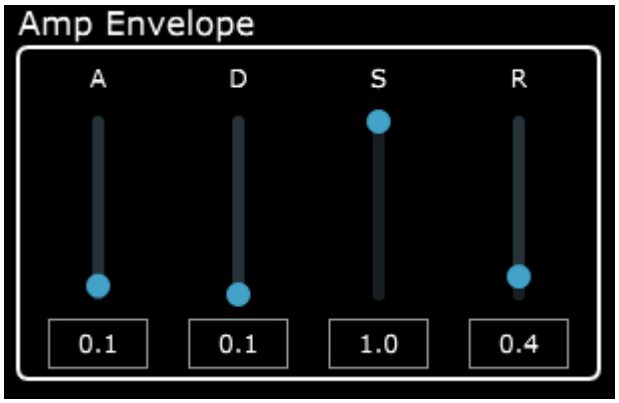

Fig. 4. The ADSR component

3.6.3 The Filter

The Synthesizer features a filter envelope which has the power to filter out certain frequencies of the waveform depending on what type of filter is chosen. The three options the user can pick from are, Low-pass filter, Band-pass filter and High-pass filter. These filters all block out certain frequencies depending on what the user chooses. The low pass filter blocks out any higher frequencies within a chosen range giving the sound a muddied and bass heavy sound, similar to a bass guitar. The band pass filter blocks out higher frequencies and and lower frequencies while allowing frequencies in the mid range to pass through, such as vocals. The high pass filter blocks out the lower range of frequencies which removes any bass [12]. A high pass filter would be usually used on high end drums like high hats and snares, to leave more space for lower frequencies like a bass guitar. The user can not only choose the type of filter they want, but they can also change the filter frequency and filter resonance. The filter frequency decides at which level the filter begins to cut out frequencies, while the resonance is an additional controlled amplification of that cutoff frequency, creating secondary peak forms and colors the original pitch [10].

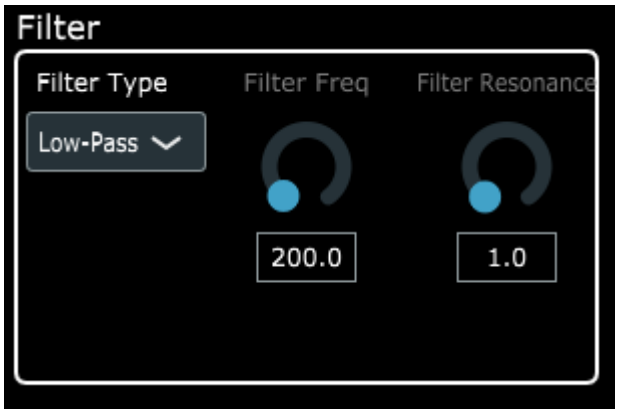

Fig. 5. The Filter

3.6.4 The Filter Modulation Component

The filter modulation envelope is the final piece of the synthesizer that was developed. This part of the synthesizer controls the attack, decay, sustain and release of the filter envelope, rather than the volume envelope. This adds another layer of customization to the filter and greatly increases the types of sounds that the user can make by just cutting out certain frequencies of the waveform [12].

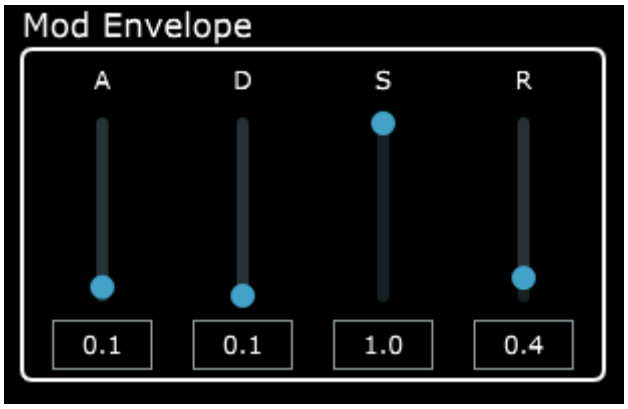

Fig. 6. The Filter Modulator

3.7 Distribution of the Synthesizer

This section will go over what steps were taken to deploy the synthesizer to the public and what technologies had to be used in order to achieve this. These tools weren't necessarily decided at the beginning of planning, but after some further research, the following technologies seemed as they would achieve the goal quickly and effectively.

3.7.1 Netlify

Netlify was the deployment technology used to get the blank website up and running. Netlify is a web framework for creating front-end heavy websites using REST-ful API's with Javascript. The netlify platform uses your git repository to run a build process to pre-render all your pages in static HTML. Netlify creates its own repository that pushes to Github and its own microservices. It then executes and distributes content across a wide CDN to deliver pre-built static websites to visitors. Netlify is extremely efficient as it selects the best CDN and distributes content. This means that the developed website runs faster than on traditional hosting networks. Instead of loading the site each time you visit, the visitor gets a pre-loaded version straight from the nearest server. This process greatly reduces loading times. [13]

Netlify was the perfect choice to host the website to distribute this software as it solidified my knowledge with GIT and really aided me in my development with such an efficient method of pushing new commits online. Any time, some new functionality was developed, the local machine was used for testing and then pushed to the github which would have the updated website online within minutes.

3.7.2 Google Drive

In order to make the software available to somebody accessing the website, a download functionality had to be created. Google drive was used to achieve this. The final plugin file is hosted on good drive, which has its download link attached to a download button on the 'Download' section of the website.

3.8 Synthesizer Website View

This section will focus on the views available for the website of the Synthesizer. The view is the part of the system that displays the data stored in the server. The view also allows users to interact with the system. In the context of this dissertation the view will encompass the web client for desktop and any mobile solutions that are discussed.

3.8.1 Synthesizer Website on desktop

KealansSynth.com is a web application that is built to be run on a desktop computer. Running in the browser means that it does not need to be installed on a computer nor does it restrict users to specific computers. This level of portability is beneficial to the user as it makes the information accessible anywhere in the world provided they have access to a desktop or internet enabled device. As it is run in a web browser it does have mobile support, however this is not a complete on-the-go solution and will be discussed further in section 3.8.2.

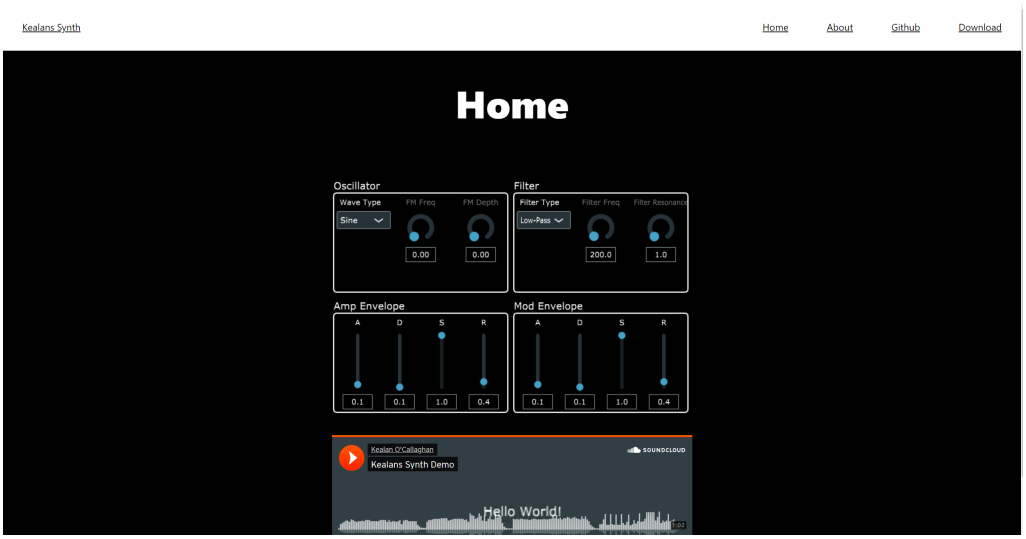

Fig. 6. The website from a desktop

The desktop web application is currently the most used and most supported way of using the Synthesizer website, there are other ways of accessing the data via mobile devices etc. but this is still the most used route. With users however becoming more mobile and competitor products offering mobile solutions the desktop version will get left behind if it is not supplemented by an effective mobile solution.

3.8.2 Synthesizer Website on mobile

The synthesizer website is supported on mobile devices, but only through the browser. This means that users are able to access information on the go provided they have an internet connection and that they are in a suitable place to navigate the website. A problem with navigating the browser on a mobile device is that it requires precise selections and a high level of dexterity from the user. This where a dedicated mobile application could be of use and provide the on-the-go functionality that the website users need, although they may have no use of a mobile app, considering the software package can only be installed on a desktop. The mobile version is purely to read information regarding the synthesizer.

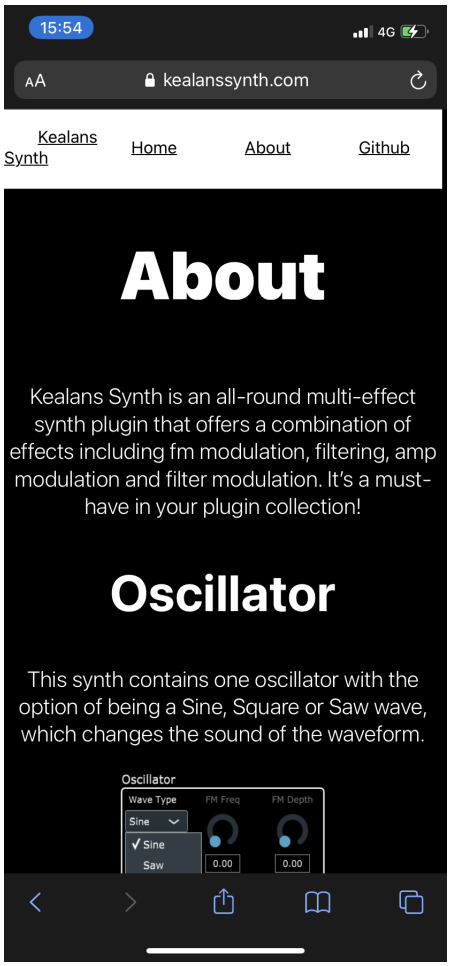

Fig. 7. The website from a Mobile browser

Chapter 4

System Design

4.1 Introduction

This chapter discusses the implementation of the system and how the system came to its current final version. Section 4.2 provides the technical information about the system, including the system and software design decisions taken. Section 4.3 outlines the structure of the system, showing the various directories and file organisation. This section also covers the activities, tools, audio and UI components of the system.

4.2 Technical Information

4.2.1 System and Software Design

As per the requirements the application needs to perform certain tasks, but before they could be implemented the use cases need to be scrutinised and a method drawn up for how each use case would be executed. This section of the paper aims to outline the approach taken to how each component of the application would interact with one another and how they will be executed in the final product. This part of the paper also aims to provide a good idea as to how each part will work.

The system was developed using a hybrid of the "waterfall" model and the agile model. This meant that the benefits of the structured approach offered by the waterfall model could be combined with the prototype approach used in the agile development model. The waterfall model starts off with the definition of requirements. This feeds into the system and software design phase, which establishes an overall system architecture and involves representing the

software system functions in a form that can be converted into executable functions. In the implementation and unit testing phase, the software design is converted into programs and each unit (or program) is tested individually.

The software development process of this project used a combination of both models. It followed the waterfall model in so far as it started with a set of requirements as outlined in the introduction. The overall system and software were designed based on these requirements. The system was then implemented and each module or part of the synthesizer was individually tested. Thus, at a high level, i.e. at an overall system level, the waterfall model was followed.

The project also followed the agile model in certain phases of the development cycle as there were some functionality which could not be executed until other modules had been completed. Once the fourth and final module had been completed, the developer had to retrace back to the second module to fix some functionality before the synthesizer could be marked as completed.

Software design activities include the architectural design, system specification, interface design (i.e. the interface between the sliders and the audio processor), and component design. The overall architecture of the system is shown in figure 8.

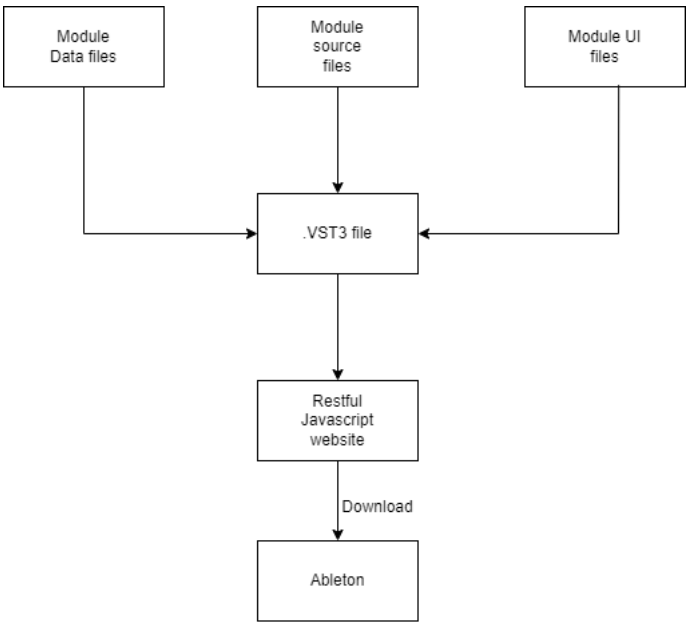

Fig. 8. System Architecture

Project Management

Good software project management is essential in developing and delivery of software projects. Usually the goal is "on schedule and within budget". Although not driven by external or commercial forces, this project had certain "intangible" deadlines, among them the delivery of a system to the supervisors within a time frame not too long after the source data was collected. With software management it is often difficult to estimate the time required to develop the system, especially when new technology is being used. Thus, planning and estimating are iterative processes. Project milestones can be used to monitor the development progress of the project at certain key points.

This project differed from commercial projects (but not research projects) in that the project manager, designer, developer and tester were all the same person. While this provided tremendous insight into the strengths and weaknesses of the development "team" from the management point of view, it meant that more self-discipline was required as there was no "external boss" to answer to. Initially, several development milestones were completed, such as creating the git repository, learning how to initialize a JUCE project and getting the project build to produce a VST file with the ability to be ran in the developers DAW of choice. It was difficult to estimate the time required for development, as it was a completely new technology to the developer (and indeed to the software development community in general). However, as the project progressed and the learning curve traversed, it became easier to estimate the time required (which diminished progressively as the project developed).

Design Quality

One important issue in this project was design quality. Many factors contribute to the quality of the design, including cohesion, coupling, understandability and adaptability. Cohesion is a measure of how closely the parts of a component relate to each other – it is the internal relationship between the parts of a component. It is desirable to have close cohesion as it means that a unit represents a single part of the problem solution. Coupling is a measure of the strength of component interconnections – it is the external relationship between components.

It is desirable to have weak coupling and it means that components are (almost) independent of each other. Understandability is important because in order to be able to change the design it must first be understood. Factors that affect understandability include cohesion and coupling, naming conventions, documentation and complexity. Adaptability refers to how easy it is

to change the design.

This project endeavoured to have strong cohesion and weak coupling, by having all the related components together in their own classes, which were further grouped into their own folders (UI, Source and Data). Thus, section information could be found in one file and any changes required could be effected on the section component alone, causing no impact on the other components.

Data file names (e.g. ADSRData.cpp and OSCData.cpp) and UI file names (e.g. ADSRComponent.cpp and OSCComponent.cpp) aim to be clear. Complexity, as understood here, is an intuitive concept. Given the complexity of working with a new technology (Audio development, C++), it was decided the .cpp files would be written in as straightforward a way as possible, even if this meant not using the latest "expert level" way of doing something or writing it in a more concise way.

Software Reuse
The template aims to be a reusable piece of software. In order to achieve this, certain design guidelines were borne in mind. A modular approach was adopted to the development of the Synthesizer, thus that another developer could take one of the four modules of the Synthesizer, without a huge amount of refactoring and moving to be done to fit in another project. Care was taken with naming conventions. Individually reusable components (e.g. Oscillator) were extracted and imported into individual modules. Where possible, parts of the system were generated. For example, at a low level, the names of the default methods were auto generated.

System portability is another key issue to consider. Given the fact that Synthesizer technologies have been used and that one of their goals is to be computer and operating system independent, it is hoped that the system is portable for future versions of JUCE and DAW's.

4.2.2 Design Decisions

Why make a website? Distribution of the Synthesizer was something the developer knew would be essential to the project in order to make the project more tangible as a real world piece of software. It allowed the incorporation of multimedia elements into a page in an almost seamless manner in order to explain the different components of the Synthesizer. This was done in order to make the software more accessible to the general public, as the majority

of music producers may not be informed on pulling down some software from Github.

Secondly, the Internet allows for updates to the Synthesizer learning system. Errors can be corrected and new material can be provided. Thirdly, communications between learners can potentially be established via email, discussion groups and bulletin boards. This could encourage a sense of community amongst learners, which can enhance the music learning process.

GUI and UX
GUI and user experience design are an important feature in application development. The user never gets to see what is under the GUI, so as far as they are aware what they see represents everything the application is capable of. It is important the GUI tells the user everything they need to know about the application and that it allows them to fully utilise its features. In the case of the Synthesizer, the JUCE framework equipped the developer with effective tools to create and shape the GUI. Although the GUI is simple, it utilizes the different tools effectively to mould the users sound to their liking:

- **Dropdown Boxes:** For items in the Synthesizer which must be chosen such as oscillator wave type and filter type, a dropdown box was chosen as it reduces clutter and neatly packs functionality into its own convenient section.

Fig. 9. Dropdown Box

- **Vertical Sliders:** For values which needed to be changed that didnt have a large range (ADSR), vertical sliders were the perfect choice as they are accurate and offer ease of use to the user.

Fig. 11. Vertical Slider

- **Circular Sliders:** For values with a larger range and where accuracy isn't as important, circular sliders are a perfect fit as they are compact

and can be automated easily, almost emulating twisting a knob on a real life synthesizer.

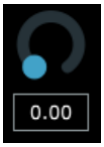

Fig. 12. Circular Slider

File Structure
During the early stages of development, most of the components had began to be developed from 2 to 3 'god' classes, but as development went on and components started to build up, it was clear a more organized approach was necessary. Each of the four modules on the Synthesizer has 4 classes. For example, in the case of the oscillator module, it has the following files:

- OscComponent.cpp
- OscComponent.h
- OscData.cpp
- OscData.h

All of the component classes were kept in a folder called 'UI' while the data files were kept in a folder called 'Data'. This ensured that the UI and audio development would be kept completely separate to one another.

The other files which done a large amount of the audio processing and cohesion were kept in a folder called 'Source' which was auto-generated at the beginning of development and not created by the developer like the other two folders.

4.2.3 System Structure

The Synthesizer contains many different components. This section provides a succinct overview of the structure of the system. Figure 8 (p28) shows the overall system architecture, but within the three folders of the synthesizer code, there is a certain architecture which was followed in an attempt to make the code as portable and standard abiding as possible.

Synthesizer structure

The Synthesizer system is structured as follows:

- There is four modules in the synthesizer.
- Each module has its own inputs that the user can tweak at a UI level.
- Each of these inputs whether it be a button, slider or dropdown box is tied to a method which takes values.
- These values are then passed through to the data files as attachments.
- The data files process each value passed through to it differently depending on what type of audio method it is connected to.
- The audio signal is then passed back to the audio renderer in the PluginProcessor.cpp file which is then output to the speakers of whichever computer the synthesizer is being ran on.

A breakdown of the files and their purpose is listed below.

File	Purpose
Data Files	
ADSRData	Updates the value of the adsr in the process block
OSCData	Updates the value of the Oscillator in the process block
FilterData	Changes filter on input and send waveform to process block
UI Files	
ADSRComponent	Designs ADSR sliders and gathers data from user
OSCComponent	Designs oscillator dropdown box and gathers data from user
FilterComponent	Designs filter sliders and gathers data from user
Proscessing File	
PluginEditor	Sets padding and bounds for synthesizer UI
PluginProcessor	Process all audio variables before being sent to output
SynthSound	Stores data such as midi input and midiButtonPressed method
SynthVoice	Plays actual sound processed by processor class

Website structure

The website is structured as follows:

- There are Javascript files which compile into the UI you see at KealansSynth.com
- The website is very UI heavy and does not contain much background functionality
- There is an audio file demonstrating what sounds the synthesizer can make hosted on soundcloud which has a player embedded into the homepage of the website.
- The Github repository for the software is also embedded in the website.
- The synthesizer itself is downloadable from the download page using google drive.
- This app is then hosted on Netlify.

4.3 Synthesizer Functionality

Within each module of the Synthesizer, there are different methods and algorithms which apply the different effects on the audio signal. This section will single out each module and explain how they worked at the granular level.

4.3.1 Oscillator

As mentioned previously, the oscillator is the part of the synthesizer which produces the base waveform itself. In our Synthesizer, the Oscillator class from the JUCE DSP module gets called from our OSCData class. The developer created a method called setWaveType() which uses various mathematical functions to achieve the desired wave form as seen below:

- **Sine wave:** return std::sin(x);
- **Saw wave:** return x / juce::MathConstants¡float¿::pi;
- **Square wave:** return x ¡ 0.0f ? -1.0f : 1.0f;);

Once the wave type is chosen using this method, the frequency of the waveform gets sent to the process block.

4.3.2 ADSR

As mentioned previously, the ADSR is the part of the synthesizer which changes the volumes of the waveform frequency at different intervals. In our synthesizer, the ADSR class from JUCE is called from the synthVoice class. The SynthVoice is where the volume of the sound is declared at a certain point eg. If you were pressing two keys at once there would be two different voices initialized.

- adsr.applyEnvelopeToBuffer(synthBuffer, 0, synthBuffer.getNumSamples());

4.3.3 Filter

As mentioned previously, the Filter is the part of the synthesizer which blocks out certain frequencies and thus giving the sound of the synthesizer a low, high or mid range sound. In the synthesizer, the developer made use of the "StateVariableTPTFilterType" method. This gave the develper an option of HighPass, LowPass and BandPass which was made accessible to the user using a switch case statement.

```
case 0:
filter.setType(juce::dsp::StateVariableTPTFilterType::lowpass);
break;
case 1:
filter.setType(juce::dsp::StateVariableTPTFilterType::bandpass);
break;
case 2:
filter.setType(juce::dsp::StateVariableTPTFilterType::highpass);
break;
```

4.3.4 Filter Modulator

As mentioned previously, the Filter Modulator is the part of the Synthesizer which changes the level and timecodes of the filter being applied to the waveform, meaning the user can shape how much the filter is applied to the sound and when it happens. This was done differently to the previous two modules. The Filter Modulator was built into the Filter class as it was not necessary to rewrite all of the code for something that was constantly going to be updated at the same time as the filter. This saves a lot of processing power and allows for more loose coupling within the architecture of the code.

```
filter.setCutoffFrequency(modulatedFrequency);
```

Chapter 5

System Evaluation

5.1 Introduction

This chapter will outline the different steps taken to ensure that the Synthesizer developed was robust and tested to a satisfactory level. Section 5.2 reviews the testing that took place at two different levels. The concept testing asks the question as to whether the concept of learning from the simplicity of the synthesizer was effective or not. Software testing refers to the testing of the produced Synthesizer from a functional standpoint. It looks at questions such as: whether the UI is generated correctly, whether all the sliders work, whether the midi input works are breaking point levels and if the synthesizer is in tune when effected by all of its different modules. Section 5.3 covers the evaluation phase of the system. It reviews the objectives and requirements of the system. It reports on the template implementation and whether it works as a real world Synthesizer. Section 5.4 reviews the implementation of the Synthesizer. Section 5.5 highlights some unexpected outcomes of the project. Section 5.6 provides a summary of the chapter.

5.2 Testing

This section covers the testing of the template. It deals with software testing and testing the concept behind the project itself. Figure 13 shows the chronology to the testing phase of the project. The combination of the waterfall and agile models adopted in the development of the project (see section 4) meant that the development and testing occurred in parallel and were closely intertwined.

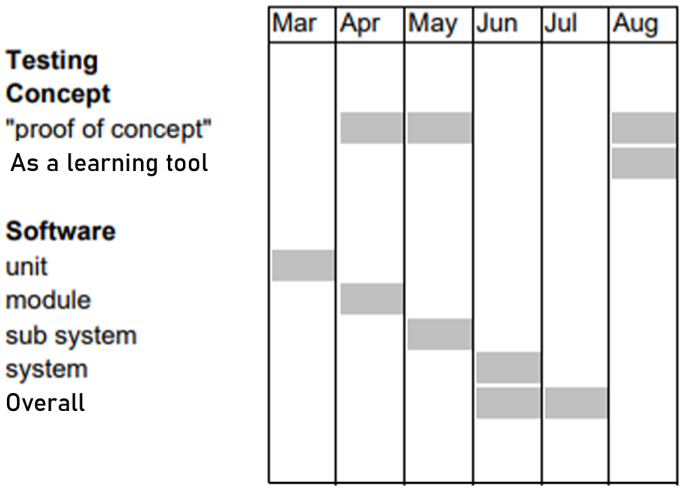

Fig. 13. Test Phases

5.2.1 Concept Testing

At the highest level, the project had to be tested at the ”proof-of-concept” point of view. That is, does the concept of creating a Synthesizer as a learning tool actually work as this function? The Synthesizer developed along with the informational website was distributed amongst a group of like-minded music producers to see if the aim of using the Synthesizer as a means of learning more about digital audio synthesis had succeeded or not.

The general consensus among the group was that it was ”refreshing” to see such a bare bones representation of the often complicated versions of the Synthesizer. Considering that the Synthesizer had the informational website to go with it, it seemed that a lot more conceptual learning was done while using the Synthesizer rather than just using an out of the box Synthesizer from one of the conglomerate music companies such as Roland. Below is some of the feedback received by the group.

Fig. 14. Feedback

In terms of robustness, the Synthesizer scored 5/5 meaning that in terms of the modules it possesses, it carries them out to their full potential. In terms of ease of use, it also scored 5/5 meaning that the UI is user friendly and accessible to those with some prior knowledge of using Synthesizers. As a learning tool it scored 4/5, meaning that it is effective as a learning tool but some feedback that I got was that it would have been nice to show some examples of using the synthesizer first hand on the website, as a video. Finally, it scored 3/5 when compared against other synthesizers, meaning that although it is an effective piece of music creating software, it cant compare to the larger applications as it does not come with some vital features like presets and its own midi keyboard.

5.2.2 Software Testing

Software testing involves both verification and validation (Somerville, 1996). Verification involves checking that the program conforms to its specification, while validation involves checking that the program as implemented meets the expectations of the user. Static checking techniques include program inspections and analysis. Dynamic techniques (tests) involve exercising the system. The testing process normally has five stages. Firstly, individual units are tested in unit testing. Module testing tests modules (usually a collection of dependent units). Sub-system testing tests collections of modules and often exposes sub-system interface mismatches. System testing tests the system as a whole and finally, there is (user) acceptance testing.

In this project, unit testing was carried out on each individual module of the Synthesizer, eg. each slider/option of each module. Module testing was

carried out on each logical unit (e.g. section of the Synthesizer such as the oscillator module or filter module). Subsystem testing mainly involved testing the Synthesizer through the DAW, meaning other possible variables were used in conjunction with the synthesizer to see how it would hold up. System testing was then carried out for the complete system, using all possible inputs and variables.

When carrying out software testing, there are several strategies that can be adopted. The two most basic strategies are top-down and bottom-up testing. Top-down testing starts with the most abstract component and works downwards while bottom-up testing starts with the fundamental components and works upwards. Top-down testing helps to detect unnoticed design errors at an early stage and can provide a psychological boost to the development team as a limited, working system is available at an early stage in development. Bottom-up testing of the low level system components is almost always necessary.

Both strategies were used in the development of the project. Top-down testing was used to test the overall look-and-feel and structure of the system. It checked that the system hung together and was not missing any key components. Bottom-up testing was used to test each component and module. As it is more important to test commonly used parts of the system rather than rarely exercised parts, the Oscillator part of the system was heavily tested, as it can react in strange ways when used in conjunction with different effect, and cause undesired issues like volume clipping and distortion.

5.2.3 Synthesizer Testing

Synthesizer testing took place on several levels. Firstly, there was the overall system testing. The Synthesizer was tested to ensure that all the effects were correctly carried out and that the sounds correctly matched their slider or button.

The Oscillator

Testing the oscillator was arguably the most important part of the whole testing process, as it is after all the part of the Synthesizer which creates the noise we hear. For the first half of the project, there was an issue where there would be a clicking noise coming from the oscillator every time a key was pressed. This is due to the fact that as the oscillator was moving the waveform up and down, every time a new note was pressed, the oscillator

was not at the 0db mark, causing a ”skip” in the waveform which caused the clicking noise. This was resolved later in development which initialized the oscillator's waveform to 0 when a key is pressed, removing the clicking. Fortunately, apart from this no further changes were needed for the oscillator, although continual testing was important to ensure it didn't clash with any new features.

The Filter

The filter was also thankfully just as unproblematic as the oscillator in that there were no show stopping errors with it, although there was quite a bit of fine tuning to find what the right frequency ranges would sound best for the filter to cut out and act upon.

The ADSR

The ADSR was the most problematic module of the development cycle. The release functionality would not work for almost the whole duration of the project. About one month before the end of development, the developer stripped out the module and built it up from scratch as the release functionality was still not working. This thankfully fixed the issue and the ADSR provided the synthesizer with a much need release (Echo) functionality.

5.3 Evaluation

This section covers the evaluation phase of the project. It reviews the objectives and requirements of the project and evaluates the implementation of the template. It answers the question as to whether the system works. Furthermore, it discusses the software and user evaluation of the project.

5.3.1 Review of Objectives and Requirements

A brief review is provided here of the objectives and requirements of the project. Chapter 1 contains more complete coverage of the project objectives, requirements and constraints.

Objectives

The main objective of this project was to create a plugin which can be ran on any digital audio workstation as a VST3 file. In addition, there were secondary objectives. The first was to have the Synthesizer have Midi in and

out capability which was completed, partially by JUCE itself. The second was to create a synth with various filters and envelope modulation capabilities. The final major objective was to create a website to give info on the program and host it, giving a visitor the option to download and test the plugin themselves.

Requirements

There were also several requirements that had to be borne in mind. The plugin had to have the capability to be ran across all platforms including DAW's and operating systems. The second was to gain user feedback and have interactive testing for future development of the Synthesizer. The final requirement was that the website would have the option of downloading the Synthesizer so that it could be easily shared and distributed.

Does it Work as a Digital Music Learning Program?

One of the main aims of this whole project was to create an open source and freely available method of learning some basic digital audio concepts and music theory. From the developers perspective, a huge amount of information was taken on board through the course of development. But, from a new users point of view, it is important that they also take some valuable information from the Synthesizer and website as they did not get to see behind the scenes of development.

The developer demonstrated the full website and program to a group of like minded music enthusiasts and the general feedback was that it was a very useful tool which made understanding the basic laws of digital audio very understandable. They appreciated the honest and bare bones nature of the Synthesizer and found it very effective in carrying out all of the basic functions a modern day Synthesizer should have.

5.4 Software Evaluation

5.4.1 Chosen technology

A decision was made early on to use JUCE as the framework for this project. Apart from being the mainly used framework for audio plugins, its helpful documentation and internet resources really lended to itself. There was the option of using another framework such as Synth-edit, but considering this was a project for a software degree, the right choice was to choose JUCE as

it is the most code dependant of all the frameworks.

In relation to the website, it was an easy choice making a RESTful Javascript application as it was not something the developer was overly familiar with and offered some good benefits such as fast loading times and easy deployment using Netlify.

5.4.2 Synthesizer Evaluation

Overall, the Synthesizer was a very successful and robust piece of software, as it completed all of its objectives without any major downfalls or setbacks. There are definitely some improvements to be made to the Synthesizer but they all fall within the realm of additions and not changes to the actual features it currently contains.

The only limitation with a project like this is time, as there can always be more functionality added as the possibilities are endless. If there was more time for development, the developer would have added a dedicated midi keyboard to the synthesizer, meaning that notes of the synth could be played right in the box without having to assigned to a channel in an audio interface.

Another addition that would have suited this project perfectly would be a waveform display which shows the different shapes that the waveform goes through as the wave is changed and effected with FM modulation and filtering.

5.5 Other Objectives

The other two objectives which were not necessarily related to the synth were to:

- Learn how to program with C++ and work within a comprehensive development environment.
- Become adept at programming by one's self and learn new skills for future projects.

These objectives were achieved as the developer now feels very comfortable programming from start to end by one's self and has a great handle on a new language which they had not studied in detail before, both of which will stand to the developer in future jobs and their career as a software developer.

5.6 Unexpected Outcome

When the developer showcased their website and synthesizer to the community of music producers, they had hoped that a few people would be mildly interested in the program. The developer was quite surprised by the reaction. Almost all the producers who seen it were impressed and were interested in trying it out for themselves and even implementing it into their own workflow of music creating due to its simplistic nature.

Not only did this course of events inspire the developer to keep actively researching into audio development, but gave them a project which has some real substance behind it and that they knew would be useful in their future of audio development and software development entirely.

5.7 Summary

This chapter covered the testing and evaluation phases of the project. It discussed the testing of the project from several different points of view. The ”proof-of-concept” testing was the most fundamental testing that was necessary for the project. As with any software development, the program had to be tested from a software point of view. However, even if the software is tested as required, the developed Synthesizer is virtually useless if its contents are not correct. Thus, the module testing was an important aspect of the testing phase.

Evaluation has often been a neglected part of the development process. However, this project has endeavoured to incorporate it at various stages during development, from early user evaluation of a system prototype to user evaluation in the intended implementation situation. The various aspects of evaluation that are important in a project such as this were all described in this chapter.

Chapter 6

Conclusion

6.1 Introduction

This chapter summarises the conclusions of the project. Section 9.2 discusses both the technical and general findings of the project. The technical and logistical difficulties are reviewed. Suggested improvements are outlined and the limitations of the project are reported. Section 9.3 identifies some suggestions for future work, while section 9.4 provides the overall conclusions on the project and section 9.5 gives a summary of the chapter.

6.2 Project Summary

6.2.1 Benefits of this Project

Synthesizer Produced

The project produced a Synthesizer that is capable of being played on any DAW as a fully functioning instrument with the possibility of being morphed into a variety of different sounds including bass, strings, plucks and kicks. It achieves this with its Oscillator, Filter, ADSR module, FM modulator and ADSR envelope modulator. It provides the base for a much more powerful plugin due to its modular nature, it would not be difficult to add another module and increase its use cases exponentially.

Educational Website Produced

The project also produced a fully functional RESTful Javascript website hosted with Netlify. The website informs the user of all of the functionality of the Synthesizer while also briefly going into the realm of music theory and

digital audio concepts. The website also contains an embedded soundcloud link which showcases some of the sounds which have been made by the Synthesizer as well as a link to the Github, driving forward the Open Source nature of the project.

6.3 Findings

6.3.1 Technical Findings

The use of the JUCE framework proved very useful during the project development. It allowed the developer to code in an object oriented fashion, meaning they could create modules to carry out the different functions, rather than one intertwined program full of dependencies. Each module could be swapped and changed with other functioning modules with very little refactoring as all of the modules work almost independently of each other.

The degree to which object oriented programming applies to a project such as an audio plugin was very surprising to the developer and was very interesting to see how the concepts carried over to a project like this which was mainly made with different modules and independent moving parts.

6.3.2 General Findings

The project showed that it is possible to create an effective Synthesizer and that using the modular influenced approach proved to be the most logistical and robust method. Planning is important to ensure that things go smoothly when working with the modular approach. One module of the Synthesizer may work flawlessly while another could slow development time and cause issues in conjunction with other modules so having a solid plan in place as to where time and resources will be spent is essential to a successful development process - especially when developing by one's self.

The ability to include some examples of the sounds the Synthesizer had the ability of creating proved to be very helpful in trying to convince those viewing the webpage that it was worth their time downloading the Synthesizer due to the simplistic look of the User Interface.

The Synthesizer could be vastly improved upon considering that by just

adding one more module, you would be increasing the possible outputs of the synthesizer exponentially, which almost left the feeling for the developer that there is more work to be done than anticipated, even after submission of the project, and that the possibilities for the future of the Syntheizer and website are endless.

The level of interest in the project was also refreshing for the developer. After showcasing to friends and like-minded music producers, the feedback was more than welcoming. They were all impressed that such software could be developed by a singular person in the frame of a few months, as it is only a few iterations behind the software that they are familiar with and use on a daily basis to create music.

6.4 Difficulties Encountered

6.4.1 Technical Difficulties

The main technical difficulty encountered during the development of the Synthesizer was the steep learning curve associated with audio plugin software development. In comparison to some other common technologies such as React development or game development, there is significantly less online resources to get a solid base from. The larger companies with the most popular plugins don't share their source code, meaning that the resources online are quite niche and specific to the person developing the software. In addition, working with a relatively new technology meant that there was not a high level of local expertise, so a lot of the learning had to be undertaken from scratch.

Although the documents for JUCE are considered some of the best in relation to other frameworks, it was still quite difficult for the developer to get good at understanding and developing using pretty much just the JUCE documentation. This was not a huge downside though, as it gave the developer a good understanding of how to interpret and apply documentation which plays a vital role in the life of any software developer.

6.4.2 Logistical Difficulties

The only logistical difficulty that was faced during the duration of this project was the loss of the project due to the developers hard drive becoming corrupted near the end of development. This was solved by pulling down the

most recent version of the project from Github, and although some progress was lost, it took no more than a few hours to get back up and running with the JUCE framework and back to the Synthesizer version.

6.4.3 Different Roles

During the development of this project, the developer had to take on more than afew roles than they were used to in their history of programming. Skills in relation to web development, object oriented development, learning documentation and UI design were all utilised extensively throughout the course of the project.

6.5 Improvements

With a project such as this, there is always room for improvement. In relation to the UI, more information could have fitted into the empty space or into an info box which appears when being hovered upon. This would give even more information on the functionality of the synth without having to rely on the website as much.

As mentioned previously, it would also have been a large improvement to the website if some informational videos and tutorials were available to give the user of the synth a head start into production, as a lot of users may not have the skill set to use the plugin to its full potential.

6.6 Limitations

Obviously a project such as this could not hope to deliver a system with all possible elements. There are some limitations that must be noted. The main one being time limitations. Generally, a Synthesizer is developed between a team working for a large company such as Roland Soundtoys or Ableton. For one person, there is a certain amount of development which is possible in a few months for a project such as this, which was achieved within the timeframe for this project while considering other subjects and deadlines.

6.7 Suggestions for Future Work

As the field of digital audio and plugins is so diverse, the list of suggestions for future work goes on. Several in particular stand out. One is the addition

of a dedicated midi keyboard in the actual Synth. This would allow the user to make sounds in the box without the need of a DAW. This ramps up the portability and accessibility of the Synth, considering modern DAW's average out at around 500 euro for a regular license.

Further development of the website is also in the pipeline for this project. An interactive experience would go hand in hand with an educational project like this, as it adds another layer of interest for the average user to delve deeper into the field of digital audio production.

In the Synthesizer itself, it was obvious throughout development that a visualization of the produced audio frequency would be very beneficial to a user as it would teach them more about the theory behind Synthesizers and give them feedback in real time to their inputs with the plugin.

Finally, and perhaps most importantly - Presets are a vital and often overlooked feature of any Synthesizer. In most musical Synthesizers, they are packed with sometimes hundreds of presets to give the user some inspiration or to just use straight from the box, rather than building a sound from scratch each time. The presets would change the inputs to the Synthesizer with the click of a button and introduce the user to methods they may not have thought of by themselves.

6.8 Conclusions

This project succeeded in creating an Audio Synthesizer in C++ with the JUCE framework. A simple but effective Synthesizer was created which can be ran on any DAW across any operating system, using the newest plugin technology - VST3.

The use of modern technology and straight-forward audio development practices to teach music theory can help boost the perception of such a field and make it more accessible to potential learners. Potential learners are being motivated to delve into the world of audio engineering and people have been downloading copies of the Synthesizer.

While carrying out research for this project, it was difficult to find information on audio engineering in an educational format. Education on audio engineering was plentiful and information on plugins was available, but there was very little educational information available to bridge the gap between

the two fields. This project managed to address this largely unreported yet intriguing area. Audio engineering plugins have been developed for education but it tends not to incorporate modern basic teaching techniques. This project demonstrated that it was possible to do so through the use of a simplistic and sleek Javascript website.

Often the only literature available on a specific audio engineering is that provided by linguists. It tends to be technical in nature and is not immediately applicable to basic digital music theory. This combination of software and information allows anyone with a slight interest in the technology to clearly understand the core concepts before delving deeper to the subject by experimenting with the software themselves. The technology allows the information to be presented in a manner more accessible to non-linguists than might normally be the case with interfaces that focus more on linguistics.

The positive reaction to the Synthesizer indicates that the project has identified and addressed a real need, as many people are interested in this fun and exciting field, but often don't know where to start with the huge amount of information out there which can be daunting.

In conclusion, the project was more successful that could have been hoped for at the start. There were several unknown elements at the start of the project which meant that it had a certain element of risk. These included not knowing if it was even possible for a solo developer to create a plugin capable of being independent of operating system or audio workstation. There was also risk with the difficulty of obtaining information about basic audio engineering and the uncertainty of knowing how well the project would be received within the music community. However, these potential problems were successfully overcome. Non-expert computer users were able to use the website and modify the Synthesizer to their liking after minimal studying of the theory behind it - while more adept users found it amusing to use a more straight-forward version of the often over complicated and flashy Synthesizers on the current market. The project paved a path in the right direction for the developer as they are excited with how much further they can go in the field of audio programming with this new found knowledge.

6.9 Summary

This chapter summarised the benefits of the project, including the powerful yet simplistic Synthesizer capable of being modified into something even more

powerful and robust while educating its users on the very core concepts of music theory and digital audio. The technical difficulties encountered during the development of the project were reported, as were the logistic problems that were faced. Possible improvements to the system were identified, including more information to be included in the Synthesizer along with some informational videos on the website. There are many possible themes for future work based on this project, such as "in-the-box" midi functionality, stripping the Synthesizer of its dependability to a DAW and adding some visualization to the waveform of the sound created by the user in real time. The chapter concluded with the comment that the project succeeded in developing an educational Synthesizer as originally envisaged which, while not perfect has shown itself to be usable and that anybody can enjoy the benefits of music when used in conjunction with modern technology.

.1 Appendices

This contains the link to the Synthesizer website and Github repository.

.2 Github Links

Project repository
https://github.com/kealanocallaghan2000/AppliedProject

Website repository
https://github.com/kealanocallaghan2000/Synthesizer

.3 Website Link

Website
https://www.kealanssynth.com/

.4 Instructions

To run the Synthesizer on a local machine, the file must be downloaded from the mentioned website and installed to a Digital Audio Workspace.

Bibliography

[1] C. Leider, *Digital audio workstation*. McGraw-Hill New York, 2004.

[2] C. Rasmussen, *Evaluating the usability of software synthesizers: An analysis and first approach*. PhD thesis, University of Guelph, 2018.

[3] M. Matthews, *The 'Father of Computer Music'*. Packt Publishing Ltd, 2006.

[4] W. C. Pirkle, "Vst3 programming guide," in *Designing Audio Effect Plugins in C++*, pp. 32–53, Routledge, 2019.

[5] B. Clark, "The differences between vst2 and vst3," 2022.

[6] S. Tanabu and Y. Watanabe, "The possibility of cross-platform library "juce"–the reason why programmers for music production software choose juce," *IEICE Technical Report; IEICE Tech. Rep.*, vol. 118, no. 149, pp. 147–151, 2018.

[7] E. Hein, "Ableton live 11," *Journal of the American Musicological Society*, vol. 74, no. 1, pp. 214–225, 2021.

[8] S. Cann, *How to Make a Noise: A Comprehensive Guide to Synthesizer Programming*. Simon Cann, 2007.

[9] T. Pinch and F. Trocco, *Analog days: The invention and impact of the Moog synthesizer*. Harvard University Press, 2004.

[10] B. Van der Pol, "The fundamental principles of frequency modulation," *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, vol. 93, no. 23, pp. 153–158, 1946.

[11] D. Creasey, *Audio Processes: Musical Analysis, Modification, Synthesis, and Control*. Routledge, 2016.

[12] Y. Jhung and S. Park, "Architecture of dual mode audio filter for ac-3 and mpeg," *IEEE transactions on consumer electronics*, vol. 43, no. 3, pp. 575–585, 1997.

[13] N. Biswas, "Deploying the site in netlify," in *Extending Gatsby*, pp. 149–175, Springer, 2021.