UNIVERSITY OF SUSSEX

School of Engineering&Informatics

Design and implementation of a subtractive synthesiser with a fixed signal path approach using the JUCE Framework

Computer Science & Artificial Intelligence BSc

Supervisor: Dr Kingsley Sage

Candidate Number: 221455

Signed statement

This report is submitted as part requirement for the degree of BSc (Hons) Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged. I hereby give permission for a copy of this report to be loaned out to students in future years.

Submission year: 2022

Abstract

In this project, I developed a subtractive synthesiser that allows users to create the sound they want using C++ and the JUCE Framework. Various functions were implemented without restrictions on oscillators, effectors, and modulation sources used in the subtractive synthesiser. Through this project, I investigated understand the software functions of synthesisers distributed in the modern synthesiser market and the JUCE libraries. This synthesiser program includes oscillators, filters, an envelope, effectors, a MIDI keyboard, and an oscilloscope. Also, a standalone version of the program and a VST version of the file exists separately. User testing was performed to ensure quality and to receive feedback on the program's satisfaction as well.

Contents

List of Figures

1. Summary

This synthesiser program uses the JUCE Framework and is developed with a fully interactive GUI. Users can manipulate the program by adjusting the value of each function of knobs by clicking&dragging the mouse or pressing their own keyboard for execution. Each component of the user interface is provided by the JUCE Framework. The program was implemented in the Visual Studio 2019 with C++ 17.
The program consists of two oscillators (sine wave (Figure 1), square wave, saw wave, R-sawTooth wave (x-axis symmetry the saw wave; $-saw(x)$), triangle wave, and white noise, saw smooth wave, softsine wave (Figure 2), hardsine wave (Figure 3)) parts, a filter (Low-pass filter, High-pass filter, Band-pass filter, Cut Frequency, and Resonance) part, an envelope (Amplifier ADSR, Filter ADSR) part, a compressor (Threshold, Ratio, Attack, and Release) part, a phaser (Rate, Depth, Center Frequency, Feedback, and Mix) part, a chorus (Rate, Depth, Center Delay, Feedback and Depth) part, a reverb (Size, Damping, Width Dry, Wet, Freeze and Delay) part, and a meter & a virtual keyboard part. The standalone version incorporates a virtual keyboard; the VST version responds to MIDI data.
Since this program was created through the JUCE Framework, the process of installing JUCE in the development environment where users will run the program should be preceded in order to run it. JUCE's installation work is briefly described on the next page.
This paper includes all project processes and an explanation of the implemented synthesiser program. The report consists of the following 11 sections as table of contents above:

1. Abstract 2. Summary 3. Introduction 4. Professional and Ethical Considerations 5. Background Knowledge 6. Project Design 7. Implement Project 8. Evaluation 9. Conclusion 10. Further Work 11. Appendices

The **Project Design** section describes why I chose a JUCE Framework application as a development environment, as well as the benefits that have been brought to the development process and the application itself. It also contains the requirements for the project, how the objectives were set up, and plans for time management during the project. The **Implement Project** section describes the methodology used to implement the synthesiser and indicates the problems encountered during the project and its solutions. The **Evaluation** section provides a reflection of the user tests performed. In addition, it contains the requirements to show how far I met the project requirements. The evaluation acknowledges positive feedback and concludes that the application is actually suitable for the program users based on the user test result. Considering all previous discussions in the **Conclusion** section, the report concludes how the project has contributed to personal and positive learning experiences. In the **Future work** section, additional work ideas were presented for the possible continuation of the project.
In addition, all the code for the synthesiser program can be found on my GitHub. Therefore, all users can easily download the program file through 'git clone' command. (E.g. git clone https://github.com/didot05/Final_Year_Project.git)

The github link of the synthesiser program:
https://github.com/didot05/Final_Year_Project

OSC: Sine, Softsine, HardSine waveform graph - Oscilloscope

The softsine and hardsine waveforms are waveforms that I made for various OSC waveforms types. Figure 1 shows the basic sine waveform, and Figure 2 is more round than the basic sine wave, so I assigned labels as 'SoftSine' arbitrarily. Figure 3 is a bit like a square waveform, so I assigned labels as 'HardSine' arbitrarily:

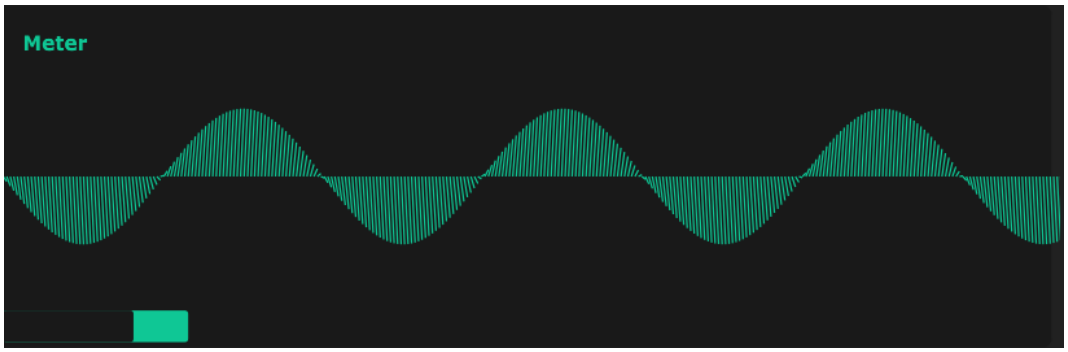

Figure 1: OSCILLOSCOPE; Sine waveform

```
return std::sin (x);
```

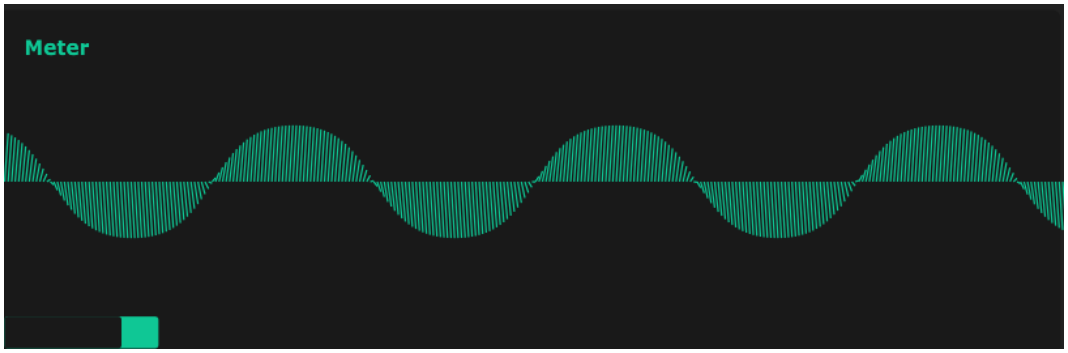

Figure 2: OSCILLOSCOPE; SoftSine waveform

```
return tanh(1.2 * std::sin(x));
```

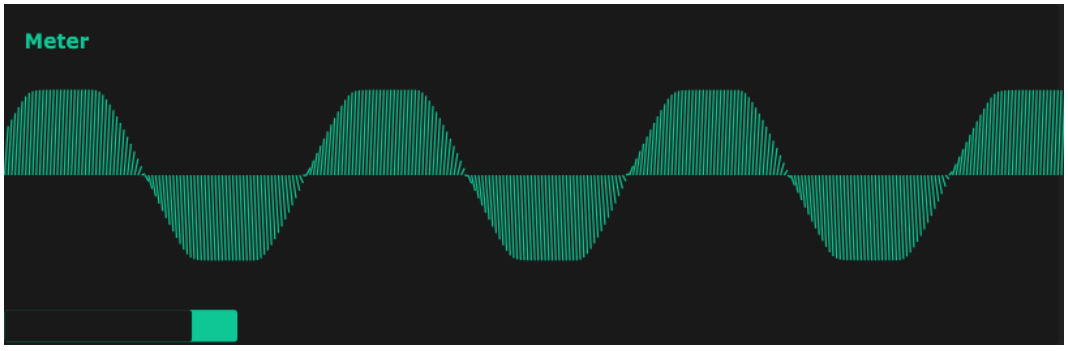

Figure 3: OSCILLOSCOPE; HardSine waveform

```
double AudioOutput = 1.3 * (sin(x));
  if (AudioOutput > 1) AudioOutput = 1;
  if (AudioOutput < -1) AudioOutput = -1;
  return AudioOutput;
```

1.1. Implementation the JUCE Framework

To run this synthesiser program, users need to download the JUCE Framework beforehand. JUCE is an open-source cross-platform C++ application framework to implement high-quality desktop and mobile applications including VST, VST3, AU, AUV3, RTAS, and AAX audio plug-ins.[6] JUCE can be easily integrated with existing projects through CMake or used as a project creation tool through Projucer. Projucer supports Xcode (macOS and iOS), Visual Studio, Android Studio, Code: Block and Linux Makefiles, and Source Code Editor. This program was written with the Visual Studio 2019 and C++17, however, users can export & compile the code that they preferred. JUCE Framework is available on the GitHub (https://github.com/juce-framework/JUCE)[51] and can download by using "git clone" command. Also, the following link will help those installation processes[52];
https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone
The following is a brief explanation of how to compile and open these JUCE file:

1. Go to the JUCE github site. (https://github.com/juce-framework/JUCE)
2. Clone the JUCE Repo to the specific location you want
3. Build/Open the Projucer file (JUCE' project management tool)
: Use the appropriate solution in "... /JUCE/extras/Projucer/Builds/ ..." [Figure 4]
4. Run the Projucer
5. Select the appropriate exporter & Build the solution
6. Run the standalone build or add the VST3 build to your VST3 folder to use it in your DAW

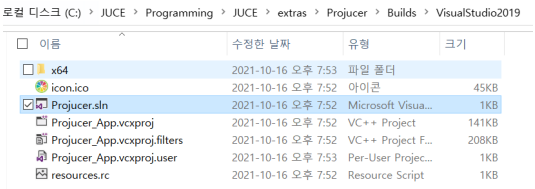

Figure 4: Open the Projucer file (1)

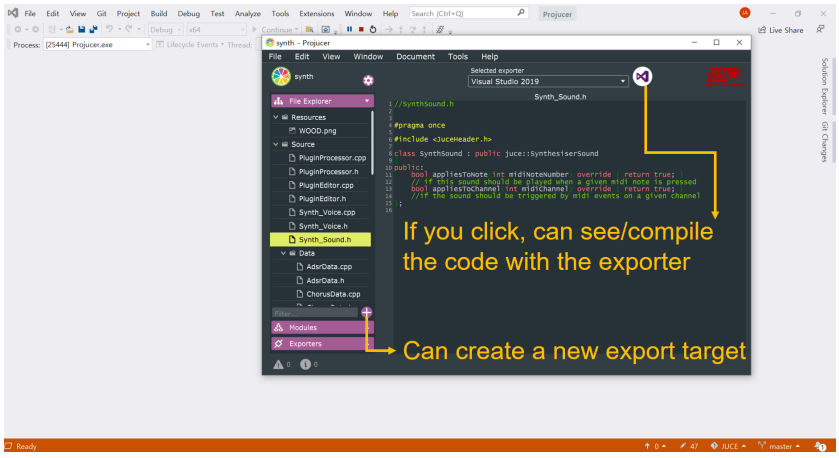

Figure 5: Open the Projucer file (2)

After compiling the 'Projucer' file, click the 'File' on the caption and open the 'synth.jucer' file. Then, just run/compile the 'synth.jucer' file's code. Finally, you can find the compiled .exe file in the following directory: *\Final_Year_Project\Builds\VisualStudio2019\x64\Debug\Standalone Plugin\synth.exe

2. Introduction

2.1. Aims

The aim of this project is to implement and design a fixed signal path subtractive synthesiser using C++ and the JUCE Framework[6].
Software synthesisers are typically plugin programs that conform to interface specifications, such as VST and AU[51], that are specific to sound playback in personal computing environments. It is a software reproduction of the sound of an air-vibrating instrument, either imitating a real instrument, or a software reproduction of the sound of a synthesiser that produces artificial electronic sound. There are several components used to create such sounds, including oscillators, effectors, and modulation sources. The use of such software synthesisers and homologous virtual instruments is rapidly changing the current trends in popular music. Many pop music artists are already using virtual instruments for most of their sound sources, away from the way performers used to play and record results, which will continue to be more frequent in terms of time and cost savings. This digital music technological development has enabled digital music production without the expensive equipment needed to produce records, expanding the creativity and diverse efforts of young musicians.(Sung, K. et al. (2019)[4]) As a result, the need for the advanced development of virtual instruments with functions that can be applied to various genres demanded by artists is being highlighted these days. Musical instruments are all products that make musical thinking and create sound. Therefore, an instrument must be able to express its characteristics well in various forms of music as a tool for expressing musical sounds. A synthesiser is an advanced tool that can meet that requirement relatively easily.
New forms of music derived from the development of computer music technology are culturally resonant. However, related technologies such as MIDI and VST are only accessed and used by experts in the music field or those who are particularly interested in sound synthesis. No matter how good the technology is, poor accessibility can be a fatal weakness. Therefore, this paper includes the process of implementing an "easy-to-use" synthesiser that is easily accessible to non-music experts. During the project period, I aimed to explore sound synthesis technology through proper research, understood and learned about sound synthesis engines, and finally designed and developed my own subtractive synthesiser that users can access right away after installing JUCE Framework properly.

Aims of the project:

- Create a subtractive synthesiser that composed fixed signal path
- Research the various music techniques & Investigation of Sound Synthesis
- Research and understand the component of synthesiser and the method to implement as a software; Oscillator, Filter, Amplifier, Modulation sources, Effectors
- Implement an easy-to-use subtractive synthesiser with JUCE Framework with a deep understanding of the JUCE libraries

3. Professional and Ethical Considerations

All projects should consider the professional & ethical considerations for sharing what I know, complying with standards, and always acting professionally and fairly. However, this is a highly practical project and there are not many ethical considerations related to or applied to the project. I have not involved participants in the project until the end of evaluating the quality of the project.

3.1. Public Interest

The project used a virtual graphical user interface that executes inside the computer, which does not affect the real world and real person. Therefore, there is no concern about the welfare or well-being of the general public and also basic human rights such as public health, privacy, and security.

3.2. Professional Competence and Integrity

This project topic was selected by me, and I found out the audio programming development using the JUCE framework is clearly possible with my project. I was able to successfully reach the general aims of the project by continuing to develop professional knowledge, skills, and competence on a continuing basis, maintaining awareness of technological developments. This is a personal project, plus, the JUCE Framework which is the fundamental source of the program is open source for everyone. Therefore, there is no legislation to comply with. In the process of implementing the program, I received appropriate feedback from the supervisor and respected and accept honest criticism. Also, I have not accepted any bribery or unethical inducement for this project.

3.3. Duty to the Profession

As a BCS member, I have tried to improve my undergraduate student-level expertise by participating in the development and avoiding any action which negatively affects the BCS Profession during the project period. I have tried to upgrade my professional knowledge and skill and shall maintain awareness of technological developments. I am also have been strived to promote public knowledge and understanding of computing and information systems and technologies in professional practice and to counter harmful false or misleading statements, which are detrimental to the Profession. In addition, as before, I have been encouraging each other's development with integrity and respect attitude toward peers and members who are in the informatics department.

3.4. Duty to the Relevant Authority

I have been to have had an 11 times personal meetings regularly with the supervisor, and during the meeting, discussed the development stage of the project. I have been working on my project according to the base plan that I built in advance with sufficient care and diligence. Instead of only relying solely on the feedback from the supervisor, I have been taken full responsibility for the project and controlled the entire stage of the work. These contents are briefly recorded in the meeting log and weekly log which is in **Appendix A. Progress logs** section (A.1 Weekly logs & A.2. Meeting logs) in this report. I have been always respected my supervisor's feedback and accepted it positively, and avoid conflict. Also, I have not used confidential information for the benefit of individuals or third parties, and I only delivered the correct information about the results of the project where any technical specifications be presented.

4. Background Knowledge

4.1. Overview of the synthesiser

A synthesiser is a musical instrument designed to produce sounds using an electronic oscillator. After all, it is necessary to analyze the elements of the sound in order to produce a synthesiser as it is a machine for making notes. Sound is caused by the movement or vibration of a particular object. The frequency of vibration or frequency of vibration repetition per second is called frequency. The period it takes for one cycle of a sound wave is the period, and the frequency is the inverse of the cycle. The vibration causes the pressure of the medium (typically, the medium is air) surrounding the object to change into the same pattern as the object's movement. Sound is caused by vibrations that exhibit the movement of a particular object that the form of a signal wave. In addition, most of these sounds are complex tones, synthesized into fundamental frequency and their harmonics. At this time, the sound energy distribution in each spectrum is different, even if the basic sound and the chord structure are the same, and the sound's timbre will be different.(Reiss, J. D. et al(2015)[1])
Thus, in order to create a synthesiser, a basic component is required that generates the basic sound such as the sine wave in which is called an oscillator (DCO, Digital Control Oscillator). Then, a subtractive synthesiser refines the sound by processing such components through various filters (DCF, Digital Control Filter) such as a LPF (low pass filter), effects, modulation sources, etc., to change the frequency of sound, or add effects to sound. In addition, there is a filter ADSR also exists which is not the same as an amplifier ADSR. This is a way in which a synthesiser using a basic subtractive synthesis creates sound.[11]
Finally, all sounds go through the following process over time after it is made from the oscillator, i.e., increasing the amplitude of the sound (attack), then decreasing slightly (decay), then keeping it constant(sustain), and finally releasing the amplitude(release) with Amplifier ADSR. That Envelope (DCA, Digital Control Amplifier) processes create each unique sound.
The below Figure 6(Sievers, B., (2007)[39]) is a diagram of the typical process of the synthesis of sound:

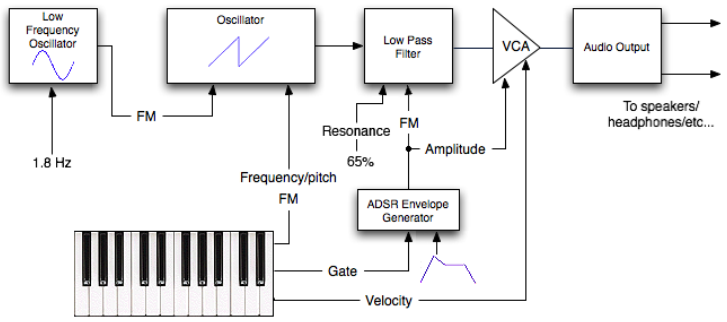

Figure 6: The progress of synthesiser synthesis sound

Synthesisers are designed with three types of components (Pirkle, W. C. (2015)[3]):

- sources: things that render an audio signal, such as oscillators and noise generators
- modifiers: things that alter the audio signal that the sources generate, such as filters and effects
- controllers: things that control parameters of the sources and/or modifiers; these might be fixed controls such as a knob on a MIDI controller or continuously variable controllers such as envelope generators

The synthesiser is usually defined vaguely as an electronic musical instrument that produces and controls sounds.[3] Following are the details of the basic components of the synthesiser in this project:

4.2. Basic components of the synthesiser

4.2.1. Oscillator (DCO)

Oscillators render audio with frequency/pitch value. This program uses several types of oscillators that synthesize various waveforms. The generating method of synthesiser has many options.
Below are the four types of method to generate oscillator(Pirkle, W. C. (2015)[3]):

- mathematically
- using the lookup 'wavetables'
- using the extracting audio samples contained in WAV files
- using the Frequency Modulation (FM) and Phase Modulation (PM)

In this program, the oscillator was implemented using a mathematical method. Each of the sound waves is calculated by mathematical equations. Those equations were written in the 6.4 section. (E.g. sine wave: AudioOutput = $sine(x)$)

4.2.2. Filters (DCF)

The filters in the synthesiser modify the frequency content of an audio signal, so that they are called modifiers.

- Low-Pass : attenuates high frequencies, can be resonant
- High-Pass : attenuates low frequencies, can be resonant
- Band-Pass : allows a band of frequencies to pass through by attenuating both low and high frequencies, may be resonant but usually is not
- Band-Stop: attenuates a band of frequencies by allowing both low and high frequencies to pass through, may be resonant but usually is not. (Band stops cannot "self-oscillate", and care must be taken to distinguish between resonance and self-oscillation.)

Here, the 'resonance' is not a filter type, but a property that most filters have. By default, users can increase or lower the volume of the frequency near the cutoff point, which is the "Q" parameter of the filter.[9]

4.2.3. Amplifiers (DCA)

The amplifier amplifies or attenuates the signal, therefore, it is also a modifier. The implemented DCA module controls the output amplitude in synth as well as panning function processing. The controller can be connected to modulate the amplitude and/or panning. The oscillator of the synthesiser feeds a filter. For DCA, the modulation is implied to be amplitude modulation. In JUCE, the envelope generator creates an envelope applicable to all types of modulation targets, such as an oscillator, filter cutoff frequency, or DCA amplitude. This envelope generator creates an amplitude curve divided into segments called 'attack', 'decay', 'sustain', and 'release'. This is why an envelope is typically called 'ADSR.' [19]

4.3. Basic Synthesiser Architecture

The following two figures(Sievers, B., (2007)[39]) are the basic flow diagram of the synthesiser. Figure 7 shows the most basic synthesiser architecture. When the user inputs the pitch value through a MIDI keyboard, the Oscillator output the sound. This is also the based model of the synthesiser, which reflected fallback objectives that were described in the next, the **5.2 Project Design: Objectives section.**

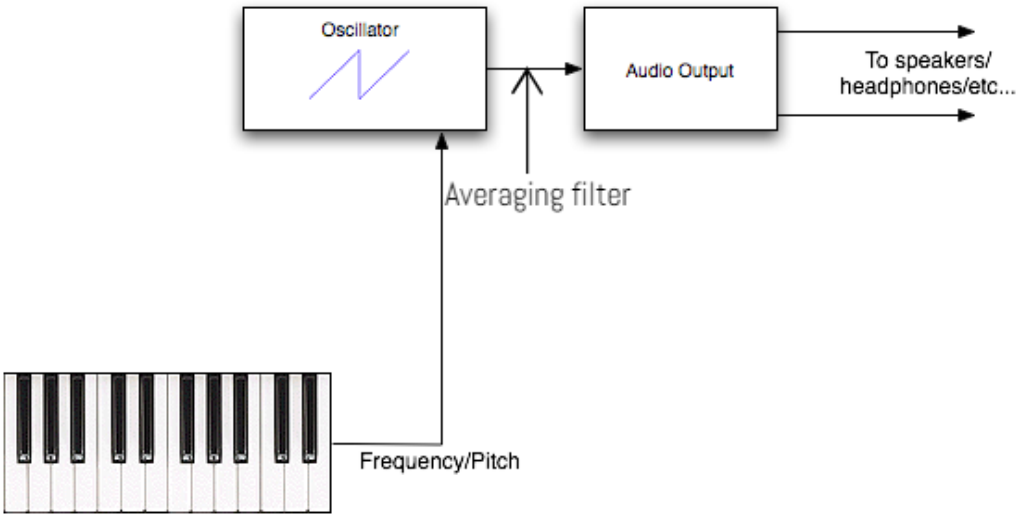

Figure 7: Diagram of a typical synth flow: Basic model - Reflect on the fallback objectives

Figure 8 shows the advanced version of the synthesiser architecture compared to Figure 7. When the user inputs the pitch/Frequency value through a MIDI keyboard, the Oscillator output the sound through filter and various modulation sources. This is also the model of the synthesiser, which reflected the core objectives that were described in the next, the **5.2 Project Design: Objectives section.**

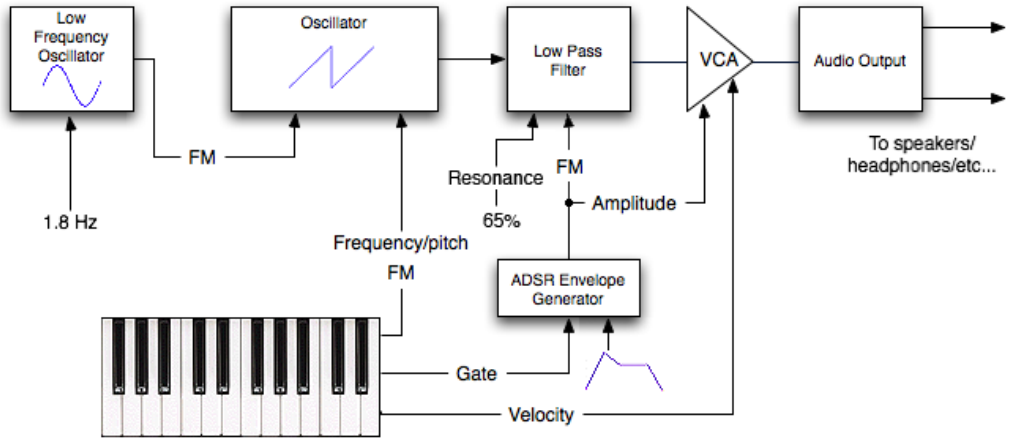

Figure 8: Diagram of a typical synth flow: Aim model - Reflect on the Core objectives

4.4. About JUCE Framework

This project uses JUCE Framework for implementing the whole synthesiser. JUCE is a GPL license library for writing C++ software, especially for creating audio applications. JUCE means 'Jules Utility Class Extensions', which was written by Julian Storer.
This is a framework for low-latency applications, with cross-platform GUI libraries to get the apps running on Mac OS X, Windows, Linux, iOS, and Android. JUCE is a tool in the Cross-Platform Desktop Development category of a tech stack. Also, is an open-source tool on GitHub and has an open-source repository as referred to above.

4.4.1. JUCE Framework's features

1. Audio & plug-ins
2. Interactive User Interface & Graphics
3. For desktop & mobile, implementing powerful and complex applications

At the beginning of implementation, numerous open-source cross-platform libraries were considered, including Qt, GTK+, FLTK, wxWidgets, openFrameworks, GLUT, and Clutter. I selected to use JUCE as a tool for the project for the advantages stated in the next section.[39]

4.4.2. JUCE Framework's advantages

1. JUCE is completely open-source, allowing users to adjust their libraries as needed.
2. There is no need to worry about related laws and regulations because there is no license restriction for non-commercial applications.
3. Intuitive class structure based on components.
4. JUCE is a completely cross-platform, therefore, users can compile the same code for Linux, Windows, and Mac
5. Has a lightweight compared to other frameworks, you can include the source code in the GitHub repository easily.
6. Has a great documentation: (https://docs.juce.com/master/classes.html)
7. Has a helpful forum where the JUCE's users can get help from Jules himself: (https://forum.juce.com/)
8. Since the JUCE library was originally created for audio processing applications, it contains a well-developed class for manipulating audio data (E.g. MIDIInput Class[30])
9. Has many useful tutorials for starting: (https://juce.com/learn/tutorials) exist

4.4.3. JUCE Framework's disadvantages

- It is not as widely adopted compared with Qt, which means users purchase JUCE books or find a wide range of online cases.

  - However, JUCE's online documents and forums were sufficient for learning and problem-solving.

4.4.4. Comparing the JUCE with Alternatives & Comparisons other applications

- Qt : Qt is the leading cross-platform application and UI framework. This framework allows users to implement applications once and distribute them to the best desktop, embedded, and mobile targets.(Viktor (2017)[43])

  - Qt and JUCE are primarily classified as "Cross-Platform Mobile Development" and "Cross-Platform Desktop Development" tools respectively. This program is running on the desktop so the JUCE framework is selected.

- Faust : This is a stream processing library that ports ideas from Kafka stream to Python. It allows users both stream processing and event processing, sharing similarities with tools such as Kafka Stream and Apache Spark/Storm/Samja/Flink.(Orlarey et al. (2009)[41])

  - Faust has many fancy framework tools related to DSP same as the JUCE framework, however, I felt that the JUCE framework's tutorials series are more systematic and easy to refer to when I implemented the synthesiser, so I select the JUCE.

- T3 : T3 is a little different from most JavaScript frameworks. It means a small part of the overall architecture that can create scalable client-side code. T3 is clearly not an MVC framework. It is a framework that allows users to create loosely-coupled components while determining other parts required for a web application. Users can use T3 with other frameworks such as Backbone or React, or users can use T3 alone.[40]

  - T3 is the JavaScript framework in general, and I am more familiar with C++, so I selected the JUCE Framework rather than T3.

- React Native: React Native allows users to build a world-class application experience on a native platform using a consistent JavaScript and React-based developer experience. React Native's focus is on the developer efficiency of all platforms of interest (one learning and can be used anywhere). Facebook uses React Native on several production apps and will continue to invest in React Native.(Marcak, M. (2021) In addition, React is more of a front-end technology for UI development.)[42]

  - React Native Seed is part of the "Cross-Platform Mobile Development" category of the tech stack, while JUCE can be primarily classified under "Cross-Platform Desktop Development". I choose the working environment as desktop so I select the JUCE similar to the Qt case.

4.4.5. JUCE key classes

- AudioSampleBuffer - It is used to shuttle continuous data through a processorGraph. Hold the float array of the floats of size channels x samples.
- MIDIBuffer - It is used to shuttle event data through ProcessorGraph. Store discrete events with timestamps and several bytes of data. In the JUCE source code of this repository, the MdIBuffer class is modified so that each event can hold arbitrary number of bytes. This is because spike events can be transmitted from the processor to the processor through the MIDIBuffer.
- Component - Almost everything a user can interact with is a component. Components store the other components as children, that inform their parents when they receiving user input such as clicks. (E.g. UI Component)
- Graphics - It is used to draw within Components(functions of the program).

4.4.6. The reasons of selecting the JUCE as a tool for the project

- JUCE Framework is a compatible application and easy to access. It supports various development environments such as PC, Mac, and Linux.
- It is cross-Platform, so easy to convert the development environment (cross-platform plugin framework) through 'Exporters'. (Can create a export target: Xcode(macOS/IOS), Visual Studio Code(2015/2017/2019), Linux Makefile, Android, Code::Blocks(Windows/Linux), CLion)
- JUCE Framework suits for developing the audio program since it provides the development of various plugin formats: VST, AU, AAX.
- The JUCE Framework makes it easy for users to create a versatile UI that can run on any platform and integrate OpenGL (2D rendering engine, font functionality, image formatting). It also expands to all screen sizes.
- JUCE Framework provides host VST (AudioPluginHost) and AU plugins inside to the own applications.
- C++ is a default development language in JUCE and it is a familiar language for me.
- JUCE is open-source and does not have related law&legality so that it helps me to do not violate professional competence and integrity.

The JUCE Framework was selected for my synthesiser implement tool because it is easy to access and can be supported on various platforms and is also compatible with various development environments. Therefore, making it easy for other users to use the invented program. This is in line with the project's aim of developing a synthesiser that is easy to use for the general public and students who are not experts also.

4.5. Example synthesiser application is implemented with JUCE

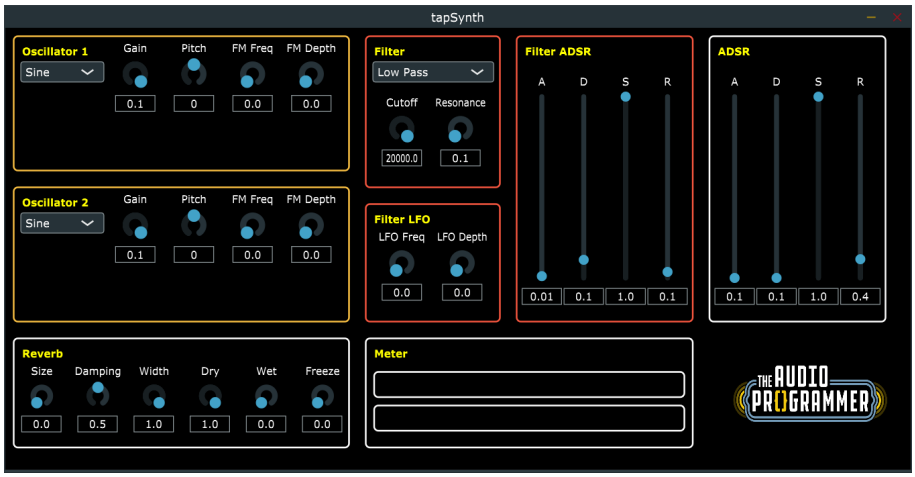

Figure 9: Sample synth program on the Github

The above snapshot (Figure 9) is a simple synthesiser application using the JUCE framework found in Github (https://github.com/TheAudioProgrammer/tapSynth). This **VST** file has two oscillators, LFO, Filter, Envelope, Effector (Reverb), and a meter. In addition, all components were implemented on the screen through a simple GUI. I implemented a **standalone** synthesiser by adding similar but more detailed effectors, functions, and a MIDI keyboard.

4.6. Requirement Tasks of the project from Background Knowledge

- Recording the project progressed
  - Writing & submit the project report
- Need to code for program
  - Implement the Oscillator - several waveforms
  - Implement the Amplifier; Envelope - Attack, Decay, Sustain, Release
  - Implement the Filter - Low Pass Filter, High Pass Filter, Band Pass Filter
  - Implement the various Effects - Phaser, Compressor, Chorus, Reverb
  - Implement the various Modulation source (E.g. FM source)
  - Implement the meter&virtual keyboard (only for standalone synthesiser, not VST)
  - Design the GUI interface - Background, Knob for each function etc.
- Evaluate & Test the program
  - Testing and Evaluating the whole synthesiser software
  - User & System testing

5. Project Design

5.1. Planning Phase

5.1.1. Project plan #1

During the project progress, it was necessary to plan the process until completing the individual project and clearly state the results of the plan. Among the various charts that manage schedules, the PERT chart has the advantage of being easy to use when it is difficult to predict the time required and also has the merit of being able to evaluate and review all programming. Therefore, I decided to create the PERT chart in the planning stage of the project. The creation of the PERT chart became a good start to make more efficient use of time resources until May 10, 2022, the deadline for individual project. Although the completion time (subjective expectations arbitrarily predicted) used to create a PERT chart is not absolute, developing a PERT chart has had sufficient advantages in projecting at least predicting approximate completion time and managing the entire process in working. The result and evaluation of the plan established were discussed in the evaluation section.
Below is the PERT chart (Figure 10) and descriptions of chart (Table 1) that was created on the planning process:

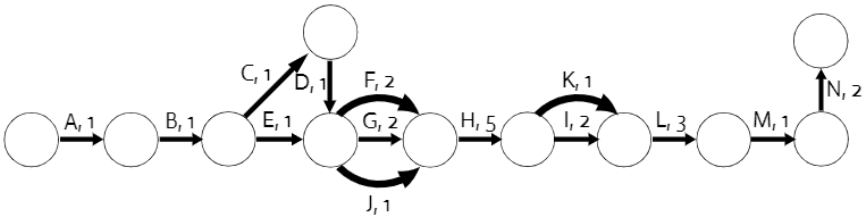

Figure 10: Project Plan: PERT Chart

Task	Description
A	Meet with project supervisor and discuss about project
B	Researching about synthesiser
C	Plan out tasks for implementing synthesiser by creating a PERT chart
D	Carry out a risk management
E	Design first prototype
F	Unit testing
G	Implement OSC & LPF (necessary elements)
H	Implement more various/detailed functions for synthesiser
I	Design GUI for each functions
J	Design Background Interface
K	System testing (User Test)
L	Finish implementing synthesiser
M	Evaluate the project
N	Submit project with review a of final project report

Table 1: Project Plan : PERT Chart Descriptions and Calculations

Tasks	Weeks				Critical Path
	Earliest Start	Earliset Finish	Latest Start	Latest Finish	
A	0	1	0	1	**YES**
B	1	2	1	2	**YES**
C	2	3	3	4	**NO**
D	3	4	4	5	**NO**
E	2	3	2	3	**YES**
F	3	5	5	7	**NO**
G	3	5	3	5	**YES**
H	5	10	5	10	**YES**
I	10	12	10	12	**YES**
J	3	4	5	6	**NO**
K	10	11	16	17	**NO**
L	12	15	12	15	**YES**
M	15	16	15	16	**YES**
N	16	18	16	18	**YES**

Critical Path: A, B, E, G, H, I, L, M, N

- All tasks must be completed by 19th April, 2022 for feedback. (Assignment due date 10th May, 2022).
- The time planner above is not absolute, should be managed by flexible way.

5.1.2. Detailed explanation of the plans

- A; Meet with project supervisor and discuss about project (1 week)

  - Should have an individual meeting with a supervisor to be more properly guided about my tasks, receive guidance on what I need to do in the future, and prepare for project planning.

- B; Researching about synthesiser (1 week)

  - Research synthesisers and various functions (E.g. effectors) related to them by reflecting on the discussions with the supervisor in 'First Meeting'

  - The type of synthesiser and the program to be used should be reflected in the 'Project Proposal Document'.

- C; Plan out tasks for implementing synthesiser by creating a PERT chart (1 week)

  - Consider the necessary steps in implementing a program by designing a PERT chart, and estimate the time it takes for each step. (Even if it is not accurately estimated, it becomes a meaningful challenge.)

  - The created PERT chart should be reflected in the 'Interim Report'

- D; Carry out a risk management (1 week)

  - Follow the following steps to carry out the task, organize the following problems, and come up with alternatives.

  - Should be organized with 'Risk identification', 'Mitigation'.

- E; Design first prototype (1 week)

  - Develop the first prototype of the subtractive synthesiser for brief guidance.

- F; Unit testing (2 weeks)

  - In the completed programming, verify that a particular module in the source code works exactly as intended.

  - If there is an error, it should be corrected before moving on.

- G; Implement OSC & LPF (necessary elements) (2 weeks)

  - Various waveforms should be implemented and LPF that essential elements in the subtractive synthesis method should be developed also.

- H; Implement more various/detailed functions for synthesiser (5 weeks)

  - Based on the data obtained through research, the various functions (E.g. Effectors) of synthesiser should be added for more completeness.

- I; Design GUI for each functions (2 weeks)

  - Each of the developed functions should be designed using various GUI components provided by JUCE Framework. (LookAndFeel class)

- J; Design Background Interface (1 week)

  - The background should be suited for the implemented functions.

- K; System testing (User Test) (1 week)

  - Overall, the implemented functions should be inspected and confirmed.

  - Investigate the overall completeness and satisfaction of synthesiser from other students.

- L; Finish implementing synthesiser (3 weeks)

  - Integrate the implemented functions, check the objectives list, and complete all coding required for synthesiser design.

  - Supplement the synthesiser by reflecting the result of the system test & user test (Task K)

- M; Evaluate the project (1 week)

  - Evaluate whether the project progressed smoothly compared to the initial plan by myself.

  - In the same way as the user test, the synthesiser program should be evaluated the satisfaction and completeness by myself.

- N; Submit project with a review of final project report (2 weeks)

  - Submit the 'final report' and source code written while working on the project to Canvas by the time.

  - The due date of the final report is 10th May 2022, and the overall task must be completed by 19th April 2022 for feedback.

5.1.3. Project Plan #2

After working on the project for several weeks, I found that the details of the PERT chart should be updated based on biweekly for efficiency. Allocating the time based on the meeting with the supervisor, it was easy to immediately reflect the feedback and useful advice obtained from the meeting in the software and make it simple to modify the program since it was not in a more advanced state.

Task	Description	day
A'	Receive the feedback from supersvisor and reflect to the software	3 day
B'	Research about function	2 day
C'	code the functional task (Data folder)	5 day
D'	code the non-functional task (GUI)(UI folder)	1 day
E'	Test the created function & write the short weekly log	1 day
F'	Risk management & Update the information on the final report	2 day
-	-	14 days

Figure 11: Detailed Project Plan Descriptions and Calculations

Task	Description	week
A	Meet with project supervisor and discuss about project	0.5 week
B	Researching and Selecting about synthesiser & effect unit	0.5 week
C	Plan out tasks for creating synthesiser by creating a PERT	0.5 week
D	Design first prototype	0.5 week
E	Implement Function of software - OSC, Envelope	2 week
F	Implement Function of software - Modulation	2 week
G	Implement Function of software - Filter	4 week
H	Implement Function of software - Effects	4 week
I	Revision&Supplement Function of software (Before move on standalone synthesiser)	4 week
J	Implement Meter&Virtual keyboard (only for standalone version of synthesiser)	2 week
K	Design GUI for each functions	0.5 week
L	Design Background Interface	0.5 week
M	System testing and modify the error	2 week
N	Submit completed project with final project report	-
-	-	23 weeks

Figure 12: Updated Project Plan : Project Plan Descriptions and Calculations

Figure 11 was the detailed schedule plan for the code&implement stage, for task E, F, G, H, I, J on Figure 12. In particular, in the case of task G, H, and I, the number of related functions varies, and therefore, the period was planned to be longer (4 weeks) since there was a high possibility of adding and developing more functions according to the result of research. The time planned for the other development function step was all two weeks, and the time for two weeks should be managed according to Figure 11, as referred above.
The detailed work of risk management and unit&programming testing was also performed during the assigned two weeks of the function implementation period, and nothing has changed in the overall amount of tasks to be worked on.
The total time required for programming development was 23 weeks in the plan, and the plan was likely to change since various unpredictable situations such as vacations & other test periods

should often occur during the semester. However, this became a good guide for dealing with constrained time.

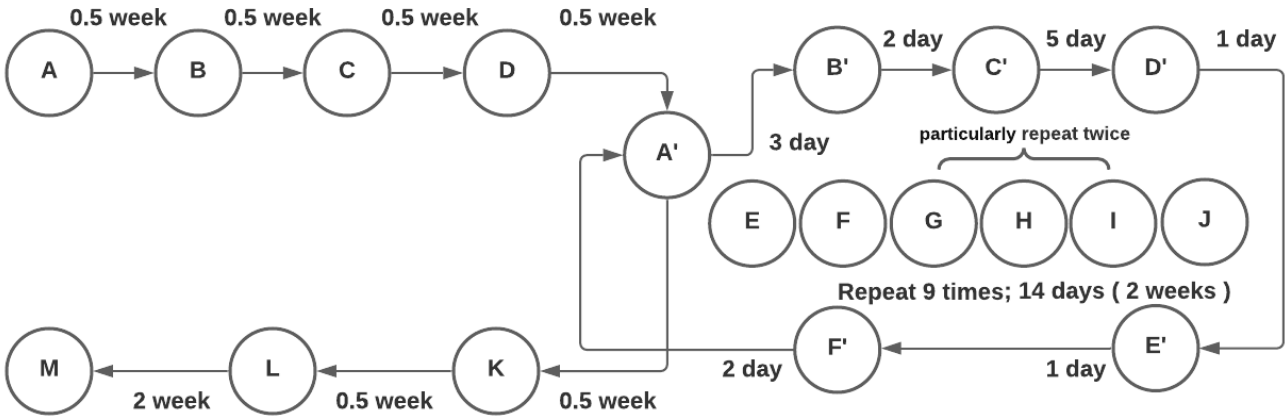

Figure 13: Supplemented PERT Chart

Above Figure 13 was the updated version of the PERT chart that contains information on how the planned time to complete the synthesiser project was allocated within the constrained time.

5.1.4. Gantt Chart for project plan

The table(Figure 14) and graph(Figure 15) below are brief data on the project plan described above. It is easy to see at a glance the planned time.

SYNTHESISER PROJECT	START DATE	Time Estimates in Day
A	2021-09-27	3.5
B	2021-10-01	3.5
C	2021-10-04	3.5
D	2021-10-08	3.5
E	2021-10-22	14
F	2021-11-05	14
G	2021-12-05	28
H	2022-01-05	28
I	2022-02-05	28
J	2022-03-05	14
K	2022-03-09	3.5
L	2022-03-12	3.5
M	2022-03-26	14
N	*	*

Figure 14: Table of Gantt Chart for Project Plan

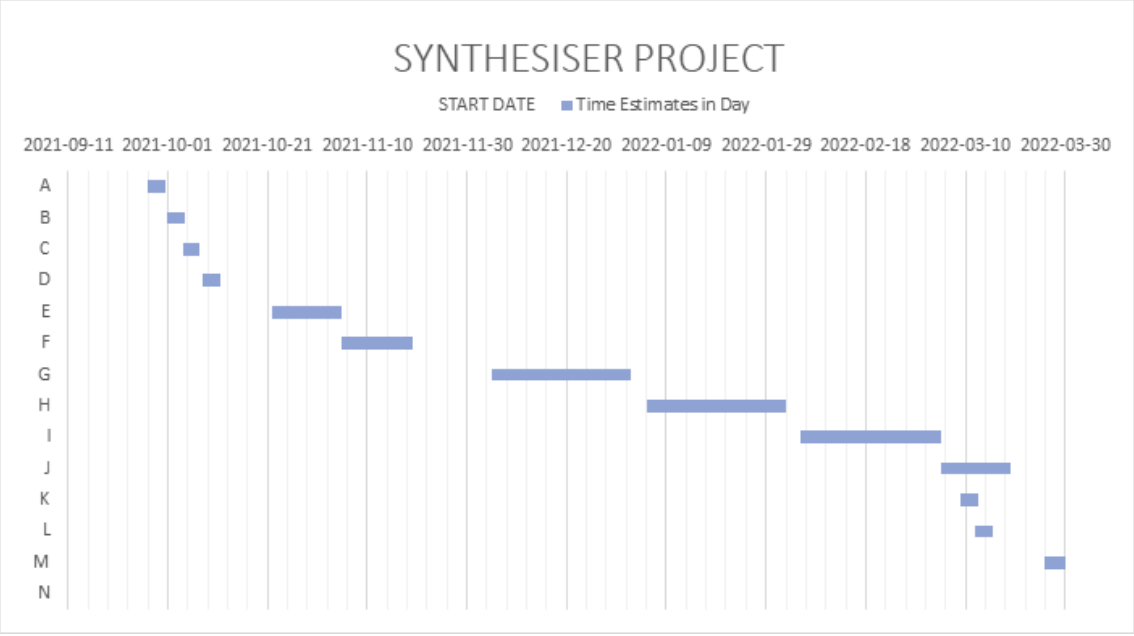

Figure 15: Graph of Gantt Chart for Project Plan

5.1.5. Gantt Chart for project progress result

The table(Figure 16) and graph(Figure 17) below show the project progress result. It is easy to see at a glance the time consuming of the project.
Overall, the task was carried out according to the planned time smoothly, and the work was completed 5 days earlier than the expected completion schedule of the M task.

SYNTHESISER PROJECT	START DATE(YY.MM.DD)	CONSUMING DATE
A	2021-09-27	5
B	2021-10-01	4
C	2021-10-04	8
D	2021-10-11	5
E	2021-10-15	9
F	2021-10-23	24
G	2021-11-15	29
H	2021-12-13	22
I	2022-01-03	39
J	2022-02-10	33
K	2022-03-14	5
L	2022-03-18	4
M	2022-03-21	51
N	2022-05-10	*

Figure 16: Table of Gantt Chart for Project Progress Result

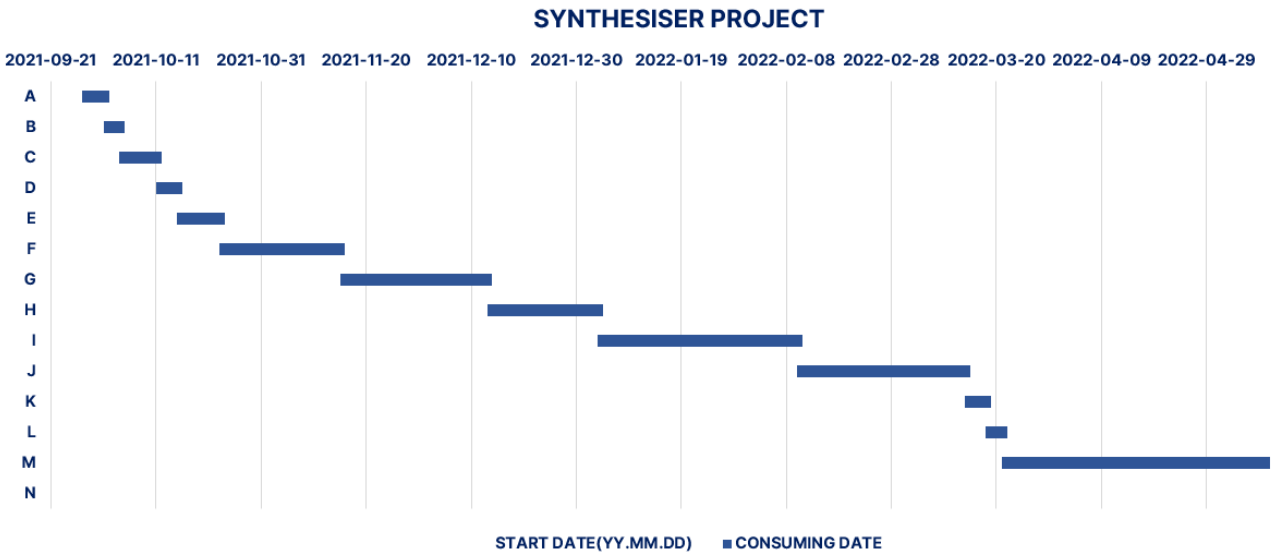

Figure 17: Graph of Gantt Chart for Project Progress Result

5.2. Objectives

5.2.1. Core Objectives

The following five core objectives below were selected, mainly for the elements that are essential when developing synthesisers as goals that should be completed first as the project progresses, set in the planning stage.

1. Research and understand the functions and characteristics of subtractive synthesisers that distinguish them from other types of synthesisers and design & implement the program using C++ & JUCE Framework
2. Choosing various options from oscillators, filters, amplifiers, and modulation sources that subtractive synthesisers should have as functions for implementation.
3. Allow the user to play the unique sound by just using a mouse & keyboard through the AudioPluginHost app so that it can be used as a desktop-based VST application plugin.
4. Implement a interactive interface(GUI) that allows a user can handle the synthesiser's function with knobs to play sounds described in Core Objective 2.
5. Provide a sound wave graph and show it through the interface for analyzing the waveform.
6. Create subtractive synthesiser with polyphonic capability.

5.2.2. Extended Objectives

The following extended six objectives below after completed core objectives (5.2.1), these are the lists that need to be researched a little more and implemented. Although not essential, the completeness of the program can be further improved when achieved.

1. Working with a MIDI interface.
2. Provide a standalone version of the subtractive synthesiser. (virtual keyboard and oscilloscope/meter functions should be added with extra work)
3. Provide recording & playback functions.
4. Provide Digital Audio Workstation (DAW) capability that allows users to execute MIDI sequencer/MIDI editing functions; audio recording, audio editing, MIDI editing, mixing, mastering, etc.
5. Implementing various Effectors; Reverb, Chorus, Compressor, Distortion, Delay, Phaser(Reiss, J. D. et al(2015)[1] etc & develop a filter with variable Resonance

5.2.3. Fallback Objectives

The following fallback four objectives below are the minimum achievement conditions in performing the task. If it is too much to achieve extended objectives and core objectives, it is a list that specified a minimum viable deliverable.

1. Implement and design the simplest possible VST plugin synthesiser.
2. Implementing at least a monophonic systems with an averaging filter which is a simple way to design a low pass algorithm.
3. Design a simple synthesiser using basic GUI sources provided from JUCE.

5.3. Project Relevance

The project requires clear comprehension of Fourier formulas used to generate a sound waveform, file formats of VST/Standalone, software engineering, GUI development, understanding of DSP, and time-management ability to implement desktop-based applications with C++ within a fixed time. Therefore, this works requires me to develop my own computing skills while designing the subtractive synthesiser, and progress various capabilities such as planning, development, documentation, testing, and evaluation of software systems.

Skills can be developed:

- Computing & Software Engineering skills
- Math
- GUI Design skills
- Time-management skills

Related skills to the project:

- Programming & Software Engineering
- Math
- Music technology
- Digital Signal Processing (DSP)

5.4. Resources Required

- The desktop that becomes a development environment should be able to install the JUCE Framework.
- A "Projucer" in which is a project management tool for implementing a VST3 plugin needed[8].
- Access to C++ and compilers is required, and an environment for these tasks should be established.

5.5. Requirements analysis

5.5.1. Functional Requirements

- F1 - Mandatory
  : The synthesiser shall have at least one oscillator with a choice of 4 waveforms: Sine, Saw, Square, Triangle. The oscillator shall receive the pitch/gain value from the users for rendering.

- F2 - Mandatory
  : The synthesiser shall have at least a low pass filter that is able to manipulate the cutoff frequency (Hz) and resonance (Q; the value of gain at the cutoff frequency – no units). The cutoff frequency value shall be adjusted between in 20-20,000Hz (Audible Frequency).

- F3 - Desirable
  : The synthesiser should have a low-frequency oscillator (LFO) module for producing a vibrato effect(Gordon (2000)[23]) by causing the oscillator's pitch to swing slightly up and down.

- F4 - Mandatory
  : The synthesiser shall have at least one effector (E.g: Reverb (Chrissy Tignor (2021)[49])) to generate a unique sound.

- F5 - Mandatory
  : The synthesiser shall have the envelop(filter ADSR and Amplifier ADSR) for making all kinds of changes to the overall sound.

- F6 - Mandatory
  : The synthesiser shall be implemented in two types: VST and Standalone.

- F7 - Mandatory
  : The standalone version synthesiser shall have the virtual keyboard to receive the MIDI-Input.

- F8 - Desirable
  : The program should have the proper oscilloscope (meter) to allow the users to confirm the exact sound waveform.

- F9 - Desirable
  : By using a "reset" function, users should be able to return knob(s) to the default state. The default state of the program should be that of showing a pure sine wave on the oscilloscope. (Waveform(OSC): sine, Resonance value: minimum value, Cutoff value(Filter): 20,000Hz for preventing cutoff a specific frequency)

- F10 - Desirable
  : The synthesiser should have a save/load function to store the specific state of the program or load the stored state file thereafter to allow the user to use the desired sound again.

- F11 - Desirable
  : The synthesiser should develop with a polyphonic system with 5 voices.

- F12 - Mandatory
  : The software representation shall pop-up on the screen smoothly such that the user can manipulate the sound. Also, the screen shall be resizable so that it can be the right size for each user's window.

- F13 - Mandatory
  : The knobs of the functions that contribute to the generation of each note shall be placed with an appropriate label so that the user knows what functions the knobs are for.

- F14 - Mandatory
  : The numerical value from each function through knob is adjusted shall be visible to the user. (E.g. Gain: 0.3, Freq: 800Hz)

5.5.2. Non-Functional Requirements

- N1 - Mandatory
  : The software shall be written in the C++ programming language. C++ is widely supported, which allows for maximum portability shall the software be extended.

- N2 - Desirable
  : The software should not crash or have lag whilst a session is in progress and unexpected noise should not be present in the output.

- N3 - Mandatory
  : For a description of the installation of the program to the users well organized README.txt file shall be included in the program files.

- N4 - Desirable
  : The synthesiser should have a help feature that allows the user to briefly know how to use or function the program. (E.g. a corresponding button opening a pop-up Help window)

- N5 - Mandatory
  : The virtual keyboard and functions of each knob shall be able to respond to keyboard-/mouse input for adjusting functions values.

5.6. Description of implemented functions in program

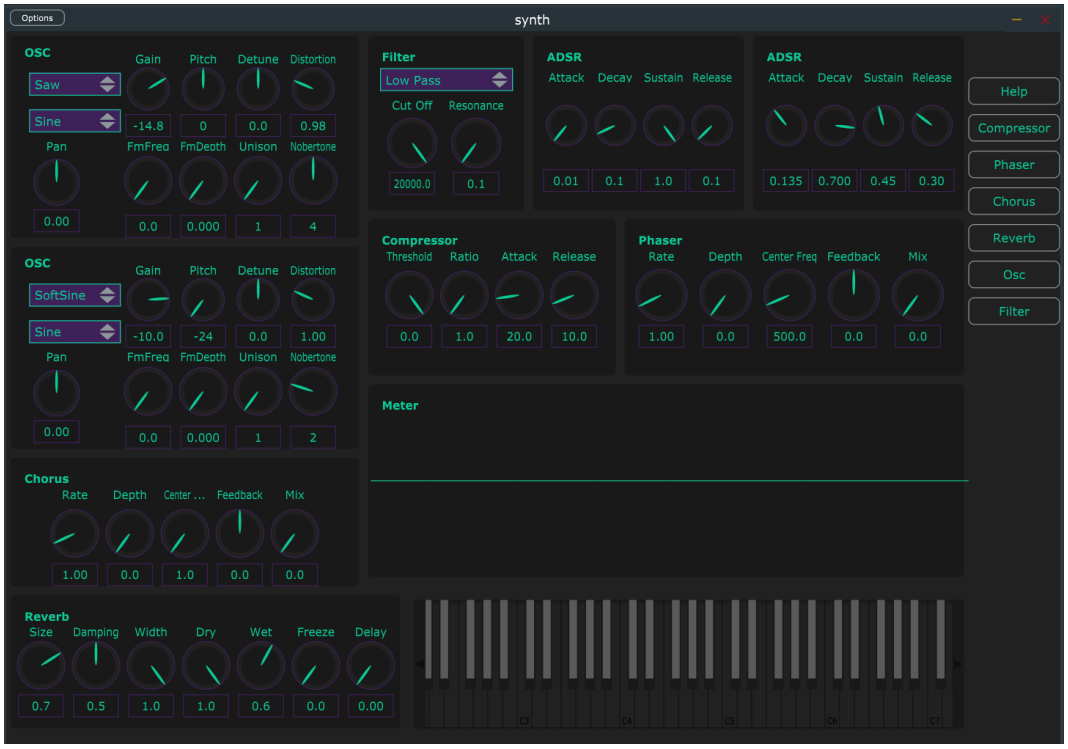

Figure 18: Snapshot of the synthesiser program

Above Figure 18 is the snapshot of the final version of the synthesiser program implemented in the project.

5.6.1. Oscillator

Two oscillators of the synthesiser are implemented with a choice of nine waveforms such as sine wave, square wave, saw wave, triangle wave, white noise, saw smooth wave, R-saw wave, softsine wave, and hardsine wave. Each waveform is defined by a mathematical method. In addition, the volume of the sound is controlled by the gain (amplitude) and the pitch of the sound also can be controlled by pitch (frequency) knobs. Plus, detune, distortion, FmFreq, FmDepth, Unison, and Nobertone functions are implemented in the Oscillator part.

- OSC1: sine wave, square wave, saw wave, R-saw wave, triangle wave, white noise, saw smooth wave, softsine wave, hardsine wave
- OSC2: sine wave, square wave, saw wave, R-saw wave, triangle wave, white noise, saw smooth wave, softsine wave, hardsine wave

5.6.2. Filter

This filter of the synthesiser is implemented with the three filters such as low-pass filter, band-pass filter, and high-pass filter. The cut-off frequency value of each filter can be controlled by the user and users are also able to gain the filter by handling the resonance (Q value) of the sound.

- Low-pass Filter (Cut-off frequency, resonance)

- High-pass Filter (Cut-off frequency, resonance)
- Band-pass Filter (Cut-off frequency, resonance)

5.6.3. Envelope

This envelope of the synthesiser is implemented with the four sections such as attack, decay, sustain, and release. There are two ADSRs that exist in the program, one(right) is for ADSR for the whole sound (Amp ADSR) and the other one(left) is just for a filter (Filter ADSR).

- Attack
- Decay
- Sustain
- Release

5.6.4. Effector

The different four effectors implemented in this program.

- Compressor (Threshold, Ratio, attach, release)
- Chorus (Rate, Depth, Center Frequency, Feedback, Mix)
- Phaser (Rate, Depth, Center Frequency, Feedback, Mix)
- Reverb (Size, Damping, Width, Dry, Wet, Freeze, Delay)

5.6.5. Standalone synthesiser: Virtual Keyboard & Meter

This synthesiser provides the Meter function (=oscilloscope) for checking the current waveform of the sound. It is useful for confirming whether the function is working well or not by checking the waveform directly. While controlling the knob responsible for each function, the user can see the changing graph and see the effect of each function on sound with the user's own eyes, and this Meter information is updated in the program every time the user plays sound in real-time. Also, the virtual keyboard is set on the screen for providing the MIDI note value to the program. This keyboard feature is implemented only in the standalone version of the program. In the case of the VST version, there are virtual MIDIkeyboard provided on the screen of the AudioPluginHost (MIDI Input plugin).

- Virtual Keyboard (MIDI keyboard component)
- Meter

5.6.6. State manipulation button

With this function, the user can save the state of the synthesiser as a whole when the desired sound is produced and load it to use again later. Resetting the state function is also implemented.

- Save
- Load
- Reset

5.7. Class Diagrams of the program

The following two figures: Figure 19, Figure 20 are the class diagram of the synthesiser. There are two different class diagrams because there are two different folders that contain each class in the source code: Data folder/UI folder.
The Figure 19 is the class diagram for implementing GUI (store in UI folder), and the Figure 20 is the class diagram for implementing synth's functional Data (store in Data folder) and files for Audio processing (not located in UI/Data folder).

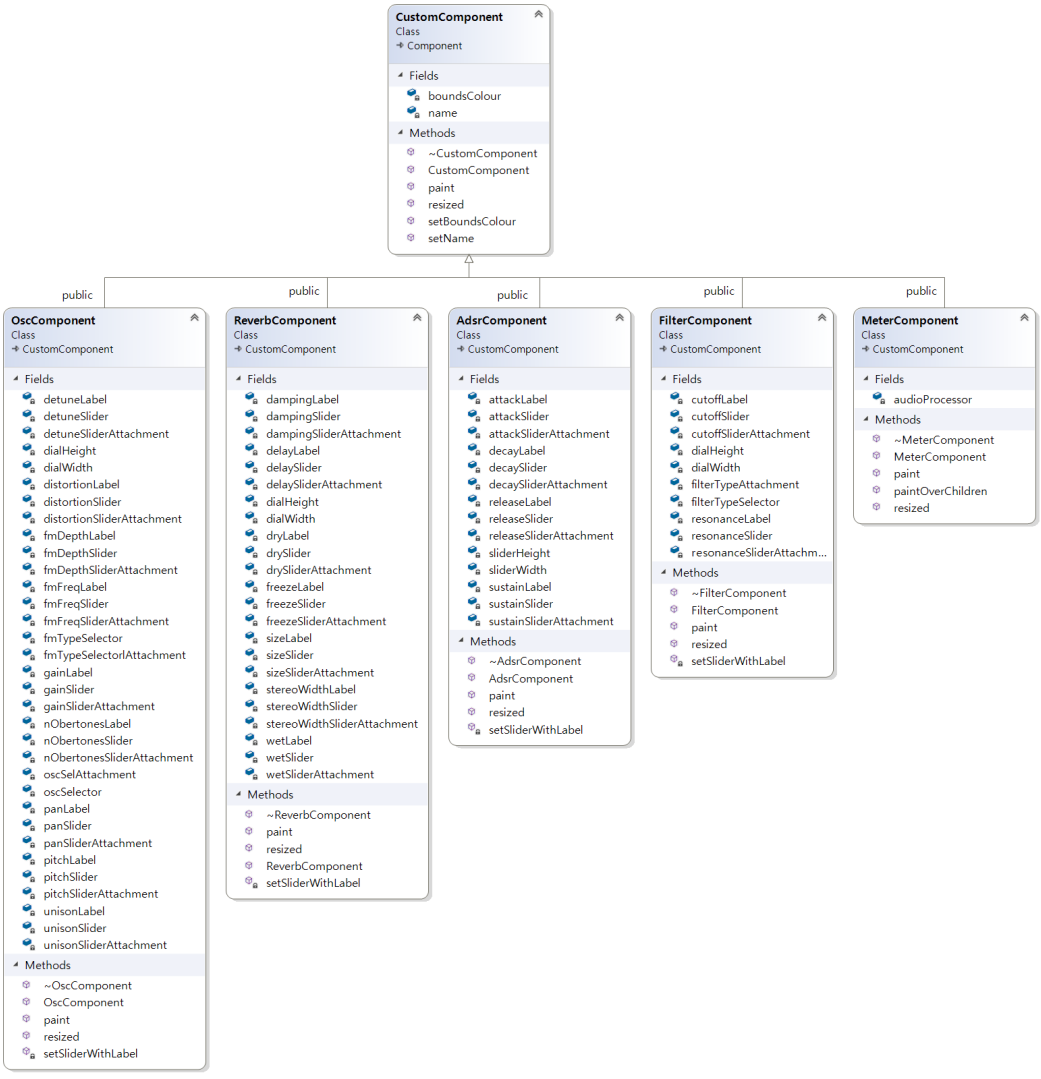

Figure 19: Figure of Class Diagram (include in UI folder) 1

Class Diagrams of the program

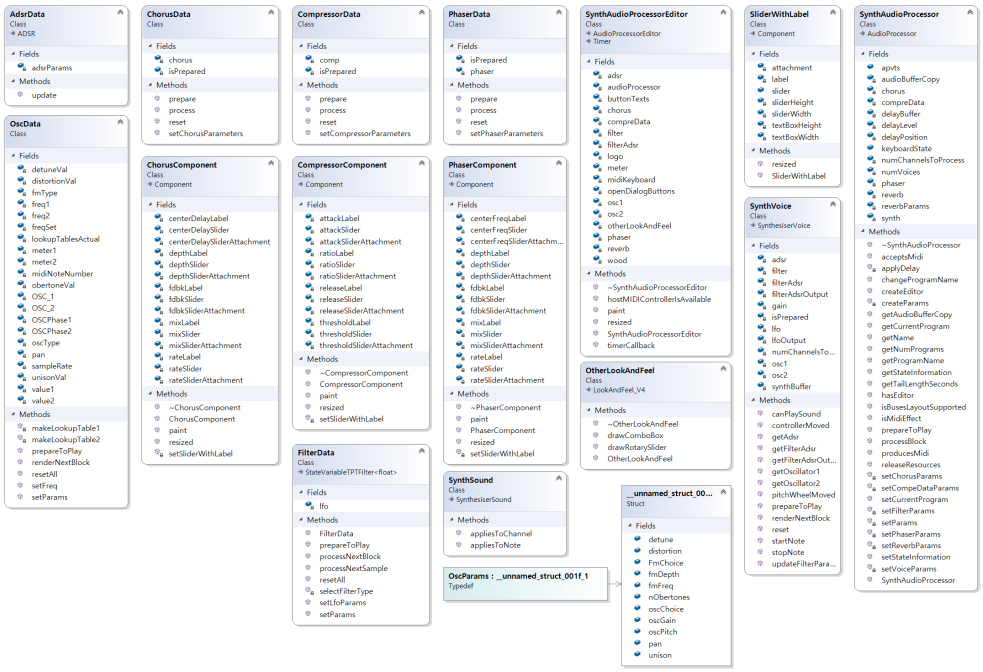

Figure 20: Figure of Class Diagram (include in Data folder and Others) 2

- Functional elements are stored in each function of the data .cpp/.h file located in the Data folder & GUI elements are stored in each function of component .cpp/.h file located in the UI folder
- Linking GUI components with the audio processor through 'AudioProcessorValueTreeState' (=apvts, defined in SynthAudioProcessor)
- Process of sending sound out to the user working on the audio buffer.
- The value of each function can be modified through sliders and combo boxes implemented through UI component files.
- The two classes PluginProcessor and PluginEditor provided by the JUCE framework are used to deal with Audio Processing and UI, respectively.
- The PluginProcessor has a juce::synthesiser member with SynthVoice instances. Each instance can generate audio through its own two oscillators (including switchable waveforms) and process manipulations through frequency modulation and filter sections.
- Synthesisers that can be created through the JUCE require an instance of SynthSound.
- The PluginProcessor also has the following JUCE instances.

  juce::AudioProcessorValueTreeState (=apvts), which operates as an interface between UI and audio processing.
- Functions that update different parameters are configured in the class of the Data folder.
- Plus, PluginEditor includes instances of all UI elements. It is organized in a class of UI folders except for on-screen keyboard components that require little setting compared to other elements.

6. Implement Project

6.1. Standalone & VST synthesiser

The synthesiser program was implemented for this project in two types: 1) Standalone 2) VST. Users can easily play with the standalone synthesiser by just opening and compiling the 'synth.jucer' file or executing the 'synth.exe' file. However, in the case of the VST synthesiser (synth.vst) should open and use with the 'AudioPluginHost.jucer' file. Users can the brief steps of building the basic plugin in the JUCE tutorial, which is on the following link:
Tutorial: Create a basic Audio/MIDI plugin;
(https://docs.juce.com/master/tutorial_create_projucer_basic_plugin.html)

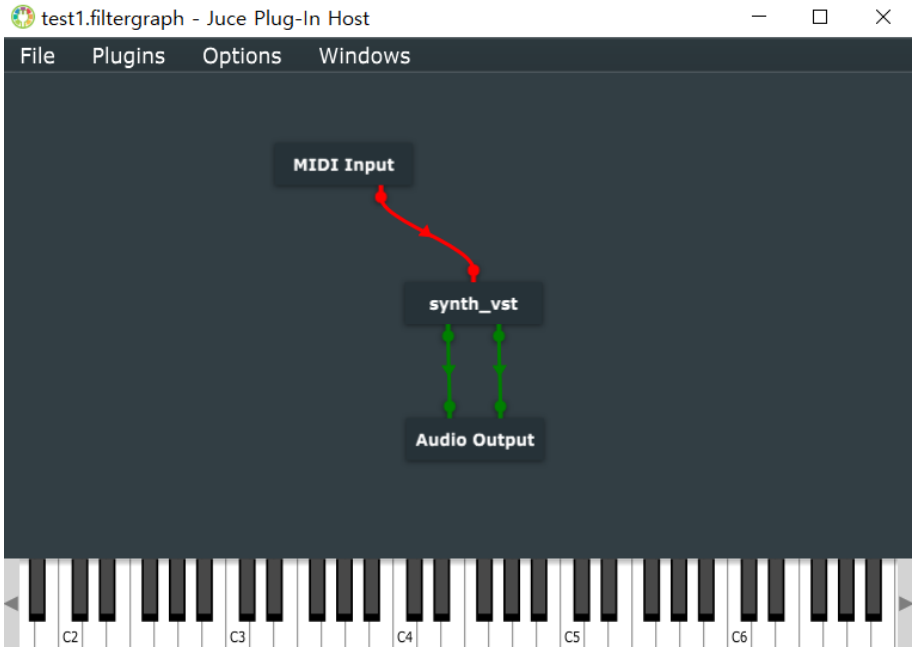

Figure 21: Plugin the VST synthesiser in the AudioPluginHost

In order to run the VST file and listen to the synthesiser sound, the user should execute the 'AudioPluginHost.jucer' file that JUCE provided it separately and existing in '../JUCE/extras/AudioPluginHost' and plugin the VST. Also, the included 'synth.vst' file should load to the Plug-In Host window.

Standalone synthesiser

- Easy-to-use
- Users can open a standalone synthesiser similar to a regular computer program (can be open by synth.exe executable file), giving them access to use the sounds of the VST synthesiser without the need to open an AudioPluginHost.

VST synthesiser

- There are some basic steps to take in order to run the program (Open/Compile AudioPluginHost.jucer, load the .vst file. etc).
- If users load another VST format file that the user already had into the AudioPluginHost file, that VST file can also be used.

Standalone & VST synthesiser

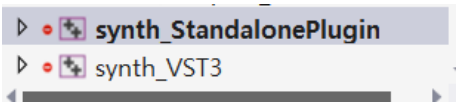

Figure 22: Snapshot of the VisualStudio code - Building .exe / .vst file

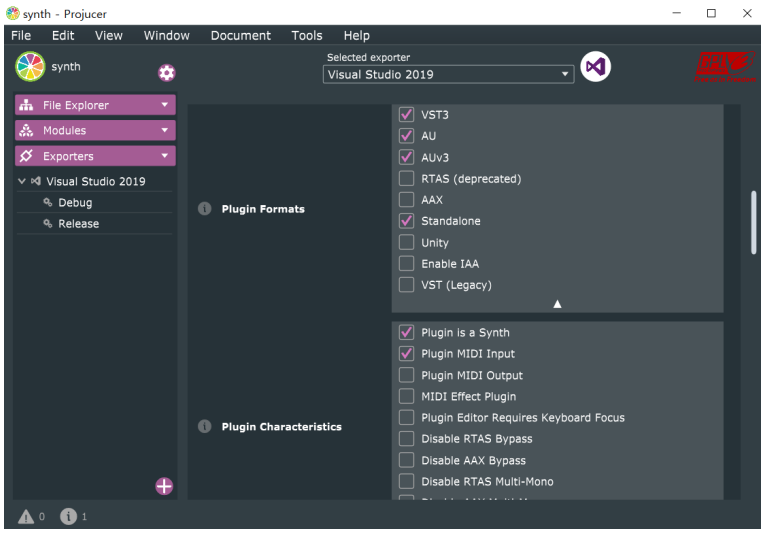

Figure 23: Snapshot of the Projucer

The JUCE Framework allows users to select the format of a plugin (Figure 20) that they can build when creating *.jucer files for the first time. Following the objective list that I made when I designed the program, I tried to develop the two formats of the synthesiser: Standalone version and VST version. Therefore, those two options were selected through the producer file (Figure 22). Then, I developed the synthesiser as input the meter(oscilloscope) and virtual keyboard into the synthesiser, adding several functions to the program based on a standalone synthesiser.
However, after development, when I tried to build the standalone and VST version of the files respectively, I found that there was a problem with the VST versions of the files. To verify that the VST version of the file runs well, the plugin operation was completed on the AudioPluginHost file, however, the sound was not come out when the virtual keyboard of the AudioPluginHost file was pressed. When the synthesiser program contain its own MIDIKeyboard, it seemed that the keyboard for MIDIInput of AudioPluginHost could not be used, since it could lead to a duplicate issue of MIDIInput. So I removed the MIDIKeyboard component from the overall code and created a VST version of the synthesiser separately where everything is the same except for the standalone version and the presence or absence of the virtual keyboard, and included the built **synth.vst** file in the uploaded implementation code file.

6.2. Implementation issue: Creating the basic waveform with mathematical formula

A synthesiser is basically an instrument that synthesizes sound through DCO, DCF, and DCA, which are responsible for pitch, tone, and volume, respectively. Among them, the Digital Control oscillator (DCO) serves to select and oscillate waveforms capable of adjusting pitch

with a digital control oscillator. It is also the most important part of the synthesiser because it cannot make any sound at all without oscillator. There are four main types of oscillators: [**Sine wave, Square wave, Triangle wave, Saw wave**] Each of the waveforms is implemented in this program with the mathematical method. All of the sounds can be expressed by proper equations. Following is the equation of the four waveforms written on the OscData.cpp file.

6.2.1. Four basic waveform formulas of the oscillator

Sine wave: The SINE waveform has its own sound with no overtone. They are sometimes described as 'pure tones' because they represent a consistent, single oscillation. The sine wave is the simplest of all waves, and that's why I set the sine wave as the default state of the program. The following formula is a used waveform equation:

- Sine wave ; $y = sine(x)$

```
AudioOutput = sin(x);
```

Listing 1: Sine Wave Formula

Square wave: SQUARE waveform is a waveform that has only odd-grained sounds and has a wide interval of overtones. Also, it is called a square wave because the width of the top and bottom squares is constant. The following formula is a used waveform equation:

- Square wave ; $y = floor(sin(x)) * 2.0 + 1.0$

```
AudioOutput = floor(sin(x)) * 2.0f + 1.0f;
```

Listing 2: Square Wave Formula

Saw wave: A representative waveform of subtraction synthesis is a SAW waveform that has both integer harmonics, i.e., odd harmonics and even harmonics. Saw is a suitable waveform for creating sound by reducing the number of overtones used by filters. The following formula is a used waveform equation:

- Saw wave ; $y = arctan(tan(-x)) * 2.0/\pi$

```
AudioOutput = atan(tan(-x)) * 2.0f / pi;
```

Listing 3: Saw Wave Formula

Mathematically, $arctan(tan(-x)) = -x$ for every x. However, the converse is not true and it cannot be, since the tangent is not an injective function. The equation (1) below is valid only when the range of θ is $(-\frac{\pi}{2} < \theta < \frac{\pi}{2})$:

$$\theta = arctan(tan(\theta)) \tag{1}$$

Recall that $arctan(x)$ just returns a number in the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$. So it has the equality that $y = arctan(tan(x))$ if and only if $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$. A formula can be given for any θ: so just need to "reduce" the angle to the right interval by subtracting an integral multiple of π so that, the following formula could be derived:

$$-\frac{\pi}{2} < \theta - k\pi < \frac{\pi}{2}$$

$$= -\frac{1}{2} < \frac{\theta}{\pi} - k < \frac{1}{2}$$

$$= 0 < \frac{\theta}{\pi} - k + \frac{1}{2} < 1$$

$$= k < \frac{\theta}{\pi} + \frac{1}{2} < 1 + k$$

Then, this equation could be derived as:

$$k = \left\lfloor \frac{\theta}{\pi} + \frac{1}{2} \right\rfloor$$

$$=> arctan(tan(\theta)) = \theta - \pi \left\lfloor \frac{\theta}{\pi} + \frac{1}{2} \right\rfloor \tag{2}$$

Therefore, for example, if $\theta = \frac{15\pi}{4}$,

$$\left\lfloor \frac{\theta}{\pi} + \frac{1}{2} \right\rfloor = \left\lfloor \frac{15}{4} + \frac{1}{2} \right\rfloor = 4$$

so,

$$arctan(tan(\frac{15\pi}{4})) = \frac{15\pi}{4} - 4\pi = -\frac{\pi}{4}$$

Since the θ value of the saw wave transmitted through the audio output is not limited to the following range: $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$, a graph is shown following to the above equation (2).

Triangle wave: The TRIANGLE waveform has only odd harmonics and little overtone. The triangle wave is often used in the sound synthesis where its timbre is less harsh than the square wave since the amplitude of its upper harmonics drops more rapidly. The following formula is a used waveform equation:

- Triangle wave ; $y = arcsin(sin(x)) * 2.0/\pi$

```
AudioOutput = asin(sin(x)) * 2.0f / pi;
```

Listing 4: Triangle Wave Formula

Similar to the case of the saw wave, mathematically, $arcsin(sin(x)) = x$ for every x. However, the converse is not true and it cannot be, since it is not an injective function. The equation (3) below is valid only when the range of θ is $(-\frac{\pi}{2} < \theta < \frac{\pi}{2})$:

$$\theta = arcsin(sin(\theta)) \tag{3}$$

Recall that $arcsin(x)$ just returns a number in the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$. So it has the equality that $y = arcsin(sin(x))$ if and only if $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$. Therefore, the range of θ should be considered for drawing graph of them:

1) in range of $(-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2})$

$$if(-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2})$$

$$=> arcsin(sin(\theta)) = \theta \tag{4}$$

2) in range of $(\frac{\pi}{2} \leq \theta \leq \frac{3\pi}{2})$

$$if(\frac{\pi}{2} \leq \theta \leq \frac{3\pi}{2})$$

$$\frac{\pi}{2} \leq \theta \leq \frac{3\pi}{2}$$

$$= \frac{-\pi}{2} \leq \theta - \pi \leq \frac{\pi}{2}$$

$$= \frac{-\pi}{2} \leq \pi - \theta \leq \frac{\pi}{2}$$

$$=> arcsin(sin(\theta)) = arcsin(sin(\pi - \theta)) = \pi - \theta \tag{5}$$

3) in range of $(\frac{3}{2} \leq \theta \leq \frac{5\pi}{2})$

$$if(\frac{3}{2} \leq \theta \leq \frac{5\pi}{2})$$

$$\frac{3}{2} \leq \theta \leq \frac{5\pi}{2}$$

$$= \frac{-\pi}{2} \leq \theta - 2\pi \leq \frac{\pi}{2}$$

$$=> arcsin(sin(\theta)) = arcsin(sin(\theta - 2\pi)) = \theta - 2\pi \tag{6}$$

Looking at the Figure 24 below, it can be seen that a part of the triangle wave graph is drawn. The blue line indicates $y = \theta$ (equation (4)), the red line indicates $y = \pi - \theta$ (equation (5)), and the green line indicates $y = \theta - 2\pi$ (equation (6)). The shape of these graphs is repeated every 2π cycle to form a triangle wave.

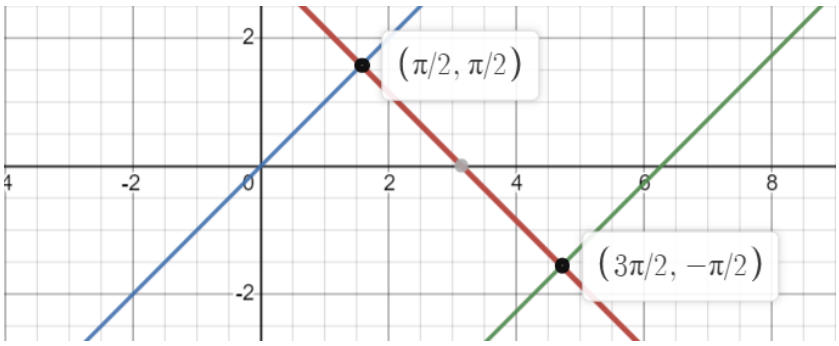

Figure 24: Desmos Graph of triangle wave formula

Therefore, saw&triangle wave formula graphs do not show the linear $y = x$ graph since it is not an injective function.

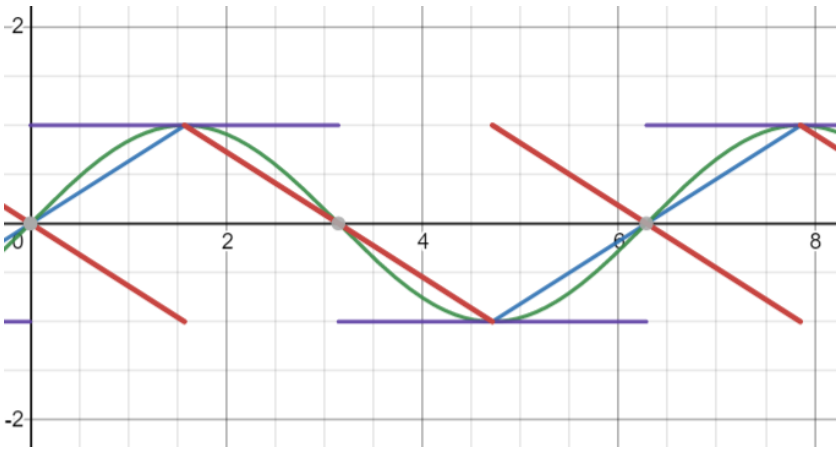

Figure 25: Desmos Graph of waveforms formula

The above Figure 25 is a graph of current waveforms created using the Desmos site. The green line represents 'sine wave', the purple line represents 'square wave', the blue line represents 'triangle wave', and the red line represents 'saw wave'. Every waveform has its own unique graph, and the waveform that produces sound must also be the same as that graph. For such comparison, this synthesiser contains an 'Oscilloscope' as a named 'meter', which represents the waveform of each sound.

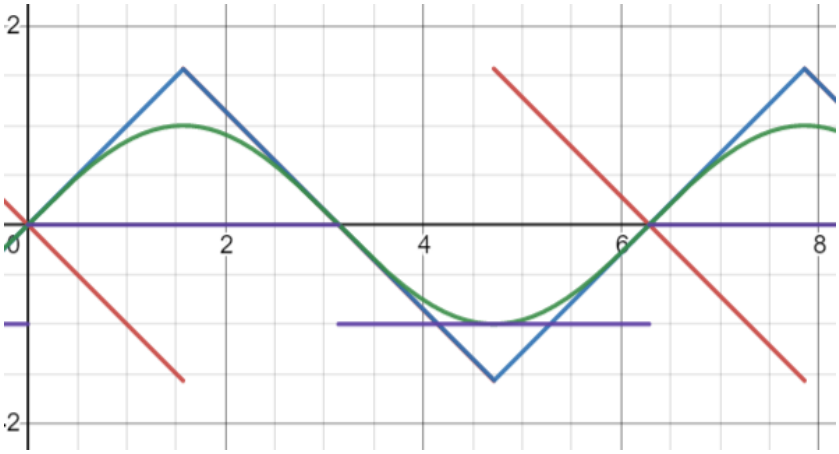

Figure 26: Desmos Graph of waveforms formula without constant values

Each waveform can be represented just by sine formulas excluding adding or multiplying some constant values. That was what I implement the first time. However, I found the problem with the oscilloscope that the difference in amplitude of each waveform. The reason why constant values are used with the formulas is to make the amplitude value of each waveform equal to the basic sine waveform's amplitude. Above Figure 26 shows the graph that not used constant values.

Waveform	**Formula without constant variables**	**Max value**	**Min value**
Sine wave	$y = sine(x)$	1, $(x = \frac{\pi}{2})$	-1, $(x = \frac{3\pi}{2})$
Square wave	$y = floor(sin(x))$	0	-1
Triangle wave	$y = arcsin(sinx)$	$\frac{\pi}{2}$, $(x = \frac{\pi}{2})$	$-\frac{\pi}{2}$, $(x = \frac{3\pi}{2})$
Saw wave	$y = arctan(tan(-x))$	$\frac{\pi}{2}$, $(x = \frac{\pi}{2})$	$-\frac{\pi}{2}$, $(x = \frac{3\pi}{2})$

Waveform	Formula with constant variables	Max value	Min value
Sine wave	$y = sine(x)$	1, $(x = \frac{\pi}{2})$	-1, $(x = \frac{3\pi}{2})$
Square wave	$y = floor(sin(x)) * 2 + 1$	1, $(x = \frac{\pi}{2})$	-1, $(x = \frac{3\pi}{2})$
Triangle wave	$y = arcsin(sinx) * 2/\pi$	1, $(x = \frac{\pi}{2})$	-1, $(x = \frac{3\pi}{2})$
Saw wave	$y = arctan(tan(-x)) * 2/\pi$	1, $(x = \frac{\pi}{2})$	-1, $(x = \frac{3\pi}{2})$

Through the graph (Figure 25), it can be seen that all of the maximum and minimum values have changed equally with the constant variables. That is, the difference in amplitude of each waveform could be solved by appropriately using a constant variable.

6.2.2. Confirm the waveforms of OSC sound by oscilloscope

I made my own oscilloscope for the synthesiser, however, tried to plugin another VST file to the synthesiser for checking whether the VST synthesiser was linked with other VST files well or not. Therefore, used another simple oscilloscope VST plugin (SimpleAudioScope.vst3) for confirming the waveform shapes.
When analyzed the each waveform, all of them(Figure 27, Figure 28, Figure 29, Figure 30) were worked out correctly and show exactly same form with the Desmos graph(Figure 25) above.

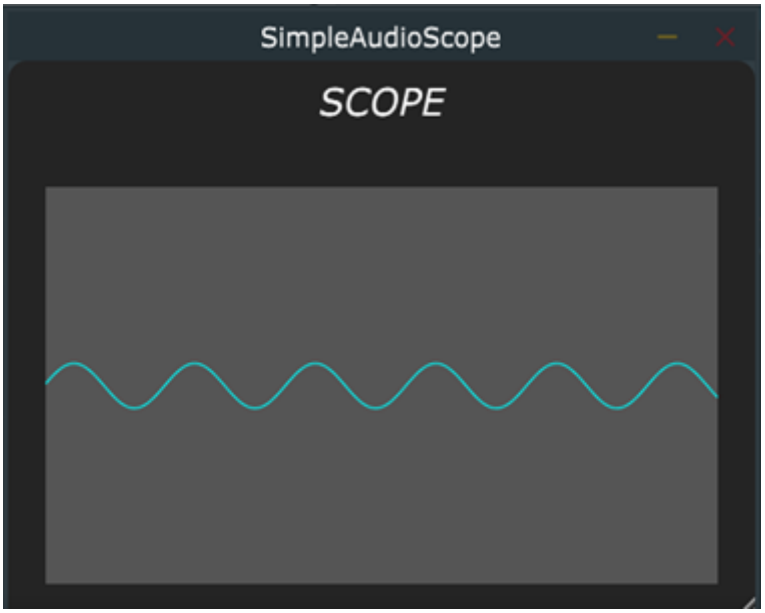

Figure 27: OSCILLOSCOPE; sine wave

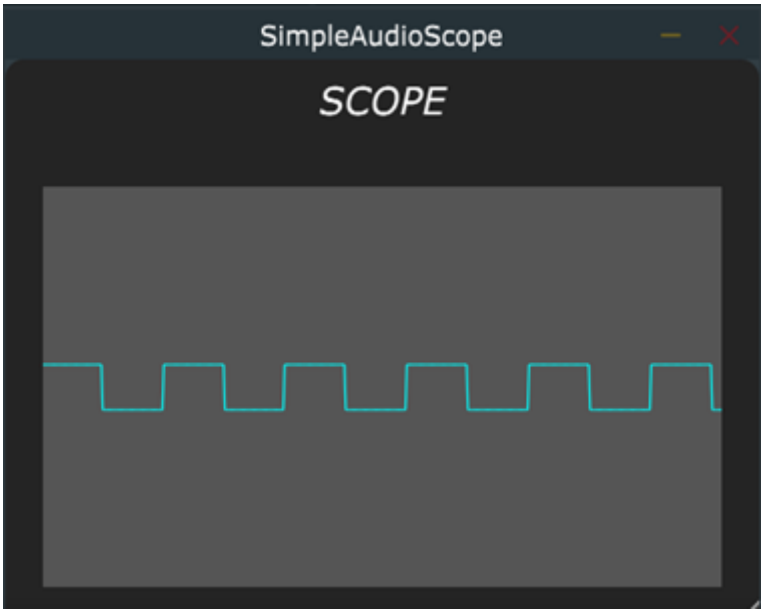

Figure 28: OSCILLOSCOPE; square wave

Waveforms of OSC sound

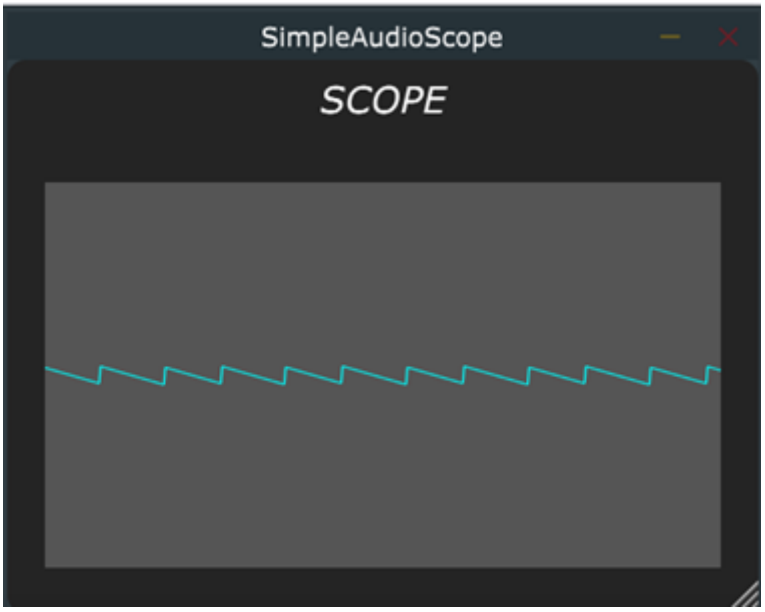

Figure 29: OSCILLOSCOPE; saw wave

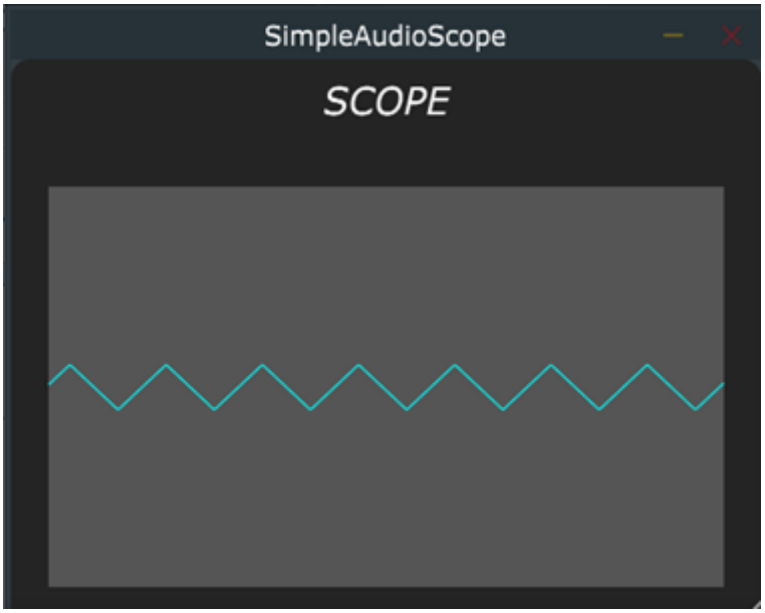

Figure 30: OSCILLOSCOPE; triangle wave

6.3. Implementation issue: Creating the filter with FIR or IIR

Filters are one of the most important parts of a subtractive synthesiser.
The user is able to control the cutoff frequency(f_C) and resonance(Q; the gain of a filter at its cutoff frequency) value with knobs in this synthesiser(Figure 32). This synthesiser contains the following three types of filters:

- Low-Pass Filter(LPF) - The method of cutting off the upper part of the cutoff frequency and saving the lower part is called a low-pass filter
- High-Pass Filter(HPF) - The method of cutting off the lower part of the cutoff frequency and saving the upper part is called a high-pass filter
- Band-Pass Filter(BPF) - The method of erasing all the rest, leaving only some narrow bands (defined by the frequency and Q value, or the center of the two frequencies)

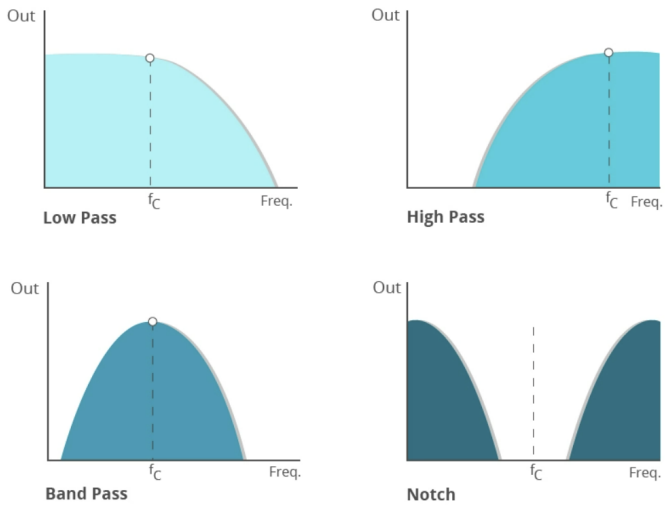

Figure 31: Type of the filters (Andrew (2016)[53])

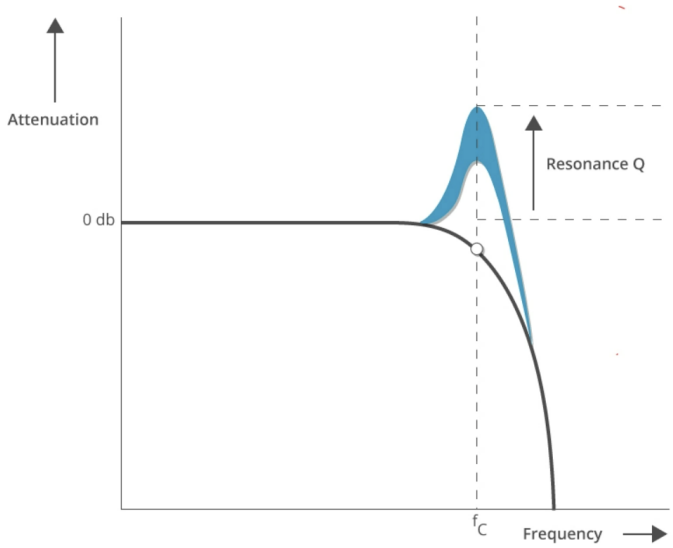

Figure 32: Resonance(=Q) of the filter (Andrew (2016)[53])

In the JUCE Framework, filters are mainly made in two ways: the first is the **Finite Impulse Response(FIR)** filter, and the second is the **Infinite Impulse Response(IIR)** filter.
In this synthesiser, the filter was created using the IIR method using the **Topology-Preserving Transform(TPT)** method that can perform low, band, and high-pass filtering on an audio signal, with 12 dB of attenuation per octave, designed for fast modulation.

TPT: Trapezoidal integration replacement technique as the topology-preserving bilinear transform (TPBLT). Also, referred to as topology-preserving transform (TPT). TPT method explicitly aims at emulating the time-varying behavior of the analog prototype structure, which aspect is completely ignored by the classical transform approach.(Vadim Zavalishin (2015)[32])

IIR filters are also used for preventing the loud audio artifacts often encountered when manipulating cutoff frequency values using IIR filter classes and other filter simulation structures. In addition, the following Figure 33 is a diagram of TPT structure. In particular, 'z' denotes the time delay of 1 sample unit.

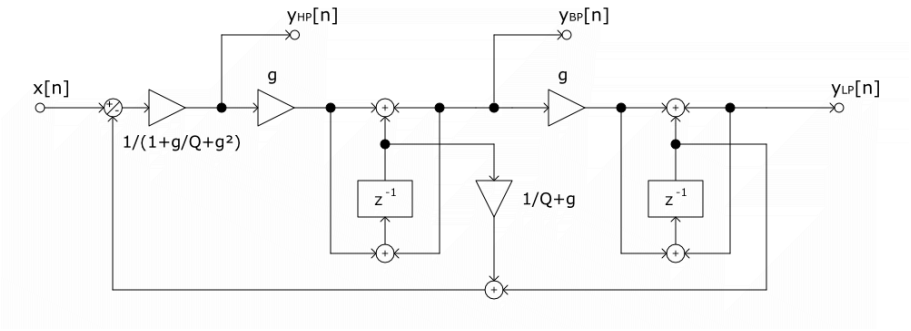

Figure 33: Structure of the TPT (Vadim Zavalishin (2015)[32])

Comparing the IIR filter & FIR filter [48]

: IIR Fiters [33]

- IIR filters are the most efficient type of filters implemented by DSP.
- IIR filters are generally provided as "biquad" filters.
- The processing amount required for IIR to calculate the "biquad" is less than the FIR filter relatively.

: FIR Fiters

- The FIR filter needs more computational time & more memory for DSP. Therefore, more powerful DSP chips are required.
- FIR filter is able to implement linear phase (the phase component of the frequency response is a linear function of frequency) filtering. It can be said that the filter is not phase-shifted across the frequency band. Alternately, the phase can be revised independently of the amplitude.
- It was able to use to revise the frequency-response errors that in a loudspeaker to a finer degree of precision than using IIRs.

FIR filters provide some advantages compared to IIR filters, however, FIR filters can have limited resolution at low frequencies, and the success of applying FIR filters relies heavily on the program used to generate filter coefficients. Plus, an IIR filter has the advantage that for a similar filter roll off as an FIR filter, a lower order or number of terms can be used. This represents that fewer computations are needed to achieve the same result, leading to the IIR's faster computation.[48]
When implemented in the program with the FIR filter for the first time, a short lag of about 3-5 seconds occurred every time when made the sound. In order to efficiently use the synthesiser, there should be no time delay problem such as a lag problem. I judged that it would be more beneficial for the synthesiser to reduce these latencies than the advantage that the FIR filter can be corrected independently of the amplitude. For preventing the lag problem, the IIR filter was implemented in programs instead of the FIR filter. The two tables below summarize the time concealing experienced when playing a synthesiser with each filter.
The synthesiser obviously has a slight latency for each function execution, however, latency below 10ms is difficult for the listener to notice in general, so is not a significant problem from a usability perspective. Therefore, if it felt like there was no latency, I filled 'N' in the blank instead of time values.

Time consuming: IIR Fiters (Implemented)

Start	Making sound	Reset the state	Save the state	Load the state	Exit
N	N	N	20 sec	20 sec	N

Time consuming: FIR Fiters

Start	Making sound	Reset the state	Save the state	Load the state	Exit
N	3-5 sec	0 sec	20 sec	20 sec	N

In addition, the below pseudocode briefly explains the TPT structure which is used for the IIR filter in the program:

Algorithm 1 Processing of the Filter with TPT structure

```
1: Start:
2:     #Pi = float pi value
3:     const float g = tanf(Pi * frequency / sampleRate); <- frequency value comes from MIDIInput by the user, g is the gain of the peak and also comes from user's input value
4:     const float h = 1.f / (1 + g / Q + g * g); <- Q is the resonance value, which comes from user's input resonance value
5:
6:     for (int i = 0; i < numSamples; i++){ # For loop start
7:         const float in = samples[i];
8:         const float yH = h * (in - (1.f / Q + g) * s1 - s2);)
9:         const float yB = g * yH + s1;
10:        s1 = g * yH + yB;
11:        const float yL = g * yB + s2;
12:        s2 = g * yB + yL;
13:        samples[i] = yL;
14:    } # For loop End
15: End
```

6.4. Efficiency issues: CPU loading

There are many discussions on the music forum about the compared efficiency of each synthesiser. The computer has limited memory and CPU, so users generally check the value of memory/CPU consumption of the program to use a more efficient synthesiser. When I run the synthesiser program on my computer and checked the result of efficiency by the computer manager run as administrator, the output was the following:

Process Name	CPU	Memory	Disk	Network
synth.exe	18.3%	17.1MB	0MB/s	0Mbps
-	a bit high compare to memory	normal	-	-

6.5. Design of the synthesiser

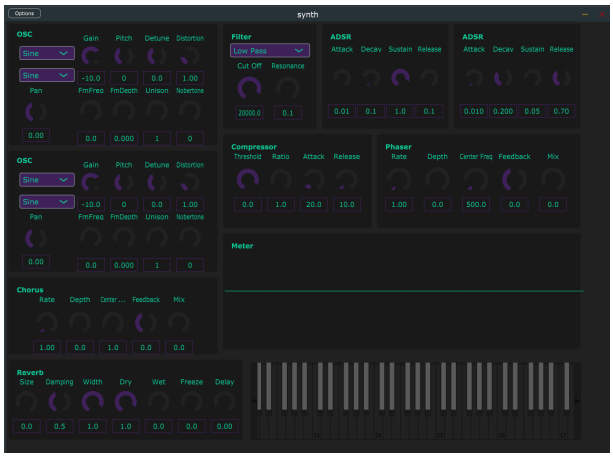

Figure 34: Snapshot of the previous version of synthesiser program

Accepting the suggestion from the user that the overall color of the program is dark and the value indicated by the knob is not intuitively known, I improved the design of the knob like a clock needle that allows users to see the value indicated by the knob immediately.

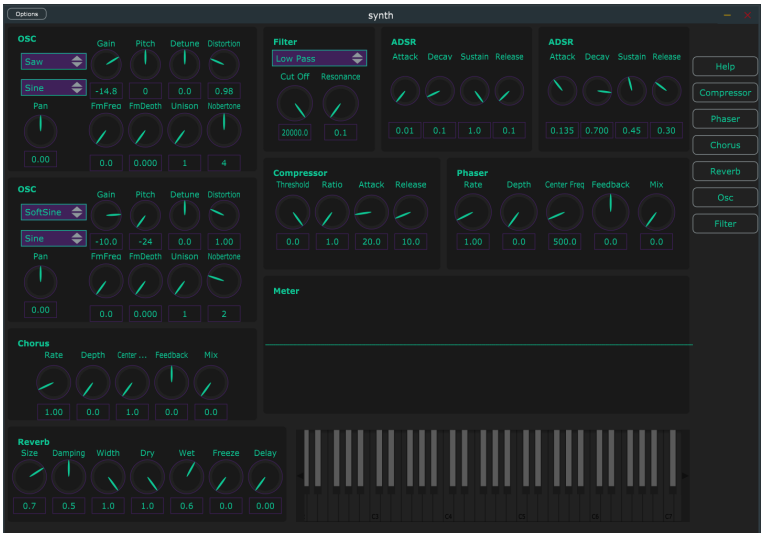

Figure 35: Snapshot of the redesigned version of synthesiser program

7. Evaluation

Following system level evaluation is judged based on the project requirements in section **5.5. Requirements analysis**. I tried to relate this evaluation to my project requirements for confirming that I met the requirements. What requirements and items are related are recorded as numbers in the first line of the description column.

7.1. System Level Evaluation: Pass

Description	Inputs	Expected Ouput	Actual Output
Related with F1: The user selects the waveform of the sound from the OSC selector box: 9 types exist	MIDIInput value is input to the synthesiser with the selected waveform. (OSC Selection box: Default setting is sine wave)	Users should hear the proper waveform sound of the MIDINoteNumber through audio.	When users press the keyboard, the appropriate waveform sound is transmitted through audio, and users hear the sound.
Related with F2: The user selects the filter from the filter selector box: LPF, HPF, BPF exist	MIDIInput value is input to the synthesiser through the selected filter by the user. (Filter Selection box: Default setting is LPF, CutOff Freq: 20,000Hz, Resonance(Q): 0.1)	Users should hear the proper sound of the MIDINoteNumber passed through the selected filter.	When users press the keyboard, the appropriate filtered sound is transmitted through audio, and users hear the sound.
Related with F3: The user controls the numeric value of the FM module (FMfreq/FMdepth) by handling the knobs.	Each numeric value (FMfreq/FMdepth) of the FM module is changed by the knobs by dragging/rolling the mouse/mouse wheel and input to the program.	The synthesiser should return the MIDI Audio Output that reflected the FM modulation.	The synthesiser returns the MIDI Audio Output that reflected the FM modulation.
Related with F4: The user controls the numeric value of each effector function by handling the knobs: Reverb, Phaser, Compressor, Chorus.	Each numeric value of the effector functions is changed by the knobs by dragging/rolling the mouse/mouse wheel and input to the program.	The synthesiser should return the MIDI Audio Output that reflected the effector functions: Reverb, Phaser, Compressor, Chorus.	The synthesiser returns the MIDI Audio Output that reflected the effector functions: Reverb, Phaser, Compressor, Chorus.

Description	Inputs	Expected Ouput	Actual Output
Related with F5: The user controls the numeric value of the filter ADSR & Amplifier ADSR by handling the knobs.	Each numeric value of the filter ADSR & Amplifier ADSR is changed by the knobs by dragging/rolling the mouse/mouse wheel.	The synthesiser should return the MIDI Audio Output that reflected the filter ADSR & Amplifier ADSR.	The synthesiser returns the MIDI Audio Output that reflected the filter ADSR & Amplifier ADSR.
Related with F6: The user presses the virtual piano's keyboard on the AudioPluginHost. (VST version)	MIDIInput value(MIDINoteNumber) is input to the program from AudioPluginHost's virtual keyboard.	Users hear the appropriate note sound of the keyboard pressed by the user through audio.	When the user presses the keyboard, the appropriate note sound is transmitted through audio.
Related with F6, N5: The user plugins other VST plugins to the synthesiser program. (VST version)	The user inputs and links the other his/her own VST plugin file to the synthsiser on AudioPluginHost.	Input VST plugin file should be able to link with the synthesiser and work(make the sound, reflect the controlled by the user) well as before. Also, the user should be able to use the VST file's function that is inserted.	Input VST plugin file is able to link with the synthesiser and work(make the sound, reflect the controlled by the user) well as before. Also, the user can use the plugin file function that is inserted.
Related with F6, F7: The user presses the virtual piano keyboard on the screen. (Standalone version)	MIDIInput value (MIDINoteNumber) is input to the program through keyboard (MIDIKeyboardComponent) component.	Users hear the appropriate note sound (MIDIOutput) of the keyboard pressed by the user through audio.	When the user presses the keyboard, the appropriate note sound of the scale is transmitted through audio.
Related with F8: The user try to confirm the waveform graph by an oscilloscope(Meter).	The user inputs the MIDIInput value to the program with the desired modulation and tries to confirm the sound graph.	The oscilloscope(Meter) should show the exact graph of the waveform in real-time.	When the user executes the program, the oscilloscope(Meter) shows the exact waveform graph in real-time.
Related with F9: The user selects the 'Reset the default state' option from the caption.	When the user presses the 'Reset the default state' option from the caption, the synthesiser loads the default state of the synthesiser.	All knobs and selector box values should go back (reset) to the initial state.	All knobs and selector box values are going back (reset) to the initial state.

Description	Inputs	Expected Ouput	Actual Output
Related with F10: The user selects the 'Save the current stated' option from the caption.	When the user presses the 'Save the current stated' option, the current state is saved on the presets folder as the ∗.synth_state_1 format file.	All knobs and selector box values should be saved as a ∗.synth_state_1 format file on the presets folder.	All knobs and selector box values are saved as a ∗.synth_state_1 format file on the presets folder.
Related with F10: The user selects the 'Load the saved stated' option on the caption.	The user can use the specific state again if it is saved. The example saved state exists on the presets folder with ∗.synth_state_1 format. When the user selects that option, the program loads the saved state from the folder.	All knobs and selector box values should be set as the value from the saved state file on the presets folder.	All knobs and selector box values are set as the value from the saved state file on the presets folder.
Related with F12: The user resizes the program screen of the synthesiser.	The user drags the edge of the synthesiser on the screen. The setResizable(true, true) method on PuginEditor.cpp allows the to user resize the size of the synthesiser.	When the user drags the edge of the synthesiser on the screen, the user should be able to make bigger or smaller the size of the program.	When the user drags the edge of the synthesiser on the screen, the user is able to make bigger or smaller the size of the program.
Related with F13: The user tries to confirm the function name of each knob/selector box before using the program.	The name of each function that is set in advance is represented to the users with the proper labels.	The names of each function should be visible to the user by the proper label.	The names of each function are visible to the user by the proper label.
Related with F14: The user tries to confirm the numeric value on the label of the knobs that are adjusted by the mouse.	The value of each function is changed by users in real-time and those changes are input into the program immediately.	The numerical value from each function through the knob is adjusted should be visible to the user by the proper label.	The numerical value from each function through the knob is adjusted and visible to the user by a proper label.

Description	Inputs	Expected Ouput	Actual Output
Related with N3, N4: The user tries to confirm brief information about each implemented function.	The users click the 'README.txt' & 'Help' button on the right side of the screen.	Users should be able to confirm the brief information of each function with README.txt file and Help pop-up box.	Users confirm the brief information of each function with README.txt file and Help pop-up box.

7.2. System Level Evaluation: Not fully achieved

Description	Inputs	Expected Ouput	Actual Output
Related with N2: Users open the synthesiser with .exe file or compile/run the source code(synth.jucer) through Visual Studio 2019.	The executable file of the synthesiser program is run with the initial state.	All knobs&Select box for each function should be set with the initial state (same as reset state) and there should be no sound until the virtual piano keyboard is pressed.	All knobs&Select box for each function should be set with the initial state, however, there is unexpected noise exist that occurs shortly for 1-2 seconds immediately after executing the program since of the code complexity.
Related with N2: The user uses the save/load/reset option on the caption.	The program save/load the $*$.synth_state_1 format file in the preset folder.	The program should reflect the save/load/reset the state of the synthesiser immediately.	There is a bit of lag that occurs when load/save the state of the synthesiser. It takes 20 sec to use those functions usually.
Related with F11: The user uses the synthesiser by the polyphonic system with 5 voices.	Users input the different 5 voices MIDIInput values.	The program should receive the different 5 voices' input values and execute them.	The polyphonic system with 5 voices not fully achieved to implement since the overall code structure should be redesigned.

7.3. User Test

I have taken for this project: User-Testing involving real participants. Since the goal of this project is to develop an easy-to-use synthesiser program, I decided that it is reasonable to investigate the overall satisfaction of the program with a User Test.
A total of five students participated in the user test, and all opinions were collected anonymously using Google Forms. The questionnaire which is consisted of 9 multiple choice questions and 2 short answer questions used exists in the: **Appendix C. User Test - Questionnaire**.

Q1 and Q8 are related to overall program general satisfaction, and Q2 and Q7 are asking about the satisfaction of program design. Q3, Q9 are asked about the satisfaction of the sound creation, whether users can make the sound with no limitation or not, and Q4 asks whether the program's help/Explanation is sufficient or not. Q5 asks whether the program is easy to use or not, especially about installation, and Q6 ask is there any Error/Lag occurs or not. Participants (total: 5) in the user test were three Informatic students, one business student, and one International Relationship student. The results of the user test were generally positive for the synthesiser program. The answers to all multiple-choice questions were divided into 1. the most positive answer(good) to 5. the most negative answer(bad), and the results were organized in the chart. Following Figure 36 is the result bar chart of the user test:

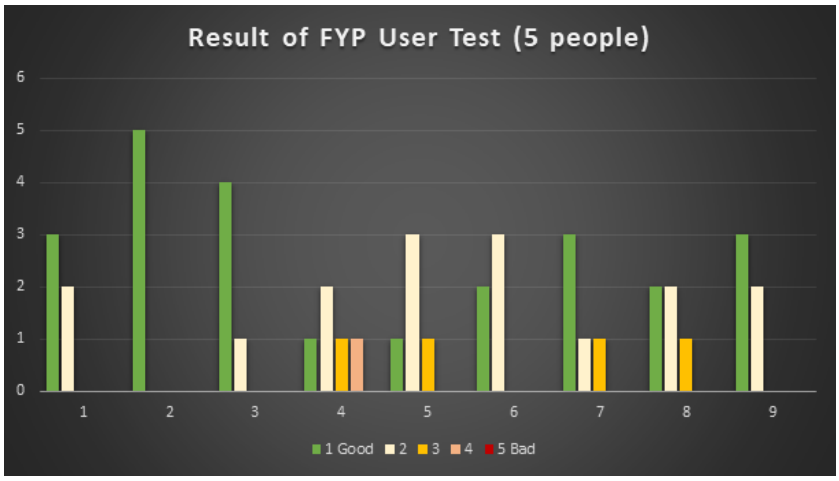

Figure 36: Result of User Test

The program user test was conducted after the participants downloaded the synthesiser program and JUCE library that I uploaded on GitHub and had time to make the desired sound on their own computers. According to the results of the User Test, most of the participants responded well to the functional part of the program, however, there was an opinion that the explanation part of the program needed to be supplemented. The followings are the arguments that participants' opinions (Q10 & Q11) for the supplement:

- **Q4. Explanation problem)** It will be good to contain some kind of 'help feature' that can be enabled to see the exact effect of the effector since some program users are not familiar with the synthesiser and its functions. (e.g. Reverb - give similar with some echo effect) **-> Help buttons were implemented**
- **Q5. Explanation problem)** It will be good to provide a more detailed explanation of the program installation progress. **-> README.txt file become more detailed.**
- **Q7. Design problem)** It is a bit difficult to check the value of the knob since the design color is dark. **-> Design slightly changed**

I confirmed that the synthesiser program works well with all of the functions that are implemented through User Test, however, also notice that there was a bit of lag occurring to load and save the current state of the synthesiser. In particular, I guess the reason why the program explanation and installation part of the program did not produce satisfactory results at first is that most students were not familiar with synthesisers.

8. Conclusion

As a result of the evaluation in the **6. Evaluation** section above, it can be said that the implementation of the synthesiser, which was the goal of the project, was well done overall, except for a few lag and time problems on the save/load option. The Core objectives in section 4.2 of the project have also been achieved almost, just except for the polyphonic part. In addition, three of the extended objectives were achieved. As described in the **4. Project Design** section and **5. Implement project** section above, the program contains all of the basic elements that the subtractive synthesiser should have. The final version of the developed synthesiser in this project provides the functions that *oscillators, filters, envelope, and effectors (including four types: Phaser, Chorus, Compressor, Reverb), meter, and virtual keyboard* to the users. Following is the Objectives list that I succeeded & not achieved:

[Success]

- Core Objectives

1. Research and understand the functions and characteristics of subtractive synthesisers that distinguish them from other types of synthesisers and design & implement the program using C++ & JUCE Framework.
2. Choosing various options from oscillators, filters, amplifiers, effects, and modulation sources that subtractive synthesisers should have as functions for implementation.
3. Allow the user to play the unique sound by just using a mouse & keyboard through the AudioHost app so that it can be used as a desktop-based VST application plugin.
4. Implement a GUI that allows a user can handle the synthesiser's function with knobs to play sounds using the many functions implemented in Core Objective 2.
5. Provide a sound wave graph and show it through the interface for analyzing the waveform.

- Extended Objectives

1. Working with a MIDI interface.
2. Provide a standalone version of the subtractive synthesiser. (virtual keyboard and meter functions should be added with extra work)
3. Creating various Effectors; Reverb, Chorus, Compressor, Distortion, Delay, Phaser etc & develop a filter with variable Resonance

[Not achieved]

- Create subtractive synthesiser with **polyphonic** capability

The only core objective that not achieved to implement in the synthesiser was the polyphonic system. I initially planned to implement the synthesiser as a monophonic synthesiser (Fallback Objective), and then add various effects, and then add a polyphonic system. However, in order to add a polyphonic system, the implementation not achieved due to the difficulty of redesigning the code of the program as a whole. I should have focused from the beginning on creating efficient code structures, such as detailed object-oriented programming so that changing a program from a monophonic system to a polyphonic system does not require redesigning the entire code.
However, the overall project can be seen as successful in that it achieved all of the core objectives except the implementation of the polyphonic system, and three of the extension objectives.
All of these components were designed through the JUCE Framework, a development application selected in the planning stage. As a result of initially selecting an application that is proper to achieve the aims of the project, it was much easier to implement the VST version of the synthesiser and the standalone version of the synthesiser. In order to develop good applications with a limited time, it was an opportunity to realize once again that it was important to properly understand the characteristics of applications that would be the first environment to be developed and to select the right development applications. Overall, the project proceeded as planned PERT chart, and as a result of making good use of holidays and vacation periods and increasing the time spent on the project during the vacation, I was able to complete the development of the project earlier than the target date. This shows that the overall time management was well done.

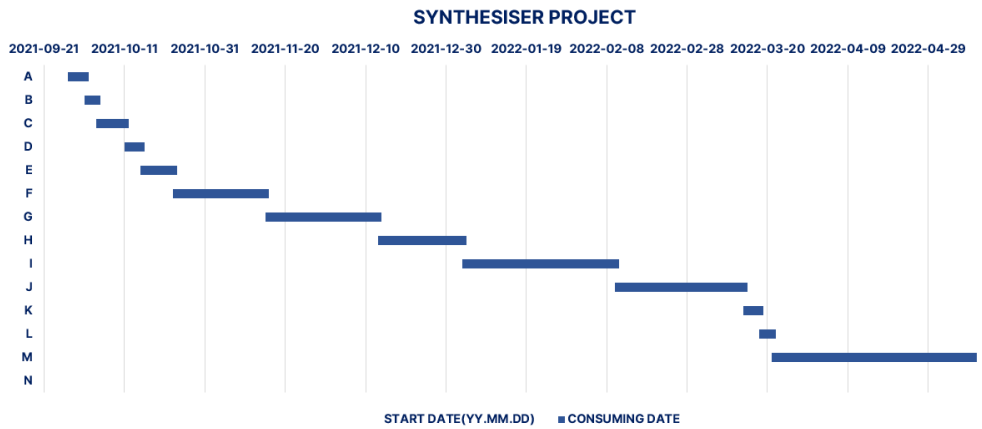

Figure 37: Result of project progress; time consuming

As I worked on this project, I became able to have a deep understanding of the overall synthesis method and functions used in the music area that I had aimed for when I design this project. In addition, I also became familiar with a unique library called the JUCE Framework.
The following two extension objectives that I decided not to implement in the synthesiser are related to recording and playback/audio recording and audio editing capabilities. After adding the function of loading/saving a certain state of the synthesiser, the recording function was ignored because the loading/saving function did a similar thing to recording the sound in this project.

- Provide recording & playback functions.
- Provide Digital Audio Workstation (DAW) capability that allows users to execute MIDI sequencer/MIDI editing functions; audio recording, audio editing, MIDI editing, mixing, mastering, etc.

9. Further Work

Get More Feedback

The purpose of this project was to develop an easy-to-use subtractive synthesiser that uses the JUCE Framework. Therefore, in order to find out the convenience of using the program, a user test was taken from other 5 students who could become users. At this time, only five people's opinions were collected for the convenience of the project, however, the program generally can further enhance its convenience of it through more testing and feedback, so future developments would like to receive more people's feedback during the evaluation stage. In particular, since only students who are not familiar with synthesisers participated in the user test in this project, it would be beneficial to receive opinions from people who have used other synthesisers or are familiar with synthesisers.

Mobile-based application & Connect with Network USB

This is a program that runs in a desktop-based environment, and I have not used a developed synthesiser program by connecting to a real instrument or a mobile environment. However, these days, various music activities take place not only limited in the desktop environment, but also in the mobile environment. Moreover, people who are personally interested in synthesisers directly connect the digital synthesiser to amplifiers to add effectors to the actual sound of their instruments such as an electronic piano with a USB network. Therefore, I would like to try to develop a synthesiser application that can be used not only in desktop environments but also in more diverse environments next time: Mobile-based application (Mobile) & Connect with Network USB (Real world)
In particular, developing synthesiser application in a mobile environment is more interesting because it is expected that develop other coding language skills such as JavaScript, which is said to be essential in the Google Android platform. Moreover, if I develop the program as a mobile application, it is expected to get various feedback easily through star ratings and comments by uploading the app to a place such as Google Play Store after production.

Implement the polyphonic system

Implementing the polyphonic system that was not achieved in this project will be a great choice for further work. Based on my experience in developing monophonic synthesiser, I would like to focus on creating efficient code structures such as more detailed object-oriented programming, so that when changing from a monophonic system to a polyphonic system from the beginning, there is no need to redesign the entire code.

Efficiency development

The efficiency could also be addressed for improving the convenience of the program. This synthesiser program currently uses a bit high CPU in the computer compared to memory, so a more well-made code should be implemented. It would be a good aim to set the next goal to use less than 10% of CPU. Also, the lag problem should be removed.

Appendices

A. Progress logs

A.1. Weekly logs

Week 1 (27th Sep 2021)

- Understand and summarizes how individual projects proceed
- Read carefully about my project description and skills
- **Plans for next week:**

  - Design a very simple program to learn about the functions of JUCE Framework

  - Create PERT chart for the first project plan

Week 2 (4th Oct 2021)

- Decided that when I should make my second individual meeting with supervisor
- Based on the first one-to-one meeting, the proposal document is supplemented and submit on 4th Oct
- Implement a simple trial program with JUCE Framework that includes oscillator module (sine, square, triangle, saw) with several waveforms and controls for pitch (frequency) and gain (amplitude). Also, work on Envelope.
- Create the first version of PERT chart
- **Plans for next week:**

  - Supplemented with oscillators, especially, triangle waveform formula

  - Design the first prototype of synthesiser

Week 3 (11th Oct 2021)

- Create the first prototype of synthesiser
- Research about OSCILLOSCOPE to check the waveform of oscillator
- Checking whether the waveform of the oscillator through using OSCILLOSCOPE is accurate or not
- Modify the triangle waveform formula
- **Plans for next week:**

  - Supplement the PERT chart based on the experience of producing synthesiser from 1-3 weeks

  - Research the FM modulator and implement it in the program

Week 4 (18th Oct 2021)

- Produce a complementary version of the PERT chart

- Implement of the FM modulator with Frequency and Depth slider

  Plans for next week:

  - Organize the contents of the Interim report

Week 5 (25th Oct 2021)

- Organize the structure & contents of the Interim report
- Change the colour of the slider; Implement CustomComponent(UI) - Colour Component

  Plans for next week:

  - Complete the draft Interim Report

Week 6 (1th Nov 2021)

- Complete the Interim Report and receive feedback from supervisor
- Implement the more function to OSC

  Plans for next week:

  - Research & Implement the Filter to the synthesiser

  - Submit the interim report with the final revision that reflect feedback from supervisor

Week 7 (8th Nov 2021)

- Submit the modified Interim report that reflect feedback from supervisor (11th Nov 2021)
- Researching and understanding about the Low pass filter (starting point: averaging filter)

  ; Cutoff frequency value should be implemented with slider for filter (GUI)

  Plans for next week:

  - Implement the averaging filter to the synthesiser (simple way to implement LPF)

Week 8 (15th Nov 2021)

- Implement the averaging filter to the synthesiser for LPF
- Researching and understanding about the High pass filter(HPF)&Band pass filter(BPF)

  Plans for next week:

  - Implement the HPF and BPF to the synthesiser

  - Research about resonance function

  - Combine the LPF filter to the my own synthesiser program which is separated in current,

Week 9 (22th Nov 2021)

- Combine the LPF filter and program what I made so far
- Implement the HPF and BPF to the synthesiser
- Research about resonance function
- With Oscilloscope, confirm the waveform of sound through filter (LPF, HPF, BPF)

  Plans for next week:

  - Implement the resonance function

Week 10 (29th Nov 2021)

- Implement the resonance function to the synthesiser
- With Oscilloscope, confirm the waveform of sound through filter of resonance function

  Plans for next week:

  - Research about effectors

Week 11 (6th Dec 2021)

- Researching and understanding the Effectors functions of the synthesiser.
- Select the specific functions that should be implemented to this synthesiser from various effectors; Phaser, Compressor, Chorus, Reverb
- Since the coursework and examination of other modules, the implementation part should be processed in the next week.

  Plans for next week:

  - Research about Phaser and implemented

Week 12 (13th Dec 2021)

- Implement the Phaser effector to the synthesiser: Rate, Depth, Mix Knobs
- With an Oscilloscope, confirm the waveform of sound through handling the Phaser knobs.

  Plans for next week:

  - Supplement the phaser function by adding the values and knobs required.

Week 13 (20th Dec 2021)

- Holidays - Brush up the code that is implemented so far
- Receive the feedback of the interim report through Canvas

  - Should be developed : include a description of the advantages of the JUCE framework and why the JUCE framework was selected rather than other development applications as a development application for the project in the reports
- Create the structure of the Final report & Organize the contents of the Final report

Week 14 (27th Dec 2021)

- Holidays - Brush up the code that is implemented so far
- Research about the pros and cons of the development application, especially JUCE Framework (Reinforce the Background part of the final report)

Week 15 (3rd Jan 2022)

- Implement the Phaser effector to the synthesiser: Center Frequency, Feedback knobs
- With an Oscilloscope, confirm the waveform of sound through handling the Phaser knobs.

  Plans for next week:

  - Research about the Chorus effect

Week 16 (10th Jan 2022)

- Research about the Chorus effect and implement it to the synthesiser: Rate, Depth, Mix, Center Frequency, Feedback knobs Knobs
- With Oscilloscope, confirm the waveform of sound through handling the Chorus knobs.

  Plans for next week:

  - Research about Compressor

Week 17 (17th Jan 2022)

- Research about the Compressor effect and implement it to the synthesiser: Threshold, Ratio, Attack, Release Knobs
- With Oscilloscope, confirm the waveform of sound through handling the Compressor knobs.

  Plans for next week:

  - Research about Reverb

Week 18 (24th Jan 2022)

- Research about the Reverb effect and implement it to the synthesiser: Size, Damping, Width, Dry, Wet, Freeze, Delay
- With Oscilloscope, confirm the waveform of sound through handling the Reverb knobs.

  Plans for next week:

  - Fix the effector function's few errors: sound crack

Week 19 (7th Feb 2022)

- Modified the overall the code to remove noise immediately after moving the knobs and pressing the virtual piano keyboard.
- Modified the code to resizeable so that the size of the synthesiser can be adjusted according to the user's preference.

  Plans for next week:

- Make the save/load state option to the synthesiser

Week 20 (14th Feb 2022)

- Implement the reset/save/load state option to the synthesiser
- Make the reset/save/load option on the caption so that allows the user to save/load the specific state of the knobs for using the state again if that is satisfying for the user.
- Organize the position of the select Box&knobs on the program main board.

Plans for next week:

- Implement the virtual keyboard for standalone version of the synthesiser

Week 21 (21th Feb 2022)

- Implemented the virtual keyboard (mini keyboard) to the synthesiser

Plans for next week:

- Develop own oscilloscope for standalone version of the synthesiser

Week 22 (28th Feb 2022)

- Develop own oscilloscope for standalone version of the synthesiser
- Write a questionnaire for the user test (10 Questions).

Plans for next week:

- Develop own oscilloscope for standalone version of the synthesiser

- Get confirmation from the supervisor about the questionnaire

Week 23 (7th Mar 2022)

- Develop own oscilloscope for standalone version of the synthesiser
- Get confirmation from the supervisor (whether it is a good quality questionnaire or not) about the questionnaire.

Plans for next week:

- Develop GUI

- Correct and supplement the questionnaire and get confirmation from the supervisor.

Week 24 (14th Mar 2022)

- Develop GUI
- Correct and supplement the questionnaire and get confirmation from the supervisor. - Get confirm email on the 18th March 2022.

Plans for next week:

- System test and find the error & fix

- Get feedback from at least 5 users with questionnaire

Week 25 (21th Mar 2022)

- System test and find the error & fix.
- Get feedback from at least 5 users with questionnaire.

  Plans for next week:

  - Reflect the user feedback to the synthesiser

Week 26 (28th Mar 2022)

- Reflect the user feedback to the synthesiser: Supplementing the README.txt for an explanation detailed of installation part of the program.
- Make one image file to explain the functions of effectors.
- Start to write the Final Year Project Report - build the structure

  Plans for next week:

  - Reflect the user feedback to the synthesiser

  - Start to Write the Final Year Project Report

Week 27 (4th April 2022)

- Reflect the user feedback to the synthesiser: Add the Help features to the program (help buttons for pop up message box)
- Fix the design of the synthesiser program
- Get the confirmation of the end of the technical part of the synthesiser (6th April)
- Start to write the Final Year Project Report

  Plans for next week:

  - Write the overall Final Year Project Report (Promised due date: 19th April)

Week 28 (11th April 2022)

- Write the Final Year Project Report

  Plans for next week:

  - Get the feedback of the final report by the supervisor

  - Revise the Final Year Project Report

Week 29 (18th April 2022)

- Get the feedback on the final report from the supervisor

  - Receive the initial feedback from the supervisor (20th April 2022)

- Revise the Final Year Project Report by reflecting on the received feedback

- Send the next draft final report that reflecting the feedback from the supervisor (24th April 2022)

Plans for next week:

- Revise the Final Year Project Report

Week 30 (25th April 2022)

- Get the feedback of the final report by the supervisor

  - Receive the second feedback from the supervisor (29th April 2022)

- Revise the Final Year Project Report

  - Send the next draft final report that reflecting the feedback from the supervisor (29th April 2022)

  Plans for next week:

  - Revise the Final Year Project Report

Week 31 (2nd May 2022)

- Get the final feedback of the final report by the supervisor

  - Receive the third (final) feedback from the supervisor (4th May 2022)

- Revise the Final Year Project Report with feedback and submit the final version of the report via Canvas
- Informed to have a meeting in the next few days for final year project presentation.
- Prepare the presentation (Presentation date: 16th May 2022)

  Plans for next week:

- **Submit the Final version of the report & code via Canvas**

Week 32 (9th May 2022)

- **Submit the Report & Source code of the Final year project via Canvas (Due Date: May 10th 2022)**

A.2. Meeting logs

Meeting #0 : 22nd Sep 2021
Group Meeting (Zoom)

- Meeting with other students working on an individual project under the same supervisor
- Hear from the supervisor about the student's responsibility ; (deliver my tasks on time)
- Hear from the supervisor about the supervisor's responsibility ; (help student to make a good decision)
- Decide the time to have a personal meeting with the supervisor via email

Meeting #1 : 27th Sep 2021
One-to-One meeting (Personal)

- Meet with supervisor in personally
- Getting some advice about the Proposal Document from supervisor

  1) Try to write/note the references of all materials what I research for project proposal document

  2) Try to write the 'Fallback' (what to do is everything turns out hard) section for project proposal document
- Plans for until the next personal meeting

  1) Finish and send project proposal document to supervisor

  2) Research JUCE Framework

  3) Try to produce a simple oscillator module with several waveforms and controls for pitch (frequency) and amplitude that will also should work with the necessary GUI
- Decided that have a personal meeting with supervisor would Monday 4pm meet ; once every other week

Meeting #2 : 11th Oct 2021
One-to-One meeting (Personal)

- Showed the trial program produced during Week2 to the supervisor and had a short discussion on the problem of triangle waveform; Get advise about the using of VST virtual OSCILLOSCOPE that useful tool for checking the form of waves
- Before moving on to LPF later, recommended by supervisor to learn more terminology, "Nyquist principle, DISCONTINUTY, ALIASING"

Meeting #3 : 25th Oct 2021
Group meeting (Zoom)

- Discuss about the Interim Report
- Discuss about the way of organize the Interim report

Meeting #4 : 17th Nov 2021
Group meeting (Zoom)

- Discuss about the Resonance functions in filter and its means&works
- Discuss with the further progress after finishing the Interim report

  - Make a meeting schedule before the Christmas holidays

  - was advised to keep working
- Receive a brief suggestion of the final report structure

Meeting #5 : 15th Dec 2021
One-to-One meeting (Zoom)

- Discuss with the further progress plan & aim and time organize for next few weeks of holidays

  - was advised to keep working and make a balance for preparing other modules work and project

- Show the new function of the synthesiser (Phaser) and get the feedback

- Making the next meeting date

Meeting #6 : 19th Jan 2022
One-to-One meeting (Zoom)

- After the vaccination, couldn't make a Zoom meeting due to poor condition.

- Instead of canceling the meeting, I send a four-page PPT containing working progress so far and received feedback from the supervisor about further project plan.

Meeting #7 : 27th Jan 2022
One-to-One meeting (Personal)

- Discuss the progress on the final year project and show to the supervisor the implemented synthesiser program so far.

- Discuss the extended point of the project - Receive the suggestion extended objectives from the supervisor; connect the program with a real keyboard, USB network, polyphonic, etc.

- Receive the suggestion to have some evaluation part to the project to get feedback from other users (friends, etc)

- Considering the time and priority it takes to process the extension task, I was advised by the supervisor to make a extended objectives list and choose what to do for further works.

Meeting #8 : 14th Feb 2022
One-to-One meeting (Personal)

- Discuss the progress on the final year project and show to the supervisor the implemented synthesiser program so far. - virtual keyboard, save/load/reset functions

- Choose the extended point of the project - Develop the polyphonic system (Advised to research about voice allocation from supervisor)

- Advised to try to have a 'User Test', and prepare the questionnaire from supervisor (at least five testers, at least five questions)

- Discuss about the project poster : will have group session on next week

Meeting #9 : 11th Mar 2022
One-to-One meeting (Zoom)

- Discuss the further few week's plan of the project

- Last two tasks before starting to write the final report:

1. Considering the evaluation part of the synthesiser (User Test: Maybe use google form)

2. Keep working on the develop the polyphonic system

- Discuss the final report structure
- In order to carry out the User test through a great questionnaire, talked about submitting the questionnaire and 'User Testing Compliance Form' to the supervisor by next week (18th Mar 2022).

Meeting #10 : 6th April 2022
One-to-One meeting (Zoom)

- Get the confirm the end of the technical part of the synthesiser from the supervisor
- Discuss the result of the User Test
- Discuss the further evaluation part of the program;

1. Computation - CPU (how much CPU is consuming)

2. About the JUCE Framework application - something challenging/something good

- Get the confirm the final report structure from the supervisor
- Make the end due day of the draft version of the final report (19th April)

B. User Testing Compliance Form

B.1. User Testing Compliance Form for UG and PGT Projects School of Engineering and Informatics University of Sussex

This form should be used in conjunction with the document entitled "Research Ethics Guidance for UG and PGT Projects".

Prior to conducting your project, you and your supervisor will have discussed the ethical implications of your research. If it was determined that your proposed project would comply with **all** of the points in this form, then both you and your supervisor should complete and sign the form on page 3, and submit the signed copy with your final project report/dissertation.

If this is not the case, you should refer back to the "Research Ethics Guidance for UG and PGT Projects" document for further guidance.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

Investigators have a responsibility to protect participants from physical, mental and emotional harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, physical hazards or discomfort, emotional distress, use of sensory deprivation (e.g. ear plugs or blindfolds), sensitive topics (e.g. sexual activity, drug use, political behaviour, ethnicity) or those which might induce discomfort, stress or anxiety (e.g. violent video games), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.

2. The study materials were paper-based, or comprised software running on standard hardware.

Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and tablet computers is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

Participants cannot take part in the study without their knowledge or consent (i.e. no covert observation). Covert observation, deception or withholding information are deemed to be high risk and require ethical approval through the relevant C-REC.

If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, the data is to be published or there are future secondary uses of the data), then it will be necessary to obtain signed consent from each participant. Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script (see Appendix 1).

4. No incentives were offered to the participants.

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle. People volunteering to participate in research may

be compensated financially e.g. for reasonable travel expenses. Payments made to individuals must not be so large as to induce individuals to risk harm beyond that which they would usually undertake.

5. No information about the evaluation or materials was intentionally withheld from the participants.

Withholding information from participants or misleading them is unacceptable without justifiable reasons for doing so. Any projects requiring deception (for example, only telling participants of the true purpose of the study afterwards so as not to influence their behaviour) are deemed high risk and require approval from the relevant C-REC.

6. No participant was under the age of 18.

Any studies involving children or young people are deemed to be high risk and require ethical approval through the relevant C-REC.

7. No participant had a disability or impairment that may have limited their understanding or communication or capacity to consent.

Projects involving participants with disabilities are deemed to be high risk and require ethical approval from the relevant C-REC.

8. Neither I nor my supervisor are in a position of authority or influence over any of the participants.

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any study.

9. All participants were informed that they could withdraw at any time.

All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script (see Appendix 1).

10. All participants have been informed of my contact details, and the contact details of my supervisor.

All participants must be able to contact the investigator and/or the supervisor after the investigation. They should be given contact details for both student and supervisor as part of the debriefing.

11. The evaluation was described in detail with all of the participants at the beginning of the session, and participants were fully debriefed at the end of the session. All participants were given the opportunity to ask questions at both the beginning and end of the session.

Participants must be provided with sufficient information prior to starting the session, and in the debriefing, to enable them to understand the nature of the investigation.

12. All the data collected from the participants is stored securely, and in an anonymous form.

All participant data (hard-copy and soft-copy) should be stored securely (i.e. locked filing cabinets for hard copy, password protected computer for electronic data), and in an anonymised form.

---

*This checklist was originally developed by Professor Steven Brewster at the University of Glasgow, and modified by Dr Judith Good for use at the University of Sussex with his permission.

Project title: Design and implementation subtractive synthesiser with fixed signal path approach using JUCE Framework

Student's Name: Jihye Ahn

Student's Registration Number: 21906481

Student's Signature:

Date: 16th March 2022

Supervisor's Name: Dr Kingsley Sage

Supervisor's Signature:

Date: 18 March 2022

B.2. Introduction Scripts

The aim of this study is to investigate the suitability of a new software synthesiser program that uses subtraction synthesis. We cannot tell how good this program is unless we ask those people who are likely to be using it, which is why we need to run studies like these to find out whether this synthesiser program is easy and simple for users to use or not. I will give you some time to browse the synthesiser program, before asking you to answer some questions. You will use your own computer to install the synthesiser application on your computer and use it to make the sound you want.

I will be observing you while you perform the tasks to make the desired sound through the synthesiser program. If you have any questions, please ask me, and please let me know when you are finished playing and testing the synthesiser program and have answered all of the questions on the questionnaire.

I will ask you some questions at the end of the study. During the study, you will use your own computer to install the synthesiser application on your computer and use it to make the sound you want. Please remember that it is the system, not you, that is being evaluated for the program's easy-to-use/compatibility.

You are welcome to withdraw from the study at any time without giving any reason. Please be assured that any data collected will be stored securely and in an anonymous form. Your personal information (ex. Contact detail) including names will be kept separate from the information that you submitted. Any information you provide will be used solely for this project, and not shared with any other parties.

- *Do you agree to take part in this evaluation?* Y/N
- *Do you have any questions before we start?* Y/N

B.3. Debriefing Scripts

The main aim of this study is to investigate the usability of a new software synthesiser program that uses subtraction synthesis. In particular, I am looking to see whether this synthesiser program is easy and simple for users to use or not. In addition, I want to understand whether a program works properly without errors or critical problems or not in other computing environments and general user satisfaction of the synthesiser program.

- *Do you have any comments or questions about the experiment?* Y/N

Here are my contact details, and those of my supervisor, and please let me know if you have any further questions about this study.

- Email(student): ja519@sussex.ac.uk
- Email(supervisor): k.h.sage@sussex.ac.uk

Thank you for your help.

C. User Test - Questionnaire

Low risk project (multiple choice – M, short-answer question - S)

(1) Do you generally satisfied with the synthesiser application? : Multiple choice

No rating: I don't know how to answer this question.

1. I didn't understand what this program was doing and how to deal with it.

2. I understand some parts of this program but do not know how to deal with it.

3. I understand some parts of this program but found it difficult to create the sound I wanted.

4. I understand almost all parts of this program but found it a bit difficult to create the sound I wanted.

5. I worked how I expected it to and I could make interesting sounds I wanted with this program.

(2) Do you think the size of the synthesiser application screen as presented on the computer screen is appropriate? : Multiple choice

No rating: I don't know how to answer this question.

1. I think the size of the application screen is should be bigger.

2. I think the size of the application screen is should be smaller.

3. I think the size of the application screen is suitable but the font should be bigger.

4. I think the size of the application screen is suitable but the font should be smaller.

5. I think the size of this application screen and font is suitable.

(3) Are the values of the variables/parameters of each function provided in the synthesiser application sufficient to create the unique sound you want? : Multiple choice

No rating: I don't know how to answer this question.

1. I think the range value of the variables of each function is should be wider.

1.1. If you check the [1. I think the range value of the variables of each function is should be wider.] please specify the appropriate range of variables you think.

2. I think the range value of the variables of each function is should be more narrow.

2.1. If you check the [2. I think the range value of the variables of each function is should be more narrow.] please specify the appropriate range of variables you think.

3. I think the range value of the variables of each function is suitable.

(4) Were you able to understand and use the functions of all applications through files without having to get external help or search for functions separately? Was the description of the synthesiser application provided in the attached README.txt file sufficient? (Help/explanation) : Multiple choice

No rating: I don't know how to answer this question.

1. I didn't understand what this program was doing and cannot use it with the given README.txt and description.

2. I didn't understand general parts of this program with the given README.txt and description. In order to use the program, I need to search for almost all functions on my own.

3. I understand some parts of this program with the given README.txt and description. In order to use the program, I need to search for the rest functions on my own.

4. I understand almost all parts of this program with the given README.txt and description. The given description satisfies most of my curiosity about the program, but I hope more details will be provided.

5. I worked how I expected it to and I could make interesting sounds I wanted with this program with given README.txt and description. All functions are clearly understood and no further explanation is required.

(5) Was the process of installing and using the synthesiser application for the first time simple? (Easy to use) : Multiple choice

No rating: I don't know how to answer this question.

1. I can't install and use this synthesiser program on my own.

2. The process of installing this synthesiser program is very complicated and takes a long time. I am confused about how to install it.

3. The process of installing this synthesiser program is a complicated, and the provided README.txt and description are not sufficient for installing the program. Another search process is required.

4. The process of installing this synthesiser program is a little complicated, but the provided README.txt and description are sufficient and allow me to install it on my own.

5. The process of installing the synthesiser program is simple, and the explanation of the installing process is sufficient.

(6) When using the synthesiser application, can you adjust the function smoothly as you intended without lag or errors? : Multiple choice

No rating: I don't know how to answer this question.

1. Just program shut down (critical error) when I try to make the sound.

2. When I tried to make the sound I wanted, there were many errors/lags in the program, so I found it is hard to make the sound I wanted.

3. When I tried to make the sound I wanted, there were not many errors in the program, but have a heavy lags problem. It takes a long time to handle the program.

4. There are a few errors when running the program, and a few lag problems have been experienced, but they are not serious.

5. The program is implemented smoothly without errors or lags, and there was no problem with the system while making the desired sound.

(6-1) If there error/lag problem exists, please write the brief circumstance that which the problem occurs. Plus, if it is a lag problem, Please write down the time it took. : Short-answer question

(7) Are you satisfied with the synthesiser application screen design? (GUI) : Multiple choice

No rating: I don't know how to answer this question.

1. The design elements of the program are not attractive, and the interface is not visually clear.

2. The design elements of the program are a bit attractive, and the interface is not visually clear.

3. The design elements of the program are a bit attractive, and the interface is a bit visually clear.

4. The design elements of the program are attractive, and the interface is a bit visually clear.

5. The design elements of the program are attractive, and the interface is visually clear.

(8) Would you use this application in your own music production? : Multiple choice

No rating: I don't know how to answer this question.

1. No. There are no merits to using this program. It is hard to install and use it.

2. Maybe not. I found some good parts of the program, but it is hard to install and use.

3. Not sure. I found some good parts of the program and it is easy to install but it's hard to use.

4. Maybe I consider it. I found some good parts of the program even there are a few problems.

5. Yes. I found many good parts of the program and I can make the wanted sound with this program.

(9) Did you feel that you could create a full range of interesting and original sounds with this synthesiser application? : Multiple choice

No rating: I don't know how to answer this question.

1. No. Through this program, I could only make limited sounds.

2. Maybe not. I could make a narrow range of sound with this program, but I felt it is limited.

3. Not sure. I could make some range of sound with this program, but I felt it is limited.

4. Maybe yes. I could make almost all of the sounds that I wanted with this program.

5. Yes. I think I was able to make various sounds through this program and create endless sounds through built-in functions.

(10) Are there any features you would like to see added to this synthesiser application? (Extra) : Short-answer question

(11) Feel free to leave any good points or problems you felt while using the synthesiser application. (Extra) : Short-answer question

D. User Test - Result

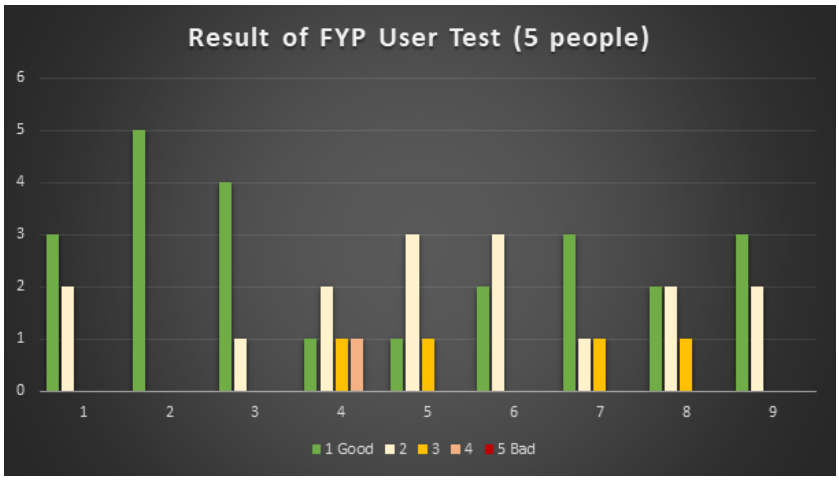

Figure 38: Result of User Test; Answers to Multiple choice questions)

Answers to Short answer Question

(Q. 6-1) 2 answers:

- There is some small lag problem exists when I try to load or save the state. (about 15 sec)

- Load a state file that exists in the presets folder, the program shows a short pause of about 20 sec.

(Q. 11) 3 answers:

- It will be good to contain some kind of 'help feature' that can be enabled to see the exact function of the effector. I guess many users who are unfamiliar with synth should search about the effectors or other functions if there are no Help features before using the program.

- Providing a more detailed explanation of the program installation progress will help to improve your program's completeness.

- I think need to provide a bright color version of the program or change some parts of the design.

E. Bibliography

References

[1] Reiss, J. and McPherson, A., 2015. Audio effects. Boca Raton, FL: CRC Press.

[2] Robinson, M., 2013. Getting Started with JUCE. Birmingham: Packt Publishing.

[3] Pirkle, W. C. (2015). Designing software synthesizer plug-ins in C++ for rackafx, VST3, and audio units. Focal Press.

[4] Sung, K., & Lee, Y. (2019). A Study of Analysis about Virtual Musical Instruments' Timbre - Focused on Violin, Erhu, Haegeum -. Journal of the Korea Entertainment Industry Association.

[5] JUCE Framwork github [online] Available at: https://github.com/juce-framework/JUCE [Accessed 27 Sep 2022].

[6] JUCE official site [online] Available at: https://juce.com/ [Accessed 27 Sep 2021].

[7] Sievers, B., 2007. Synthesis Basics. [online] Beausievers.com. Available at: http://beausievers.com/synth/synthbasics/ [Accessed 28 Sep 2021].

[8] Docs.juce.com. 2020. JUCE: Tutorial 02: Projucer Basics [online] Available at: https://theaudioprogrammer.com/tutorial-02-projucer-basics/ [Accessed 29 Sep 2021].

[9] The Basics of Synthesis and Sound Design [online] Available at: https://stikeys.co/tutorials/the-basics-of-synthesis-and-sound-design/ [Accessed 29 Sep 2021].

[10] ALBOUY, A., 2017. Audio Plugins with Faust and JUCE. [online] Ccrma.stanford.edu. Available at: https://ccrma.stanford.edu/workshops/faust-juce-21/ [Accessed 15 December 2021].

[11] Understanding Sound Waves and Waveforms [online] Available at: https://stikeys.co/tutorials/understanding-sound-waves-and-waveforms/ [Accessed 29 Sep 2021].

[12] Docs.juce.com. 2020. JUCE:: Tutorial: Create A Basic Audio/MIDI Plugin, Part 1: Setting Up. [online] Available at: https://docs.juce.com/master/tutorial_create_projucer_basic_plugin.html [Accessed 29 Sep 2021].

[13] Docs.juce.com. 2020. JUCE:: Tutorial: Build a MIDI synthesiser [online] Available at: https://docs.juce.com/master/tutorial_synth_using_MIDI_input.html [Accessed 29 Sep 2021].

[14] Sound synthesiser #2 - Oscillators & Envelopes [online] Available at: https://youtu.be/OSCzKOqtgcA [Accessed 29 Sep 2021].

[15] Let's Build a Synth with Juce Part 5 - Creating ADSR Sliders [online] Available at: https://youtu.be/PfaYCyVwB60 [Accessed 29 Sep 2021].

[16] Learning Synthesis: Noise [online] Available at: https://www.perfectcircuit.com/signal/learning-synthesis-noise [Accessed 29 Sep 2021].

[17] Sine, Square, Triangle, Saw [online] Available at: https://www.perfectcircuit.com/signal/difference-between-waveforms [Accessed 29 Sep 2021].

[18] Faronbi, D. and Gilmore, A., 2021. Synthesizer Parameter Approximation by Deep Learning. [online] DigitalCommons@UNO. Available at: https://digitalcommons.unomaha.edu/university_honors_program/151/ [Accessed 12 October 2021].

[19] Quick Guide to Envelopes [online] Available at: https://making-music.com/quick-guides/envelopes/ [Accessed 18 Oct 2021].

[20] Wikimedia Foundation. Waveform. [online] Available at: https://en.wikipedia.org/wiki/Waveform [Accessed 18 Oct 2021].

[21] Let's Build a Synth with Juce Part 8 - FM Frequency Modulation Synthesis [online] Available at: https://www.youtube.com/watch?v=Q9AynPrcHls&t=1s [Accessed 18 Oct 2021].

[22] Analog Synthesizer Tutorial - AM/FM Modulation [online] Available at: https://youtu.be/RISVHAzHO7s [Accessed 19 Oct 2021].

[23] Gordon Reid. 2000. An Introduction To Frequency Modulation. [online] Available at: https://www.soundonsound.com/techniques/introduction-frequency-modulation [Accessed 19 Oct 2022].

[24] The "Sound" of Performance Data, Part 2 - Aliasing [online] Available at: https://www.aternity.com/blogs/the-sound-of-performance-data-part-2/ [Accessed 19 Oct 2021].

[25] Sampling, Aliasing & Nyquist Theorem [online] Available at: https://youtu.be/yWqrx08UeUs [Accessed 19 Oct 2021].

[26] Docs.juce.com. 2020. Tutorial: Introduction to DSP [online] Available at: https://docs.juce.com/master/tutorial_dsp_introduction.html [Accessed 1 Nov 2021].

[27] Docs.juce.com. 2020. Tutorial: Build a white noise generator [online] Available at: https://docs.juce.com/master/tutorial_simple_synth_noise.html [Accessed 10 Nov 2021].

[28] Docs.juce.com. 2020. Tutorial: Build a white noise generator [online] Available at: https://docs.juce.com/master/tutorial_simple_synth_noise.html [Accessed 10 Nov 2021].

[29] Docs.juce.com. 2020. dsp::StateVariableTPTFilter< SampleType > Class Template Reference [online] Available at: https://docs.juce.com/master/classdsp_1_1StateVariableTPTFilter.htmll [Accessed 3 Dec 2021].

[30] Docs.juce.com. 2020. MIDIInput Class Reference [online] Available at: https://docs.juce.com/master/classMIDIInput.html [Accessed 10 Dec 2021].

[31] Docs.juce.com. 2020. MIDIKeyboardComponent Class Reference [online] Available at: https://docs.juce.com/master/classMIDIKeyboardComponent.html [Accessed 10 Dec 2021].

[32] Zavalishin, V., 2015. [online] Native-instruments.com. Available at: https://www.native-instruments.com/fileadmin/ni_media/downloads/pdf/VAFilterDesign_1.1.1.pdf [Accessed 13 December 2022].

[33] Digital IIR Filter, Ivan Cohen [online] Available at: https://youtu.be/esjHXGPyrhg [Accessed 15 Dec 2021].

[34] The digital state variable filter [online] Available at: http://www.earlevel.com/main/2003/03/02/the-digital-state-variable-filter/ [Accessed 15 Dec 2021].

[35] Matching of Forward Euler SVF to Trapezoidal SVF [online] Available at: https://cytomic.com/index.php?q=technical-papers [Accessed 15 Dec 2021].

[36] Digital State-Variable Filters [online] Available at: https://ccrma.stanford.edu/~jos/svf/ [Accessed 20 Dec 2021].

[37] TIME-VARYING FILTERS FOR MUSICAL APPLICATIONS [online] Available at: http://www.dafx14.fau.de/papers/dafx14_aaron_wishnick_time_varying_filters_for_.pdf [Accessed 20 Dec 2021].

[38] Docs.juce.com. 2020. dsp::Phaser< SampleType > Class Template Reference [online] Available at: https://docs.juce.com/master/classdsp_1_1Phaser.html [Accessed 8 Jan 2022].

[39] What is JUCE? [online] Available at: https://stackshare.io/juce [Accessed 9 Jan 2022].

[40] JUCE vs T3 [online] Available at: https://stackshare.io/stackups/juce-vs-t3 [Accessed 9 Jan 2022].

[41] Orlarey, Yann & Fober, Dominique & Letz, Stephane. (2009). FAUST : an Efficient Functional Approach to DSP Programming. [Accessed 9 Jan 2022].

[42] Marcak, M., 2021. REACT NATIVE PROS AND CONS IN 2021 [online] Pagepro.co. Available at: https://pagepro.co/blog/react-native-pros-and-cons/ [Accessed 9 Jan 2022].

[43] Viktor, 2017. Cross-Platform App Development Or Why Should I Choose Qt? [online] Available at: https://blog.vakoms.com/qt-cross-platform-application-framework-is-it-worth-it/ [Accessed 9 Jan 2022].

[44] Docs.juce.com. 2020. dsp::Chorus< SampleType > Class Template Reference [online] Available at:
https://docs.juce.com/master/classdsp_1_1Chorus.html [Accessed 10 Jan 2022].

[45] Docs.juce.com. 2020. dsp::Compressor< SampleType > Class Template Reference [online] Available at:
https://docs.juce.com/master/classdsp_1_1Compressor.html [Accessed 20 Jan 2022].

[46] Docs.juce.com. 2020. dsp::Reverb< SampleType > Class Template Reference [online] Available at:
https://docs.juce.com/master/classdsp_1_1Reverb.html [Accessed 22 Jan 2022].

[47] Docs.juce.com. 2020. LookAndFeel Class Reference [online] Available at:
https://docs.juce.com/master/classLookAndFeel.html [Accessed 22 Jan 2022].

[48] FIR vs IIR filtering [online] Available at:
https://www.minidsp.com/applications/dsp-basics/fir-vs-iir-filtering [Accessed 22 Jan 2022].

[49] Chrissy Tignor. 2021. 5 Basic Audio Effects Every Vocalist Should Know [online] Available at:
https://www.izotope.com/en/learn/5-basic-audio-effects-every-vocalist-should-know.html [Accessed 30 Jan 2022].

[50] JUCE Forum [online] Available at:
https://forum.juce.com/t/where-to-begin/14588/3 [Accessed 18 Mar 2022].

[51] MusicRadar. 2022. The best VST/AU synth plugins 2022: the top analogue, wavetable, modular and free synthesizers. [online] Available at:
https://www.musicradar.com/news/best-vst-synth-plugins [Accessed 25 Mar 2022].

[52] Tutorials - git clone [online] Available at:
https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone [Accessed 12 April 2022].

[53] Andrew Eisele. 2016. A Guide to Analog Subtractive Synthesis with the Moog Phatty Keyboard. [online] Available at:
https://www.musicradar.com/news/best-vst-synth-plugins [Accessed 20 April 2022].

F. Project Proposal Document

Digital Music Synthesiser

Individual Project Proposal Documents

Working title
: Design and implementation subtractive synthesiser with fixed signal path approach using JUCE Framework

Candidate number : 221455
Supervisor : Dr Kingsley Sage
Date of Submission: 4th Oct 2021

Description

Develop a desktop or mobile based digital music synthesiser, either by modelling conventional subtractive synthesis or by other means. Design can be quite varied with functional modules including, although not limited to, oscillators, filters, amplifiers, effects, modulation sources (such as envelope generators and low frequency sources). Could be a fixed signal path instrument or a modular/semi modular approach. Could be developed as an AU/VST plugin instrument or a standalone application.

Skills

- Programming
- Maths
- Music technology
- Software Engineering

Topics

- Music Technology
- DSP
- Digital Signal Processing

Aims

The aims for this project is implementing and designing the fixed signal path subtractive synthesiser among various kinds of digital music synthesisers via C++ and JUCE Framework. Synthesiser refers to an electric instrument that can create a new sound by synthesizing sounds of various frequencies or waveforms and electronically modulate stored tones according to the user's capabilities. When using a synthesiser, users are able to handle an expedient method to record complicate songs by imitating scores that are difficult to play unless users are a

keyboard or notes of other instruments, and since users do not need actual instruments, they can play much more economically. In addition, the synthesiser also serves as an input device for DAW programs with MIDI sequencer/MIDI editing functions running on a computer. Skilled experts even make up an orchestra with this single electronic instrument. Moreover, when creating simple musical instrument sounds, it can add various effects for notes, (such as voice, vocoding, or effect using each internal effector of synthesiser), and those advantages making it indispensable to modern musicians.
The project aims to develop desktop-based subtractive synthesiser applications using subtraction synthesis among digital synthesisers that play a major role in this modern music genre. Among the VST and Standalone formats, the Standalone format will be adopted so that users can easily create unique sounds using a keyboard and mouse their own through the interface provided by the JUCE Framework. Various functions will be implemented without restrictions on filters, oscillators, effectors and modulation sources used in the subtractive synthesiser to understand the software functions of synthesisers distributed in the modern synthesiser market and develop software engineering skills.

Core Objectives

1. Research and understand the functions and characteristics of conventional subtractive synthesisers to distinguish them from other types of synthesisers and design & implement the program using C++ & JUCE Framework.

2. Choosing various functions from oscillators, filters, amplifiers, effects, and modulation sourses that conventional synthesisers should have for being implemented.

3. Making the user can play the unique sound by just using a mouse & keyboard so that it can be used as a desktop-based standalone application program.

4. Design a GUI that allows users to play sounds using the many functions implemented in Core Objective 2.

5. Provide the sound wave graph played by the user and show it through the interface for analyzing the waveform which notes.

Extended Objectives

- Working with MIDI interface
- Provide VST version of the subtractive synthesiser
- Provide recording & playback functions
- Provide sequencer(DAW) fuctions
- Generating polyphonic synthesiser
- Should contain various effect unit ; Distortion, Chorus, Phaser, Flanger, Reverb, Echo, Delay, Loop, Compressor etc.
- Could be developed as synthesiser use modular approach

Fallback Objectives

- Implementing and designing the simplest standalone synthesiser
- Input basic filters, effects, modulation sources provided from JUCE Library

  - Implementing at least functions ; monophonic systems with a simple Low-Pass filter and a virtual keyboard

- Design a simple synthesiser using basic GUI sources (vertical slider) provided from JUCE Library

Relevance

The project requires clear comprehension of Fourier formulas used to generate a sound waveform, file formats of VST/Standalone, software engineering, GUI development, understanding of DSP, and time-management ability to implement desktop-based applications with C++ within a fixed time. Therefore, this works leads particularly immersed in developing my own computing skills while designing the subtractive synthesiser, and progressing various capabilities such as planning, development, documentation, testing, and evaluation of software systems.

- Computing Skills, Software Engineering skills, Math, GUI Design skills, Time-management

Personal Weekly Time Table

<table>
<tr><th></th><th>Monday</th><th>Tuesday</th><th>Wednesday</th><th>Thursday</th><th>Friday</th><th>Saturday</th></tr>
<tr><td>09:00</td><td rowspan="3">IAM</td><td></td><td>Knowledge Reasoning</td><td>Knowledge Reasoning</td><td></td><td></td></tr>
<tr><td>10:00</td><td></td><td></td><td></td><td></td><td rowspan="2">Project</td></tr>
<tr><td>11:00</td><td></td><td rowspan="2"></td><td></td><td></td></tr>
<tr><td>12:00</td><td></td><td></td><td></td><td>Individual Project</td><td></td></tr>
<tr><td>13:00</td><td></td><td rowspan="2">Project</td><td></td><td rowspan="2">Project</td><td></td><td></td></tr>
<tr><td>14:00</td><td>Knowledge Reasoning</td><td></td><td></td><td></td></tr>
<tr><td>15:00</td><td></td><td></td><td></td><td></td><td rowspan="2">Cognitive Neuroscience</td><td></td></tr>
<tr><td>16:00</td><td></td><td></td><td></td><td>Individual Project</td><td></td></tr>
<tr><td>17:00</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>18:00</td><td rowspan="3">Project</td><td></td><td rowspan="2">Project</td><td></td><td></td><td></td></tr>
<tr><td>19:00</td><td></td><td></td><td></td><td></td></tr>
<tr><td>20:00</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

- Should spend 10-15 hours a weeks for individual project
- May use more/less time for the project due to each weeks' schedules.
- The time planner above is not absolute, will be managed by flexible way.

References

[1] Faronbi, D., & Gilmore, A. (2021). Synthesizer Parameter Approximation by Deep Learning

[2] Pirkle, W. (2014). Designing software synthesizer plug-ins in C++: For RackAFX, VST3, and Audio Units. Routledge.

[3] The Basics of Synthesis and Sound Design [online] Available at: https://stikeys.co/tutorials/the-basics-of-synthesis-and-sound-design/ [Online; Accessed 29-Sep-2021]

[4] Understanding Sound Waves and Waveforms [online] Available at: https://stikeys.co/tutorials/understanding-sound-waves-and-waveforms/ [Online; Accessed 29-Sep-2021]

[5] Docs.juce.com. 2020. JUCE:: Tutorial: Create A Basic Audio/MIDI Plugin, Part 1: Setting Up. [online] Available at: https://docs.juce.com/master/tutorial_create_projucer_basic_plugin.html [Online; Accessed 29-Sep-2021]

[6] Docs.juce.com. 2020. JUCE:: Tutorial: Build a MIDI synthesiser [online] Available at: https://docs.juce.com/master/tutorial_synth_using_midi_input.html [Online; Accessed 29-Sep-2021]

[7] Sound Synthesizer #2 - Oscillators & Envelopes [online] Available at: https://youtu.be/OSCzKOqtgcA [Online; Accessed 29-Sep-2021]

[8] Let's Build a Synth with Juce Part 5 - Creating ADSR Sliders [online] Available at: https://youtu.be/PfaYCyVwB60 [Online; Accessed 29-Sep-2021]

[9] Learning Synthesis: Noise [online] Available at: https://www.perfectcircuit.com/signal/learning-synthesis-noise [Online; Accessed 29-Sep-2021]

[10] Sine, Square, Triangle, Saw [online] Available at: https://www.perfectcircuit.com/signal/difference-between-waveforms [Online; Accessed 29-Sep-2021]