Evan Sitt
C3JI5D
Application Development Documentation

The following contains the user guide and developer guide for the synthesizer. Release builds are included within the repository along with a sample FL Studio project utilizing the synthesizer.

The repository for this project can be found at

https://github.com/ParadoxChains/AppDev-Git-SITT_EVAN-C3JI5D

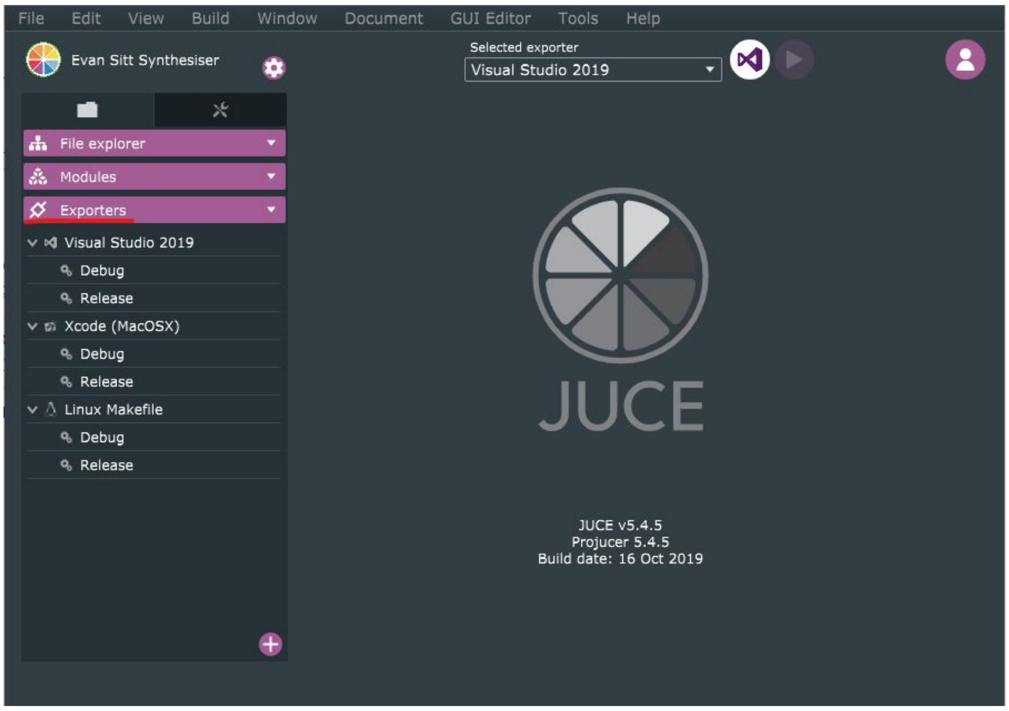
Required software:

- **JUCE** framework
- **Projucer**
- **ASIO** drivers
- Compatible Host Program
- **C++ 11** or newer

<u>User Guide</u>

Installation

1. Clone or Download the release of the synthesizer from https://github.com/ParadoxChains/AppDev-Git-SITT_EVAN-C3JI5D
2. If you are using the prebuilt binaries, please skip to step 7.
3. For building the source code, the **Projucer** development application is required, it can be downloaded from https://juce.com/discover/projucer
4. After starting up the **Projucer** program, open the `.jucer` file found at `TOP-LEVEL-DIRECTORY\C3JI5D-Synthesiser\C3JI5D-Synthesiser.jucer`
5. Select your chosen exporter from the lower left menu. If your preferred exporter is not present, use the button in the lower right side of the Exporter menu to add a new exporter of your preference.



Selected exporter: Visual Studio 2019

JUCE v5.4.5
Projucer 5.4.5
Build date: 16 Oct 2019

6. Upon opening your exporter of choice, export the project.
7. Determine if you would like to use the **VST3** or **Standalone** version of the synthesizer.
    a. If you are using the **VST3** version of the plugin, please open your choice of host program (typically a **Digital Audio Workstation**).

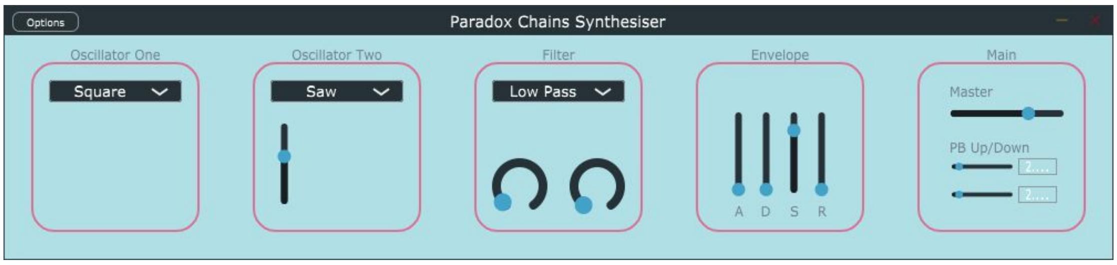
i. Set up the required routing for MIDI input to the plugin's MIDI input port.

ii. Set up the required routing for audio output from the plugin's two audio output ports to the respective ASIO.

b. If you would prefer to use the **Standalone** version of the synthesizer, please perform the necessary setup to route MIDI input and audio output to your computer's hardware.

User Interface

The synthesizer graphical user interface is as follows:



- The **Oscillator One** box contains the dropdown menu to select the waveform type of the primary oscillator.
  - Options for the waveform type are Square, Saw, Sine.
- The **Oscillator Two** box contains the dropdown menu to select the waveform type of the secondary oscillator. An additional slider is present to allow for adjusting the level of the second oscillator.
  - Options for the waveform type are Square, Saw, Sine.
  - The slider will adjust the linear multiplier applied to the secondary oscillator's output.
- The **Filter** box contains a dropdown menu and two rotary dials.
  - The dropdown menu contains three frequency-based filter options.
    - The available filters are the Low Pass filter, High Pass filter, and Bandpass filter.
  - The rotary dials control the parameters of the selected filter.
    - The left dial controls the frequency cutoff of the filter.
    - The right dial controls the resonance of the filter.
- The **Envelope** box contains four sliders to control the ADSR Envelope.

    - A corresponds to Attack time.
    - D corresponds to Decay time.
    - S corresponds to Sustain level.
    - R corresponds to Release time.
- The **Main** box contains three sliders to control the final output.
    - The Master slider controls the output volume.
    - The second slider controls pitch bend up.
    - The third slider controls pitch bend down.

Developer Guide

Overview

This digital waveform synthesizer is built upon the **JUCE** framework, built within the **C++** language. The **Projucer** development application is required to generate the projects for the exporters. The **Maximilian** framework is also used for digital synthesis operations.

The following sections will describe each of the main classes, their purposes, and their relations to each other.

PluginEditor

The **PluginEditor** class is the main body of the application. As such, the **PluginEditor** will contain the **PluginProcessor** class and one instance of each graphical user interface class.

The constructor of the **PluginEditor** is responsible for initializing each element to connect to the **PluginProcessor**, and adding each element to the graphical user interface.

PluginProcessor

The **PluginProcessor** class handles the MIDI input and audio output of the synthesizer. The functions within it correspond to various parts of the processing pipeline. A number of these are implemented by default by the **JUCE** framework. Additionally, the **PluginProcessor** also includes a **Synthesiser** and a **SynthVoice**.

The constructor function initializes the **Synthesiser** and **SynthVoice**, along with constructing and initializing the `AudioProcessorValueTree` which stores the current state of the synthesizer and handles passing the state to and from the host program. The `prepareToPlay` function initializes the processor parameters and state. This is separated from the constructor for better and clearer handling. The `processBlock` function is the main portion that handles digital synthesis. This function also serves to retrieve the current state of the synthesizer.

SynthSound

SynthSound is a simple class that can allow for sophisticated synthesizers which will operate conditionally on notes and channels, however, this synthesizer maintains a simplistic implementation.

SynthVoice

The **SynthVoice** class is the main processing portion of the synthesizer. This class defines a possible voice for the synthesizer, of which multiple may be added to the synthesizer as needed for polyphony.

In this synthesizer, the different waveforms are united within a singular voice class. This **SynthVoice** class contains functions that interface with the graphical user interface controls and retrieves their parameters. The `renderNextBlock` function will be called by the `processBlock` from the **PluginProcessor**, and as such will contain the main processing loop that will handle input from and output to the audio buffer.

The `startNote` and `stopNote` functions control parameter initialization and cleanup for the voice. The setOscType function will take the waveform type parameters to generate values based upon the frequency provided, and output the resulting value to be written to the audio buffer.

Oscillator

The **Oscillator** class is a graphical user interface class. Its purpose is to set up the respective portion of the GUI for the primary oscillator and connect the GUI controls to the `AudioProcessorValueTree` so that the respective parameters can be handled to and from the host program.

The **Oscillator** class contains only a dropdown menu for selecting the waveform type of the first oscillator. The `paint` function handles the graphical drawing. The `resized` function handles redrawing upon resizing the window.

Oscillator2

The **Oscillator2** class is a graphical user interface class. Its purpose is to set up the respective portion of the GUI for the secondary oscillator and connect the GUI controls to the `AudioProcessorValueTree` so that the respective parameters can be handled to and from the host program.

The **Oscillator2** class contains a dropdown menu for selecting the waveform type of the secondary oscillator and an additional slider for controlling the mix level of the secondary oscillator output. The `paint` function handles the graphical drawing. The `resized` function handles redrawing upon resizing the window.

Filter

The **Filter** class handles the graphical user interface for the filter. Its purpose is to set up the parameter controls for the filter and connects the controls to the `AudioProcessorValueTree` for the parameters to be handled between the synthesizer and the host program.

The **Filter** class contains a dropdown menu for selecting the type of frequency-based filter, a rotary dial for controlling the filter frequency cutoff, and a rotary dial for controlling the filter resonance. The `paint` function handles the graphical drawing. The `resized` function handles redrawing upon resizing the window.

Envelope

The **Envelope** class is a graphical user interface class. This class sets up the envelope portion of the GUI and connects the controls to the `AudioProcessorValueTree` for the parameters to be handled to and from the host program.

The **Envelope** class contains four sliders: one for the attack time, one for the decay time, one for the sustain rate, and one for the release time. The `paint` function handles the graphical drawing. The `resized` function handles redrawing upon resizing the window.

Frontgui

The **Frontgui** class is the graphical user interface class handling the controls of the Main portion of the user interface and connects the controls to the `AudioProcessorValueTree` for the passing of parameters to and from the host program.

The **Frontgui** class contains three sliders: one for the master gain, one for pitch bend up, and one for pitch bend down. The `paint` function handles the graphical drawing. The `resized` function handles redrawing upon resizing the window.