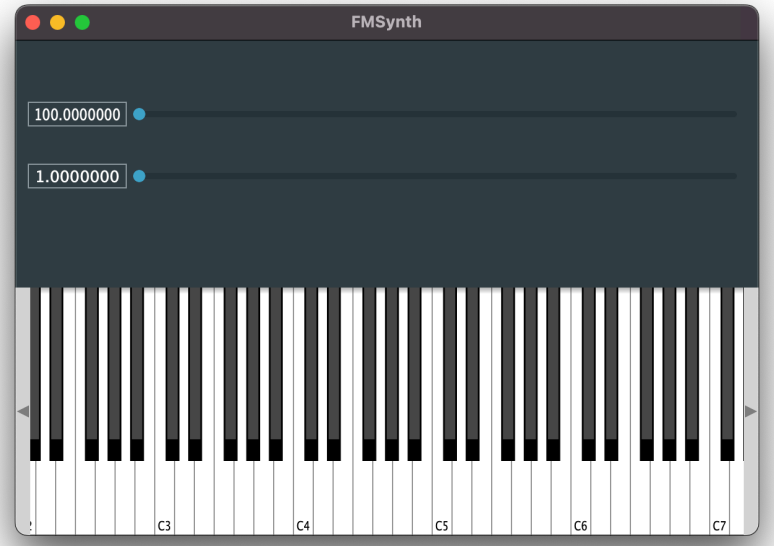
# ***FM SYNTH***

In the following document I am going to explain the main features of the standalone audio application that I have developed as a final project of Music Systems Programming 2.

For this project I have used the JUCE framework in which C++ is integrated and I have been inspired by several tutorials related to sound synthesis that are explained in the web page. Below, I will explain the main classes defined and their implementation.




Basically it is a midi synth where you can adjust the modulation index and the modulation frequency. To program the application I have adapted and modified classes from the tutorials mentioned above, mainly the program is structured as follows:

**- MainContentComponent Class:**

- Represent the UI layout and the UI parameter definition, the constructor initialises slider range, keyboard and set audio source channels.
- **resized()** method represent the bounds and layout of the UI
- **prepareToPlay()** and **getNextAudioBlock()** method overrides in the synth audio source class that is in charge of the synthesis
- **timerCallback()** inherits from juce::Timer and it executes periodically based on the setTimer defined in the MainContentComponent constructor. This method helps us to make the interaction with the user smoother when playing notes on the piano.

**- SynthAudioSource Class:**

- Inherits from juce::AudioSource and initialises the synthesiser voices in order to play it polyphonic notes
- **getNextAudioBlock()** overrides in MainContentComponent as I mention, and is where the incoming MIDI and audio process in the buffer.
- **prepareToPlay()** set the current sample rate.

**- FMWaveVoice Class:**

- Inherits from juce::SynthesiserVoice and is responsible for generating audio samples for each note.
- **canPlaySound()** method checks if the sounds its passing trough the class
- **prepareToPlay()** set the current sample rate.
- **startNote()** set the parameters when the note is played and **stopNote()** reset the values when the note is not played.
- **renderNextBlock()** this is the most important method where the processing is actually done. Managing the tail values we are able to know if the note is playing or not and it calculates the current sample using the formula.

$$S_{out} = sin(\omega + I_{mod} * sin(\omega_{mod}))$$

- **updateAngleDelta()** and **updateAngleDeltaMod()** are methods that I created to clean a little bit the code and update the phase values for the modulated and non modulated signals.

**- FMWaveSound Class:**

- Inherits from juce::SynthesiserSound and is responsible for generating audio samples for each note.
- Is the simpler class where the **appliesToNote()** and **appliesToChannel()** methods. The first function indicates that the sound is applicable to MIDI notes and the second function indicates that the sound is applicable to all MIDI channels.

I will explain better the implementation of these classes in a video that I will deliver with the project, finally I want to dedicate a last section of things to improve that will be completed with the video.

**- Things to improve:**

- More **separation between the UI and processing logic**, this could be made translating the audio application to a plugin that has more predefined classes in this sense.
- Clean up the code in the renderNextBlock to make it easier to make changes to the code for the future.
- I**mprove sound synthesis** by creating a separate class for the oscillator, this can be done with juce::dsp::Oscillator. It would also be possible to apply a table-lookup logic to create cleaner harmonics.
- **Better memory management** using smart pointers (std::unique_ptr and std::shared_ptr) and setting certain flags to prevent excessive memory consumption by the user.
- As a main goal I would like to migrate this application to iOS app or VST3 plugin. I have been following tutorials to program a plugin but I preferred to focus on the basic structure of c++, understanding the definition and implementation of different classes.separation between the UI and processing logic, this could be made translating the audio application to a plugin that has more predefined classes in this sense.