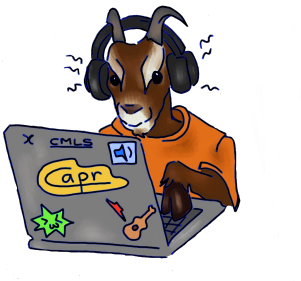
# Il Frattalatore

**CAPR** group:


**C**laudio, Eutizi (ID: 10812073)
**A**ndrés, Bertazzi (ID: 10488849)
**P**ierluigi, Tartabini(ID: 10845797)
**R**iccardo, Martinelli (ID: 10456202)


Juce Homework assignment #4
**Computer Music: Languages & Systems**
Politecnico di Milano




May 19, 2022

# Contents

# 1 Introduction

The **Frattalatore** is an additive synthesizer that exploits the fractal mathematical systems to produce singular sounds. It has been implemented using `Juce 6` and external `C++` libraries: `SFML` and `oscpack`.
The Frattalatore outputs a sum of 5 oscillators' contributions, related each other by a fractal set of parameters. All the oscillators take as input the same `MIDI` note input given by a piano keyboard or by the VST virtual keyboard, but they are set with different sets of parameters that come from the fractal computation. The Frattalatore can be used either as a standalone application or as a VST in a DAW, such as Ableton, Pro Tools etc.

# 2 What can the user do?

The Frattalatore has two modes of operations, depending on which are the user's needs. In order to exploit the full potential of the synth, the user has to open two `.exe` files: the **FractalWindow.exe** and the **Frattalatore.exe**.
Opening the FractalWindow executable file, a menu window appears and the user can choose which fractal to "play" with the Frattalatore: **Mandelbrot, Julia set 1, Julia set 2 and Burning Ship**.
Then, an image of the chosen fractal appears in another window and the user can choose a point from that (all the controls of the fractal window are described below in the next sections). In the same moment the `FM knobs` of the synth's oscillators will will be set with different values taken from the chosen fractal point and its next recursive five points. This mechanism proposes a sound, that will be more or less harmonic depending on the more or less belonging of the chosen first point on the fractal.
The user can play notes with a MIDI keyboard and the final result will be a sum of the fundamental note and the 5 sound's oscillators modulated previously.

# 3 GUIs

The GUIs of the **Frattalatore** project have been made using different libraries, and they communicate with each other through **OSC Messages**.
In particular, the GUI in the **C++** part is based on the `SFML` library that allows to manage events and complex mathematical iterations.

## 3.1 Menu Window

The first window is a Menu where the user can choose the type of Fractal. The choice must be done with the keyboard's arrows and then selected with Enter button.

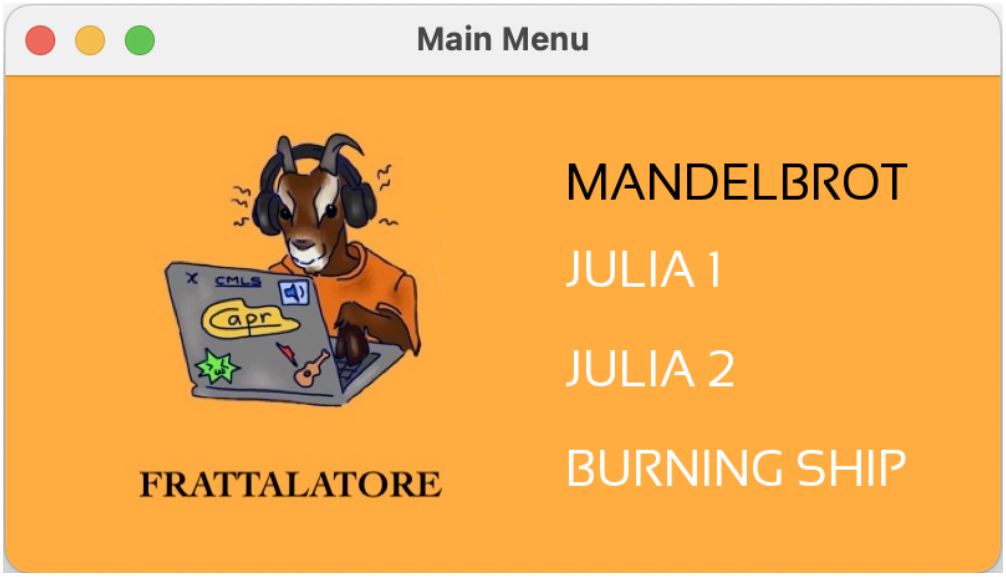
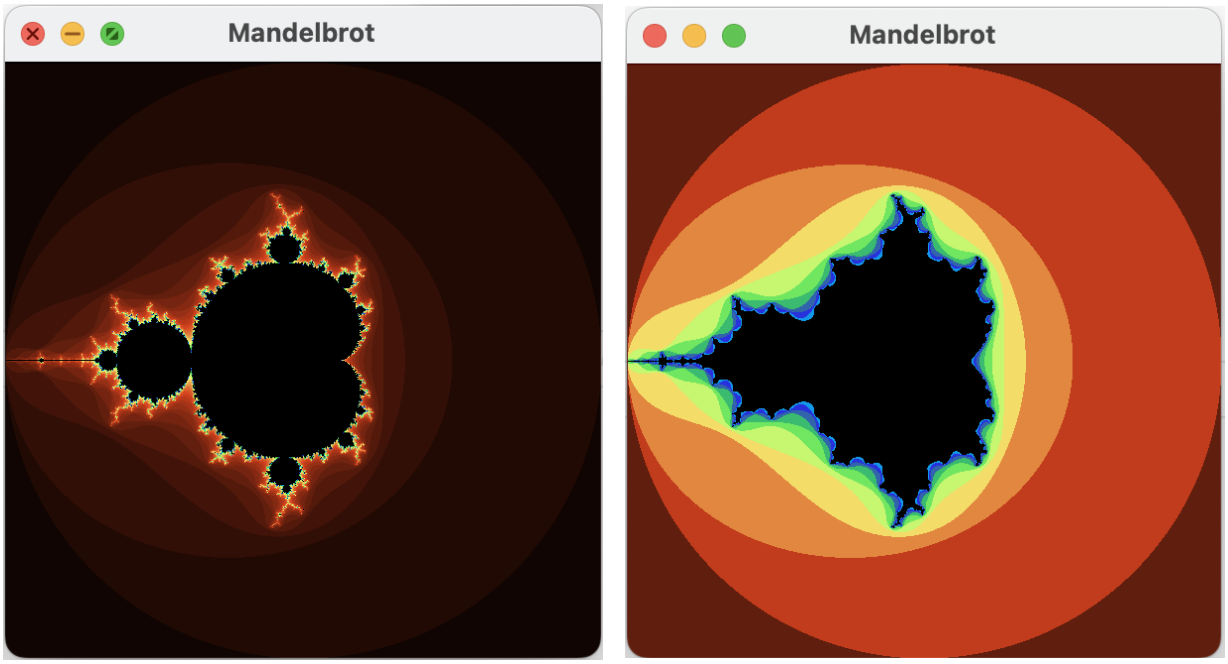



Figure 1: Main Menu Window

## 3.2 Fractal Images

After the previous choice, the user will see the second window where is drawn the fractal image.
The user can do many actions using the mouse or the computer keyboard:

- **Enter**
  The main event is selecting a point on the figure placing the mouse on it and then pressing the `Enter` key, in order to generate the sequence of coordinates that will be associated to the parameters of the oscillators.
- **Wheel Scroll**
  The user can increase/decrease the number of iterations so that he can see more details of the fractal by scrolling the wheel of the mouse.




(a) Fractal Window

(b) Fractal Window with less iterations

Figure 2: Fractal Images

### 3.3 Synth Window

- **Oscillators**
  The Frattalatore has five different oscillators.
  Each one has a selector of the type of wave (**Sine**, **Saw**, **Square**) and 4 different knobs that controls **Gain**, **Pitch**, **FM Freq** and **FM Depth**. The last two regards on the frequency modulation and select the frequency and the amplitude of it; these last two are the parameters that are set from the fractal point choice.

- **Filters**
  The Filter section allows to select three different types of filters:

  1. **High Pass Filter** (HPF)
  2. **Band Pass Filter** (BPF)
  3. **Low Pass Filter** (LPF)

  It has also two parameters that allows to select the **cut off frequency** and **Resonance** of the filter.

- **LFO (Low Frequency Oscillator)**
  It allows to modulate the output signal by selecting the **LFO Frequency** and **LFO Depth** of the modulation.

- **ADSR**
  This is a section which allows the user to modify the envelope of the signal by modifying four parameters: **Attack** (A), **Decay** (D), **Sustain** (S), **Release** (R).

- **Keyboard**
  It is a virtual **MIDI Keyboard** which can be used both with the laptop keyboard and directly on the GUI with the mouse. The Plugin also can receive MIDI inputs by an external MIDI keyboard.


Figure 3: Synth GUI

# 4 Fractals

A geometrical fractal is *"a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole"*, as described by **Professor Mandelbrot** in his publications.
Iterating a specific recurrence relation at each point in the complex space, an image is drawn more and more detailed increasing the number of iterations. To clarify, this concept can be seen also in nature where big clouds are composed with little "infinite" clouds or the mountain shapes are made with little "infinite" shapes...
Each point of the complex plane (composed by a *Real part* and an *Imaginary part*) applies an

iterative process using the fractal formula and creates a sequence of points. As long as the sequence observes a particular belonging condition, is possible to continue to iterate the sequence until the infinite (in this particular case until it reaches the maximum number of iterations ). If an element of the sequence is out from the belonging condition, the iteration ends (the sequence diverges). So each point belongs to the fractal with a particular weight, measured with the number of each point iterations.
Visually speaking, the interesting thing is that is possible to zoom the fractal "until the infinite" in particular points in order to visualize the same or a similar starting image (as seen in the following images).

## 4.1 Mandelbrot set

The Mandelbrot fractal is created by the sequences generated for each point of the complex domain. For each complex point (in this case equal to $p$), the first element of the sequence is the point itself because $z_0 = 0$. Than, the result is iterated into $z_n$ and the p remains the initial complex point.

$$f_p(z) = \begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + p \end{cases}$$

## 4.2 Julia set

We can notice that the formula of the Julia sets is the same of Mandelbrot set. The difference depends on how we set the initial conditions, but the iteration process is the same.
For each complex point p, the first element of the sequence is generated by putting the point coordinates to $z_n$ and sum the preset constant c. Then we can iterate the result into $z_n$ and c remains the initial constant.We chose 2 different Julia fractals setting 2 different constants.

### 4.2.1 Julia set 1

Setting the constant $c = -0.7269 + 0.1889i$, we have the following fractal equation:

$$f_p(z) = z_{n+1} = z_n^2 + c$$

### 4.2.2 Julia set 2

Setting the constant $c = 0.285 + 0.01i$, we have the following fractal equation:

$$f_p(z) = z_{n+1} = z_n^2 + c$$

## 4.3 Burning ship fractal

Mathematically the Burning ship fractal applies the same process of the Mandelbrot fractal. c is each complex point of the complex space.

$$f_c(z) = \begin{cases} z_0 = 0 \\ z_{n+1} = (|\Re(z_n)| + i\,|\Im(z_n)|)^2 + c \end{cases}$$

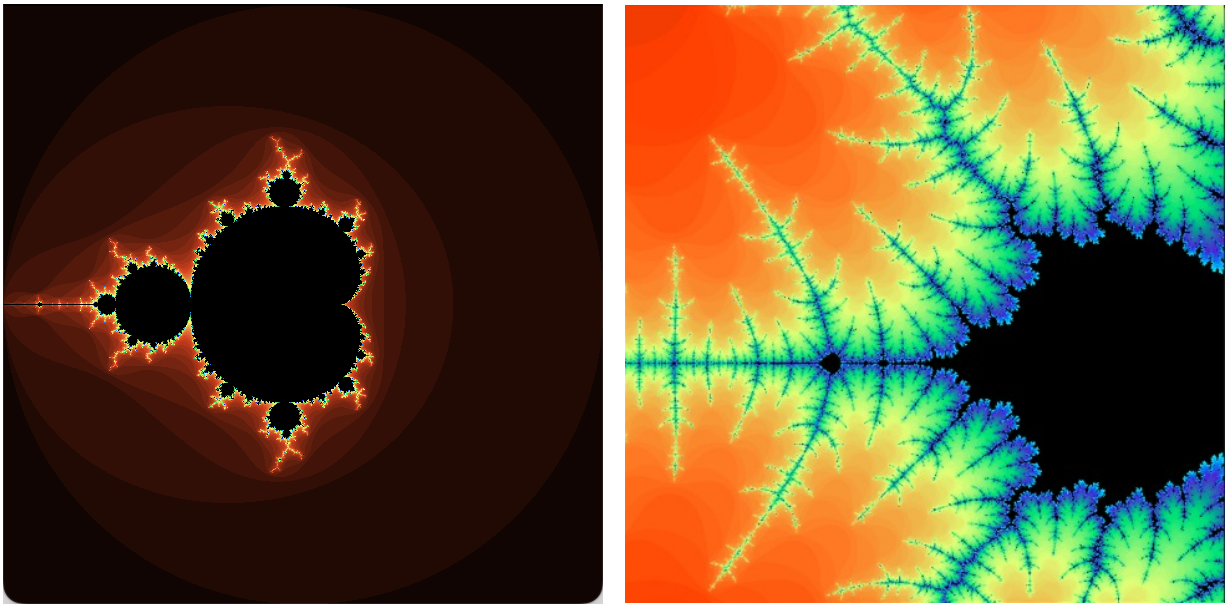
### 4.4 Fractal design and code

To implement these formula in the code it was created two classes called `Fractal` and `threeValuesArray` with their related main method types:

- `int ricorsioniMandelbrot(double cr, double ci, int max_iterations);`
- `std::vector <threeValues> calcoloMandelbrot(double cr, double ci, int iterazioni_user, int iter);`

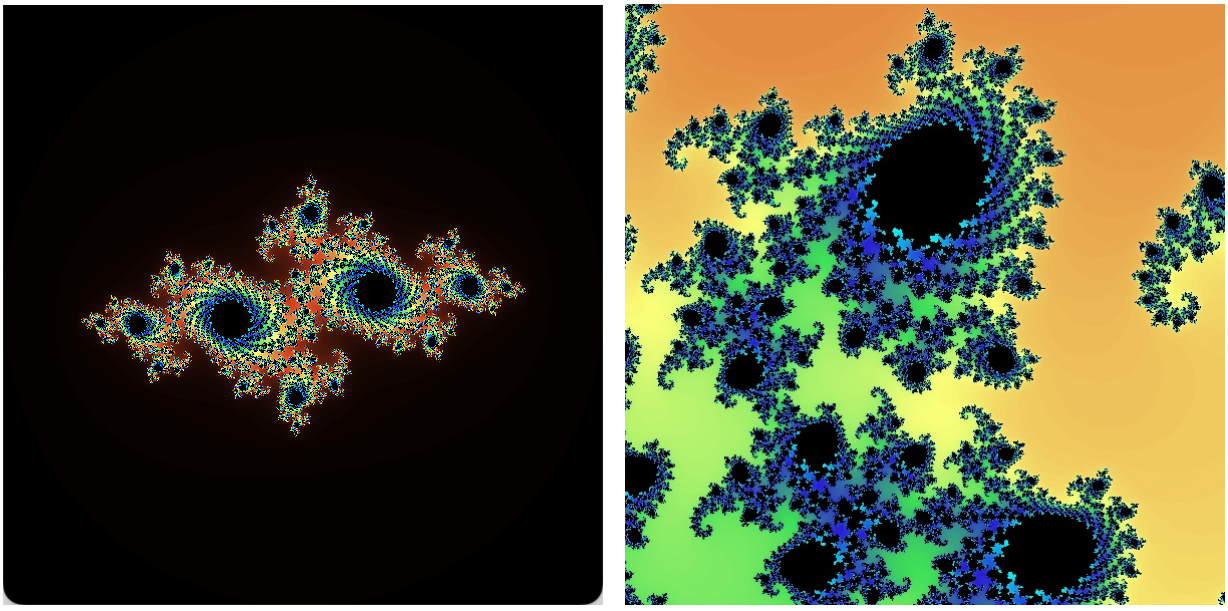
The first one is used inside two `for` cycles in which are scanned all the pixel (x,y) of the points belonging to the window. The returning `int` of the method is the number of iterations of the current point.
To draw the images has been set a high number of iterations that each point of the domain could be computed to create its relative sequence. Theoretically, the number of iterations is infinite. Thus, each point has a grade of divergence of the sequence: high grade correspond to a great belonging to the fractal, low grade correspond to a less belonging.
Each value is used to set the RGB of the each window's pixel and to draw the fractal image. Finally, the color is set with a particular interpolation that emphasizes the boundaries of the figure, using Color paint_fractal(int n, int max_iter);.
It's useful to remember that the four fractal images are represented in a complex plane and each point has 2 coordinates (Real one and Imaginary one). All fractals have a unique condition: the modulus of the complex elements of the sequence has to be less than 2: $\sqrt{(\Re(z_n))^2 + (\Im(z_n))^2} < 2$.
For this reason the complex domain and the fractal images are inside the square centered in the origin of complex space, with height and weight equal to 4.
Drawing the Fractal images with this method was be possible using `SFML` library, that optimizes the graphic.

The second method is important to send the values to **Juce** via the **OSC protocol** (as will be described in the next paragraph). It returns a vector of size 5; for each position there are 3 elements: $x$ coordinate, $y$ coordinate and number of iterations.
The first position is related to the point that the user chose from the window with the Enter keyboard button; the following are the points of the first x,y coordinates of the fractal sequence.
Each couple of coordinates has a proper sequence more or less convergent to the fractal depending on the first choosing point.



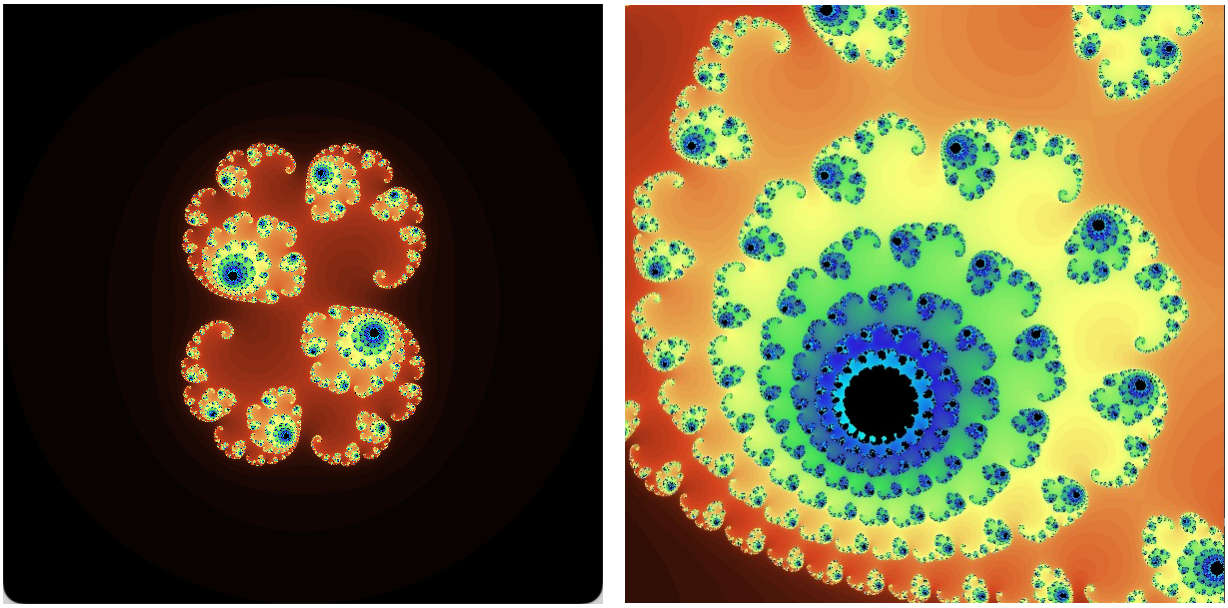
(a) Mandelbrot fractal (b) Zoomed detail

Figure 4: MANDELBROT



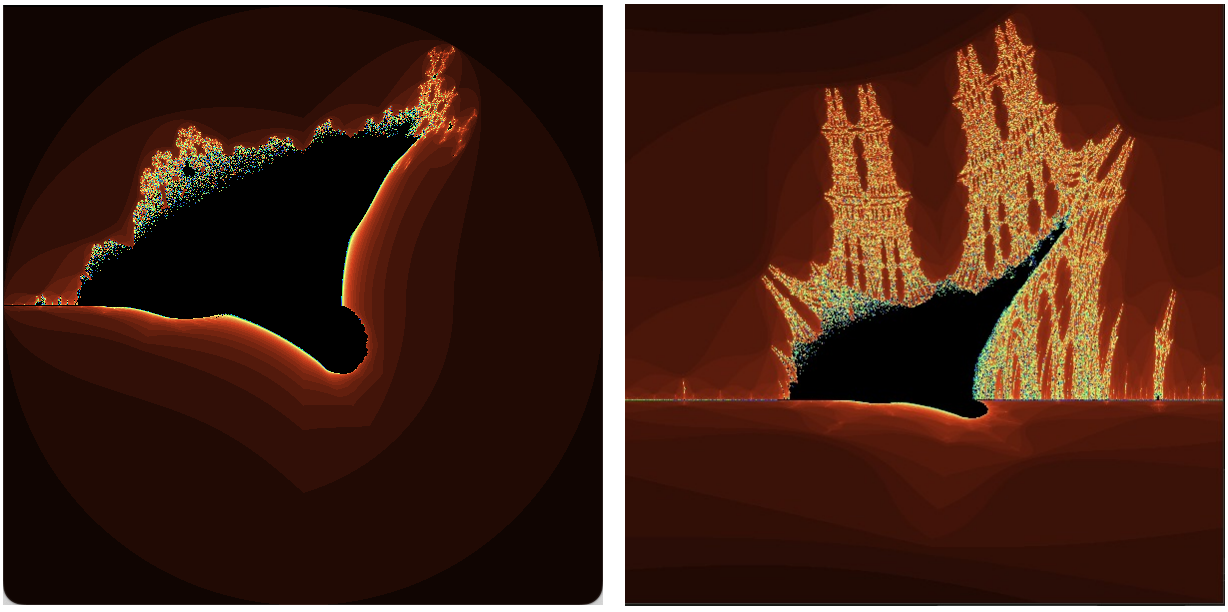
(a) Julia 1 fractal (b) Zoomed detail

Figure 5: JULIA 1



(a) Julia 2 fractal (b) Zoomed detail

Figure 6: JULIA 2



(a) Burning ship fractal (b) Zoomed detail

Figure 7: BURNING SHIP

# 5 OSC Messages

To allow the communication between the **C++ SFML** project and the **Juce** Synthesizer, a library called **oscpack** has been used. (OSC is the acronyms of *Open Sound Control*), that supports sound synthesis and communication through different multimedia devices.
The communication is Client-Server-like and it is sent through a UDP socket, for example:

```
#define ADDRESS "127.0.0.1"
#define PORT 7000
```

The sender is the **SFML** window: it initialises an output buffer that will contain the values that the **Juce** Synthesizer (receiver) uses to set the initial conditions of the FM parameters of the oscillators.
In particular the sent values are:

1. $x$ and $y$ coordinates of the chosen points of the fractal;
2. number of iterations of each sequence generated from each point.

The data is sent through 5 different UDP sockets (ports `7000, 7001, 7002, 7003, 7004`), creating one communication channel for each oscillator. The `OscComponent` class is the listener class, that inherits from the `juce::OscReceiver` and the `juce::OscReceiver::ListenerWithOSCAddress` and implements the function `oscMessageReceived` that sets the FM parameters every time a message arrives.

# 6 Frattalatore Synthesizer implementation details

Each one of the first 5 iterations of the chosen point on the fractal corresponds to an oscillator. All the oscillators play simultaneously.
The user can decide which point $p$ of the image wants to iterate, then an array of 5 triples $(\Re(p), \Im(p), \#iterations)$ of the fractal sequence is created.
Each point of this array is associated to a single oscillator and it can be itself the first element of a new fractal sequence.
As we said in the previous paragraphs the number of iterations of a sequence of a single point in the complex space corresponds to the "grade" of convergence of this fractal point. We associate the grade of divergence of each sequence (each oscillator) to the value of the **FM depth** that modulates the sound. If the grade of convergence is high (theoretically infinite), the sequence doesn't diverge and there is no modulation: FM depth $\approx 0$.
Instead, a smaller grade corresponds to an higher FM depth, until a maximum of 100 if the point diverges immediately.
Moreover we can change the **FM frequency** of each oscillator. We use the modulus of the complex number ($\sqrt{\Re(p)^2 + \Im(p)^2}$) to modulate it.
If the point chosen by the user belongs to a hypothetically infinite fractal sequence, the output sounds "good". Instead the sequence diverges out of the fractal condition, so the sound is distorted.

### 6.1 Components and Data

#### 6.1.1 Components

The resulting synth GUI that the user can play with is a composition of different `JUCE` components, each one made of sliders, labels, combo boxes etc. A `SliderWithLabel` class abstracts a generic slider with label component, that describes the graphical features of a generic GUI component and its attachment to the `AudioProcessorValueTreeState`, in order not to repeat the same lines of code for every component. Then, the classes `OscComponent, FilterComponent, LfoComponent` inherit from the generic `GuiComponent` that defines the generic structure of the synth's component.

#### 6.1.2 Data

These GUI components are views that control the model of the synthesizer modules. These are specified in the **Data** folder: `OscData, FilterData, AdsrData`. Every stage of the synth, except from the ADSR, has been implemented using **JUCE DSP** module. In every class the functions `initialize(), prepare() and reset()` of the DSP objects are called in `prepareToPlay()`, together with other utility functions that update the parameters and manage the user choices from the GUI.

### 6.2 SynthVoice

The five voices of the Frattalatore have been implemented using a class called `SynthVoice`, that inherits from `SynthesiserVoice` and makes the blocks of the synthesizer work together. It manages all the events of a voice, with functions like `startNote(), stopNote(), onPitchWheelMoved()` and the most important `prepareToPlay()` and `renderNextBlock()`. The former is called before the processing starts and sets the context parameters (`samplesPerBlock, sampleRate, outputChannels`) to every module of the synth (oscillators, filter, adsr). The latter processes every output block calling the functionalities of the modules in a certain order (oscillators outputs $\rightarrow$ gain $\rightarrow$ adsr $\rightarrow$ filtering) to produce the desired sound. The voices are then created, initialized and exploited in the `PluginProcessor` definition in order to output the generated sound and to update the parameters that come from the `AudioProcessorValueTreeState`.

## 7 Future Developments

### 7.1 Fractal Image

- **Zoom**
  Graphically the user will be able to zoom in and zoom out the fractal and find the infinite loop in particular points of the complex space.

- **Move**
  The user will be able to move on the image using the keyboard arrow: left, right, up and down.

### 7.2 Synthesizer

- **FX**
  An FX section will be added. It will contain distortions, delays, reverbs and other effects that will be also controlled with OSC messages from the fractal window. This will be useful to produce more unique and particular sounds.

- **OSC**
  A complete OSC control will be developed. Exploiting more data coming from the fractals, the way of producing more interesting and authentic sounds without even touching the Synth GUI will increase. This will be also developed mapping the data in a lot of different fashions.