ASSIGNMENT REPORT

BIAGIO NELLO SPAZIO

"in space no one can hear you scream"

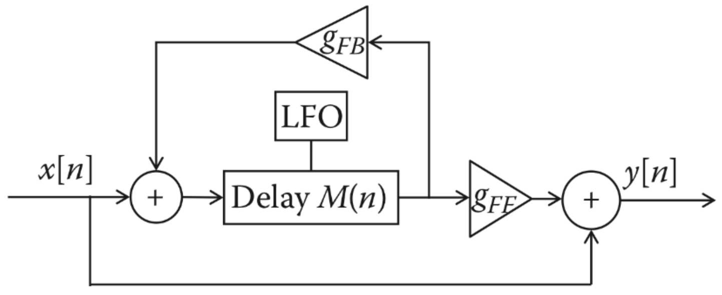
(except for Biagio)

General idea:

We have decided to implement a flanger plugin and to connect its parameters to the fate and the emotional state of our friend Biagio, floating alone in the vasteness of the interstellar space.

The flanger effect:

Before delving into the details of our implementation, let's have a look at the general working principles of a flanger with feedback.



As shown in the diagram on the left, a flanger effect consists of a system in which the input signal is processed through a delay whose value changes over time, controlled by the output of an LFO.
In a flanger with feedback also the output of the delay block is mixed with the original signal and used as input for the signal.

Finally, the original and the processed signals are mixed with a balance that reflects the effect that the user wants to achieve.
The typical implemented parameters, over which the user has control, have been renamed partially for artistical purposes and are listed below, along with a brief explanation of their role in the system:

- **Dry**: it controls the amout of original signal in the output
- **Wet**: it sets the amount of processed signal in the output
- **Sickness**: it controls the minumum amout of delay, i.e. the center value of the LFO oscillation
- **Air pollution**: it adjusts the gain with which the output of the delay is given as input to the system.
- **Speed**: it controls the frequency of the LFO
- **Distance**: it sets the amplitude of the LFO

According to the design pattern offered by the Juce platform used in this project, the code has been split in two main classes namely **DelayLineAudioProcessorEditor** and **DelayLineAudioProcessor,** which have been in charge of the implmementation of the GUI and the Audio Processing algorhitm respectively.

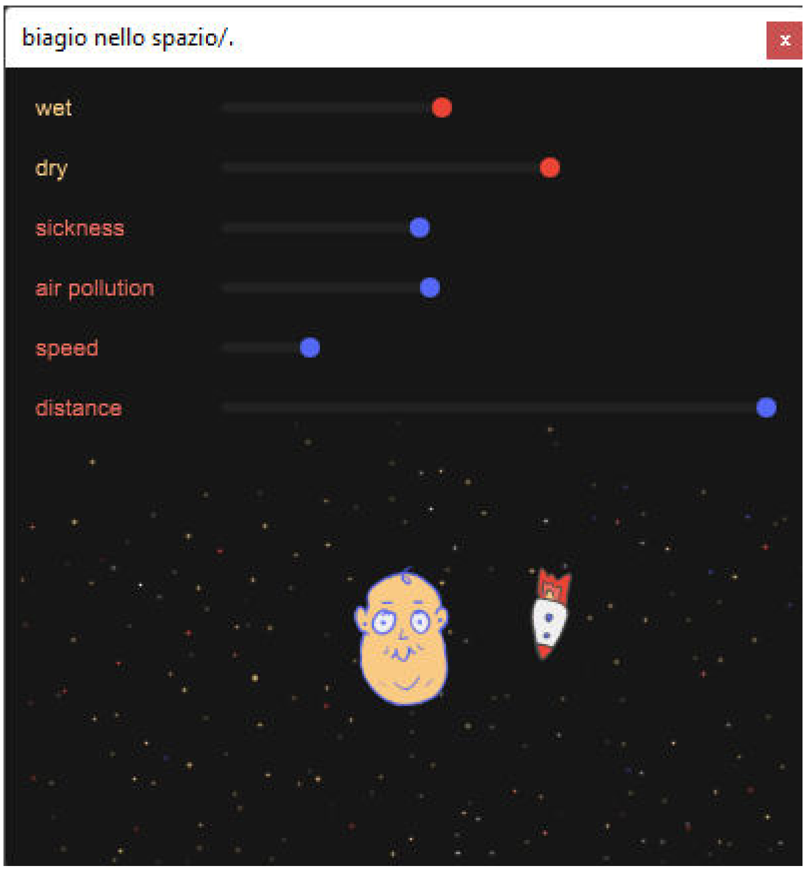
GUI:

The parameters of the effect can be modified using sliders that have accordingly been labeled. These graphic elements have been introduced in the code using the **juce::Label** and **juce::Slider** classes. We have overridden the **sliderValueChanged** method inherited from the **juce::Slider::Listener** and have implemeted in a way that enables us to connect the sliders to the parameters actually influencing the audio processing.

In addition, we declared the class **juce::Image** to add the background to our instrument along with Biagio and his spaceship.

The animated graphic part with Biagio and his spaceship has been implemented by drawing the background, the character, and the ship on photoshop. The images have been introduced in the code by projucer through the class **juce::imageCache**. The spaceship has been made to rotate around Biagio's head with an angle that is proportional to the phase of the LFO, while the other parameters (except for wet and dry) control Biagio's expressions and the size of the ship or its distance from our friend's head.

The animation of the flame coming out of the spaceship has been created by drawing three different flames which are alternated randomly.



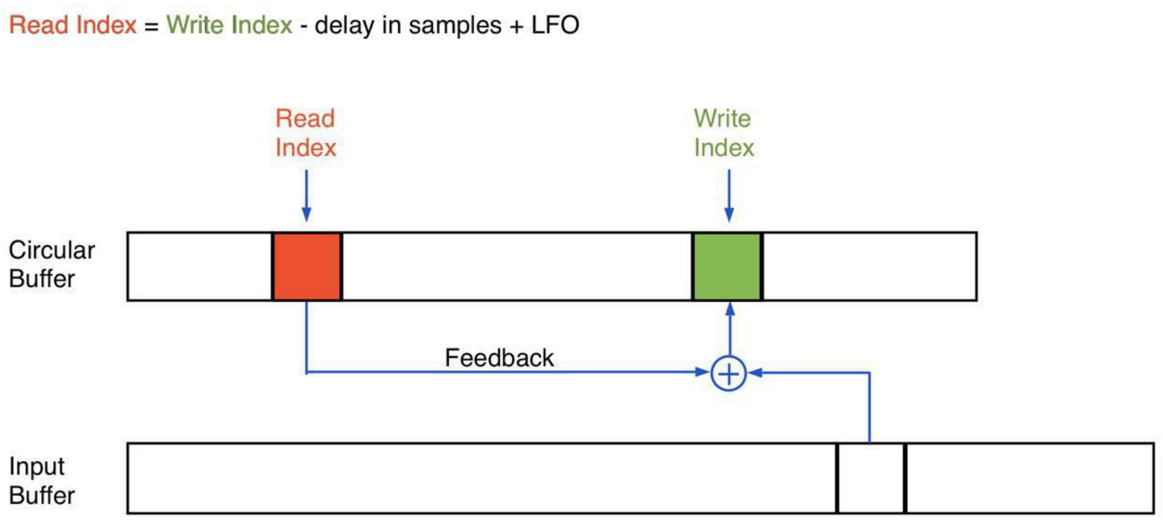
PROCESSOR:

To manage the audiostream we have used the **juce:AudioSampleBuffer** class. In the constructor of the DelayLineAudioProcessor class we have taken care of initializing the audio buffer, the parameters corresponing to the value of the sliders in the GUI (with appropiate conversion factors) and the pointers used to “move” on the buffer and actually implement the algorhitm.

We have implemented the code for the flanger circuit in the **processBlock()** method.

We have implemented the **getStateInformation** and **setStateInformation** functionalities by copying the slider values to the memory block given by the DAW. This memory block is later returned to the plugin and the values are restored. This implementation has been tested in AudioPluginHost and has proved to work on most of the DAWs.

The algorhitm has been implemented using a circular buffer with two indexes, one of which is used to write on the circular buffer and the other one that is used to read from it. Since these two indexes point to different positions in the buffer, it is safe to say what we have implemented so far is a delay. In the position pointed by the writing index, a weighted mix of the sample coming from the input buffer and the one pointed by the reading index of the circular buffer is copied, implementing a feedback in its way. To actually implement the flanger effect, the position pointed by the reading index has been modulated according to an LFO.



In addition, a particular attention has been paid to avoid glitches given the asynchronous nature of the calling of the processBlock() method. Actually, since the processoBlock() is called every time the buffer is full, the LFO could experiment some discontinuities that would lead to a glitchy effect. To avoid this a variable has been introduced to role-play and compensate the passing of time between the processing of two consecutive samples.

Moreover, since the values generated by the LFO are not necessarily integer numbers, a linear interpolation algorhitm has been used to manage this problem.