Link GitHub: https://github.com/polimi-cmls-22/group12-hw-ID-Ratatouille.git

Introduction

The idea for this *DragonBall FM* occurred to us after the implementation of an FM synthesizer for our first project. That's because, most of us was foreign to the mechanism underlying synthesizers, and discovered a whole new complex world, related to sound generation and design.

After being more acquainted with this subject, we wondered how to assist a rookie like us in getting used to synthesizers and their main features. The idea was to blend the intricacy of sound production with something we're all familiar with, and that's where DragonBall comes into play.

DragonBall FM is an application capable of merging together the main parameters of a synthesizer and video game user interface, providing an interactive first approach to digital sound making for the user.

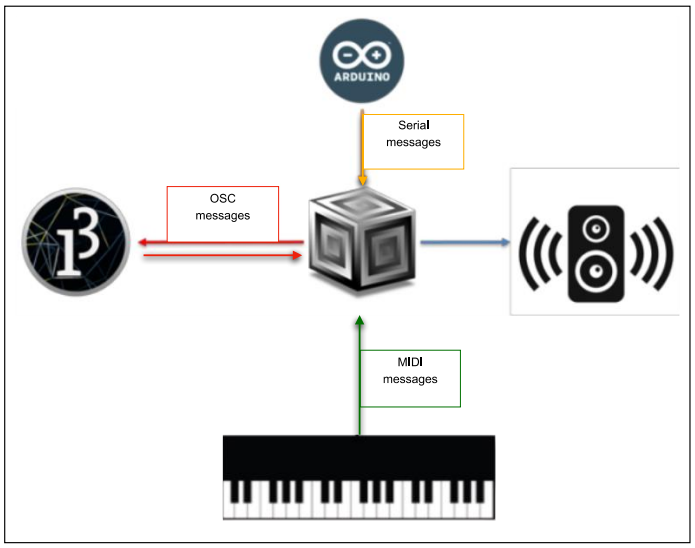
DragonBall FM offers the user the possibility to choose between seven different sounds, all created using FM synthesis. The sound can be modified in real time using specific Arduino sensors, and the input frequencies are provided by a MIDI keyboard.

Scheme

The Arduino module processes and passes data through serial connection to the SuperCollider computation module, which in turn receives MIDI messages from the keyboard. A GUI, realized with Processing, is connected to our SC engine through OSC messages (in a 2-way fashion). The interface allows both the

visualization of parameters modified via hardware by the user and the possibility to select the desired initial sound.

Figure 1: Interaction Scheme



Hardware

Arduino connections

As regards the physical sensors, we used an ambient light sensor, a triple axis accelerometer and an analog potentiometer, obtaining the following Arduino+breadboard configuration:

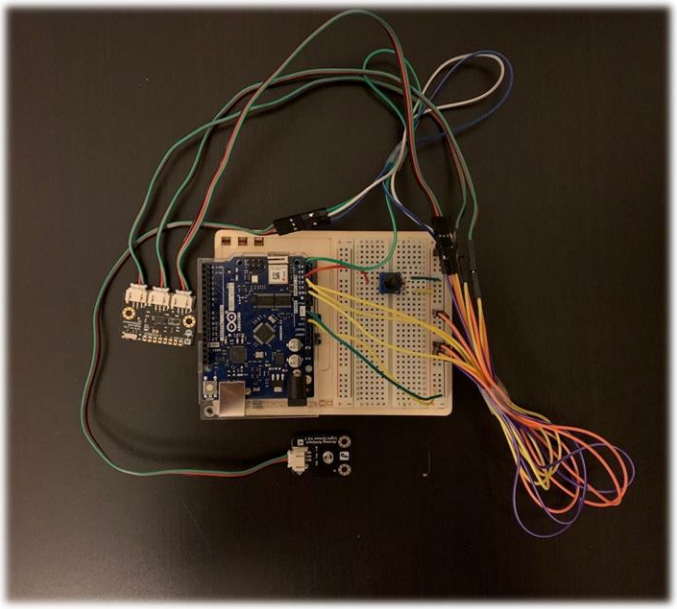


Figure 2: Our Arduino with breadboard configuration

Ambient Light Sensor

This sensor is able to detect the light density of your environment and reflect this information back via an analog voltage signal to the Arduino controller.

This sensor requires a setting section, included in the setup function, to calibrate the maximum and minimum light values. This is done within an interval of 5 seconds, marked by a temporary ignition of the built-in led.

In *DragonBall FM*, this sensor is responsible for the cutoff frequency of an *integrated* Low Pass Filter applied to the input signal. When the sensor is obscured, the frequency is lower, and the resulting sound is darker (accordingly with the color of the sky in the GUI).

Figure 3: Ambient Light Sensor

3-Axis acceleration sensor

3-Axis acceleration sensor is an electronic equipment which measures the acceleration during the object motion. Differently from the previous one, this sensor required a specific calibration in order to manage the effect of the gravitational acceleration on the final output values.

After the calibration, we found every axis initial value and the corresponding value of local gravity. This allowed us to properly initialize the values for the X and Y accelerations, ranging between -1 and +1.

In *DragonBall FM* we used it to map both the distribution of the audio signal and the frequency of a Low frequency Oscillator, by linking these values to the user's wrist motion.

Figure 4: 3-axis Accelerometer

Potentiometer

A potentiometer is a mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input, it is possible to measure the amount of resistance produced by a potentiometer as an analog value. We used the potentiometer as a facsimile of the classical gain knob, found in numerous synthesizers.

Figure 5: Potentiometer

GUI

Figure 6: Dragonball FM GUI. The fourth ball has been selected so it's illuminated.

The Graphical User Interface has the typical video game style. The seven dragon balls located on the top can be selected with a mouse click and are illuminated accordingly (see Figure 6). Goku's motion changes accordingly with the accelerometer inclination along the X and Y axis. Another graphical element related to physical sensor is the color of the sky, which varies from light to dark blue. In addition to the components related to Arduino's sensor, the GUI offers the possibility to visualize the values of the cutoff frequency of the Low Pass Filter, the one of the Low Frequency Oscillator and the volume level. These values are included inside surrounding graphical elements derived from DragonBall environment, and change real time, by means of OSC messages sent from SC to Processing.

As regards the style, the movement effect is provided by the use of gifs. To obtain the final result we had to properly set the frame rate and frame counter: we use a specific counter which increases of one unit after n frames (where n can be modified accordingly to the desired frame rate). In our case n = 6 and, as a result, the frame rate slows down of 6.

SuperCollider

Serial port

As mentioned before, we established a serial connection between SuperCollider and Arduino. The results of the measurements obtained through sensors are contemporarily written in serial port, separated by three different ASCII characters. The sequence is the following:

[*light value*]ˈxˈ[*x value*]ˈbˈ[*y value*]ˈcˈ\n

This serial reading method is efficient if the three numeric values are positive integers, while it results problematic when encountering commas and minus signs. Since the values of accelerations we obtained were between -1 and +1, we mapped them between 100 and 300, using the following simple mathematic relations, to overcome this issue:

$$x_{val} = x_{acc} + 2 * 100$$

$$y_{val} = y_{acc} + 2 * 100$$

In SC the three values are saved in three different variables and then normalized accordingly to their aim.

These values are then sent to Processing through different OSC message, to set the color of the sky and the motion of GOKU's animation inside the window.

Ambient Light Sensor - Low Pass Filter

We used the Light Ambient Sensor to control the frequency of a Low Pass Filter, applied to the input signal. Depending on the amount of light received, the sensor returns to SC a value between 0 and 1023, which is then mapped trough linexp functions between 80 and 20000 Hz.

The light value also affects the GUI interface: SC sends Processing an OSC message (/light) and to change the color of the sky (from light to dark blue), accordingly with the frequency sweeping.

Triple Axis Accelerometer - Pan / Low Frequency Oscillator

The spread of the signal between the left and right channel is controlled by the acceleration on the X axis. This value is ranged through the linlin function between -1 and 1, respectively the left and right output channel. The value of Y acceleration instead, corresponds to the rate of a Low Frequency Oscillator, applied to the output signal. The values obtained from the LFO are mapped between 0.1 and 20 Hz and are shown in real time in the upper left corner of the GUI.

These values are sent to Processing through the OSC messages (/xAxis , /yAxis and /lfofreq). The first two are then mapped consistently with the height and width of the window and represent the coordinates of Goku's position.

Potentiometer – Master volume

The analog potentiometer is used to modify the amplitude of the output signal, achieved by using *DragonBall FM's* features. As seen for the light sensor, the first value (the one printed in the serial port) goes from 0 to 1023 and is mapped between 0 and 10, to obtain a more suitable range for the master volume.

Finally, to apply pan and gain to the final output at once, a private audio rate bus is created. This bus takes as input the output of the desired patch and thus allows to control the above-mentioned parameters more easily.

Patches

We associated each sphere to one patch of our FM synthesizer. We chose to implement some flexible sounds, in order to show the user the versatility of FM synthesis.

To implement these sounds, we used up to four operators combined into different algorithms. The seven selectable patches are the listed below, each one followed by its main characteristic:

- Electric piano 1, fast attack and bright resulting sound
- Electric piano 2, softer timbre
- Pluck, short decay
- Organ, sawtooth timbre
- Bass, simple solid based tone
- MM, techno music inspired sound
- Violin, emulation of string musical instruments

Midi

The synthesizer can be controlled with a MIDI keyboard. The connection between MIDI and SC exploits the NoteOn and NoteOff functions and it's based on the creation of a synth for each note being played, which will be deleted once the note has been released.

Conclusions

The main idea at the basis of this project was to exploit our new acquaintances regarding the FM synthesis for creating a hybrid environment, able to offer a valid first approach to sound generation. *DragonBall FM* isn't conceived as a classic synthesizer, but more as a game based on FM sound modification. Accordingly with its nature, we decided to implement seven presets, all based on FM synthesis, with which the user can experience a wide variety of results. This can be done thanks to the cross superposition of various effect, such as LFO, LPF and Pan. In addition to this, since all the presets are entirely realized using FM, the user can familiarize with this typology of sounds, by just playing with them. As regards possible future improvements, *DragonBall FM* could be enhanced by adding an ad hoc playlist of at least 7 different songs (one for each preset), so that the user can experience each sound and its peculiarities and have an alternative to the MIDI input. It'd be also interesting to develop a *DragonBall FM 2.0*, capable of guiding the user towards a deeper knowledge of FM synthesis, with an eye on the algorithms and operators' mechanism.