# Padder

Lelio Casale, Marco Furio Colombo, Marco Muraro, Matteo Pettenò

10582124, 10537094, 10866647, 10868930

## 1 Introduction

### 1.1 Presentation of the project

The aim of our project is to develop a musical instrument based on a virtual pedal that, thanks to the interaction between the different blocks of our architecture, can generate chords that can be played in real time during a musical performance. This kind of approach could be seen as a handsfree way to generate chords, instead of using a piano or a keyboard. For example, it is possible to play the chords with the foot, while you're playing the guitar and singing at the same time.
Thanks to the small dimensions of the Arduino board and the sensors, it's also a more portable instrument than a keyboard or a piano. Changing the synthesizer parameters, the musician can also create many sounds, varying from the piano sound usually used to create chords.

### 1.2 Structure

Our project has five different main blocks:

- A hardware part composed by an **Arduino Uno board** and two **URM09 Analog Ultrasonic Sensors** connected to it. These two sensors give the information about the note played depending on the distance of the foot from these two sensors. This information is fed to the synth.
- A pad-like synthesizer implemented using **wavetable synthesis**, provided with high-pass and low-pass filters and a Low Frequency Oscillator modulating output's amplitude.
- The updated version of our **harMMMLonizer** takes the one-voice audio signal from the synth and creates different chords depending on the parameters of the harmonizer.
- The **touchOSC** application (Mk1 version) that represents our Graphical User Interface of the synthesizer and of the smart harmonizer. Thanks to touchOSC, it is possible to change the synthesizer and the harmonizer parameters in real time with a simple mobile device, such as a smartphone or a tablet.
- The **visualization** part, whose goal is giving to the user a feedback of the chords played with the foot. The root note is displayed. Moreover, the visualization was built in order to have different colors depending on the musical type of the played chord.

## 2 Implementation

### 2.1 Arduino

The hardware components of this project are an **Arduino One** board and two **URM09 Analog Ultrasonic Sensors** that are connected to it. Every ultrasonic sensor is connected to a voltage source pin (**5V** and **3,3V**) , to a ground pin (**GND**) and to an analog input pin (**A0** and **A1**).
An ultrasonic transceiver sends an ultrasonic ping (over the human audible range) that gets reflected from a surface and returns to the sensor. The sensor calculates the time difference between the emission and the return of the ping. Thanks to the functions *AnalogRead(A0)* and *AnalogRead(A1)*, the Arduino can receive this time and then, with a short formula, it's easy to calculate the distance between the sensor and a generic object in his range, in our case the musician's foot.

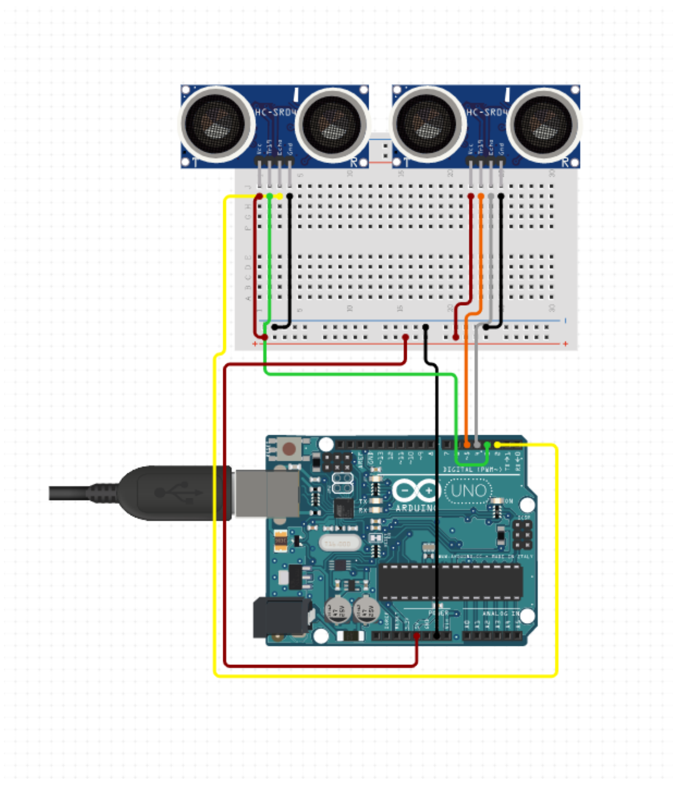


Figure 1: An example of how to connect the two sensors to the Arduino

Each sensor covers a different type of notes: the sensor connected to the A0 input, whose position is near to the musician, can play all the white notes in the C3 octave (C3, D3, E3, F3, G3, A3, B3); on the other side, the transceiver connected to the A1 pin, that is placed beyond the first sensor, plays all the black notes in an octave of a keyboard (C3#, D3#, F3#, G3#, A3#). The *MapToNote.ino* file maps all the possible positions of the foot to all the notes of the C3-C4 octave. The messages sent from the Arduino to the serial port are composed by a number, which is the MIDI number of the played note, and a letter: *'b'* indicates that the note played is black, *'w'* indicates that the played note is white, and *'s'* represents the stop note message. A possible sequence of messages could be *"48w"* (C3 note on), *"48s"* (C3 note off), *"54b"* (F3# note on), *"54s"* (F3# note off). All these messages are then mapped in *SuperCollider* and then the info regarding the MIDI note and the gate value (0 when an *'s'* message is sent, 1 in the other cases) are sent respectively to the ~***serialMidiNoteControlBus*** and to the ~***serialMidiMessageControlBus***. The synth then reads from these two buses in order to have the note that has to be played and the gate control, to stop the sound when the sensors don't see anything.

The black notes sensor has priority to the white notes sensor: in fact, if both the A0 and the A1 sensor see something in their range, the message sent is a **black note** message. Then, if the black note sensor doesn't see anything but the white note sensor sees something near, the Arduino sends a **white note** message. As the last possibility, when the musician is not playing anything placing his foot on the virtual keyboard, the two sensors do not see anything in their range and a **stop note** message is sent. The Arduino sends a stop note message also when the played note is changed by the musician, changing the foot position on the virtual keyboard.

The maximum ranges set are approximately 20cm for the black note sensor connected to the analog pin A1 and 27cm for the white note sensor connected to the analog pin A0.

## 2.2 Wavetable Synthesizer

The **Wavetable Synthesizer** aims to create pad-like sounds suitable for background harmony. Basically, wavetable synthesis is a sound synthesis tecnique that is based on periodic reproduction of a **single-cycle waveform**. Substantially, values representing the single cycle are stored in what is called a **wavetable** and then read from this data-structure for the signal to be played. The synth that was developed is a mixture of two different sounds, each one consisting of six partials with different amplitude ratio. This allows to achieve a more complex and interesting timbres. The wavetable synth was implemented through *SuperCollider* language within a *SynthDef* object that also features some effects to be applied on the synth itself. The two wave types that we developed are a **sinusoidal** wavetable and a **Chebyshev** wavetable and the sound can be further shaped thanks to the chain of effects and filters that are applied to the final harmonized signal.

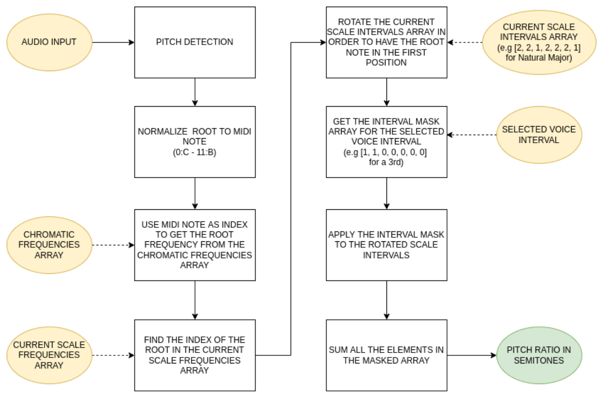
The synthesizer takes the informations regarding the MIDI note played and the gate control respectively from the ~***serialMidiNoteControlBus*** and the ~***serialMidiMessageControlBus***. Then, the output signal of the synthesizer is fed to the ~***inputAudioBus***, which is the input signal of the harmonizer block.

## 2.3 HarMMMLonizer

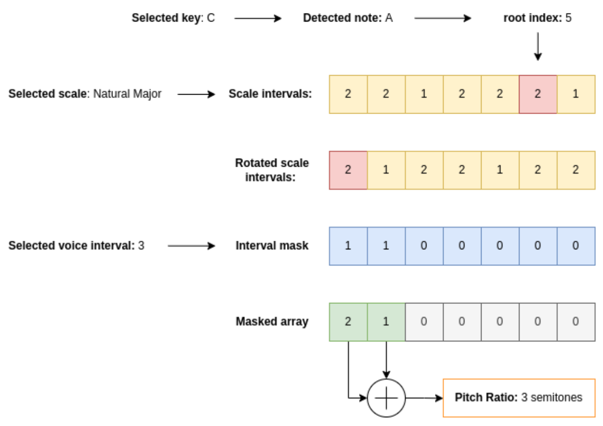
In order to obtain a rich musical background, we needed something able to turn the selected root note into a chord in tune with the harmony of the song. Therefore we improved and expanded the Harmmmlonizer that we implemented for the first homework.

The new developed features are:

- **Smart pitch ratio computation**: the user has now the opportunity to set the **key** and the reference **scale**, and for each voice the **desired interval** respect to the root note, which we assume to always be the detected one. Fig. 2 shows a high-level diagram of the algorithm we have implemented to find the desired pitch ratio in semitones, while Fig. 2 shows an example of its application. The algorithm is implemented by the **pitchRatioManager** *SynhtDef* in Supercollider.
- **An optimized pitch shifter** *SynhtDef* that uses the *PitchShiftPA* pseudo-UGen which implements the **PSOLA** algorithm allowing to maintain the formants while changing the pitch with (particularly useful if the input signal is a human voice).
- **Improved master section**: to the final output signal the user can now apply a **reverb** with dry/wet, room size and high damp controls, a **high-pass filter** and a **low-pass filter** in which the cutoff frequency and the resonance can be modified, a **Low Frequency Oscillator** modulated by the user in frequency and phase and an **ADSR envelope**. These effects were implemented in order to give to the user more possibilities to shape the sound. All these parameters are controlled by the user through a more portable and compact GUI created with **TouchOSC**.


AUDIO INPUT
PITCH DETECTION
NORMALIZE ROOT TO MIDI NOTE (0:C - 11:B)
CHROMATIC FREQUENCIES ARRAY
USE MIDI NOTE AS INDEX TO GET THE ROOT FREQUENCY FROM THE CHROMATIC FREQUENCIES ARRAY
CURRENT SCALE FREQUENCIES ARRAY
FIND THE INDEX OF THE ROOT IN THE CURRENT SCALE FREQUENCIES ARRAY
ROTATE THE CURRENT SCALE INTERVALS ARRAY IN ORDER TO HAVE THE ROOT NOTE IN THE FIRST POSITION
CURRENT SCALE INTERVALS ARRAY (e.g [2, 2, 1, 2, 2, 2, 1] for Natural Major)
GET THE INTERVAL MASK ARRAY FOR THE SELECTED VOICE INTERVAL (e.g [1, 1, 0, 0, 0, 0, 0] for a 3rd)
SELECTED VOICE INTERVAL
APPLY THE INTERVAL MASK TO THE ROTATED SCALE INTERVALS
SUM ALL THE ELEMENTS IN THE MASKED ARRAY
PITCH RATIO IN SEMITONES

(a)


Selected key: C
Detected note: A
root index: 5
Selected scale: Natural Major
Scale intervals: 2 2 1 2 2 2 1
Rotated scale intervals: 2 1 2 2 1 2 2
Selected voice interval: 3
Interval mask 1 1 0 0 0 0 0
Masked array 2 1 0 0 0 0 0
Pitch Ratio: 3 semitones

(b)

Figure 2: Harmmmlonizer pitch ratio computation: (a) Algorithm (b) Example

## 2.4 Graphical User Interface: TouchOSC

The **Graphical User Interface (GUI)** was implemented through **TouchOSC** (Mk1 version), a fully modular control surface that runs on all iOS and Android devices. TouchOSC supports sending and receiving messages through **OSC (Open Sound Control)** and **MIDI (Musical Instrument Digital Interface)** protocols. The TouchOSC Editor application allows you to arrange and freely configure all the control elements you need to manage within your GUI.
All the graphical elements are contained within what is called a layout. Basically, a layout is a file that contains one or more pages that, in their turn, hold controls element with their individual settings such as OSC-mapping or MIDI-mapping, look and feel configurations and other options in function of the specific element type. Layouts can be transferred to the target device over **WI-FI network** using TouchOSC Editor app or copied through an USB cable connection. For further information, we recommend visiting TouchOSC Mk1 official website.
The layout that was developed consists of three different pages. The first one is devoted to general controls over the final output and the Pitch Shifter. The second page is related to controls over pitched voices, whereas the third one provides controls over Synth Pad parameters. The musician can switch from one page to the other by simply selecting the correct page on the tab-bar section located on top of the screen. Messages are exchanged with the software through OSC communication protocols. Thus, each section within both pages has its own OSC address patterns hierarchy. The Layout was developed for iPad or other tablet devices.

### 2.4.1 Page 1: General Controls

As brought up before, the first page consists of two main sections: **Master** and **Pitch Shifter**.

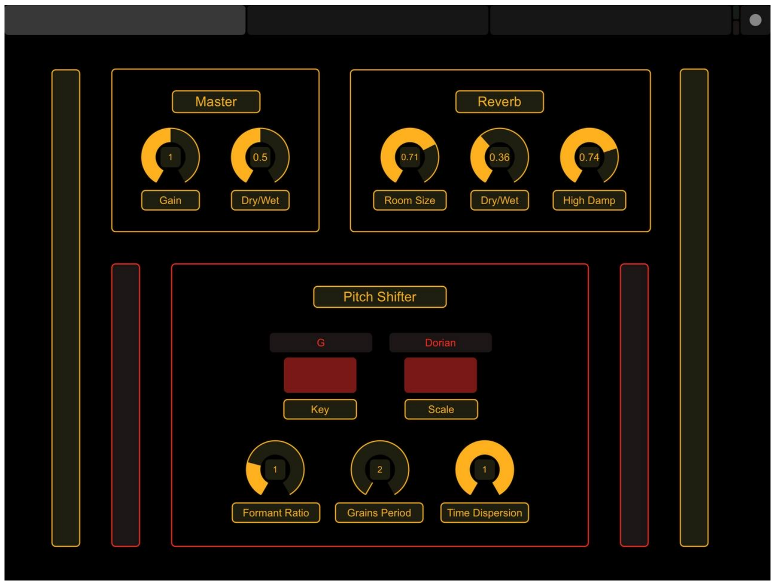


Figure 3: General controls page within TouchOSC GUI.

The **Master** section allows the musician to set master gain and dry/wet balance between input signal and pitched voices. Furthermore, some controls over reverberation effect to be accomplished on the final output are provided. Indeed, three knobs are available for Room Size, Dry Wet and High Damp parameters.
The **Pitch Shifter** section makes possible for the artist to set the key and the musical mode for pitch shifting. This controls elements consist of two buttons. By clicking on a button, the user can switch from the current value that is displayed within the corresponding label to next one. Moreover, three knobs are available, each one devoted to a specific parameter related to the pitch shifting. This section enables the musician to set **Formant Ratio**, **Grains Period** and **Time Dispersion**.

### 2.4.2 Page 2: Pitched Voices Controls

The **Voices** pages is divided into four sections, one for each voice, as four voices harmony is supported. For each section, **Gain** and **Pan** knobs make possible to set voice gain and panning values. **Voice Interval** slider allows to set the interval for pitch shifting, whereas the two buttons on its left side enable to set the number of octaves and direction for pitch shifting, which can be accomplished within the same octave as the input note or one/two octaves upward/downward.

Moreover, some controls over feedback delay line are available. The musician can choose the desired **Feedback Mode** by clicking on the **FeedbackMode** button and set **DelayTime** and **Feedback amount** values for feedback effect to be applied on the single pitched voice.

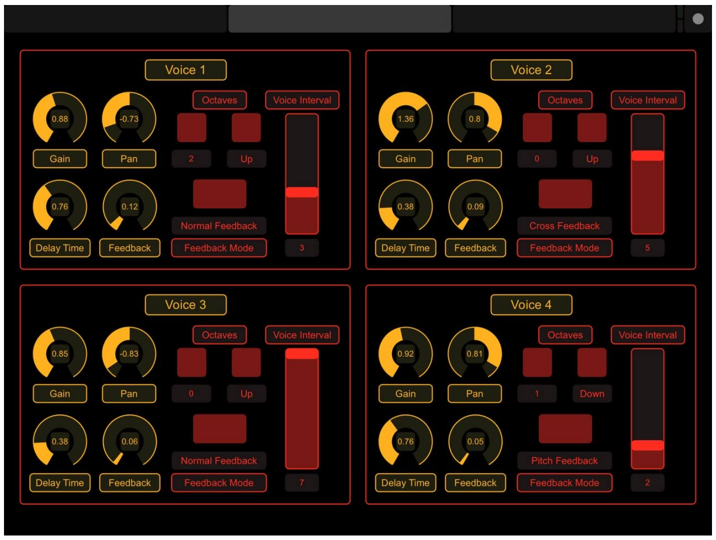


Figure 4: Pitched voices page within TouchOSC GUI.

### 2.4.3 Page 3: Synth Pad Controls

The third page is devoted to **Synth Pad** controls and it consists of two sections. As the synth pad sound is a mixture of two different sounds, the first section provides the musician with the possibility of controlling the **balance** between them. Indeed, two gain knobs, one for each synth, are available.

The second section makes possible to control parameters related to **effects** and **ADSR envelope** to be applied on the synth sound. As a matter of fact, it consists of three subsections. The first one provides controls over **Frequency** and **Resonance** of the High Pass and Low Pass filters. Then, two knobs allows the musician to set Amplitude and Frequency of the LFO. Moreover, four sliders are available to control **Attack**, **Decay**, **Sustain** and **Release** of the ADSR envelope.

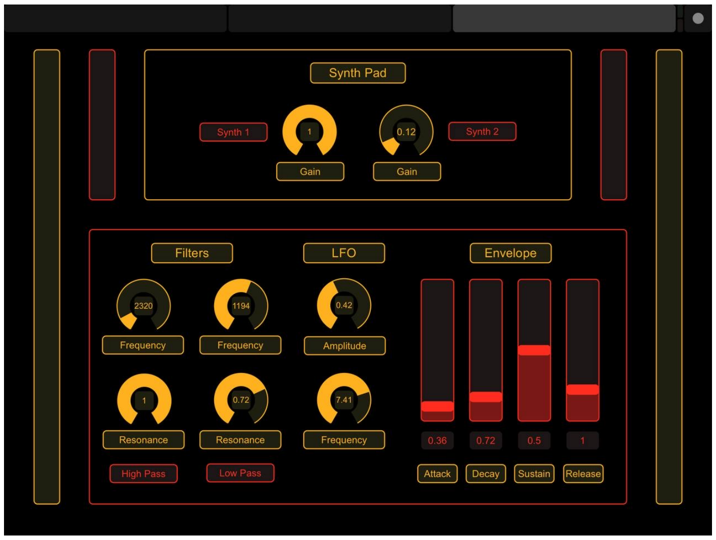


Figure 5: Synth Pad page within TouchOSC GUI.

## 3 Visualization

### 3.1 Introduction

The visualization is implemented using the *Processing* ability to render rapidly changing shapes and exploiting the Object-Oriented paradigm of Java, upon which the Processing language is based upon.

The main idea behind the visualization is to offer a functional graphic section pleasantly integrated with some catchy graphics that could reflect and respond to the user actions.

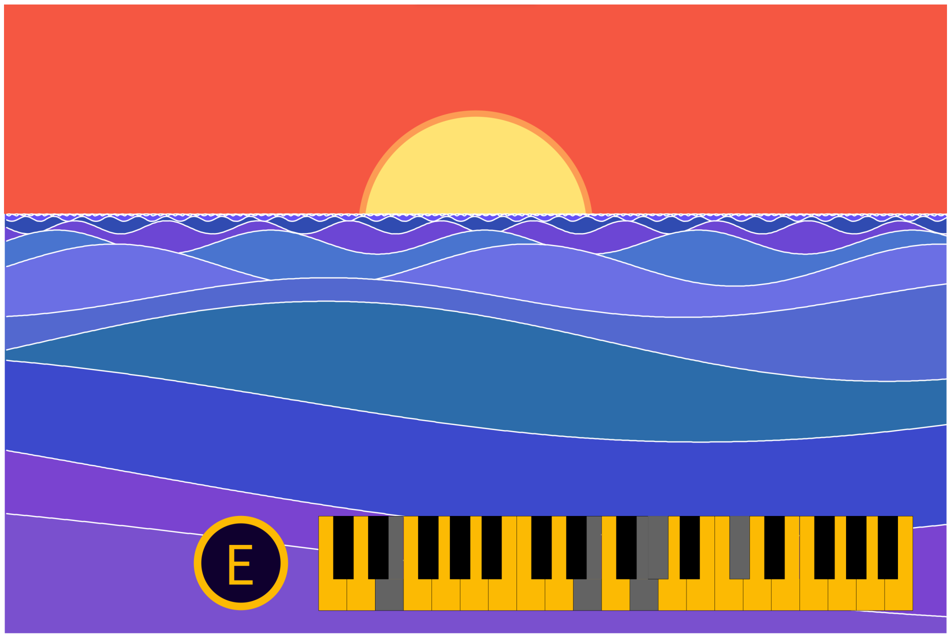


Figure 6: Visualization of the keyboard and animated landscape

## 3.2 Functionalities

Our project is basically a proof of concept for a highly portable keyboard that could ideally be operated hands-free and autonomously generates chords. Therefore the visualization features a functional block that aims to aid the user to clearly see how the provided tools are operating. On top of that a beautiful landscape that reflects the music produced by the user.

The functional block at the bottom of the screen includes:

- The **note** played by the user on the left.
- A **three-octave keyboard** on the right, displaying the root note received from the sensor connected to the Arduino and the generated notes that compose the chord.

The animated landscape that reacts to the user's actions, updating in real time the parameters that define the shape of each element appearing in the picture:

- The sun is animated and its radius oscillates between its maximum and minimum set dimension as notes are played.
- The waves amplitude, velocity and color palette are updated independently:
  - The **amplitude** is scaled each time a note is played.
  - **Velocity** is increased or slowed down and can even determine a change of direction of the wave.
  - The palette that determines the color of each wave in the whole sea is changed accordingly to the **chord** generated by the harmonizer:
    - Major chords generate rgb colors with values of red and blue fixed at 255 and 0 respectively.
    - Suspended chords generate rgb colors with values of green and blue fixed at 255 and 0 respectively.

∗ Minor Seventh chords generate rgb colors with values of red and green fixed at 0 and 255 respectively.

∗ Minor chords generate rgb colors with values of red and blue fixed at 0 and 255 respectively.

∗ Major Dominant seventh chords generate rgb colors with values of green and blue fixed at 0 and 255 respectively.

∗ Major seventh chords generate rgb colors with values of red and green fixed at 255 and 0 respectively.

∗ Chords with other alterations generate a custom designed palette that includes colors from each other palettes.

Both colors and chord types are mapped with corresponding moods and feelings that they resemble, as shown in Fig. 7. For example, cold colors are associated with unhappy feelings, as minor chords do. Exploiting this, it is possible to create a relation between the chords played by the harmonizer and a color palette. Of course it is to be noted that those relations are subjective.

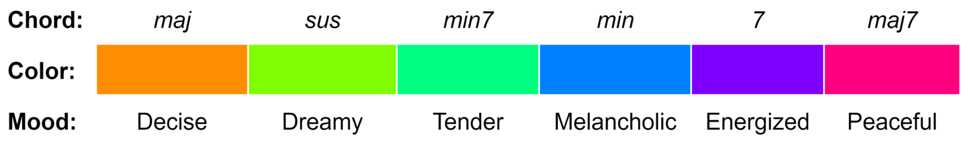


Figure 7: Mapping between chords, mood and colors.

## 3.3 Connection

The connection between the visualization and the rest of the system is actuated using OSC messages coming from the harmonizer implemented in *SuperCollider*. *Processing* needs to receive a message that indicates if the user has played a new note or if stopped playing, which root note was selected and the harmonized notes produced by the harmonizer. In order to make this possible, we created an *OscManager* SynthDef that collects the needed data and generates a custom OSC message, with the structure shown in Fig.8.

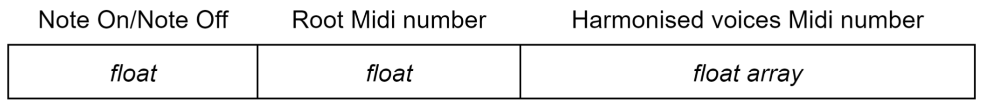


Figure 8: Osc message structure.

# 4 Conclusions

On the *Processing* side, the *oscEvent* method provided by *OscP5* and *NetP5* libraries recognizes the incoming OSC message. Once triggered, it creates an event that parses the message using a custom *OscHandler* class and then updates the parameters of the classes generating the keyboard and the landscape.

## 4.1 Implementation

The implementation paradigm follows as much as possible the Object-Oriented paradigm of Java, upon which Processing is based. Therefore we created classes and handlers for each element shown in the landscape to keep the code scalable and intelligible. In order to achieve this we avoided as much as

we could explicit calculations inside the *draw* method.
The implemented class creates from scratch each element of the visualization. Diving more in detail:

- The *RootNameDisplayer* class renders the chord root note name, following the English note convention.
- The *PianoKeyboardDisplayer* class renders the keyboard block, allowing to set which notes are currently being played by the user and by the harmonizer.
- The *Wave* class creates a single wave and offers methods that change its amplitude, velocity and color.
- The *SunHandler* class renders the sun and offers a method that changes its radius.
- The *PerspectiveHelper* class has a method that provides the distances used to create a perspective illusion, based on a set vanishing point.
- The *ChordBasedColorPicker* class creates a twelve colors palette that changes accordingly to the chord being played.
- The *ColorIntepolator* class allows us to interpolate color, creating smooth transitions.
- The *ChordRecogniser* class has a method that helps recognize the chord root and chord type being played given the root note and the notes created through the harmonization process.
- The *OscHandler* class enables the correct parsing of the incoming Osc messages, which are required to provide the type of incoming message (*NoteOn* or *NoteOff*), the root note and the harmonized notes frequencies.

## 4.2 Results

The final result of our project is a prototype for a highly portable tool that could help musicians to enrich their live sets, in particular in situations where a solid harmonic base is needed as happens in semi-acoustic sets. Nevertheless it is to remember that what we produced is still a proof of concept. For example, with better performing sensors we could afford to use a larger keyboard, easily improving the usability of the instrument.

## 4.3 Future Improvements

As a future improvement, we could use other kind of sensors connected to Arduino to increase the maximum range of each group of notes.
Regarding the harMMMLonizer at the moment only heptatonic scales are supported but it will be nice to add pentatonic, hexatonic and octatonic scales support as well.
Another idea of possible improvement is to connect an external device different from the TouchOSC application to change the parameters of the synthesizer and the harMMMLonizer. For example, connecting a joypad could be another smart and convenient way to modulate the parameters of our computer music system.
On the design part, we had the idea to build a wooden keyboard instead of using simple paper. Even if this choose makes our project less easily portable, it could be a good add for the use at home of our computer music system.