CMLS HW3 - DrumKit SuperCollider

{ 10482867 10521088 10539533 10702368
10751919 }@mail.polimi.it

June 1, 2020

Abstract

In this assignment we implemented a DrumKit application. A total of 8 different drum sounds are synthesised on SuperCollider, along with multiple user interfaces to control them using different input devices.

In this report, we are going to explain the generation of the drum sounds, the interactive system design and the MIDI/OSC protocol communication between different interfaces. The project is also available online at: `https://github.com/Lorenzoncina/DrumKit-Supercollider.git`.

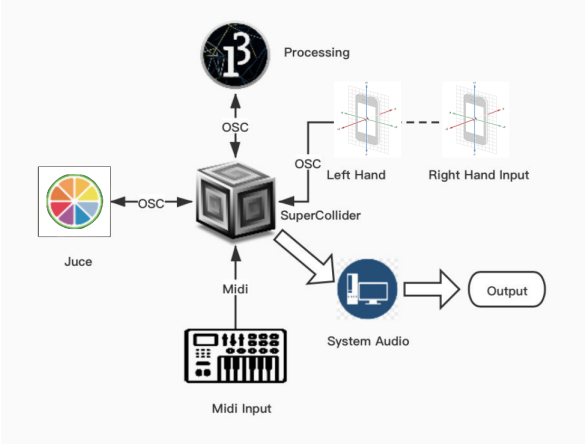

Figure 1: System Design of DrumKit Application

1 Introduction

The DrumKit application design was based on classic analog drum machines, like the Roland TR-808, but the controllers are split up on several different platforms through which it's possible to interact with the sounds. The goal was to allow the user to play the DrumKit using any of these different platforms.

In order to have multiple controllers on different devices, we made three different interfaces separately on SuperCollider, JUCE and Processing. Also, we are able to take Midi inputs from both hardware and software Midi devices and take motion movements inputs from mobile phones with OSC protocols. All the controllers are communicating and interacting with the main program on SuperCollider, which produces an output stream of music.

As always in this kind of application, it is important to respect the Model-View-Controller design pattern, in order to keep the different components that build up the whole structure well separated. In this case the Model is represented by the sound bank and their synthesizers; the View is the audio feedback produced by the machine; the Controller is represented by the different available GUIs, through which the user can interact directly (using keyboard and mouse) or indirectly (using the phone or a Midi controller).

Here in **Figure 1** we explain the connections and the flows inside our implementation.

No.	Name	Sound
1	bd	Bass Drum
2	hh5	Hi-hat
3	sn	Snare
4	cb	Cowbell
5	kc	Kick
6	fm	FM
7	ce	Clap
8	tom	Tom

Table 1: List of SuperCollider SynthDef

2 Sound Synthesis

Sounds are generated in the SuperCollider framework through Synth Definitions. We used a basic set of drum sounds enriched with non-traditional ones: bass drum, hi-hat, snare drum, cowbell, kick, frequency modulator, clap electron, tom. To better perform the Sound Synthesis some parameters of the synths can be modified in real-time by the user.

We creating 8 different Synth nodes on SuperCollider. Here in Table 1 is a list of the drum sound we generated using SynthDef Class in SuperCollider.

3 Protocols

As to the protocol part, we used both Musical Instrument Digital Interface(MIDI) and Open Sound Control(OSC) protocols for communication between different components inside our DrumKit, as **Figure 1**.

3.1 OSC communication

The main idea of communicating using the OSC protocol is as simple as it is effective: the applications involved communicate with each other by exchanging messages. First of all it is necessary to connect both SuperCollider and Processing to a network, in this case the scsynth server is the localhost. After that, the sender and the receiver must be tuned to the same port in order to create a communication channel. The OSC messages are then created during the interaction with the user and sent to SuperCollider, that will perform the requested action.

There are 3 parts of this DrumKit System that require OSC message communication.

- On the interface of Processing part, if a user triggers the *mousePress* event to ask sound generation from SuperCollider. An OSC message would be sent forward to SuperCollider and another message would be back as the feedback of that message.
- On the phone controller, the motion movements would be detected. When the sound generation action was triggered by the acceleration of movements on the phone, an OSC message with certain information would be sent back to SuperCollider as well.
- On JUCE the user has an interface inspired by a classic drum machine; OSC messages are sent to Supercollider in order to connect each graphic component to the relative audio parameter and to set the sequence patterns to be played.

3.2 MIDI communication

The MIDI commmunication for this project was limited to the mapping of a couple of different MIDI instruments to the different drum sounds. Such mapping was done manually by checking which note number the different keys/pads of the controllers we experimented with were sent in the noteOn messages and associating those note numbers to the different sounds. MIDI instruments can thus be used while also controling the

sound parameters via the Supercollider GUI, or even the JUCE one.

This mode of communication was tested with two different keyboards and an electronic drumkit.

4 Interaction

For this DrumKit application, the way how a user can interact with the main program from SuperCollider is not strictly defined by the assignment. Thus, we are free to choose whatever interaction and input device with the specific protocols to implement the message communication between each element.

We applied a combined interaction way, both using Processing and JUCE for visualization. As for the input device, we turned midi keyboards, phones and computers for user control as input peripherals. All the sound of the DrumKit are defined and generated using SuperCollider main program.

4.1 SuperCollider GUI

The SuperCollider GUI was intended to manage some parameters of every sound. The knobs have been set within a specific range, with minimum and maximum value to choose from to adjust the sounds we are going to play with pads through MIDI devices, computer keyboard or Processing GUI.

There are two or three parameters, therefore two or three knobs for every pad, that can be changed as we please to hear the sound differently from the initial settings (the values we set when creating our SynthDefs). Most of the knobs control the tone (pitch) and the amplitude of the SynthDef; other knobs were employed specifically on the characteristics of the synth, e.g. for the hi-hat we can also change the bandwidth of the noiseOsc. The window is partitioned so that for every column there is a pad relative to the synth and its knobs just below. The buttons 'assign' allow the user to assign that specific synth to a key pressed on the computer keyboard. As a result, it is possible to control the drum kit using the buttons, the keyboard keys and a midi controller at the same time.

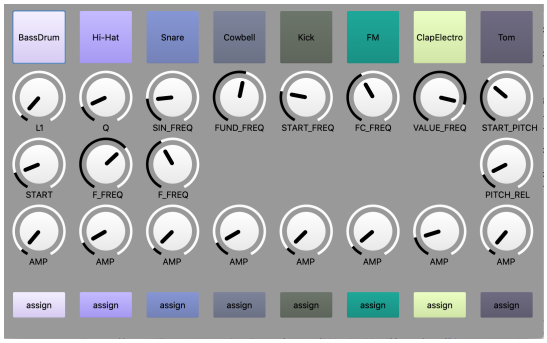

Figure 2: Graphical User Interface for SuperCollider

4.2 JUCE GUI

We implemented a GUI that allows the user to control the drum kit and all its synths' parameters via a set of slider and buttons; this interface includes a simple but fully functional 16-step sequencer that can be used to control Supercollider Patterns remotely and play them. The layout of the interface can be seen in Fig. 3. First of all, like in all drum machines, the tempo slider changes the bpm, while the instrument selector slider selects a specific sound to be sequenced using the pads on the bottom right part of the window. For each sound we also have a volume slider and other parameters, which are specific to the different sounds. The Start/Stop buttons, as the name suggests, starts and stops the reproduction of the sounds.

4.3 Processing

In the Processing framework, we let user click the screen via the specially designed interface. Different regions on the screen serve as differ-

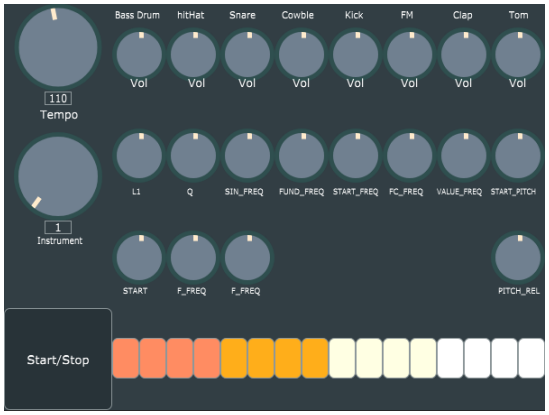

Figure 3: Juce GUI

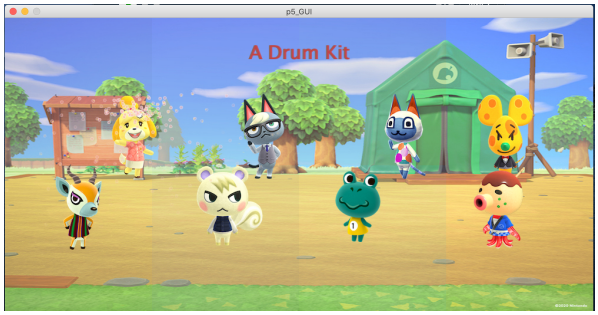

Figure 4: GUI screenshot for Processing

ent pads on drum pads. So user can click the screen to trigger different sounds.

Actually by touching and clicking different area, we are sending OSC messages to ask SuperCollider to play the sounds that have been defined before in **Section 2**. Also, with the help of oscP5 library, we can make Processing communicate fluently with SuperCollider.

We also put a particle system in the Processing visualization part, to give user some feedback after each click. When a click is detected, it will start a particle system with just one particle at the origin position from the clicking point. With the increasing of time, this particle system would increase in the number of particles inside as well. For each particle has been defined a limited life time(from 255 to 0), so after it was generated it would remain for some time on the screen and then "burnt out". If there were no existing particles inside a particle system, then the system would be dead. Also, in order to make the visual part a little more interesting, we also put the background image and some avatars from the Nintendo Game: Animal Crossing(https://animal-crossing.com).

Here are the screenshot of the visualization effect while playing and generating sound in Processing part.

4.4 Motion Control on iPhone

Nowadays there are a lot of existing OSC control surface application. With those kind of already developed app and integrated motion detection service on a modern mobile phone, we can make the device serve as drum sticks while holding, rotating or moving it to a certain position or velocity. Thus, we decided to use physical motion movement as another interactive way to control the generation of different sound.

The OSC control application we used in this application is called "Syntien". We used this app available for iOS to send OSC messages of Acceleration data(on X,Y and Z axis) to the SuperCollider server.

We also set an OSC receiver in SuperCollider to read the messages, check and map different combinations on the Acceleration values. Different value sets would trigger different defined sound in real time, to generate a series of sound we can listen to while playing.

Here in **Figure 5** is the reference axes on phone devices. We can see how to monitor the acceleration inputs from the phone device and map it as triggers the generation of the sounds. Also, in **Table 2** shows the mapping from different acceleration inputs to its sounds triggers.

With two phones at hand, we can even play this Virtual Air DrumKit in a more physical and realistic way.

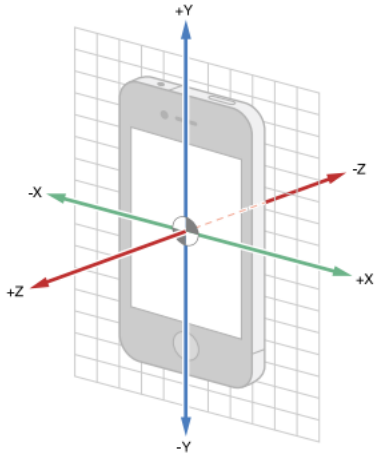

Figure 5: Reference axes on phone devices

No.	Name	Map to
1.1	$x < -5, y > 5, z > 5$	Kick
1.2	$x < -5, y < -5, z > 5$	FM
1.3	$x < -5, y > 5, z < -5$	Clap
1.4	$x < -5, y < -5, z < -5$	Tom
2.1	$x > 5, y > 5, z > 5$	Bass Drum
2.2	$x > 5, y < -5, z > 5$	Hi-hat
2.3	$x > 5, y > 5, z < -5$	Snare
2.4	$x > 5, y < -5, z < -5$	Cowbell

Table 2: Example Mappings on Two Device Input(both hands)

5 Conclusions

As we have been given many free choices for this homework, we let our fantasy fly to let our product to be as unique, and functional, as possible.

This freedom bounces back to the users when interfacing with the software we designed: they can choose to operate with it in many different ways, even more than one at the same time. From a user's perspective a software should be friendly and easy looking, in order to obtain the best experience possible.

With the final result of this interactive DrumKit, we have achieved this goal along with the potentiality of customized sounds. A further implementation of this application could be the recording of a performance, producing an output file, which can be done later someday.