ASSIGNMENT REPORT

A Multi-sound Instrument

Developed by: SOGADADO

General Idea:

We decided to create an application that contains many different Synths with a common framework: all of them must respond to different frequencies (in order to be playable via MIDI) and have four parameters that the user can set. Each of the Synths has these four parameters assigned to arbitrary elements which determine how it sounds.

The project is divided into three main files that implement various functionalities as following:

Midi.scd: This file contains the code that plugs the midi input to the Synths we created. Every time a user selects a different Synth the ~midiSynthConnect function is called, which releases any ongoing notes, frees the previous Synth, and assigns a new one using the MIDIFunc.noteOn() function. Additionally, this file boots the server and does the MIDI and audio initialization.

Gui.scd: This major part consists of the ~mainWindow global object which represents the main window. In the file “gui.csd” you can find initWindow() and showWindow() methods that initialize and show the main window respectively. The initialization process can be described as follows:

1. First, it creates and initializes all needed widgets: 2 PopUpMenus *popUpSounds* and *popUpOutput*, 4 knobs (related to 4 different effects) and corresponding labels which are objects of StaticText class;
2. Next, All these widgets are placed in horizontal or vertical layouts to make the window nicely resizable;
3. Then it sets the palette (*QtGUI.palette = palette;*) with default colours: red for buttons/knobs, black for text colour, white for background colour. We decided to make our app in the Japanese style, that’s why these colours were chosen.
4. Finally, the content of *popUpSounds* is initialized using another global object: ~*sounds*. It’s a dictionary which contains the information about our created sounds:

~sounds = Dictionary.new;

~sounds.put("ditzy", ~knobsDitzy);

…

~knobsDitzy = [["Width", ~action1], ["Max partials",~action2], ["Amp", ~action3], ["Fundamental", ~action4]];

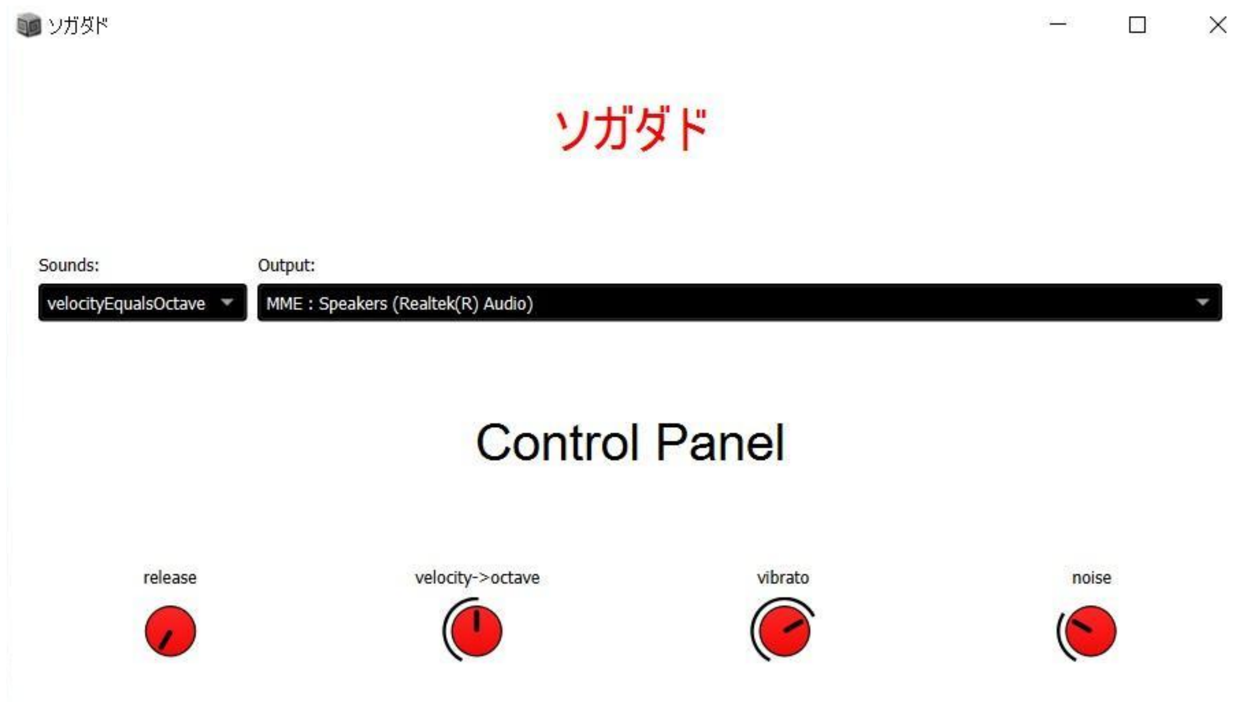
So, the key (“ditzy”) is a synthdef name, and the value (~*knobsDitzy*) is an array of pairs [knobName, knobAction]. In this example, the first knob is called “Width” and ~*action1* is executed when the ‘Width’ knob value is changed.

- ~*currentSound* global object is a synthdef name that is currently chosen. Of course, we update it every time popUpSounds item is changed.
- also, ~mainWindow is responsible for killing all the processes when it’s closing (~mainWindow.onClose_({Server.killAll;});)

Also, there are 2 other methods *initKnobs* and *updateKnobs*. The former creates knobs, sets the initial values and labels (according to the ~*sounds* and ~*paramValues* global objects) and places them on the appropriate layouts. The latter is executed every time the user changes the sound (by clicking *popUpSounds* widget).

Main.scd: This file loads all the other files and does some further global configuration like defining initial knob values and setting a different value for memory allocation.

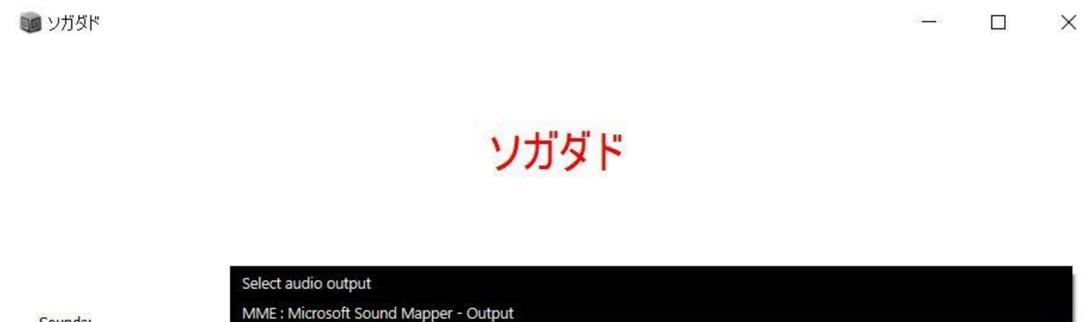
This is the general view of our application’s main interface:

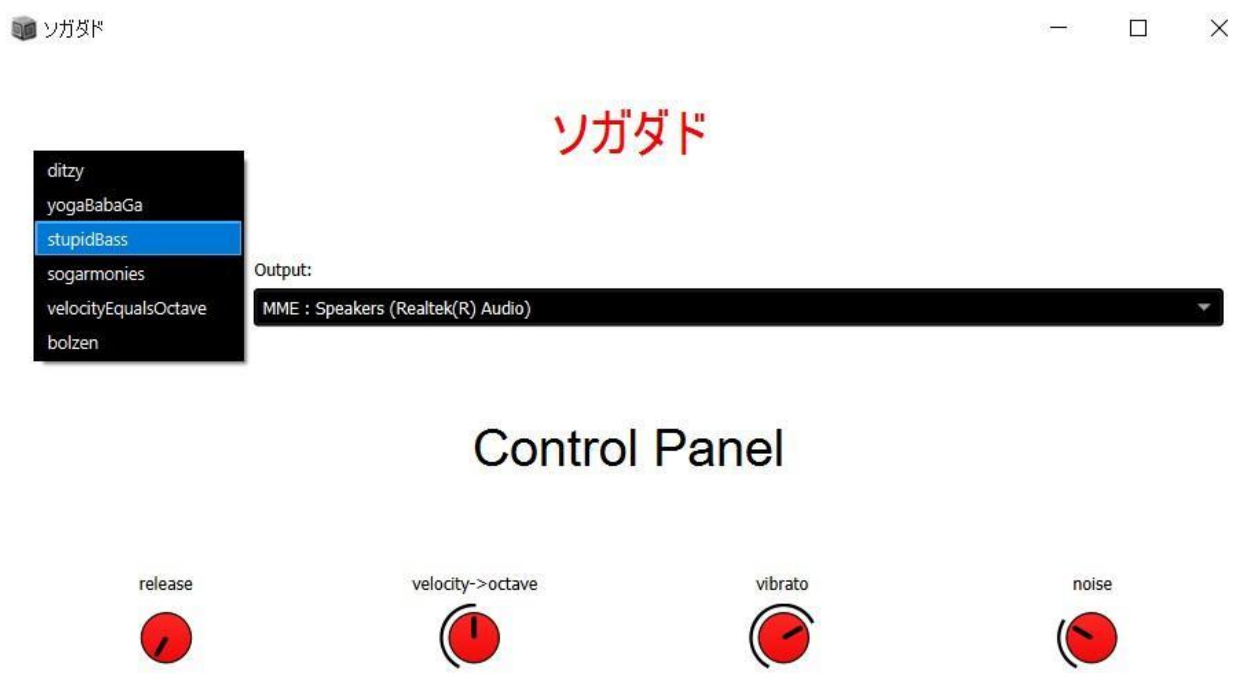


Sounds: velocityEqualsOctave

Output: MME : Speakers (Realtek(R) Audio)

Now clicking on each pop-up buttons, you can see the extended options that you can choose:





SYNTHDEFS:

velocityEqualsOctave: A dreamy sound made with additive signals. There are three sinusoids and some white noise on top. The second sinusoid is the third harmonic of the first sinusoid. The third sinusoid is at the same frequency of the first one but with a longer envelope and a velocity-controlled vibrato. Velocity also controls the amplitude and, more interestingly, the octave. The frequency is re-computed inside the Synth by mapping it over a range of octaves based on the velocity. The range is then controlled by the second gain knob providing us with a range of one octave (when the knob is at 0) up to four octaves (when the knob is at 1). The other knobs control the release time, the vibrato and the white noise amounts.

yogaBabaGa (graph below): In an attempt to create a faux microtonal sound, this Synth was constructed by five triangle waveforms at a frequency detuned proportionally to the velocity and to the first knob. The first three of these LFTri are the first three "partials". The other two are even higher partials, played through a delay. Another LFTri acts as a tremolo with its range controlled by the second knob. The third knob controls the delay and the fourth sets the amplitude of the delayed partials.

stupidBass: complex waveform created by summing 20 (!!) pulse generators at multiple frequencies. Each one is filtered by a very narrow band pass filter centered in the same frequency. As if this sound was not boring enough, the four knobs control the attack, decay, sustain and release of the envelope. Best played in the lower octaves, where it sounds like any old cheesy synth bass sound from the 80s.

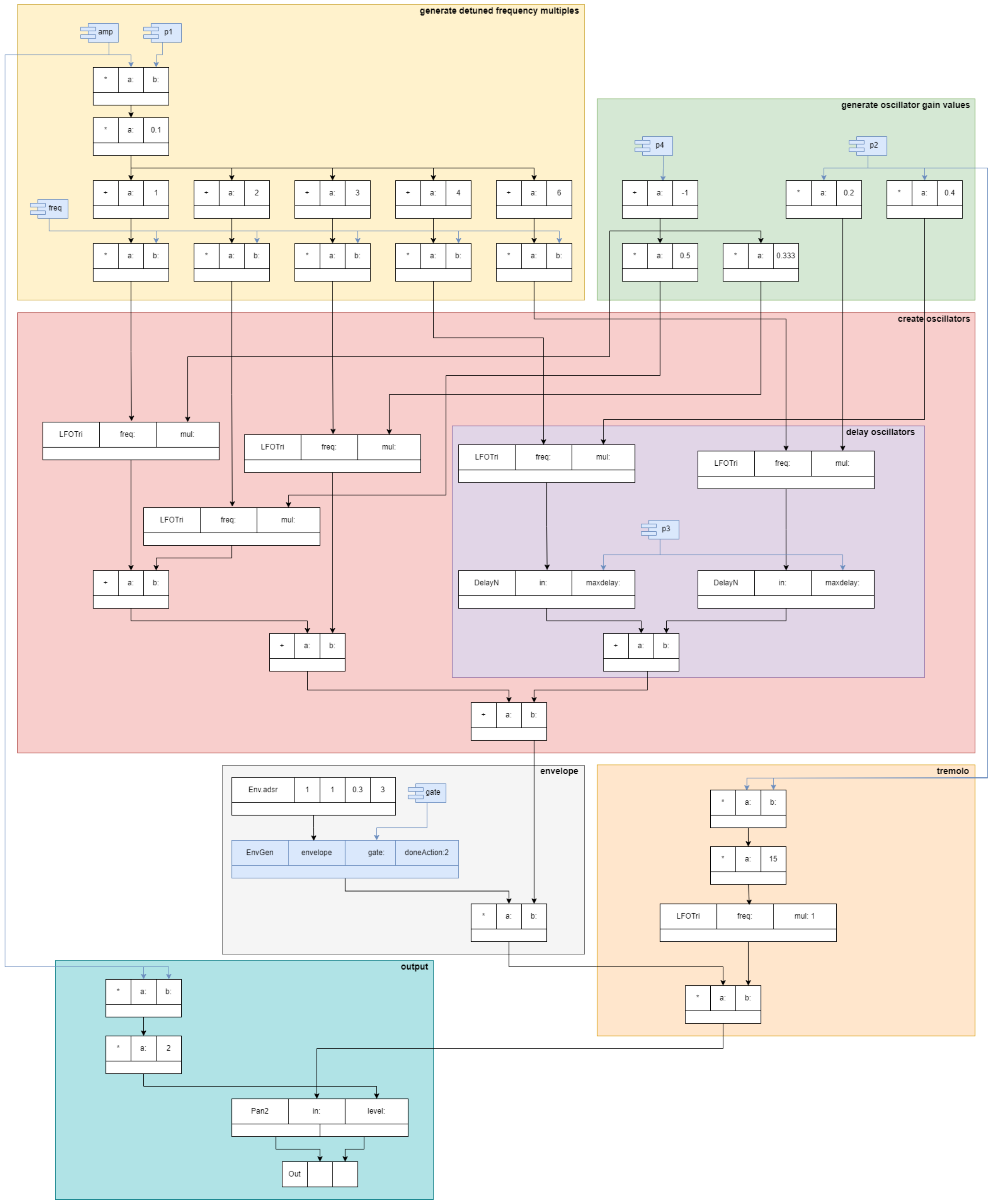
Sogarmonies: it is to serve as a simulator of a typical accompanying group of strings in an orchestra. The main tone is basically the sum of a sawtooth wave and a pulse of the same frequency. Obviously, the fundamental frequency of the tone is chosen by the midi key that is pressed by the user; however, as one would expect a set of strings, major third (1.25*fundamental frequency), and perfect fifth (1.5* fundamental frequency) of the particular note are also added to the final mix. An LFPulse with a very short duty cycle was used to control the amplitude whose frequency is determined by a knob called “harmonicajam”. The smaller its frequency is, the more it sounds as if there is a harmonica in the jam (achieved through trial and error). At the end, an envelope with a constant attack time and other properties are multiplied by the signal and a reverb whose room size is controlled by a knob is there to help realize the strings sound even better. The volume and panning are also managed by two other knobs.

Ditzy: A complicated Synthdef in which random notes are generated with octave jumps and different panning values. It is composed of two signals that generate pulse waves. LFPulse was used to transform the sound from a static tone into regular pulses with two different phases on the left and right pan (0 and 0.5). Using an LFNoise, the frequencies are assigned a new value 4 times per second with a range between fund and fund*maxPartial rounded to the nearest multiple, so in this case LFNoise will throw random overtones of the fundamental. LFPulse has been benefitted from again to create octave jumps, although it ranges between 1 and 2 this time. Finally, we added a reverb on both signals, which were then assigned to output left and right separately.

bolzen: THX intro inspired sound. The sound generation starts from a pulse wave which has a frequency that changes over time. The frequency of every instant is determined by a summation of the one associated to the MIDI Note-on signal given through the input and a percussive envelope function. It has a release time that can be controlled by the "Slide time" knob. Next, the signal generated is combined with an ulterior envelope, the release time of which can be controlled using the "Release" knob in the interface. The signal obtained is then used as an input for a resonant low pass filter whose cutoff frequency can be manipulated via the "Filter" knob, but that is also a linear function of the velocity of the midi input signal. The reciprocal of the Q factor of this resonant low pass filter is imposed as a decreasing linear function of a numerical value that can be controlled using the "Q" knob in the interface. The elaborated signal is finally given in input to a Pan2 UGen node with a level that is set according to the velocity of the input midi signal.

Graph example:

Finally, on the next page you’ll find a graph describing the internal struture of the *yogaBabaGa* synth.


generate detuned frequency multiples
amp
p1
* a: b:
* a: 0.1
+ a: 1
+ a: 2
+ a: 3
+ a: 4
+ a: 6
freq
* a: b:
generate oscillator gain values
p4
p2
+ a: -1
* a: 0.2
* a: 0.4
* a: 0.5
* a: 0.333
create oscillators
LFOTri freq: mul:
delay oscillators
p3
DelayN in: maxdelay:
+ a: b:
envelope
Env.adsr 1 1 0.3 3
gate
EnvGen envelope gate: doneAction:2
tremolo
* a: 15
LFOTri freq: mul: 1
output
* a: 2
Pan2 in: level:
Out