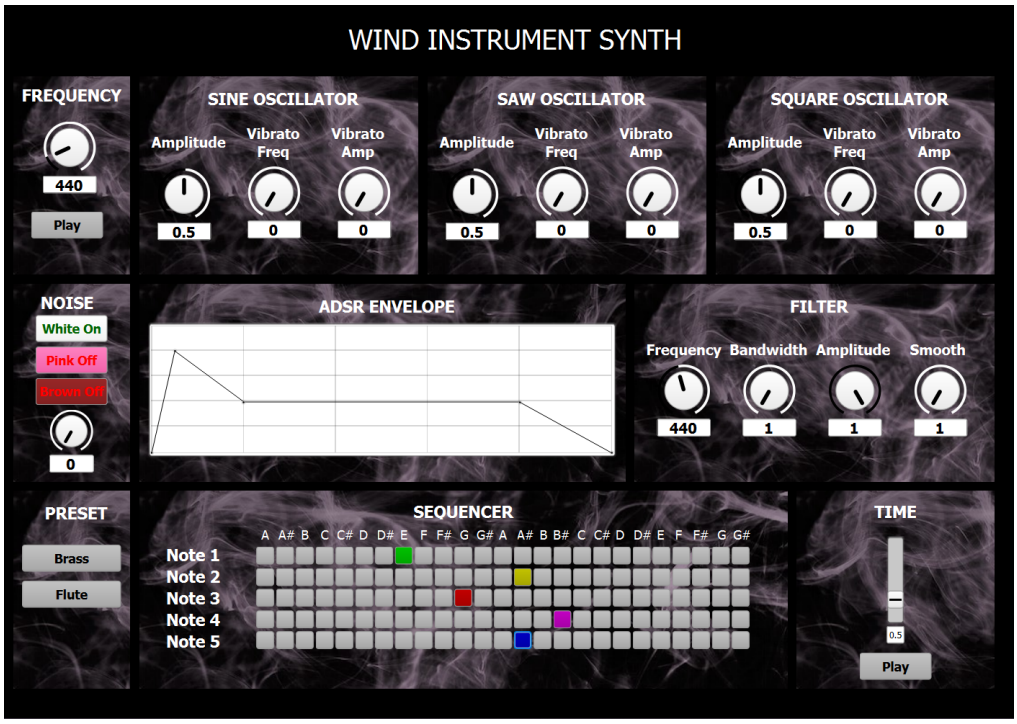
CMLS HW1 - Assignment 2: Synthesis of a wind instrument through Subtractive synthesis

Xenharmonics Group:
Baroli Gabriele
Ferrando Alessandro
Mauri Noemi
Passoni Riccardo



Wind Instrument Synth

Abstract

Wind Instrument Synth is a plugin developed with **SuperCollider**.
The goal of this assignment was to implement, in the Supercollider programming language, a synthesizer which, using subtractive synthesis techniques, is able to emulate wind instruments. Along with the instrument we also developed a GUI in order to let the user experiment with the synthesis parameters and play the instrument itself.

0.1 Introduction

Subtractive synthesis is a method of sound synthesis in which a complex waveform, or a sum of multiple simple waveforms, is filtered and modified to produce a simpler sound.

We implemented each necessary step in order to perfrom this particular synthesis technique, namely: multiple oscillators, a filter, an envelope shaper and a noise generator (this last optional step was added in order to better emulate wind instruments).

1 Implementation - GUI and sequencer

Our project's GUI is shown in the front page of this report.
We'll now provide a detailed explanation of each of the blocks present in the instrument, describing each parameter that the user can interact with.

- In the top left corner of the instrument the frequency selector is located. It provides a knob with which the synth's frequency can be set, it also allows to hear the selected tone generated by the synth by toggling the *play/stop* button placed underneath the pitch selector.

- Still in the same first row, there are three remaining blocks. These are used to interact with the parameters of the three oscillators involved in the signal generation of the instrument. These oscillators respectively create sine, sawtooth and a square waves. Each oscillator provides three parameters the user can interact with:

Amplitude: selects the range of the wave, by default it's ± 1

Vibrato Frequency: frequency of the vibrato effect applied to the oscillator

Vibrato Amplitude: depth of the vibrato effect

- In the row below, starting from the right, the GUI gives access to the parameters related to the synth's noise generator. It presents the user with a knob that controls the amplitude of the noisy signal and with a choice between different noise types:

  White Noise: sound that contains all frequencies at equal intensities, energy is constant across the spectrum

  Pink Noise: noise that decreases in intensity as frequency increases (inversely proportional)

  Brown Noise: type of noise with an intensity that decreases by 6 dB per octave as the frequency increases

- Further to the right, we have a visual controller that allows to customize the envelope of the signals, providing the option to edit the synth's ADSR values. The points in the graph can be moved in order to set the envelope times and levels.

```
1  b = EnvelopeView(envBox, Rect(0, 0, 520, 150))
2      .drawLines_(true)
3      .selectionColor_(Color.red)
4      .drawRects_(true)
5      .resize_(5)
6      .step_(0.05)
7      .action_({arg x;
8     switch (x.index,
9        0,    { start_time=x.value[0][0]; endattack_level=x.value
             [1][0] },
10       1,    { attack_time=x.value[0][1]; attack_level=x.value
             [1][1] },
11       2,    { decay_time=x.value[0][2]; decay_level=x.value
             [1][2] },
12       3,    { sustain_time=x.value[0][3]; sustain_level=x.value
             [1][3] },
13       4,    { release_time=x.value[0][4]; release_level=x.value
             [1][4] }
14 );})
15     .thumbSize_(5)
16     .value_([[start_time, attack_time, decay_time, sustain_time
          , release_time],[endattack_level, attack_level,
          decay_level, sustain_level, release_level]]);
```

Listing 1: Implementation of the instrument's envelope block

- The last component in the audio processing chain is the filter block: we've opted for a *bandpass* filter, whose **cutoff frequency**, **bandwidth**, **amplitude** and **cutoff curve smoothing** (the slope of the filter at the cutoff) are controllable with the provided knobs.

- Trying out different combinations of parameters, we've found two specific combinations which emulate two instruments, that we provide in the instruments as presets. These resemble respectively a **brass** instrument and a **flute**.
  These presets are accessed using the labeled buttons in the bottom left part of the instrument, these buttons work the same way as the "play" button present in the first block of the instrument.

- The last block at the bottom of the plugin controls the sequencing algorithm.
  This section allows the user to listen to the resulting waveform in a loop of notes, without having to click the play button for each note. We've implemented a five stage sequencer, for each stage a row of buttons lets the user select the corresponding note following the labeled notes on top:

```
b_1_1 = Button.new(seqBox, bounds:Rect(0,0,20,20)).states_([
    [" ", Color.black, Color.grey(0.7)],
   ["", Color.green(0.7), Color.green(0.7)]
]).font_(Font("Monaco", 14, true))
.action_({|b|
    if (b.value == 1,
     {ampnote1_1=mute1; ampnote1_2=mute2; ampnote1_3=mute3;
         note1=1; b_1_2.value=0; b_1_3.value=0; b_1_4.value=0;
         b_1_5.value=0; b_1_6.value=0; b_1_7.value=0; b_1_8.value
         =0; b_1_9.value=0; b_1_10.value=0; b_1_11.value=0;
         b_1_12.value=0; b_1_13.value=0; b_1_14.value=0; b_1_15.
         value=0; b_1_16.value=0; b_1_17.value=0; b_1_18.value=0;
          b_1_19.value=0; b_1_20.value=0; b_1_21.value=0; b_1_22.
         value=0; b_1_23.value=0; b_1_24.value=0; b_1_25.value
         =0;}, {  ampnote1_1=0; ampnote1_2=0; ampnote1_3=0;
})});
```

Listing 2: Implementation of the sequencer (1)

- The sequencer is started and stopped by clicking the *Play* button, once pressed it starts a sequence which cycles over the MIDI note numbers corresponding to the active buttons. The playback speed of the sequencer is controlled by the slider above the button, it specifies the interval that passes between two consecutive notes.

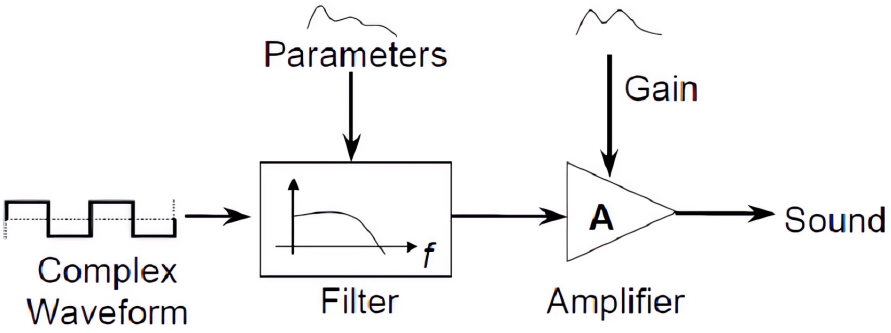
Note to the user: excessive duration of the envelope with respect to the sequencer timing may result in overcrowding the server. **Caution is advised**.

```
playSeq = Button.new(seq_time, bounds:Rect(0,0,80,30

)).states_([
     ["Play", Color.black, Color.grey(0.7)],
    ["Stop", Color.red(0.7), Color.grey(0.7)]
]).action_({|b|
    if(b.value == 1,
        {p = Pseq([note1, note2, note3, note4, note5], inf).
            asStream;
        q = Pseq([ampnote1_1, ampnote2_1, ampnote3_1, ampnote4_1,
             ampnote5_1], inf).asStream;
        r = Pseq([ampnote1_2, ampnote2_2, ampnote3_2, ampnote4_2,
             ampnote5_2], inf).asStream;
        t = Pseq([ampnote1_3, ampnote2_3, ampnote3_3, ampnote4_3,
             ampnote5_3], inf).asStream;{
inf.do{
    c = Synth(\wind_instrument, [ . . . ]) ; (tempo).wait ;
        }}.fork;},
        {p.stop;c.free;})}).font_(Font("Monaco", 14, true));
```

Listing 3: Implementation of the sequencer (2)

2 Implementation - Subtractive Synthesis



As we've mention in the beginning of this report, subtractive synthesis is a method of sound synthesis in which a complex waveform (usually a sawtooth, square or triangle wave) is filtered and modified to produce a simpler sound. Certain frequencies then get removed in the following steps.

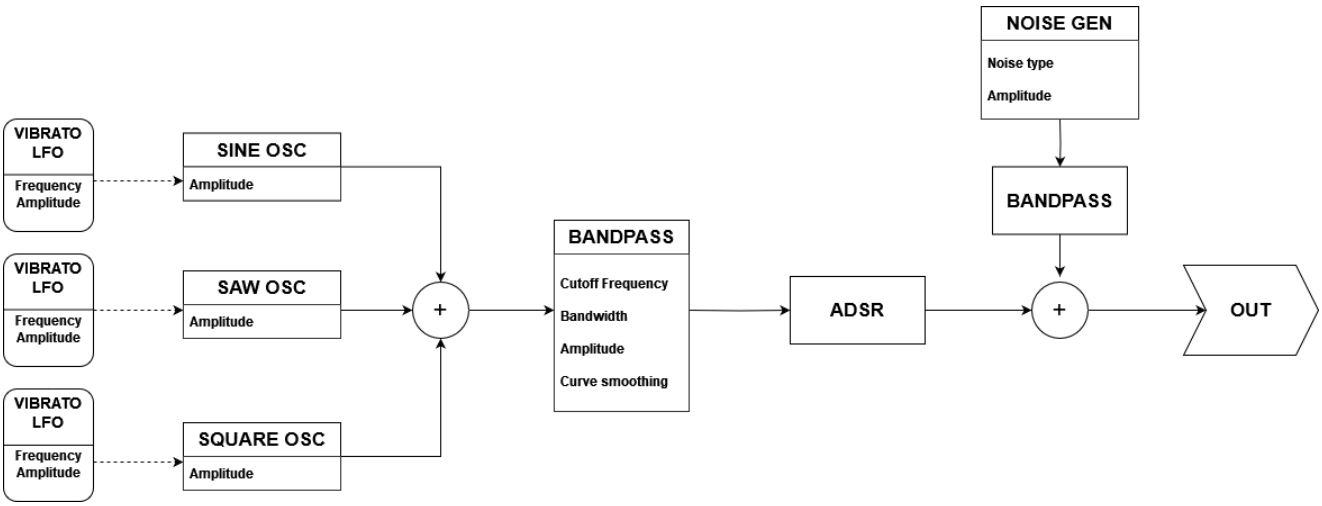


Figure 1: Block diagram of the plugin

This is usually performed by using a filter, which may be controlled by an LFO. Then as a last step the envelope of the signal can be shaped.

The waveform in our instrument is created by using three oscillators (sine, saw, square), this is implemented using the deterministic generator Supercollider classes *SinOsc*, *Saw* and *Pulse*. For each, the first five harmonics with respect to the chosen frequency are summed, furthermore he amplitude of the harmonics is geometrically tapered (this is all performed using *.geom*. A layer of noise is then added, to emulate the "breath" effect, using the respective classes for each noise type. All this constitutes the complex waveform which is then filtered to produce the desired sound.

```
1  osc1=SinOsc.ar([freq,freq] * SinOsc.kr(rate1,0,depth1
       *0.01,1) * harms, mul:Array.geom(num, 1*mute1, amp1*mute1
       ));
2  osc2=Saw.ar([freq,freq] * SinOsc.kr(rate2,0,depth2*0.01,1) *
        harms, mul:Array.geom(num, 1*mute2, amp2*mute2));
3  osc3=Pulse.ar([freq,freq] * SinOsc.kr(rate3,0,depth3*0.01,1)
        * harms, mul:Array.geom(num, 1*mute3, amp3*mute3));
4
5  sig = sig*amp*env;
6  white_noise = WhiteNoise.ar(white_noise*0.005);
7  pink_noise = PinkNoise.ar(pink_noise*0.01);
8  brown_noise = BrownNoise.ar(brown_noise*0.01);
```

Listing 4: Oscillators and noise

The oscillator signals and noise are both filtered through a bandpass, the class *BBandPass* is used for this purpose, using all the appropriate user controlled parameters.

```
1 sig = BBandPass.ar(osc1+osc2+osc3, filter_freq * XLine.kr(0.1,
    smooth,0.01), filter_bw, filter_mul);
2
3 noise = BBandPass.ar(white_noise + pink_noise + brown_noise,
    filter_freq * XLine.kr(0.1,smooth,0.01), filter_bw,
    filter_mul) * env;
```

Listing 5: Oscillators and noise

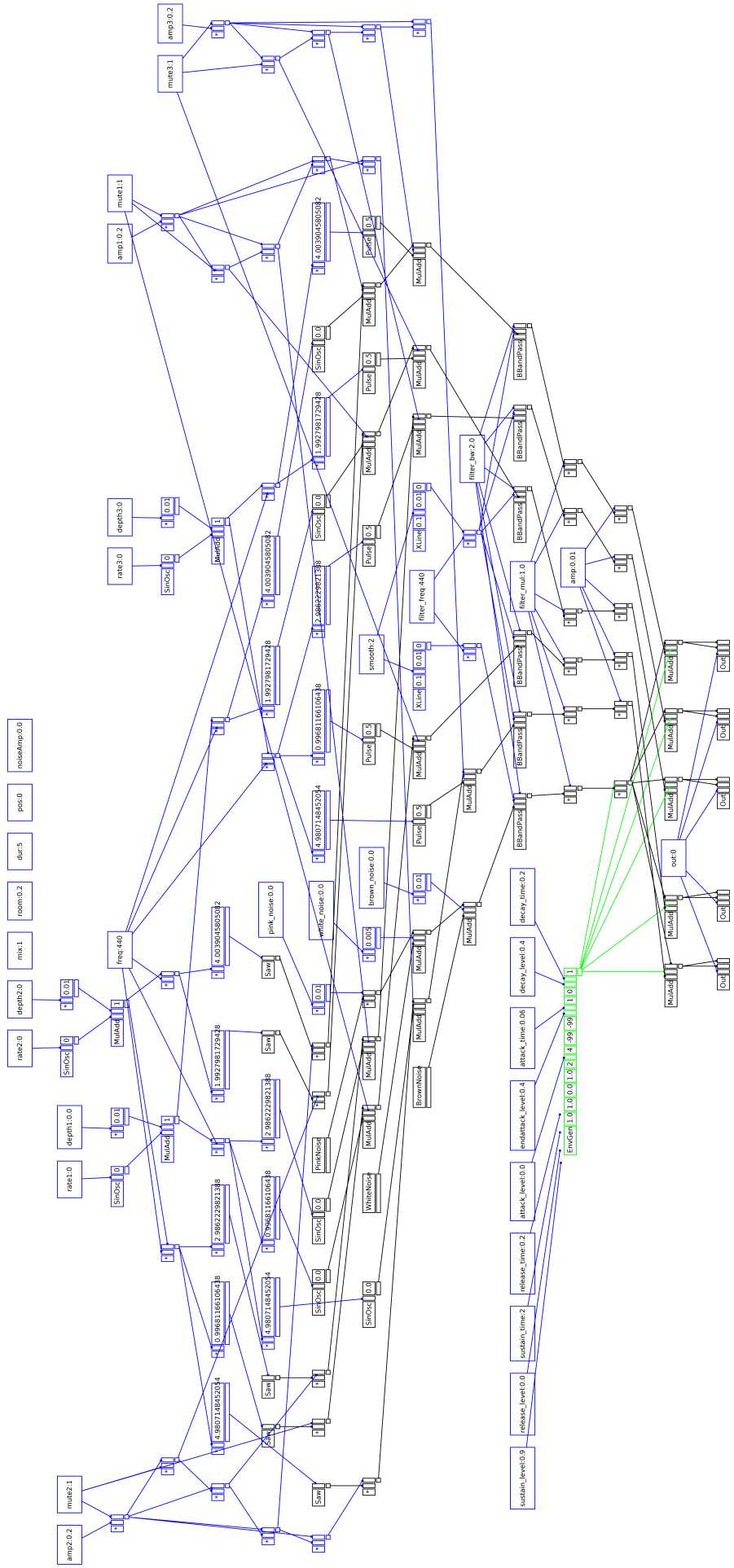
For the last step, to complete the subtractive synthesis processing chain, we apply the ADSR envelope to the signal. The env variable, created using the *EnvGen* class (controlled by the graph present in the GUI), is multiplied by all the signals we've just discussed.

```
1 env = EnvGen.ar(Env.new([ attack_level, endattack_level,
    decay_level, sustain_level, release_level], [attack_time,
    decay_time, sustain_time, release_time], 'linear' ),
    doneAction:2);
2
3 . . .
4
5   Out.ar(out, (sig+noise)!2);
```

Listing 6: Oscillators and noise

The synth is wrapped into a *SynthDef* called **"wind instrument"**, which is then used to initialize the generators needed to output sound.
In order to run the plugin the synth definition must be ran, followed by the parenthesis-wrapped code below.

The resulting generated UGen graph can be seen in the following page.


amp3:0.2
mute3:1
mute1:1
amp1:0.2
depth3:0
rate3:0
freq:440
noiseAmp:0.0
pos:0
dur:5
room:0.2
mix:1
depth2:0
rate2:0
depth1:0.0
rate1:0
mute2:1
amp2:0.2
filter_bw:2.0
filter_freq:440
filter_mul:1.0
amp:0.01
decay_time:0.2
decay_level:0.4
attack_time:0.06
endattack_level:0.4
attack_level:0.0
release_time:0.2
sustain_time:2
release_level:0.0
sustain_level:0.9
EnvGen
out:0
smooth:2
brown_noise:0.0
white_noise:0.0
pink_noise:0.0
BrownNoise
PinkNoise
WhiteNoise
MulAdd
SinOsc
Saw
Pulse
XLine
BBandPass