Politecnico di Milano
Computer Music: Languages and Systems
Academic Year 2022-2023
Eray Özgünay – Sebastian Mendez – Harry Foley

POLITECNICO
MILANO 1863

M-RAM
Musical Room Ambience Monitor

Overview

M-RAM, Musical Room Ambience Monitor, is designed to provide generative ambient music and sounds for a user's bedroom. The project was inspired by the concept behind Brian Eno's pioneering album, 'Ambient 1: Music for Airports", where he played with the idea that an indoor space should have a soundtrack that is suited to it. He felt that indoor environments should have music that "induce calm and a space to think" while remaining "as ignorable as it is interesting".

From this idea we thought it would be interesting to create a system that is suited to a bedroom. So for example, at night we play a very soft chord on a pad with the sound of rain to provide a calming feeling while the user tries to sleep. And then in the morning wake them with a gentle slow melody before during the day transitioning to a more stimulating fast paced piece.

M-RAM uses the time of day, ambient light and motion to alter the sound and music that is played. It is primarily based in SuperCollider and also uses Arduino for data acquisition and Processing for GUI.

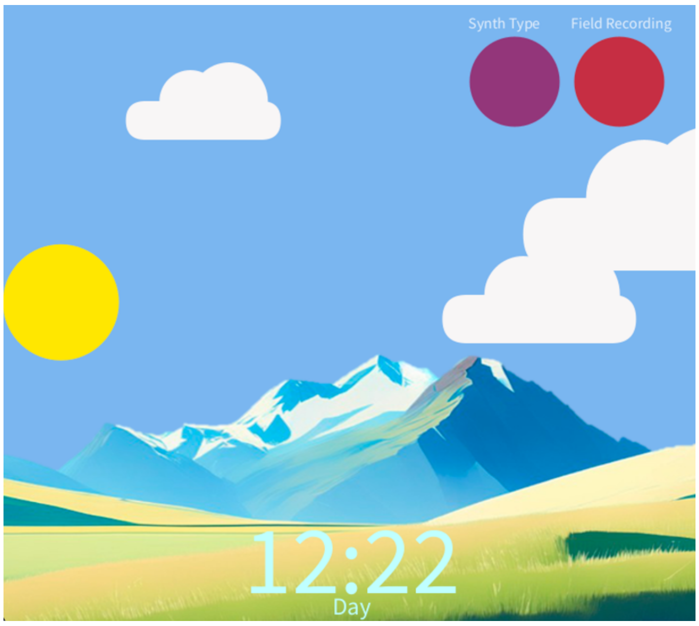


Figure 1: M-RAM GUI

Block Diagram

As we can see in the block diagram below, SuperCollider is at the heart of the M-RAM system. Supercollider is responsible for both deciding the behaviour of the system and the generation and playback of sound.

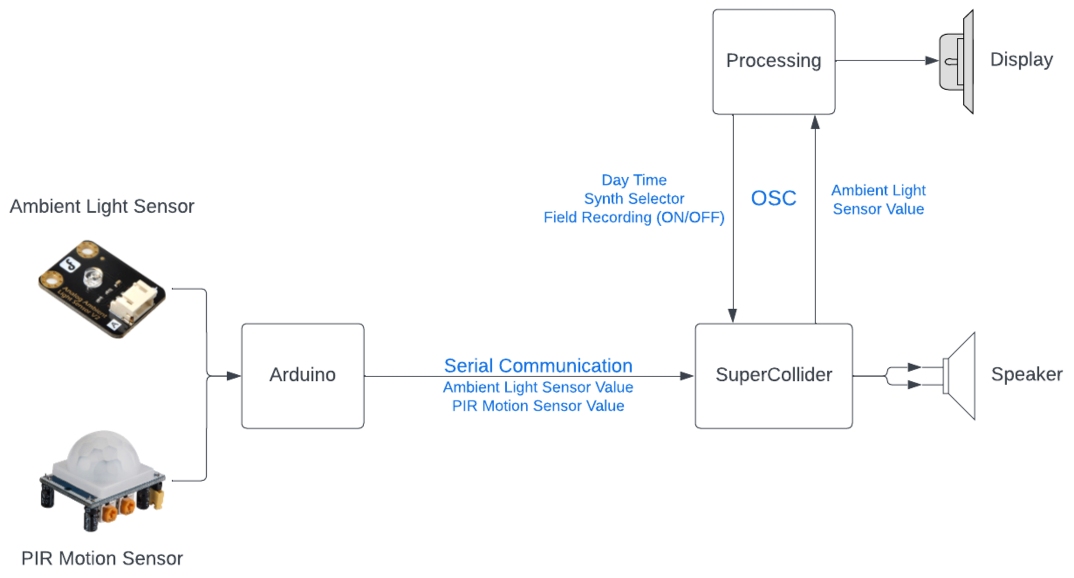


Figure 22: Block Diagram of M-RAM

SuperCollider receives serial messages from Arduino about the data acquired from its sensors. The Arduino uses a PIR Motion Sensor and an Ambient Light Sensor to acquire data from the environment. The PIR Motion Sensor should be placed facing into the room where the user will be. The Ambient Light Sensor should be placed facing out of a window in the room, as to acquire the natural ambient light outside.

The values transmitted from the Ambient Light Sensor range from 0 for total darkness to 1024 for maximum brightness. The PIR Motion Sensor is a digital sensor, it transmits a series of HIGH values when motion is detected and a series of LOW values when no motion is present. The temporal sensitivity of the sensor can be adjusted using a variable potentiometer on the sensor's board.

SuperCollider also interfaces with Processing to provide a GUI. Processing sends SuperCollider OSC messages containing information on the time of day and user selected controls, that are selecting different synth types or deciding whether the background field recordings played or not.

SuperCollider

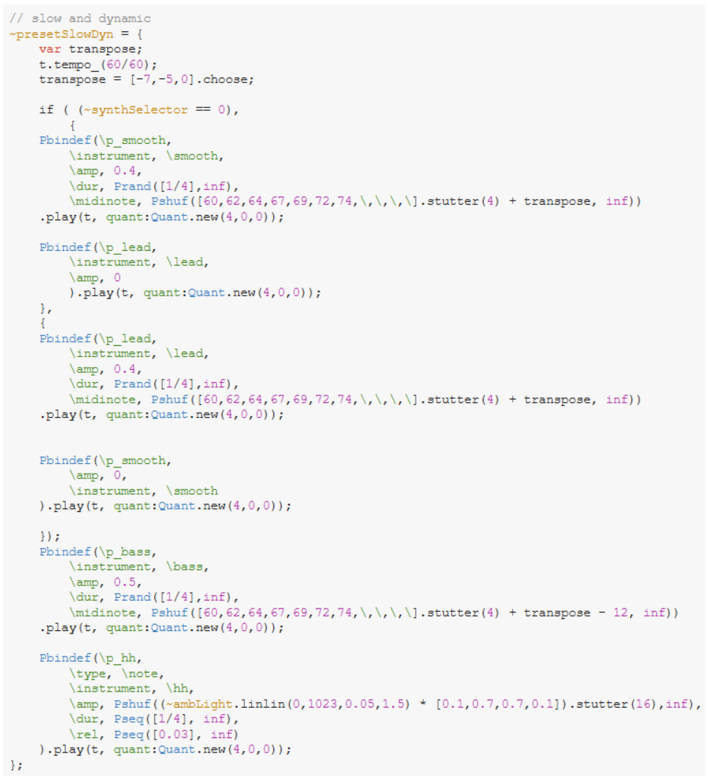
SuperCollider is at the centre of the M-RAM system.

First we define 7 synths that will be used by the programme: A smooth synth, a lead synth, a relaxing pad synth, a bass synth, a hihat, a buffer player synth and a white noise synth. Each of these was created in the conventional manner using SynthDef.

Next we create a series of functions that represent different patterns that can be played by the system. These are named Slow Repetitive, Slow Dynamic, Fast Repetitive, Fast Dynamic. As an example we will examine the function Slow Dynamic. In Figure 2.

First the global tempo is set to 60 BPM. Next, we define three patterns using Pbindef. The first Pbindef sets the behaviour for the saw tooth synth names "p_smooth". The duration of each note is simply set to a quarter note.

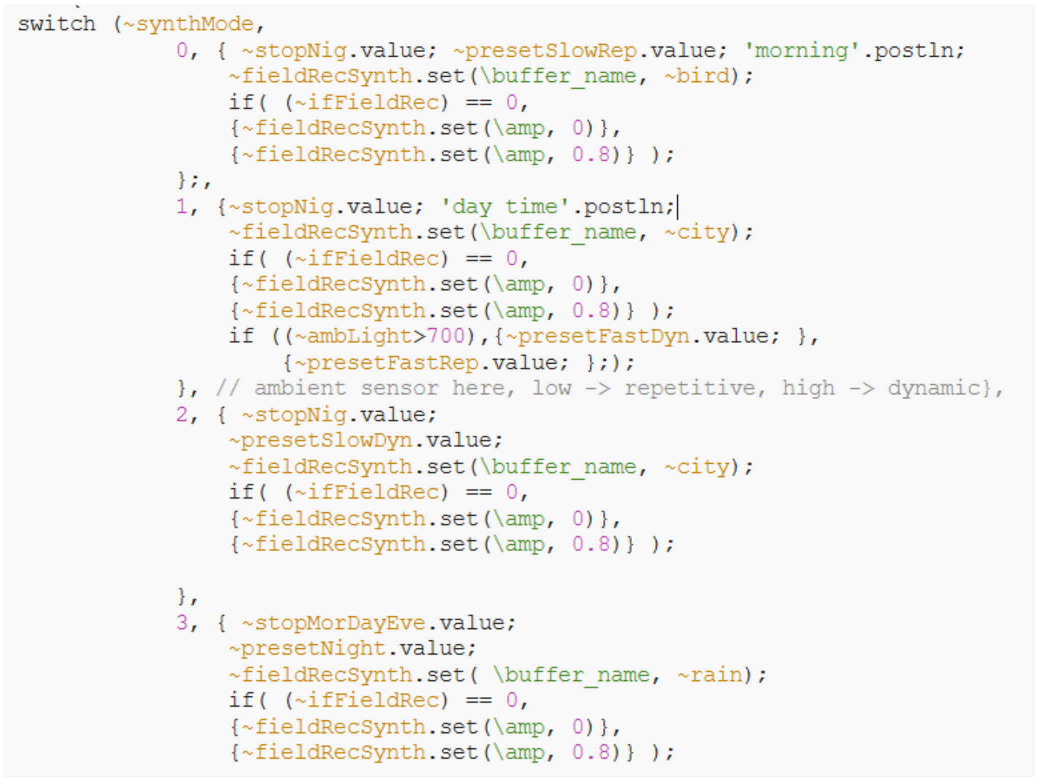
The midinote of the synth is interesting. Inside an array we have a mixture of midinote values and rests. Attached to this is the stutter function, which repeats the array 4 times.
We also then add to the current midinote number a -5, -7 or 0 using the choose function to add variation. This equates to altering the note to a perfect 4th, perfect 5th or unchanged respectively. The order the notes in the array are played back is altered each time the Pbindef is called using the Pshuf function. Also for Pbindef's, Quant feature is utilised. This restricts when the Pbindef can start playing to every four beats. This is used on all patterns in the programme to guarantee that they start at the beginning of the next bar.



```
// slow and dynamic
~presetSlowDyn = {
    var transpose;
    t.tempo_(60/60);
    transpose = [-7,-5,0].choose;

    if ( (~synthSelector == 0),
        {
    Pbindef(\p_smooth,
        \instrument, \smooth,
        \amp, 0.4,
        \dur, Prand([1/4],inf),
        \midinote, Pshuf([60,62,64,67,69,72,74,\,\,\,\].stutter(4) + transpose, inf))
    .play(t, quant:Quant.new(4,0,0));

    Pbindef(\p_lead,
        \instrument, \lead,
        \amp, 0
        ).play(t, quant:Quant.new(4,0,0));
    },
    {
    Pbindef(\p_lead,
        \instrument, \lead,
        \amp, 0.4,
        \dur, Prand([1/4],inf),
        \midinote, Pshuf([60,62,64,67,69,72,74,\,\,\,\].stutter(4) + transpose, inf))
    .play(t, quant:Quant.new(4,0,0));


    Pbindef(\p_smooth,
        \amp, 0,
        \instrument, \smooth
    ).play(t, quant:Quant.new(4,0,0));

    });
    Pbindef(\p_bass,
        \instrument, \bass,
        \amp, 0.5,
        \dur, Prand([1/4],inf),
        \midinote, Pshuf([60,62,64,67,69,72,74,\,\,\,\].stutter(4) + transpose - 12, inf))
    .play(t, quant:Quant.new(4,0,0));

    Pbindef(\p_hh,
        \type, \note,
        \instrument, \hh,
        \amp, Pshuf((~ambLight.linlin(0,1023,0.05,1.5) * [0.1,0.7,0.7,0.1]).stutter(16),inf),
        \dur, Pseq([1/4], inf),
        \rel, Pseq([0.03], inf)
    ).play(t, quant:Quant.new(4,0,0));
};
```

Figure 33: Slow Dynamic Preset

Changing the playing presets (depending on the daytime) is done as in the following code block, inside our main Routine:

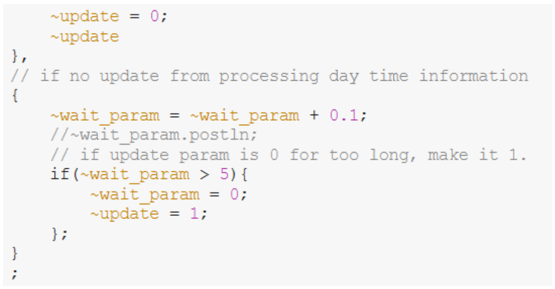


```
switch (~synthMode,
            0, { ~stopNig.value; ~presetSlowRep.value; 'morning'.postln;
                ~fieldRecSynth.set(\buffer_name, ~bird);
                if( (~ifFieldRec) == 0,
                {~fieldRecSynth.set(\amp, 0)},
                {~fieldRecSynth.set(\amp, 0.8)} );
            };,
            1, {~stopNig.value; 'day time'.postln;
                ~fieldRecSynth.set(\buffer_name, ~city);
                if( (~ifFieldRec) == 0,
                {~fieldRecSynth.set(\amp, 0)},
                {~fieldRecSynth.set(\amp, 0.8)} );
                if ((~ambLight>700),{~presetFastDyn.value; },
                    {~presetFastRep.value; };);
            }, // ambient sensor here, low -> repetitive, high -> dynamic},
            2, { ~stopNig.value;
                ~presetSlowDyn.value;
                ~fieldRecSynth.set(\buffer_name, ~city);
                if( (~ifFieldRec) == 0,
                {~fieldRecSynth.set(\amp, 0)},
                {~fieldRecSynth.set(\amp, 0.8)} );

            },
            3, { ~stopMorDayEve.value;
                ~presetNight.value;
                ~fieldRecSynth.set( \buffer_name, ~rain);
                if( (~ifFieldRec) == 0,
                {~fieldRecSynth.set(\amp, 0)},
                {~fieldRecSynth.set(\amp, 0.8)} );
```

Figure 44: Switch statement in the main Routine to set which presets and field recordings are playing

The switching between the presets playing is done by a switch. For morning, day and evening, we first stop the night preset in case it is playing, and vice versa. Then we call their corresponding functions. Update is done if the ~update variable is 1, and ~stopFlag (a Boolean flag that determines if the sounds should stop depending on the infrared sensor) is 0. If ~stopFlag is 1, then all the amplitudes go 0 and the sounds stop.

With ~wait_param we are making sure to call the same day time function after some time to prevent same melodies playing over and over since its melody is shuffled when each function is called. In that way, if the ~update parameter is 0 for too long, we prevent the same melody playing for too long.



```
    ~update = 0;
    ~update
},
// if no update from processing day time information
{
    ~wait_param = ~wait_param + 0.1;
    //~wait_param.postln;
    // if update param is 0 for too long, make it 1.
    if(~wait_param > 5){
        ~wait_param = 0;
        ~update = 1;
    };
}
;
```

Figure 55: Updating the melody even if there is no update but the same melody is playing for too long

Looping and resetting the ~main routine which contains the previous code block is done by a task, it is waiting 0.01 seconds in each iteration to prevent looping constantly and crashing everything. We have also 0.01 seconds wait in every Routine to prevent each routine to looping infinitely without giving a chance to routine to reset.

```
~mainTask = Task({
    loop {
        ~stopFlag = ~infRed;
        ~main.reset;
        0.01.wait;
        //'waited 1'.postln;

    }
});
```

Figure 66: The main task that is calling the routine

Field Recordings Buffer

To load the field recordings, we utilize Buffers. We save the field recordings to some buffers using global variables:

```
~loadAudioIntoBuffers = {
    arg samples;
    ~numframes = samples;
    d = Buffer.alloc(s, ~numframes);
    ~rain = Buffer.read(s,Platform.resourceDir +/+ "sounds/rain_traffic.mp3",0,~numframes);
    ~crickets = Buffer.read(s,Platform.resourceDir +/+ "sounds/crickets_light_rain.mp3",
0,~numframes);
    ~bird  = Buffer.read(s,Platform.resourceDir +/+ "sounds/bird-foley.wav",0,~numframes);
    ~city  = Buffer.read(s,Platform.resourceDir +/+ "sounds/sunny_day.mp3",0,~numframes);
};

~loadAudioIntoBuffers.value(44100*5);
```

Figure 7: Loading the files into buffers

In SynthDef, we send the audio signal to output Bus using PlayBuf in audio rate:

```
SynthDef(\fieldSamplePlayer, {
    |buffer_name, out, amp = 0.3|

    Out.ar([0,1],PlayBuf.ar(1, buffer_name, BufRateScale.kr(d), doneAction: Done.freeSelf, loop:
1.0)) }).add;
```

Figure 78: SynthDef for the buffer player

In our main Routine, depending on the state of the day, we set the arguments buffer_name (global variable's name corresponding to the field recording) and amplitude. The field recordings depending on states of the day are given below:

Table 1: Summary of the behaviour of M-RAM

State of Day	Light Sensor	Motion Sensor	SuperCollider Behaviour	Field Recording
Morning	Mapped to hihat volume and dynamicity of the melody	Mapped to Volume	Slow Repetitive Or Slow Dynamic	Bird Song
Day	Mapped to hihat volume	Mapped to Volume	Fast Dynamic	City Scape
Evening	Mapped to hihat volume	Mapped to Volume	Slow Dynamic	City Scape
Night	Mapped to LPF	N/A	Soft Chord	Rain

Arduino

For the development of this project, we implemented two sensors and a microprocessor board Arduino MKR Wi-Fi 1010. The first sensor, which acquires the ambient light level has an analogue input and the second sensor, the motion detector provides a digital input, both connected directly to the board as showed in the following figure.

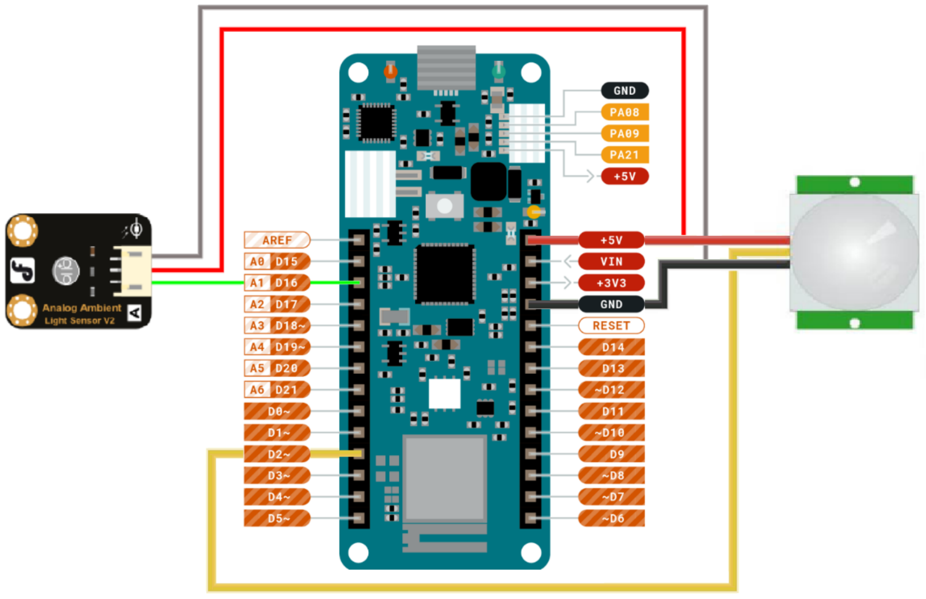


Figure 89: Pin connections of Arduino

In the setup function the pin mode for the digital signal is assigned, while in the loop function the values obtained by the sensors are printed in the monitor console. After each value of the Ambient Light sensor, it is printed the letter “a” and after the values of the motion sensor it is printed the letter “b”, thus Supercollider is able to interpret the Ascii value corresponding to each sensor and then save the value of the sensor lecture to be process.
The motion sensor board is designed to take snapshots of the IR light in the environment. It compares new IR light values to the previously take snapshot, if there is no difference in light level it continuously sends low values to the board. If there is a difference in light level the sensor will send a series of high values to the Arduino, until it accepts the new light levels as baseline and resumes sending low values.
With the sensor transmitting short bursts of high values there is a chance that SuperCollider may miss these values if the burst falls between the period that SuperCollider reads serial messages.
To counter act this behaviour we have programmed a counter element into the Arduino code. It operates as follows:

1. If the sensor sends a HIGH value, a counter values is set and a HIGH value is sent to SC.
2. If the sensor then goes LOW the counter is decremented and a HIGH values is sent to SC.
3. Once the counter goes to 0, LOW values are sent to SC, until step 1 is triggered again by new HIGH values from the sensor.

```
int M_sensor = 2;            // the pin that the sensor is attached to
int counter = 0;
void setup()
{
  pinMode(M_sensor, INPUT);   // initialize sensor as an input
  Serial.begin(9600);                // initialize serial
}
void loop()
{
  int light, motion;
  delay(100);
  light = analogRead(1);      //connect light sensor to Analog 0
  Serial.print(light);        //print the value to serial
  Serial.print("a");          //print "a" to indicate that is light sensor value

  motion = digitalRead(M_sensor);   // read sensor value
  Serial.print("\n");

  if (motion == 1) {            // check if the sensor is HIGH
    counter = 10*5;           // reset counter
    Serial.print(1);              // send 1 to SC
    Serial.print("b");          // print "b" to indicate that is motion sensor value
  }
  else if (counter > 0){     // if counter > 0 continue sending 1 to SC
    Serial.print(1);
    Serial.print("b");
    counter = counter - 1;
  }
  else {                            // else there has been no motion in the defined period of
recent time
    Serial.print(0);            //so send SC 0
    Serial.print("b");
  }
```

Processing

Processing is responsible for providing the user with a real time display of M-RAM's state. As this GUI would always be displayed in the user's room we wanted it to be simple, functional and fun.

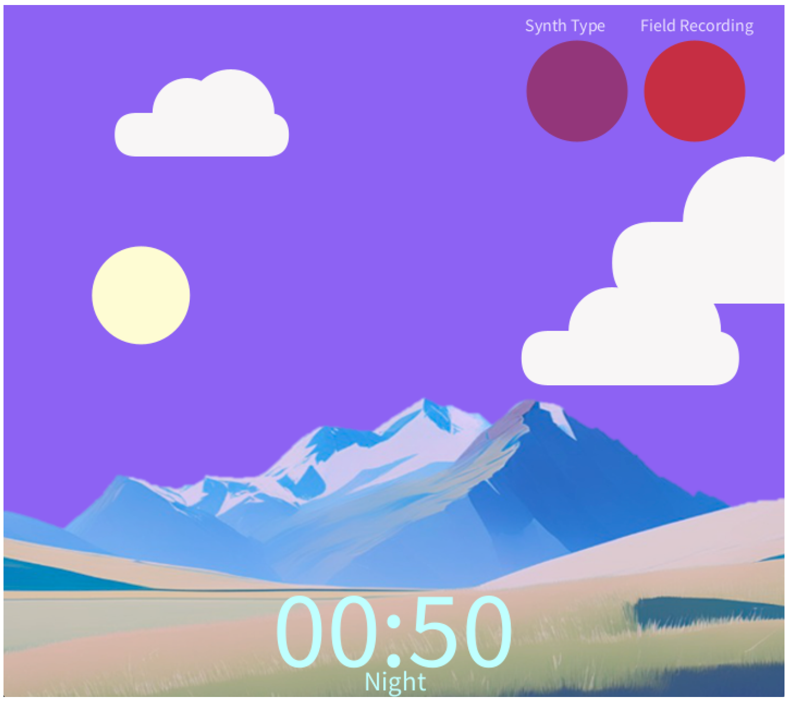


Figure 910: GUI view at night time

The user is shown a digital clock face with the current time and the current 'state' of the day: Morning, Day, Evening and Night. We have programmed a Sun and Moon that are synced to the time of day, and rise and set accordingly.
The position of the Sun and Moon effect the colour of the sky and the transparency of the PNG of the landscape to give a feeling of transitioning to night time.
The time of the clock has been quickened for demonstration purposes. It has a speed of 1 minute lasting 25 milliseconds.

Processing uses OSC to communicate with SuperCollider. It sends three parameters to SuperCollider; the user's selection of synthesizer, the user's selection of turning Field Recordings on or off and the state of day.

Processing receives from SuperCollider information on the ambient light in the environment. This is used to control an animation of cloud coverage. Due to the rapidly varying nature of the ambient light data we needed to average it over time to stop the clouds from oscillating about their position. To do this we use a queue data structure. The cloud position is controlled by the average value of the queue. A queue is an ideal collection for this as it operates on a First In First Out policy. In this way old irrelevant data is removed as new relevant data is received. This solution is very effective and creates a very smooth motion to the cloud animation.