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

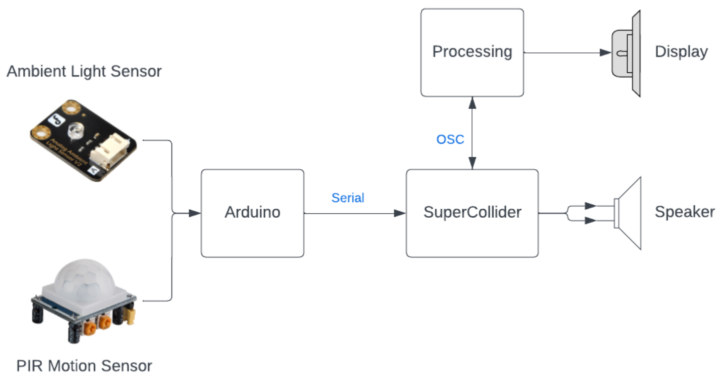
## Overview

M-RAM, Musical Room Ambience Monitor, is designed to provide generative ambient music and sounds to a user's indoor space. Inspired by Brian Eno's sentiment that it is a waste to spend years designing a space and then put no thought into what music is played in it.

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




SuperCollider receives serial messages from Arduino about the data acquired from it's 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 the use with a GUI. Processing sends SuperCollider OSC messages containing information on the time of day and user selected controls, such as selecting different melodies or sounds.

## SuperCollider

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

First we define 5 synths that will be used by the programme: A saw tooth synth, a bass synth, a hihat, a Rhodes style piano and a drone synth. Each of these was created in the conventional manner using SynthDef.

Next we created 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 x.
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 more interesting. Inside an array we have a mixture of midinote values and rests. Attached to this is the stutter function, which makes the array 4 times longer. 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 4$^{th}$, perfect 5$^{th}$ 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. Online 163 we can see that the Quant feature is utilised. This restricts when the Pbindef can start playing to every for beats. This is used on all patterns in the programme to guarantee that they start at the beginning of the next bar.

```
154 (
155 // slow and dynamic
156
157 ~presetSlowDyn = {
158 t.tempo_(60/60);
159 //change the params
160 Pbindef(\p_smooth,
161     \dur, Prand([1/4],inf),
162     \midinote, Pshuf([60,62,64,65,67 ,69,71 ,72,\,\,\,\,\].stutter(4) + [ -3,-5].choose , inf))
163 .play(t, quant:Quant.new(4,0,0));
164
165 Pbindef(\p_bass,
166     \dur, Prand([1/4],inf),
167     \midinote, Pshuf(([60,62,64,65,67 ,69,71 ,72,\,\,\,\]/2).stutter(4) + [ -3,-5,-7 ].choose, inf))
168 .play(t, quant:Quant.new(4,0,0));
169
170 Pbindef(\p_hh).play(t, quant:Quant.new(4,0,0));
171 }
172
173 )
```

Changing the playing presets (depending on the day time) is done as it was in the following code block:

```
~main = Routine.new(
    {

        {
            0.01.wait; //repeat the routine every .1 secs
            // if synths should change
            if(   ~update == 1,
                {
                    if ( (~stopFlag != 1),
                        {~stopMorDayEve.value; ~stopNig.value;},
                        {
                    switch (~synthMode,
                        0, { ~stopNig.value; ~presetSlowRep.value; 'morning'.postln},
                        1, {
                            ~stopNig.value; 'day time'.postln;
                            if ((~ambLight>700),{~presetFastDyn.value; },  {~presetFastRep.value; };);
                                }, // ambient sensor here, low -> repetitive, high -> dynamic},
                        2, { ~stopNig.value; ~presetSlowDyn.value; },
                        3, { ~stopMorDayEve.value;~presetNight.value }
                    );
                        }
                    );
                    ~update = 0;
                    ~update.postln
                },
                // dont update
                {
                    ~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;
                    };
                }
                ;
            )
    }.loop;}
);
```

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 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.

Looping and resetting the ~main routine in 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.

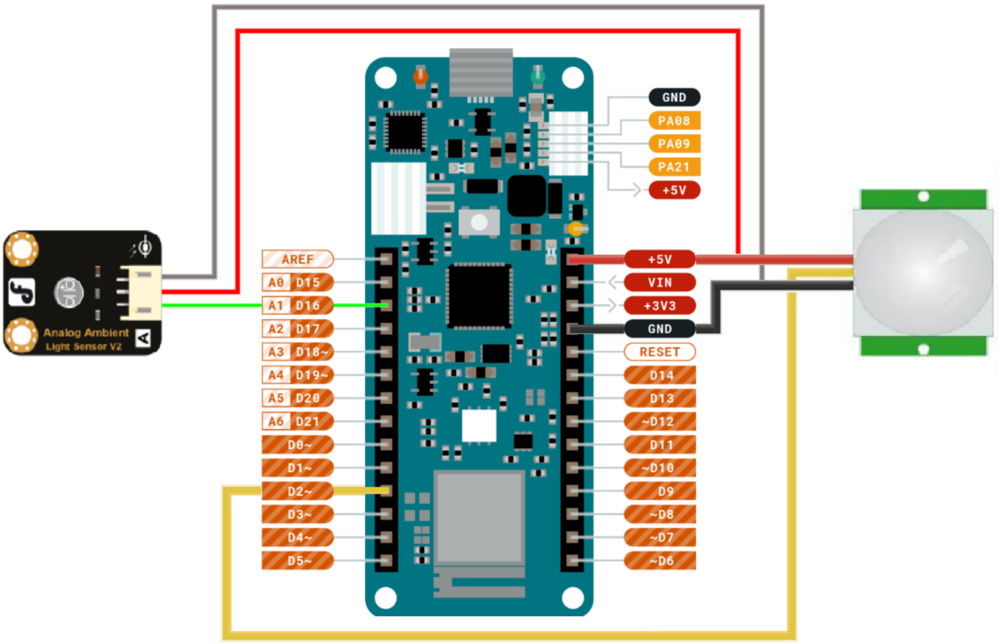
```
~mainTask = Task({
    loop {
        ~stopFlag = ~infRed;
        ~main.reset;
        0.01.wait;
        //'waited 1'.postln;

    }
});
```

| 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.




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 values 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

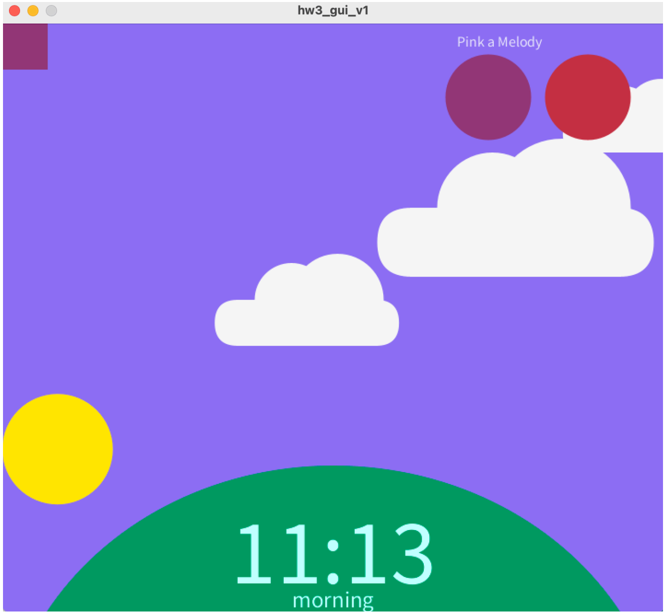
```
int M_sensor = 2;             // the pin that the sensor is attached to
int val = 0;                  // variable to store the sensor status (value)
int counter = 0;
void setup()
{
  pinMode(M_sensor, INPUT);   // initialize sensor as an input
  Serial.begin(9600);         // initialize serial
}
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 countinue 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.




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 also 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 in the background.
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 information of the state of day and any user controlled parameters. It receives from SuperCollider information on the ambient light in the environment as well as other data related to the current behaviour of the system.

The ambient light data is used to control an animation of cloud coverage during the daytime hours.