- Node Package Manager (npm)
  - https://www.npmjs.com/
- ExpressJS
  - https://expressjs.com/
- Socket.io door Automattic
  - https://socket.io/
- Process Manager 2
  - https://www.npmjs.com/package/pm2
- p5.JS door Lauren Lee McCarthy
  - https://editor.p5js.org/
- Hydra Synth door Olivia Jack
  - https://hydra.ojack.xyz/

`.abstraction()`

source code

Timo Hoogland, 2023

Introductie

Abstraction is een interactieve live coding installatie die je mee neemt in de wereld van creative coding, open source systemen en free culture. Een systeem bestaat uit veel lagen (*abstracties*) met verschillende functies die samenwerken. Elke functie kan op zichzelf staan en heeft een input, een transformatie en een output, maar door functies te combineren is het mogelijk een output te genereren die complexer is dan de losse onderdelen. Door gebruik te maken van open source *libraries* en *frameworks* kan je als coder eigen systemen ontwikkelen.

In de installatie kunnen twee bezoekers samen beeld en geluid *coden*. Door aan de knoppen te draaien veranderd de code op het scherm. Als de gebruikers samenwerken kunnen ze het geheel complexer maken.

Het hele systeem van deze installatie is open. Zo draait de installatie op een Raspberry Pi en wordt de electronica aangestuurd met een Arduino, beiden micro-computers ontwikkeld met het doel technologie toegankelijk te maken en te democratiseren. De software is geschreven met Javascript en HTML (de talen van het internet) tot een web applicatie die in Linux OS draait. De installatie is gemaakt met behulp van verschillende frameworks zoals NodeJS, p5.js (een JavaScript creative coding library ontwikkeld door Lauren Lee McCarthy) en Hydra (een live coding visuele synthesizer ontwikkeld door Olivia Jack).

Uitleg van de titel

Abstraction is een proces van het weglaten van niet-essentiële informatie en vervolgens te generaliseren om daarmee de fundamentele structuren zichtbaar te maken. Abstractie komt van het Latijnse abstráhere (weglaten). Een abstractie verbindt alle onderliggende processen in een groep. De term abstractie vindt haar weg door wetenschap, wiskunde en ook als kunststroming


```
    }
    for (i in init){
        io.emit('message', i, init[i]);
    }
});

// require dependency for receiving controller values
const { Server } = require('node-osc');

const oscPort = 9999;

// setup a server to receive OSC messages from controllers
let osc = new Server(oscPort, '0.0.0.0', () => {
    console.log(`OSC server listening at port ${oscPort}`);

    // receive messages from controller and forward to the browser
    osc.on('message', (msg) => {
        io.emit('message', ...msg);
    });
});
```

Gebruikte Frameworks

- Raspberry Pi
  - https://www.raspberrypi.com/
- Raspberry Pi OS (Raspbian)
  - https://www.raspberrypi.com/software/
- Chromium Browser
  - https://www.chromium.org/Home/
- Arduino Uno
  - https://www.arduino.cc/
- HD44780 LCD door Bill Perry
  - https://github.com/duinoWitchery/hd44780
- OSC For Arduino door Yotam Mann en Adrien Freed
  - https://github.com/CNMAT/OSC
- NodeJS
  - https://nodejs.org/en/

```
const socket = require('socket.io');
const shell = require('child_process');
const rpi = require('detect-rpi');

// init the app server and port to listen
const app = express();
app.use(express.static('.'));

const port = 3000;
const server = app.listen(port, () => {
    console.log(`Server is running at http://localhost:${port}`);

    if (rpi()){
        // hide mouse when not moving
        shell.exec(`unclutter -idle 1`);
        // open browser in fullscreen incognito when on rpi
        shell.exec(`chromium-browser --start-fullscreen
                    --start-maximized --incognito
                    http://localhost:${port}`);
    }
});

// connect via socket io
const io = socket(server);

// post socket id to max console
io.sockets.on('connection', function(socket){
    console.log(`Connected ${socket.id}`);

    // initialize all visuals with some values
    init = {
        '/control1/function' : 1024/5,
        '/control2/function' : 1024/5*3,
        '/control1/switch' : 1,
        '/control2/switch' : 1,
        '/control1/value' : Math.random()*1024,
        '/control2/value' : Math.random()*1024,
```


en is een belangrijk onderdeel dat onze intelligentie kenmerkt en ons in staat stelt patronen te zien en verbindingen te leggen.

Over de kunstenaar

Timo Hoogland is een digitale kunstenaar, live coder, muziek technoloog en docent uit Apeldoorn. Hij maakt live experimentele elektronische muziek met computer code en ontwikkeld daarnaast generatieve audiovisuele composities en installaties. In zijn werk haalt Timo inspiratie uit wiskundige concepten, geometrie, natuurlijke fenomenen en chaotische of deterministische systemen en experimenteert met deze algoritmes om beeld en klanken te creëeren in het digitale domein.

- timohoogland.com
- @tmhglnd op instagram
- @tmhglnd op youtube

Licentie

Bewerken — te remixen, te veranderen en afgeleide werken te maken voor alle doeleinden, inclusief commerciële doeleinden.

Deze licentie is goedgekeurd voor Free Cultural Works.

Onder de volgende voorwaarden:

Naamsvermelding — De gebruiker dient de maker van het werk te vermelden, een link naar de licentie te plaatsen en aan te geven of het werk veranderd is. Je mag dat op redelijke wijze doen, maar niet zodanig dat de indruk gewekt wordt dat de licentiegever instemt met je werk of je gebruik van het werk.

GelijkDelen — Als je het werk hebt geremixt, veranderd, of op het werk hebt voortgebouwd, moet je het veranderde materiaal verspreiden onder dezelfde licentie als het originele werk.


```
/* style for the code sections */
div {
    display: table;
    column-count: 2;
    width: 100%;
    height: 50%;
}

p {
    padding: 5%;
    width: 50%;
    height: 100%;
    border: 0;
    display: table-cell;
}

/* highlight the code and blend with background */
mark {
    background-color: magenta;
    color: cyan;
    mix-blend-mode: difference;
}

.canvas {
    z-index: -1000;
    position: fixed;
}
```

Server

De server start de applicatie en geeft de waardes van de controllers door aan de web pagina.

filename: `server.js`

```
// require dependencies for server
const express = require('express');
```

```
    // delete all the sounds when not used anymore
    this.dispose = () => {
        this.loop.stop();
        let nodes = [this.noise, this.filter, this.adsr];
        for (let i=0; i<nodes.length; i++){
            nodes[i].disconnect();
            nodes[i].dispose();
        }
    }

    // if true then pan to both speakers, else split
    this.connect = (isConnect) => {
        if (isConnect){
            this.noise.pan(0);
        } else {
            this.noise.pan(this.output);
        }
    }
}
```

filename: style.css

```
@import url('./fonts/UbuntuMono-Regular.ttf');

/* The style of the page body */
html,
body {
    font-family: 'Ubuntu Mono', monospace;
    font-size: 1.7vw;
    line-height: 1.3;
    width: 100%;
    height: 100%;
    margin: 0px;
    overflow: hidden;
}
```


Source Code

Arduino controller

filename: controller.ino

```
/*
  Written by Timo Hoogland

  The installation uses a small controller to control Audio
  and Visuals
  Uses LCD Display, 2 Potentiometers, a Button (NO) and LED

  Displays selected function on screen
  Displays the potmeter value as floatingpoint value

  Using OSC example code from Adrien Freed - Library CNMAT/OSC
  Using LCD example code from Bill Perry - Library hd44780
*/

// include the libraries needed for Ethernet, OSC messaging
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <SPI.h>
#include <OSCMessage.h>

// declare EthernetUDP object
EthernetUDP udp;

// include the libraries needed for LCD Display with backpack
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// declare LCD object
hd44780_I2Cexp lcd;

// the device's ID
```

```
char device[] = "/control1";

// Ethernet shield1 MAC Address A8:61:0A:AE:A8:6D
byte mac[] = { 0xA8, 0x61, 0x0A, 0xAE, 0xA8, 0x6D };

// Ethernet shield2 MAC Address A8:61:0A:AE:95:10
// byte mac[] = { 0xA8, 0x61, 0x0A, 0xAE, 0x95, 0x10 };

// the Arduinos IP address
IPAddress ip(128, 32, 122, 252);
// destination IP is all addresses in network
IPAddress outIp(255, 255, 255, 255);
// IPAddress outIp(169, 254, 50, 206);
// receiving port
const unsigned int outPort = 9999;

// history of values for filter
int _f = 0;
int _v = 0;
int _s = HIGH;
int thresh = 1;
bool changed = true;

// function names for display
char *functionNames[] = {"squiggle", "mosaic",
                         "smear", "glass", "paint"};

// The startup program running once
//
void setup() {
  // input for button
  pinMode(2, INPUT_PULLUP);

  // Start ethernet connection for mac address and ip
  Ethernet.begin(mac, ip);
  // Open receiving port number
  udp.begin(8888);
```



```
this.adsr = new p5.Envelope(0.001, 1, 0.05, 0);
this.noise.amp(this.adsr);
this.noise.start();
this.osc.amp(this.adsr);
this.osc.start();

this.cutoffs = [ 0, 360, 240, 3200 ];

// the note sets the frequency and triggers the sound
this.trigger = (time) => {
    let i = this.loop.iterations % this.rhythm.length;
    let j = this.loop.iterations % this.lengths.length;

    this.adsr.setADSR(0.001, this.lengths[j], 0, 0);
    if (this.cutoffs[i] < 1){
        this.adsr.triggerAttack(this.osc, time);
    } else {
        this.filter.freq(this.cutoffs[i], time);
        this.adsr.triggerAttack(this.noise, time);
    }
}

// this is the main loop that keeps triggering notes
this.loop = new p5.SoundLoop(this.trigger, 0.191);
this.loop.start();

this.rhythm = [ 100 ];
this.lengths = [ 0.1 ]

// change the following parameters with the knob
this.value = (v) => {
    let s = cos(v * pi * 2) * 0.5 + 0.5;
    this.loop.interval = s * -0.05 + 0.2;

    this.rhythm = RN.choose(cos(v*pi*3)*5+3, this.cutoffs);
    this.lengths = GN.spreadF(v*7+1, 0.001, 0.2);
}
```

```
        for (let i=0; i<nodes.length; i++){
            nodes[i].disconnect();
            nodes[i].dispose();
        }
    }

    // if true then pan to both speakers, else split
    this.connect = (isConnect) => {
        if (isConnect){
            this.osc.pan(0);
        } else {
            this.osc.pan(this.output);
        }
    }
}

// Sound function for the Paint visual
// A triangle and pink noise oscillator are used as percussive
// sounds by triggering short portions and using a filter
function Paint(out){
    // the triangle wave oscillator
    this.osc = new p5.Oscillator(73, 'triangle');
    this.output = out;
    this.osc.pan(out);
    // the pink noise generator
    this.noise = new p5.Noise('pink');
    this.noise.pan(out);
    this.noise.disconnect();

    // the filter processes the distorted noise
    this.filter = new p5.Filter('lowpass');
    this.filter.freq(230);
    this.filter.res(10);
    this.filter.gain(3);
    this.filter.process(this.noise);

    // the envelope triggers short portions of the sounds
```



```
  // Initialize LCD Display with columns and rows
  int status = lcd.begin(20, 4);
  // Status is non-zero if unsuccesful
  if (status){
    // Blink error code using onboard LED if possible
    hd44780::fatalError(status);
  }

  // Otherwise initialization succesful, load start screen text
  startScreen();
}

// The main program running continuously in a loop
//
void loop() {
  // read knobs from analog inputs and emit osc message
  int f = analogRead(A0);
  int v = analogRead(A1);
  int s = digitalRead(2);

  // threshold to change value to remove noise from potentiometer
  // only send message when value changed
  if (abs(f - _f) > thresh){
    oscSend("/control1/function", f);
    _f = f;
    changed = true;
  }
  if (abs(v - _v) > thresh){
    oscSend("/control1/value", v);
    _v = v;
    changed = true;
  }
  if (abs(s - _s) > 0){
    oscSend("/control1/switch", s);
    _s = s;
    changed = true;
```

```
  }

  // print all the text on the display and change the LED color
  // but only if any of the values changed
  if (changed){
    displayTextAndLED(_f, _v, _s);
    changed = false;
  }

  // pause the program to safe cpu load
  delay(50);
}

// a function that display text and LED color
void displayTextAndLED(int f, int v, int s){
  // control LED RGB light brightness (0-255 on PWM output)
  analogWrite(3, v/4); //red
  analogWrite(5, f/4); //blue
  analogWrite(6, (1-s)*255); //gree

  // clear the display
  lcd.clear();
  // print the function name on first line
  lcd.setCursor(0, 1);
  lcd.print(" .");
  lcd.print(functionNames[int(float(f)/1024*5)]);
  lcd.print("(");

  // generate a char array for number displaying
  char displayNumber[10];
  // convert float value to string with fixed digits
  dtostrf(float(v)/1023, 8, 6, displayNumber);

  // print the variable number from the knob as float 0-1
  lcd.setCursor(0, 2);
  lcd.print("     ");
  lcd.print(displayNumber);
```



```
// the note sets the frequency and triggers the sound
this.note = (time) => {
    let i = this.loop.iterations % this.phrase.length;
    let note = this.phrase[i];

    let f = p.midiToFreq(note + 48);
    this.osc.freq(f, 0, time);
    this.mod.freq(f * (this.detune + 0.5), 0.01, time);
    this.adsr.triggerAttack(this.osc, time);
}

// this is the main loop that keeps triggering notes
this.loop = new p5.SoundLoop(this.note, 1);
this.loop.start();

// the phrase is a list of a melody
this.phrase = [];

// change the following parameters with the knob
this.value = (v) => {
    this.detune = v * 2;

    let s = sin(v * pi * 9) * 0.5 + 0.5;
    let cf = s * 30 + 60;
    this.phrase = TL.toScale(GN.saw(16, s*5, 36, 12));

    this.loop.interval = (1 - s) * 0.5 + 0.1;
    this.delay.feedback(s * 0.3 + 0.3);
    this.filter.freq(p.midiToFreq(cf));
}

// delete all the sounds when not used anymore
this.dispose = () => {
    this.loop.stop();
    let nodes = [this.osc, this.filter, this.adsr,
                 this.mod, this.delay];
```

```
    }
}

// Sound function for the Glass visual
// Two triangle oscillators that are combined with
// amplitude modulation, delay and a melodic phrase
function Glass(out){
    // the main triangle wave oscillator
    this.osc = new p5.Oscillator(1, 'triangle');
    this.output = out;
    this.osc.pan(out);
    this.osc.disconnect();
    this.osc.start();

    // the modulation triangle oscillator
    this.mod = new p5.Oscillator(0.5, 'triangle');
    this.mod.disconnect();
    this.mod.start();
    this.mod.scale(-1,1,0,1);

    // the modulater changes the volume of the oscillator
    this.osc.amp(this.mod);

    // the filter processes the main oscillator
    this.filter = new p5.Filter();
    this.filter.process(this.osc);
    this.filter.disconnect();

    // the envelope triggers the sound
    this.adsr = new p5.Envelope(0.001, 1, 0.4, 0);

    // the delay processes the filter sound
    this.delay = new p5.Delay();
    this.delay.process(this.filter, 0.3123, 0.5)
    this.delay.drywet(0.5);

    this.detune = 0;
```



```
  // if the code is combined show text
  if (!s){
    lcd.print(" )");
    lcd.setCursor(0, 3);
    lcd.print(" .combine();");
  } else {
    lcd.print(" );");
  }
}

// A function that sends an OSC-message to specified IP and Port
// with variable address and value
void oscSend(char addr[], int val){
  // the messages wants an OSC address as first argument
  OSCMessage msg(addr);
  // add the value to the message
  msg.add(val);

  udp.beginPacket(outIp, outPort);
  // send the bytes to the SLIP stream
  msg.send(udp);
  // mark the end of the OSC Packet
  udp.endPacket();
  // free space occupied by message
  msg.empty();
}

// A function that displays the startscreen
void startScreen(){
  lcd.setCursor(0, 0);
  lcd.print("ABSTRACTION");
  delay(500);

  lcd.setCursor(0, 1);
  lcd.print("  by Timo");
  delay(500);
```

```
  lcd.setCursor(0, 3);
  lcd.print("         starting...");
  delay(1000);
}
```



```
this.lfo.start();

// the main pink noise sound
this.noise = new p5.Noise('pink');
this.output = out;
this.noise.pan(out);
this.noise.disconnect();
this.noise.amp(this.lfo);
this.noise.start();

// the filter processes the pink noise
this.filter = new p5.Filter();
this.filter.process(this.noise);

// change the following parameters with the knob
this.value = (v) => {
    let f = cos(v * pi * 8) * -0.5 + 0.5;
    this.lfo.freq(f * 10 + 0.5, 0.05);
    this.filter.freq(p.midiToFreq(f * 55 + 70), 0.05);
}

// delete all the sounds when not used anymore
this.dispose = () => {
    let nodes = [this.noise, this.lfo, this.filter];
    for (let i=0; i<nodes.length; i++){
        nodes[i].disconnect();
        nodes[i].dispose();
    }
}

// if true then pan to both speakers, else split
this.connect = (isConnect) => {
    if (isConnect){
        this.noise.pan(0);
    } else {
        this.noise.pan(this.output);
    }
```

```
    // change the following parameters with the knob
    this.value = (v) => {
        this.phrase = TL.toMidi(TL.toScale(
                    GN.cosine(8, v * 5 + 0.5, 25, 0)), 3);
        this.loop.interval = (1 - v) * 0.05 + 0.231;
        this.adsr.setADSR(0.001, (cos(v * pi * 4) * 0.5 + 0.5)
                                    * 0.4 + 0.05, 0, 0);
    }

    // delete all the sounds when not used anymore
    this.dispose = () => {
        this.loop.stop();
        let nodes = [this.osc, this.adsr, this.delay];
        for (let i=0; i<nodes.length; i++){
            nodes[i].disconnect();
            nodes[i].dispose();
        }
    }

    // if true then pan to both speakers, else split
    this.connect = (isConnect) => {
        if (isConnect){
            this.osc.pan(0);
        } else {
            this.osc.pan(this.output);
        }
    }
}

// Sound function for the Smear visual
// Filtered pink noise modulated with
// a Low Frequency Oscillator
function Smear(out){
    // the modulation Low Frequency Oscillator
    this.lfo = new p5.Oscillator(1, 'sine');
    this.lfo.scale(-1,1,0,0.9);
    this.lfo.disconnect();
```


Web App

De web app is een website waar de installatie in draait. Deze bestaat uit 3 delen:

1. Een html pagina die de inhoud van de website beschrijft.
2. Een css bestand, dit is een stylesheet die kleuren en lettertypes beschrijft.
3. Een script, dit is de code bestand met alle functionaliteiten van de installatie.

filename: `index.html`

```
<!DOCTYPE html>
<head>
    <title>ABSTRACTION | by Timo Hoogland | 2023</title>

    <!-- include hydra-synth for visuals -->
    <script src="./node_modules/hydra-synth/dist
                /hydra-synth.js"></script>
    <!-- socket.io for communication between browser and device -->
    <script src="./node_modules/socket.io/client-dist
                /socket.io.min.js"></script>
        <!-- include the p5.js library for sound processing -->
    <script src="./node_modules/p5/lib
                /p5.js"></script>
    <script src="./node_modules/p5/lib/addons
                /p5.sound.js"></script>
    <!-- include total serialism library for algorithmic composition
    <script src="./node_modules/total-serialism
                /build/ts.bundle.js"></script>

    <!-- add the stylesheet for the layout -->
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <!-- the visual canvas -->
    <canvas id="hydra-canvas" class="canvas"></canvas>
```

```
    <!-- the 3 code sections overlaying the canvas -->
    <div>
        <p id="o0"></p>
        <p id="o1"></p>
    </div>
    <div>
        <p id="o2"></p>
    </div>
    <!-- the script with all the core of the installation -->
    <script src="index.js"></script>
</body>
</html>
```

filename: `index.js`

```
const pixels = { width: 480, height: 270 };

// aliases for JS functions
let sin = Math.sin;
let cos = Math.cos;
let int = Math.floor;
let pi = Math.PI;
let rand = Math.random;

// canvas setup for the hydra visuals
let canvas = document.getElementById('hydra-canvas');
canvas.width = window.innerWidth;
canvas.height = window.innerHeight;
canvas.style.width = '100%';
canvas.style.height = '100%';
canvas.style.imageRendering = 'pixelated';

// create a new hydra instance
var hydra = new Hydra({
    canvas: canvas,
    precision: 'lowp',
    detectAudio: false,
```



```
    }
}

// Sound function for the Mosaic visual
// An arpeggiating oscillator with delay effect playing a
// short melody with changing melodic phrase
function Mosaic(out){
    // the base triangle wave oscillator
    this.osc = new p5.Oscillator('triangle');
    this.output = out;
    this.osc.pan(out);
    this.osc.disconnect();
    this.osc.start();

    // the envelope, that makes the sound short
    this.adsr = new p5.Envelope(0.001, 1, 0, 0.2);

    // the delay processes the main oscillator
    this.delay = new p5.Delay();
    this.delay.process(this.osc, 0.316, 0.7, 2500);
    this.delay.drywet(0.4);

    // the phrase is a list of a melody
    this.phrase = [0];

    // the note sets the frequency and triggers the sound
    this.note = (time) => {
        let i = this.loop.iterations % this.phrase.length;
        let note = this.phrase[i];
        this.osc.freq(p.midiToFreq(note), 0, time);
        this.adsr.triggerAttack(this.osc, time);
    }

    // this is the main loop that keeps triggering notes
    this.loop = new p5.SoundLoop(this.note, 0.231);
    this.loop.start();
```

```
// the filter processes the squarewave and is modulated
this.filter = new p5.Filter();
this.filter.process(this.osc);
this.filter.disconnect();
this.filter.freq(this.mod.scale(-1, 1, 250, 1000));
this.filter.res(20);

// the delay processes the total sound
this.delay = new p5.Delay();
this.delay.process(this.filter, 0.233, 0.6, 2100);
this.delay.drywet(0.5);

// change the following parameters with the knob
this.value = (v) => {
    let s = cos(v * 4 * pi) * -0.5 + 0.5;
    let f = p.midiToFreq(v * 2 + 38);
    this.osc.freq(f, 0.05);
    this.mod.freq(s * 2 + 0.01, 0.05);
    this.filter.res(s * 15 + 5, 0.05);
}

// delete all the sounds when not used anymore
this.dispose = () => {
    let nodes = [this.osc, this.mod, this.filter, this.delay];
    for (let i=0; i<nodes.length; i++){
        nodes[i].disconnect();
        nodes[i].dispose();
    }
}

// if true then pan to both speakers, else split
this.connect = (isConnect) => {
    if (isConnect){
        this.osc.pan(0);
    } else {
        this.osc.pan(this.output);
    }
```



```
});

// initialize an instance of p5.js for the sounds
let p = new p5((p) => {
    p.setup = () => {
        p.noCanvas();
        p.userStartAudio();
    }
});

// set resolution for the visuals
setResolution(pixels.width, pixels.height);

// initalize solid black screens
solid().out(o0);
solid().out(o1);
render(o3);

// create a mask (split screen in black/white on half width)
osc(Math.PI*2,0).thresh(0.5, 0).out(o2);

// storage for incoming parameters from controller
let c1 = { value: 0, function: 0, switch: 0, sound: null };
let c2 = { value: 0, function: 0, switch: 0, sound: null };

// A timer that lowers the sound when the installation is not used
let timer;
function resetTimer(){
    // reset timer that turns of sound
    clearTimeout(timer);
    timer = setTimeout(() => p.outputVolume(0.125, 1), 15000);
    // turn on volume if controller is touched
    p.outputVolume(0.7, 0.5);
}

// receive messages from the controllers
let socket = io();
```

```
// The socket receives all the messages from the controllers
socket.on('message', (...msg) => {
    // reset the timer when the controller is used
    resetTimer();

    // if both controllers' switch is pressed merge the visuals
    if (msg[0].match(/switch/)){
        if (msg[0].match(/control1/)){
            c1.switch = !msg[1];
        } else if (msg[0].match(/control2/)){
            c2.switch = !msg[1];
        }

        if (c1.switch && c2.switch){
            merge();
        } else {
            split();
        }
    }
    // route incoming messages and change values for code
    if (msg[0].match(/control1/)){
        // if the message comes from controller 1
        if (msg[0].match(/value/)){
            c1.value = msg[1] / 1023;
            if (c1.sound){
                c1.sound.value(c1.value);
            }
        }
        else if (msg[0].match(/function/)){
            let f = Math.floor(msg[1] / 1024 * 5);
            // only change something if the function
            // actually changes (not just the knob position)
            if (f !== c1.function){
                c1.function = f;
                // redraw the visuals with new code
                visual(c1, 'o0');
```



```
            if (ctl.sound){
                // delete the previous sound
                ctl.sound.dispose();
            }
            // generate the sound
            ctl.sound = soundSnippets[snippets[ctl.function]](out);
            // set the sound value based on the controller value
            ctl.sound.value(ctl.value);
        }

        // Libraries that help for algorithmic composition of music
        TL = TotalSerialism.Translate;
        TF = TotalSerialism.Transform;
        GN = TotalSerialism.Generative;
        RN = TotalSerialism.Stochastic;

        // the root and the scale for the melodic parts
        TL.setRoot('D');
        TL.setScale('minor_pentatonic');

        // sound function for the squiggle visual
        // A square oscillator with a modulated bandpass filter
        // and a little delay for the sense of room
        function Squiggle(out){
            // the base squarewave oscillator
            this.osc = new p5.Oscillator(1, 'square');
            this.output = out;
            this.osc.pan(this.output);
            this.osc.disconnect();
            this.osc.start();

            // a Low Frequency Oscillator that modulates the filter
            this.mod = new p5.Oscillator(1, 'sine');
            this.mod.disconnect();
            this.mod.start();
```

```
// Generate the new visual code
function visual(ctl, out){
    // get the code from the list of options
    let snippets = Object.keys(codeSnippets);
    // apply the parameter value
    let code = codeSnippets[snippets[ctl.function]](out,
                                                    'ctl.value');
    // generate the visuals
    eval(code);
    // display the code also as text on the screen
    displayCode(code, out);
}

// display the code as text on the screen
function displayCode(text, element){
    let paragraph = document.getElementById(element);
    paragraph.innerHTML = '';
    let mark = document.createElement('mark');
    mark.innerText = text;
    paragraph.appendChild(mark);
}

// All the different sound snippets for the installation
// Calls a class that is described below
// (o) is the output to the speaker
let soundSnippets = {
    'squiggle' : (o) => { return new Squiggle(o); },
    'mosaic' : (o) => { return new Mosaic(o); },
    'smear' : (o) => { return new Smear(o); },
    'glass' : (o) => { return new Glass(o); },
    'paint' : (o) => { return new Paint(o); }
}

// Generate the new sound code
function sound(ctl, out){
    // get the code from the list of options
    let snippets = Object.keys(soundSnippets);
```



```
                // remake the sound with new code
                sound(c1, -1);
            }
        }
    }
    else if (msg[0].match(/control2/)){
        // if the message comes from controller 2
        if (msg[0].match(/value/)){
            c2.value = msg[1]/1023;
            if (c2.sound){
                c2.sound.value(c2.value);
            }
        }
        else if (msg[0].match(/function/)){
            let f = Math.floor(msg[1]/1024*5);
            // only change something if the function
            // actually changes (not just the knob position)
            if (f !== c2.function){
                c2.function = f;
                // redraw the visuals with new code
                visual(c2, 'o1');
                // remake the sound with new code
                sound(c2, 1);
            }
        }
        else if (msg[0].match(/switch/)){
            c2.switch = !msg[1];
        }
    }
});

// Split the visuals and sound into 2 separate screens
function split(){
    solid().add(src(o0).mask(src(o2)).add(
        src(o1).scale(-1).mask(src(o2).invert(1))
    )).out(o3);
```

```
    // display the code in the editor
    displayCode(
        `src(o0).mask(src(o2)).\nadd(src(o1)
        .mask(src(o2).invert(1)))\n.out(o3)`, 'o2');

    // split the sound so one is left and one is right
    c1.sound.connect(false);
    c2.sound.connect(false);
}

// Merge the visuals and sound by modulating together with random
// modulation option and amount every time started
function merge(){
    let options = ['modulate', 'modulateKaleid',
                'modulateScrollX', 'modulateRotate',
                'modulateScale']
    let random = int(rand() * options.length);

    let modulation = options[random];
    let modAmount = rand()*4+0.1;
    let mergeCode = `src(o0).\n${modulation}(src(o1)
                    .scale(-1), ${modAmount.toFixed(3)})`;

    // display the code in the editor
    displayCode(`${mergeCode}\n.out(o3)`, 'o2');

    // evaluate the code and display the merged visuals
    mergeCode = `solid().add(${mergeCode}).out(o3)`;
    eval(mergeCode);

    // merge the sounds by playing in both speakers
    c1.sound.connect(true);
    c2.sound.connect(true);
}

// All the different visual code snippets for the installation
// stored in an object by name and followed by the code
```



```
// ${v} is the variable from the knob
let codeSnippets = {
    'squiggle' : (o, v) =>
        `gradient(0).pixelate(6,6)
        .hue(() => 1-(${v} * 0.5 + 0.4))
        .mask(shape(8).scale( () => ${v} * 10 + 3, 0.56))
        .rotate(() => ${v} * pi, 1)
        .modulate(noise(1.5, 0.2),
            () => cos(${v} * pi * 4) * 4 + 5.5)
        .out(${o})`,
    'mosaic' : (o, v) =>
        `osc(4,0.1,() => ${v} * 2 + 1.7).kaleid(3)
        .colorama(() => cos(${v} * pi * 6) * 0.1 + 0.1)
        .modulatePixelate(voronoi(8, 0.5, 0),
            () => sin(${v} * pi * 4) * 10 + 11)
        .out(${o})`,
    'smear' : (o, v) =>
        `src(${o})
        .modulate(src(${o}).pixelate(6,6), 0.005)
        .scale(() => sin(${v} * pi * 4) * 0.02 + 1).hue(0.001)
        .diff(osc(2, -0.1, () => ${v} * 2 + 3)
            .mask(shape(2,0.02,0).scrollY(0.5)))
        .out(${o})`,
    'glass' : (o, v) =>
        `osc(2, 0.4, 0)
        .modulateKaleid(shape(() => int(${v} * 4) + 2, 0.2,
            () => sin(${v} * pi * 9) * 0.5 + 0.5).repeat(3), 6)
        .colorama(() => cos(${v} * pi * 2) * 0.01 + 7)
        .out(${o})`,
    'paint' : (o, v) =>
        `noise(0.4, 0.05)
        .colorama(() => ${v} * 0.01 + 6)
        .hue(() => ${v} * -0.3)
        .modulate(noise(() => cos(${v} * pi * 2) * 2 + 4))
        .out(${o})`,
}
```