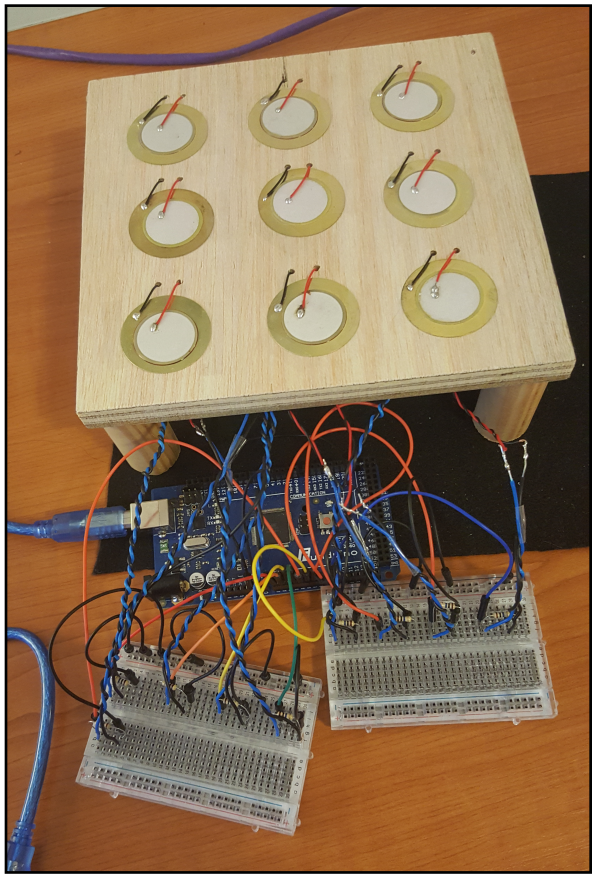
S1875574 John Kim
S1864432 Nils Rublein

PROGRAMMING AND PHYSICAL COMPUTING FINAL PROJECT M2

DJ Launchpad and 3D effects

This is a DJ Launchpad made using piezo-electric sensors, together with an Arduino Mega board. We are also using 3D visuals to react to our backing sound file.



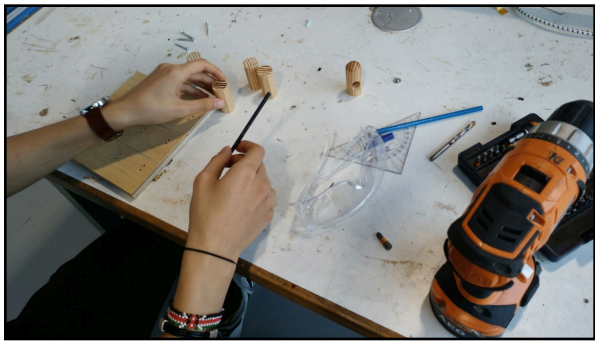
Electronics and hardware used:

Arduino Mega
Piezoelectric sensors (9)
10M ohm resistors (9)
Plywood board

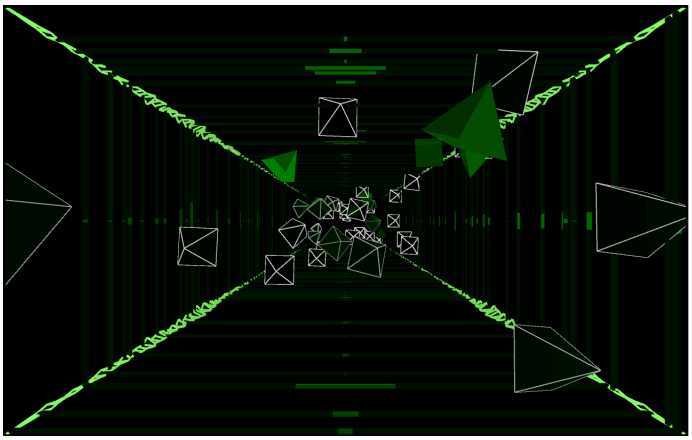
Software used:

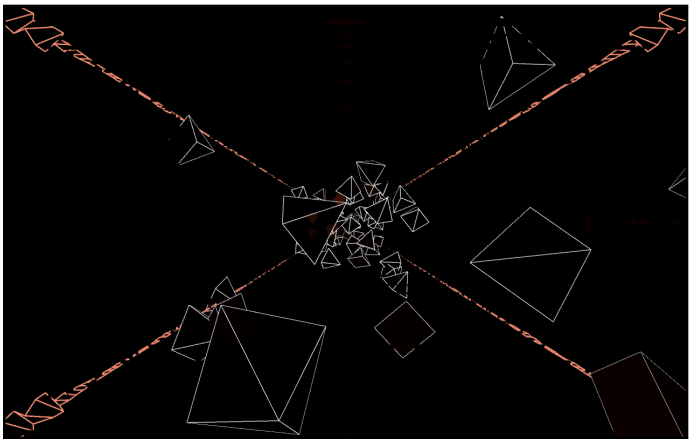
Arduino
Processing
And Firmata Library

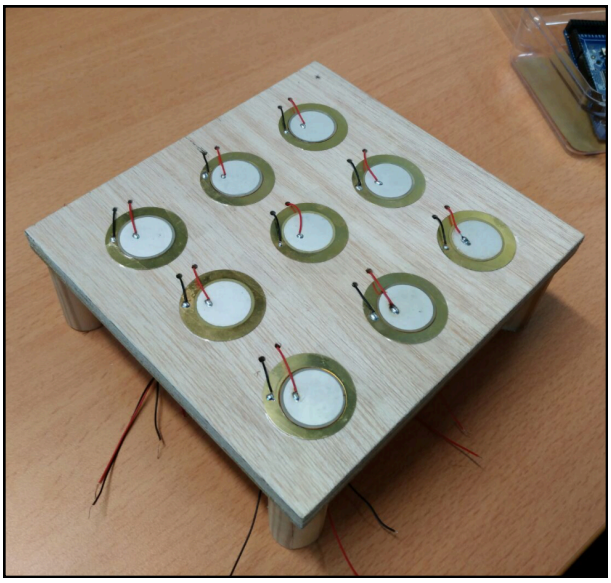
Process of making:
Due to our similar interests in music, we both had the idea of using sensors to make a certain musical instrument. After long hours or experimentation and inspiration from other projects, we decided to make a DJ launchpad using piezo-electric sensors.



These sensors are attached to the analog ports of the Arduino, and using these values we use Processing to perform certain actions; to play sounds in the sound library of processing. The wires of the sensors were too short after installation, thus they all had to be soldered to an extension wire, which are then connected to the breadboard each having a 10M ohm resistor.





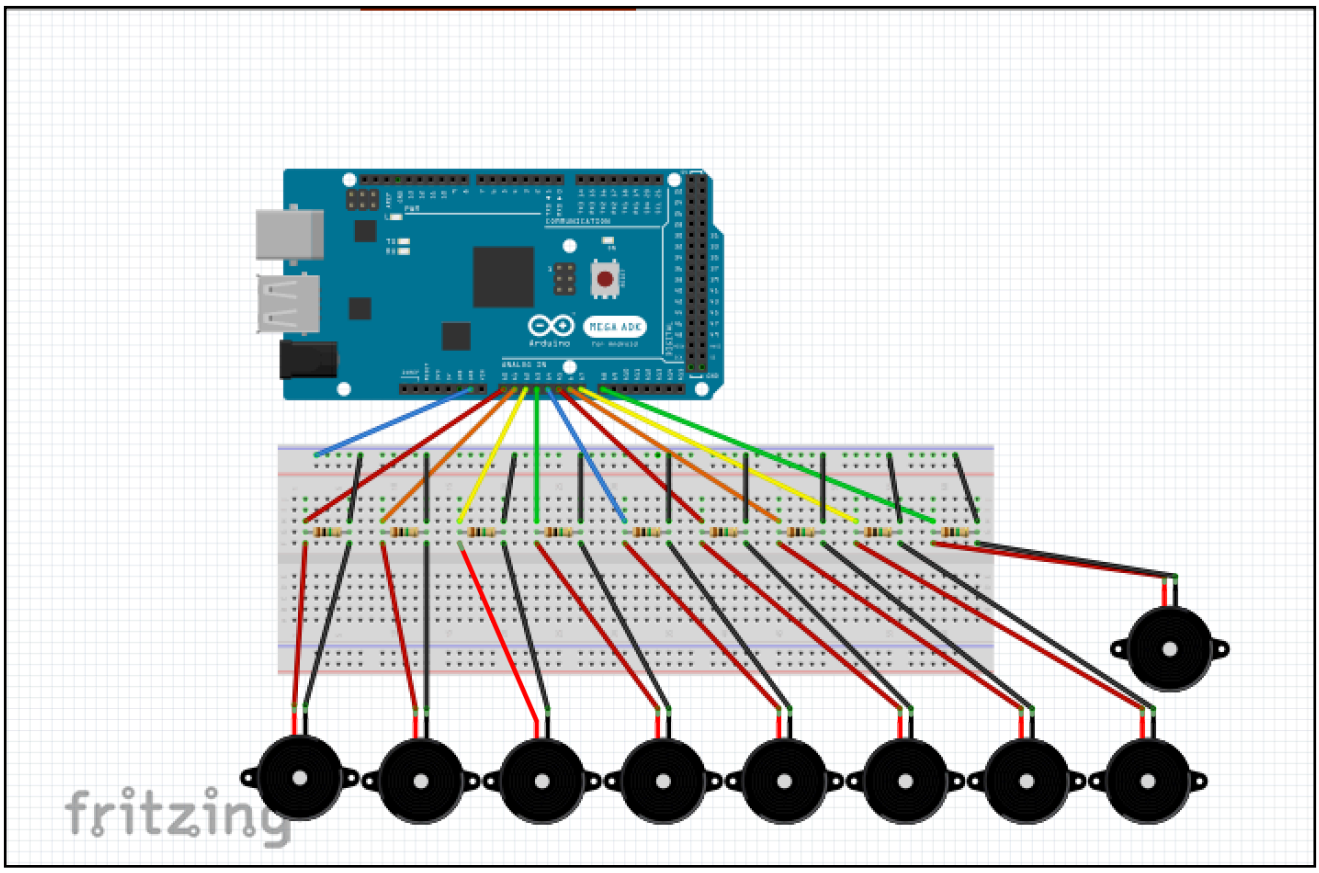


The piezo-electric sensors assembled on the wooden board before soldering.

On the left are the 3D animations that play when the processing sketch is run.

For the 3D animations, we looked for many sources of inspiration and got a lot of help from this source url https://github.com/samuellapointe/ProcessingCubes by samuellapointe. In order for us to make sense out of it, we had to translate the example code from French to English and modified it to our needs.

FRITZING



PROCESSING CODE

```
/*End of module 2 Assignment for PPAC
 Launchpad built with Piezoelectric sensors connected to an Arduino Mega and
 3D Animations that change visuals according to the music backing track.
 This requires the Processing Sound library, Minim library, Serial and Arduino Library.
 Upload the Standard Firmata page onto the arduino and run this sketch.
 Programmed on Processing version 3.2.3
 by Nils Rublein and John Kim.
 29-01-2017
 */

//import libraries for sound and for communication between Arduino and Processing
import processing.sound.*;
import processing.serial.*;
import cc.arduino.*;
import org.firmata.*;

//creating an arduino object
Arduino arduino;

// sensor objects
TriggerSound soundKit;
TriggerSound soundKit1;
TriggerSound soundKit2;
TriggerSound soundKit3;
TriggerSound soundKit4;
TriggerSound soundKit5;
TriggerSound soundKit6;
TriggerSound soundKit7;
TriggerSound soundKit8;


int threshhold = 400;
int threshhold3 = 1000;     //some piezo sensors aren't as sensitive as the others and need
therefore special threshholds
int threshhold8 = 20;

String soundSample0;       // 2 samples for each sensor-button, *sensornumber*+1 is the second
sample
String soundSample01;      // with the first tap on the sensor you trigger the first sample, if you
tap it again,
String soundSample1;       // it plays the second sample and resets afterwards.
String soundSample11;
String soundSample2;
String soundSample21;
String soundSample3;
String soundSample31;
```

```
String soundSample4;
String soundSample41;
String soundSample5;
String soundSample51;
String soundSample6;
String soundSample61;
String soundSample7;
String soundSample71;
String soundSample8;
String soundSample81;

////////////////////////////////////////////////////////////////////////////////
//The following variables are used for the 3D visuals
////////////////////////////////////////////////////////////////////////////////


//importing minim library and analysis
import ddf.minim.*;
import ddf.minim.analysis.*;

//creating minim and audioplayer object
Minim minim;
AudioPlayer song;

//using the full path of the library due to FFT fft being ambiguous
ddf.minim.analysis.FFT fft;

/*FFT stands for Fast Fourier Transform, which is a method of analyzing audio that allows you to visualize
 the frequency content of a signal. e.g visualizations in music players and car stereos*/

// Variables that define the "areas" of the spectrum
// For example, for bass, one takes only the first 3% of the total spectrum
float specLow = 0.03; // 3%
float specMid = 0.125;  // 12.5%
float specHi = 0.20;   // 20%

// This leaves 65% of the possible spectrum left unused.
// These values are usually too high for the human ear anyway.

// Scoring values for each zone
float scoreLow = 0;
float scoreMid = 0;
float scoreHi = 0;

// Previous values, to soften the reduction
float oldScoreLow = scoreLow;
float oldScoreMid = scoreMid;
float oldScoreHi = scoreHi;
```

```
// Softening value
float scoreDecreaseRate = 25;

// Pyramids that appear in space
int nbPyramids;
Pyramid[] pyramids;

// Lines that appear on the sides
int nbWalls = 500;
Wall[] walls;

////////////////////////////////////////////////////////////////////////////////
//end of the global variables for the 3D visuals
////////////////////////////////////////////////////////////////////////////////



void setup() {
  fullScreen(P3D);  //for 3d perspective

  // Check available serial ports and print in the serial monitor
  println("Available serial ports:");
  for (int i = 0; i<Serial.list ().length; i++) {
    print("[" + i + "] ");
    println(Serial.list()[i]);
  }

  //write number of the port inside the array
  arduino = new Arduino(this, Arduino.list()[1], 57600);

  // Alternatively, use the name of the serial port corresponding to your
  // Arduino (in double-quotes), as in the following line.
  //arduino = new Arduino(this, "/dev/tty.usbmodem621", 57600);

  loadSampleKits();  //function to load the sample sounds

  //load samples for each sensor
  //in the format (sensor number, sample 1, sample 1.1, this, threshhold value for each sensor).
  //the sample 1 and 1.1 (or 2&2.1/ 3&3.1) is to switch between 2 sound samples on one piezeo
sensor/button.
  //we map 2 sound samples to one sensor because there are not enough sensors to play a
wide variety of sounds e.g. daft punk)
  soundKit = new TriggerSound(0, soundSample0, soundSample01, this, threshhold);
  soundKit1 = new TriggerSound(1, soundSample1, soundSample11, this, threshhold);
  soundKit2 = new TriggerSound(2, soundSample2, soundSample21, this, threshhold);
  soundKit3 = new TriggerSound(3, soundSample3, soundSample31, this, threshhold3);
  soundKit4 = new TriggerSound(4, soundSample4, soundSample41, this, threshhold);
  soundKit5 = new TriggerSound(5, soundSample5, soundSample51, this, threshhold);
```

```
soundKit6 = new TriggerSound(6, soundSample6, soundSample61, this, threshhold);
soundKit7 = new TriggerSound(7, soundSample7, soundSample71, this, threshhold);
soundKit8 = new TriggerSound(8, soundSample8, soundSample81, this, threshhold8);


////////////////////////////////////////////////////////////////////////////////////////
//The following is code for the 3D visuals that uses the minim library.
////////////////////////////////////////////////////////////////////////////////////////

// Loading the minim library
minim = new Minim(this);

//loading song
song = minim.loadFile("daftpunk.mp3");

// Create the FFT object to analyze the song
fft = new ddf.minim.analysis.FFT(song.bufferSize(), song.sampleRate());

// One pyramid per frequency band
nbPyramids = (int)(fft.specSize()*specHi);
pyramids = new Pyramid[nbPyramids];

// As many walls as we want
walls = new Wall[nbWalls];

// Creating all objects
// Creating pyramid objects
for (int i = 0; i < nbPyramids; i++) {
  pyramids[i] = new Pyramid();
}

// Creating wall objects
// Left Wall
for (int i = 0; i < nbWalls; i+=4) {
  walls[i] = new Wall(0, height/2, 10, height);
}

//Right Wall
for (int i = 1; i < nbWalls; i+=4) {
  walls[i] = new Wall(width, height/2, 10, height);
}

//Bottom Wall
for (int i = 2; i < nbWalls; i+=4) {
  walls[i] = new Wall(width/2, height, width, 10);
}

//Top Wall
for (int i = 3; i < nbWalls; i+=4) {
```

```
    walls[i] = new Wall(width/2, 0, width, 10);
  }

  //Black background
  background(0);

  //start song
  song.play(0);

  ////////////////////////////////////////////////////////////////////////////
  //end of the setup for the 3D visuals
  ////////////////////////////////////////////////////////////////////////////
}

void draw() {
  background(0);


  //play samples
  soundKit.play();
  soundKit1.play();
  soundKit2.play();
  soundKit3.play();
  soundKit4.play();
  soundKit5.play();
  soundKit6.play();
  soundKit7.play();
  soundKit8.play();

  ////////////////////////////////////////////////////////////////////////////
  //following code is for the draw() for 3D visuals
  ////////////////////////////////////////////////////////////////////////////

  // Advance the song. On draw () for each "frame" of the song ...
  fft.forward(song.mix);

  // Calculation of "scores" (power) for three categories of sound; the low(bass), mid and high
sounds
  // First, to save the old values
  oldScoreLow = scoreLow;
  oldScoreMid = scoreMid;
  oldScoreHi = scoreHi;

  // Reset values
  scoreLow = 0;
  scoreMid = 0;
  scoreHi = 0;

  // Calculate the new "scores"
```

```
  //the function getBand returns the amplitude of the given frequency band, and 'i' is the index of
the frequency band
  for (int i = 0; i < fft.specSize()*specLow; i++)
  {
    scoreLow += fft.getBand(i);
  }

  for (int i = (int)(fft.specSize()*specLow); i < fft.specSize()*specMid; i++)
  {
    scoreMid += fft.getBand(i);
  }

  for (int i = (int)(fft.specSize()*specMid); i < fft.specSize()*specHi; i++)
  {
    scoreHi += fft.getBand(i);
  }

  // Slow down the descent so that there is a smooth transition to original score
  if (oldScoreLow > scoreLow) {
    scoreLow = oldScoreLow - scoreDecreaseRate;
  }

  if (oldScoreMid > scoreMid) {
    scoreMid = oldScoreMid - scoreDecreaseRate;
  }

  if (oldScoreHi > scoreHi) {
    scoreHi = oldScoreHi - scoreDecreaseRate;
  }

  // Volume for all frequencies at this time, with higher sounds higher.
  // This allows the animation to go faster for the more distinct sounds, which are more
noticeable
  float scoreGlobal = 0.66*scoreLow + 0.8*scoreMid + 1*scoreHi;

  // Subtle background color, slight hints of red, green and blue.
  background(scoreLow/100, scoreMid/100, scoreHi/100);

  // Pyramid for each frequency band
  for (int i = 0; i < nbPyramids; i++)
  {
    // Value of the frequency band
    float bandValue = fft.getBand(i);

    // The color is represented as red for bass, green for middle sounds and blue for highs.
    // Opacity is determined by the volume of the tape and the overall volume.
    pyramids[i].display(scoreLow, scoreMid, scoreHi, bandValue, scoreGlobal);
  }
```

```
  // Wall lines, here we keep the value of the previous band and the next one to connect them
together
  float previousBandValue = fft.getBand(0);

  // Distance between each line point, negative because on the dimension z
  float dist = -25;

  // Multiply the height by this constant
  float heightMult = 2;

  // For each band
  for (int i = 1; i < fft.specSize(); i++)
  {
    // The value of the frequency band, the more distant bands are multiplied so that they are
more visible.
    float bandValue = fft.getBand(i)*(1 + (i/50));

    // Selection of the color according to the strengths of the different types of sounds
    stroke(100+scoreLow, 100+scoreMid, 100+scoreHi, 255-i);
    strokeWeight(1 + (scoreGlobal/100));

    // Bottom left line
    line(0, height-(previousBandValue*heightMult), dist*(i-1), 0, height-(bandValue*heightMult),
dist*i);
    line((previousBandValue*heightMult), height, dist*(i-1), (bandValue*heightMult), height, dist*i);
    line(0, height-(previousBandValue*heightMult), dist*(i-1), (bandValue*heightMult), height,
dist*i);

    // top left line
    line(0, (previousBandValue*heightMult), dist*(i-1), 0, (bandValue*heightMult), dist*i);
    line((previousBandValue*heightMult), 0, dist*(i-1), (bandValue*heightMult), 0, dist*i);
    line(0, (previousBandValue*heightMult), dist*(i-1), (bandValue*heightMult), 0, dist*i);

    // bottom right line
    line(width, height-(previousBandValue*heightMult), dist*(i-1), width, height-
(bandValue*heightMult), dist*i);
    line(width-(previousBandValue*heightMult), height, dist*(i-1), width-(bandValue*heightMult),
height, dist*i);
    line(width, height-(previousBandValue*heightMult), dist*(i-1), width-(bandValue*heightMult),
height, dist*i);

    //top right line
    line(width, (previousBandValue*heightMult), dist*(i-1), width, (bandValue*heightMult), dist*i);
    line(width-(previousBandValue*heightMult), 0, dist*(i-1), width-(bandValue*heightMult), 0,
dist*i);
    line(width, (previousBandValue*heightMult), dist*(i-1), width-(bandValue*heightMult), 0, dist*i);

    // Save the value for the next loop revolution
    previousBandValue = bandValue;
```

```
  }

  // Rectangular walls
  for (int i = 0; i < nbWalls; i++)
  {
    // A band is assigned to each wall, and its force is sent to it.
    float intensity = fft.getBand(i%((int)(fft.specSize()*specHi)));
    walls[i].display(scoreLow, scoreMid, scoreHi, intensity, scoreGlobal);
  }

  ////////////////////////////////////////////////////////////////////////////
  //end of the draw() for 3D visuals
  ////////////////////////////////////////////////////////////////////////////
}

void loadSampleKits() {
  //To load a sample kit, just comment the current one out and comment the one you wish to play


  /*soundSample0 = "1.wav";
  soundSample01 = "10.wav";
  soundSample1 = "2.wav";
  soundSample11 = "11.wav";
  soundSample2 = "3.wav";
  soundSample21 = "12.wav";
  soundSample3 = "4.wav";
  soundSample31 = "13.wav";
  soundSample4 = "5.wav";
  soundSample41 = "14.wav";
  soundSample5 = "6.wav";
  soundSample51 = "15.wav";
  soundSample6 = "7.wav";
  soundSample61 = "16.wav";
  soundSample7 = "8.wav";
  soundSample71 = "8.wav";
  soundSample8 = "9.wav";
  soundSample81 = "9.wav";*/


   soundSample0 = "bom.wav";
   soundSample01 = "bom.wav";
   soundSample1 = "clap.wav";
   soundSample11 = "clap.wav" ;
   soundSample2 = "cowbell.wav";
   soundSample21 = "cowbell.wav";
   soundSample3 = "drebass.wav";
   soundSample31 = "drebass.wav";
   soundSample4 = "guitarsynth.wav";
   soundSample41 = "guitarsynth.wav";
```

```
  soundSample5 = "hihat.wav";
  soundSample51 = "hihat.wav";
  soundSample6 = "kick.wav";
  soundSample61 = "kick.wav";
  soundSample7 = "snare.wav";
  soundSample71 = "snare.wav";
  soundSample8 = "yeah.wav";
  soundSample81 = "yeah.wav";



  /*soundSample0 = "2SAD4ME.mp3";
  soundSample01 = "2SAD4ME.mp3";
  soundSample1 = "AIRHORN.mp3";
  soundSample11 = "AIRHORN.mp3";
  soundSample2 = "HITMARKER.mp3";
  soundSample21 = "HITMARKER.mp3";
  soundSample3 = "Oh Baby A Triple.mp3";
  soundSample31 = "Oh Baby A Triple.mp3";
  soundSample4 = "SPOOKY.mp3";
  soundSample41 = "SPOOKY.mp3";
  soundSample5 = "SANIC.mp3";
  soundSample51 = "SANIC.mp3";
  soundSample6  = "DAMNSON.mp3";
  soundSample61 = "DAMNSON.mp3";
  soundSample7 = "SMOKEWEEDEVERYDAY.mp3";
  soundSample71 = "SMOKEWEEDEVERYDAY.mp3";
  soundSample8 = "2SED4AIRHORN.mp3";
  soundSample81 = "2SED4AIRHORN.mp3";
  */
}


// Class for the pyramids that float in space
class Pyramid {
 // Z position of "spawn" and maximum position Z
 float startingZ = -10000;
 float maxZ = 1000;

 // Position values
 float x, y, z;
 float rotX, rotY, rotZ;
 float sumRotX, sumRotY, sumRotZ;

 //Constructor
 Pyramid() {
  // Show the pyramid at a random place
  x = random(0, width);
  y = random(0, height);
```

```
    z = random(startingZ, maxZ);

    // Give the pyramid a random rotation
    rotX = random(0, 1);
    rotY = random(0, 1);
    rotZ = random(0, 1);
  }

  void display(float scoreLow, float scoreMid, float scoreHi, float intensity, float scoreGlobal) {
    // Selection of the color, opacity determined by the intensity (volume of the strip)
    color displayColor = color(scoreLow*0.67, scoreMid*0.67, scoreHi*0.67, intensity*5);
    fill(displayColor, 255);

    // Color lines, they disappear with the individual intensity of the pyramid
    color strokeColor = color(255, 150-(20*intensity));
    stroke(strokeColor);
    strokeWeight(1 + (scoreGlobal/300));

    // Creating a transformation matrix to perform rotations, enlargements
    pushMatrix();

    //Shifting
    translate(x, y, z);

    // Calculate the rotation according to the intensity for the pyramid
    sumRotX += intensity*(rotX/1000);
    sumRotY += intensity*(rotY/1000);
    sumRotZ += intensity*(rotZ/1000);

    // Application of rotation
    rotateX(sumRotX);
    rotateY(sumRotY);
    rotateZ(sumRotZ);

    // Creation of the pyramid, variable size according to the intensity for the pyramid
    drawPyramid(70+(intensity/2));


    // Application of the matrix
    popMatrix();

    // Move Z
    z+= (1+(intensity/5)+(pow((scoreGlobal/150), 2)));

    // Replace the box in the rear when it is no longer visible
    if (z >= maxZ) {
      x = random(0, width);
      y = random(0, height);
      z = startingZ;
```

```
    }
  }
}

void drawPyramid(float t) {

  //programming the 4 sides of a pyramid
  //side 1
  beginShape(TRIANGLES);
  vertex(-t, -t, -t);
  vertex(t, -t, -t);
  vertex(0, 0, t);

  //side 2
  vertex(t, -t, -t);
  vertex(t, t, -t);
  vertex(0, 0, t);

  //side 3
  vertex(t, t, -t);
  vertex(-t, t, -t);
  vertex(0, 0, t);

  //side 4
  vertex(-t, t, -t);
  vertex(-t, -t, -t);
  vertex(0, 0, t);
  endShape();

  //bottom side
  pushMatrix();
  translate(0, 0, -t);
  rectMode(CENTER);
  rect(0, 0, 2*t, 2*t);
  popMatrix();
}



//Class to trigger sound effects based on the input of piezoeletric sensors

class TriggerSound {
//Variables for sensors, threshholds and (different kinds of) sample sound packs
  SoundFile file;
  SoundFile file2;
  int sensor;
  String sample;
  String sample2;
  int threshhold;
```

```
  boolean nextSample = false;

//explanation on PApplet p
//We need access to the PApplet instance that is automatically created for us in our Processing sketch.
//When we're in another class, the 'this' keyword refers to the instance of that class, not the sketch's PApplet instance.

  TriggerSound(int tempSensor, String tempSample, String tempSample2, PApplet p, int tempThreshhold) {    //constructor

    sensor = tempSensor;
    sample = tempSample;
    sample2 = tempSample2;
    threshhold = tempThreshhold;
    file = new SoundFile(p, tempSample);
    file2 = new SoundFile(p, tempSample2);
  }

  void play() {

    arduino.analogRead(sensor);               //read and print the input of the sensors
    println(arduino.analogRead(sensor));

    //if the intput is higher then our threshhold and the boolean nextSample is false, play sample 1 and the change the boolean to true;
    // if you then tap at the same sensor again, you will play sample 2 and reset the boolean to false

    if (arduino.analogRead(sensor) > threshhold && nextSample == false) {  //if the output is higher then our threshhold and the boolean nextSample is false
      file.play();
      delay(100);
      nextSample = true;
    } else if ( arduino.analogRead(sensor) > threshhold && nextSample == true) {
      file2.play();
      delay(100);
      nextSample = !nextSample;
    }
  }
}


// Class to display the lines on the sides
class Wall {
  // Minimum and maximum position Z
  float startingZ = -10000;
  float maxZ = 50;
```

```
  // Position values
  float x, y, z;
  float sizeX, sizeY;

  //Constructor
  Wall(float x, float y, float sizeX, float sizeY) {
    // Show the line at the specified location
    this.x = x;
    this.y = y;
    // Random Depth
    this.z = random(startingZ, maxZ);

    // The size is determined because the walls on the floors have a different size than those on
the sides
    this.sizeX = sizeX;
    this.sizeY = sizeY;
  }

  // Display function
  void display(float scoreLow, float scoreMid, float scoreHi, float intensity, float scoreGlobal) {
    // Color determined by low, medium and high sounds
    // Opacity determined by the global volume
    color displayColor = color(scoreLow*0.67, scoreMid*0.67, scoreHi*0.67, scoreGlobal);

    // Make the lines disappear in the distance to give an illusion of fog
    fill(displayColor, ((scoreGlobal-5)/1000)*(255+(z/25)));
    noStroke();

    // First band, that which moves according to the force
    // Transformation matrix
    pushMatrix();

    //Shifting
    translate(x, y, z);

    // Enlargement
    if (intensity > 100) intensity = 100;
    scale(sizeX*(intensity/100), sizeY*(intensity/100), 20);

    // Creating the "box"
    box(1);
    popMatrix();

    // Second strip, the one that is always the same size
    displayColor = color(scoreLow*0.5, scoreMid*0.5, scoreHi*0.5, scoreGlobal);
    fill(displayColor, (scoreGlobal/5000)*(255+(z/25)));
    // Transformation matrix
    pushMatrix();
```

```
    //Shifting
    translate(x, y, z);

    // Enlargement
    scale(sizeX, sizeY, 10);

    // Creating the "box"
    box(1);
    popMatrix();

    // Move Z
    z+= (pow((scoreGlobal/150), 2));
    if (z >= maxZ) {
      z = startingZ;
    }
  }
}
```

THE END