INTERFACE TECHNOLOGY FOR NEW MEDIA ART

MDM | TI

Isabel Carvalho 2019213446 | **Carlos Cabral** 2018287209
Daniel Abrantes 2019211048 | **Simão Ventura** 2019208304

TABLE OF CONTENTS

O Nosso Conceito 01
Metodologia do Design de Interação 02
Estratégia de Implementação 04
Imagens e Circuitos 10
Código Processing 16
Código Arduino 24
Artefacto Físico 29
Imagens 31

O NOSSO CONCEITO

O tema que escolhemos tratar é a New Media Art, e decidimos criar um artefacto que se insere mais no âmbito da Light Art e Sound art, que são dois ramos dessa área que, achamos nós, se podem conectar de formas interessantes.

O nosso artefacto é uma pequena instalação de Arte que consiste numa caixa oca com uma janela de acrílico e um espelho no fundo. Entre o espelho e o acrílico estarão colocadas duas fitas de LEDs que, ao reagirem ao som e à luz, vão acender/apagar e mudar de cor. Como vão estar entre o espelho e o acrílico, vão dar uma ilusão de túnel infinito,de forma semelhante ao que acontece quando pomos um espelho paralelamente em frente a outro.

Para além dos LEDs, teremos também um ou mais Servos, que também vão reagir ao som e à luz, a produzir outro efeito. A nossa ideia é que na face de cima da caixa seja cortada uma ranhura por onde o utilizador pode inserir uma folha de acrílico, e os Servos vão ter algo agarrado a eles, como um pincel ou um marcador, para fazerem desenhos na folha em questão.

Como o utilizador tem controlo total sob a folha de acrílico durante todo o processo de utilização, pode mexê-la, rodá-la, ou até retirá-la a meio da utilização e pôr outra vez, acabando assim com uma espécie de desenho personalizado criado ao mesmo tempo por si e pela música que escolheu tocar.

A simplicidade das “regras” torna o artefacto mais intuitivo e permite que o utilizador dê asas à sua imaginação para criar um produto irrepetível, quase como se tratasse de um projeto de arte generativa em que o utilizador age como o computador a executar o algoritmo.

METODOLOGIA DO DESIGN DE INTERAÇÃO

RESEARCH

Nesta fase primária e inicial da pesquisa, começámos por fazer um brainstorming de ideias para o projeto. Visto que a utilização de arduino é fundamental e o principal foco do trabalho, o nosso leque de escolhas ficou logo aí limitado. No entanto, arduino é uma ferramenta que propõe inúmeras invenções, e sabendo que teríamos de implementar obrigatoriamente dois ou mais inputs distintos (ex. som, luz, movimento, etc.), o nosso foco começou logo por se orientar nessa direção.

Sendo o nosso grupo composto por quatro elementos, conseguimos logo nesta etapa obter bons resultados do brainstorming. Os inputs que decidimos implementar nesta fase foram o som e a luz, cada um alterando uma componente distinta, que em conjunto formariam o resultado desejado.

Ainda neste aspeto, para procurar algo que todos concordassem ser o conceito adequado, procurámos algo em comum que todos apreciamos, chegando à conclusão que, como todos somos ávidos adeptos de música, seria interessante fazer um visualizador de música.

DESIGN

Com o nosso conceito definido, logicamente o passo seguinte seria descobrir a melhor maneira de transformar a ideia para um objeto, com a restrição do arduino, de forma que melhor se ajustasse aos nossos desejos, tanto estéticos como de funcionamento.

Assim sendo, optámos por um design moderno, talvez até futurista, tentando implementar acrílico e espelhos para criar ilusões de ótica, dando ao projeto a estética desejada. Começámos por tentar perceber que ferramentas funcionariam de melhor forma dada a nossa visão, e chegámos à conclusão que duas fitas de LED's satisfariam as nossas necessidades quanto ao input de luz, e quanto ao input de som, um simples sensor de som resolveria o problema.

Com as ferramentas em questão escolhidas, restava-nos apenas decidir qual o material que constituiria a estrutura deste visualizador. Nesta fase de prototipagem, estamos a pensar usar madeira para suportar toda a montagem, assim como para dar uma estética mais simples e moderna à construção.

AVALIAÇÃO

No que toca ao interior da construção, tencionamos colocar o máximo de hardware tecnológico fora da visão. Estaria tudo enclausurado dentro da estrutura de madeira, criando assim uma estética limpa para o olhar, assim como reduzir a possibilidade de desconexão acidental de alguma das componentes. No esforço de tentar ter o mínimo de cabos expostos, tencionamos também dar uso a uma ou mais pilhas para dar força ao visualizador.

Para avaliar e testar as ligações na breadboard e no arduino, utilizámos o Tinkercad, que fornece uma excelente visualização e manipulação de todas as ligações pretendidas, assim como encontrar erros inesperados. Para nos conduzir no caminho certo, a visualização das conexões na interface do Tinkercad ajuda-nos a perceber todas as conexões feitas de forma clara, assim como ajudar a perceber como essas ligações e como o hardware ficará posicionado de melhor forma dentro da estrutura de madeira.

ITERAÇÃO

Tendo portanto decidido tudo o que veio para trás, estava na hora de começar a fazer uns sketches da estrutura. Antes de definir as dimensões e forma da estrutura de madeira, começámos pelo fundamental: como ligar o arduino. Temos nesta fase duas opções: ligar o arduino com o auxílio de pilha, ou ligar um cabo USB diretamente do arduino ao computador.

Para a iteração da pilha, ponderámos colocar um botão ou uma alavanca à superfície da madeira, numa posição estratégica, para complementar a estética minimalista para a qual estávamos a apontar. Na vertente USB, pensámos em utilizar um cabo retrátil USB-A para USB-B, ficando apenas a extremidade USB-A fora da caixa, na parte de trás (o menos visível possível), podendo o utilizador esticar o cabo durante a utilização do visualizador, retraindo-o assim que terminasse a utilização do mesmo (semelhante às fichas de um aspirador).

No que toca à utilização de madeira, decidimos por agora utilizar cortada a lazer, tendo ainda que refinar como esta se estruturará para providenciar o maior suporte possível à construção. Quanto ao seu formato, pensámos incluir a madeira criando umas bordas ocas em torno no acrílico e do vidro, com especial atenção às dimensões da borda inferior, onde pretendemos incluir todo o hardware referente ao arduino. Esta poderá numa fase final ser pintada com uma cor e com um acabamento ainda por decidir.

IMPLEMENTAÇÃO

A implementação por nós feita digitalmente (código e sketch visual no Tinkercad) já está de forma geral funcional, tendo chegado a uma iteração da mesma que nos permite começar a pensar na implementação física. Quanto a esta, devido à falta de material (LED strip e servos), ainda não conseguimos criar um protótipo com alguma fidelidade, porém já estamos a direcionar forças para esse fim.

ESTRATÉGIA DE IMPLEMENTAÇÃO

Para a implementação, apesar de ainda não termos definido completamente o funcionamento final do artefacto, queremos definir como utilizar os valores dos inputs no contexto em questão. Vamos utilizar um sensor de som, cujos valores serão interpretados para definir a forma como os atuadores vão reagir, e decidimos também utilizar um sensor de iluminação para introduzir outras possibilidades, como a alteração das cores mostradas pelos LEDs.

A nossa estratégia passa por recolher os valores de saída dos sensores e utilizá-los de diferentes maneiras com cada atuador para obter efeitos variados. Queremos misturar alterações derivadas de ambos os sensores para ter essa variação, isto é, em vez de usarmos o sensor de som para um dos atuadores e o sensor de luz para o outro, ambos os atuadores têm comportamentos que são ditados pelo output dos dois sensores (por exemplo, o sensor de som define quantos LEDs estão ligados a cada instante, e o sensor de luz define a cor dos mesmos).

Estamos a pensar implementar o código por "blocos", em vez de juntarmos o código todo de uma vez vamos fazer testes isolados com cada sensor, ajustar o que for necessário para o funcionamento pretendido dos atuadores consoante os valores do output, e assim que conseguirmos um efeito satisfatório, fazer a montagem final do circuito para podermos avaliar as dimensões que vamos utilizar para a caixa.

Desta forma é mais fácil evitar bugs e confusões que possam surgir ao implementar o código todo no mesmo ficheiro antes de fazer experiências, para além de que também torna o processo de documentação do código muito mais fácil.

"BOUNCE"

Começámos por implementar e testar código para experimentar com alguns efeitos que achámos interessantes para o projeto. Para isso, ligámos um conjunto de LEDs e um potenciómetro. Neste exemplo, o potenciómetro simula o output obtido através do sensor de som, e o seu valor é o que dita a velocidade a que os LEDs se acendem e apagam. O efeito que aplicámos aqui alude a uma bola a cair no chão e a fazer ricochete, cada vez perdendo mais energia cinética até parar, e depois recomeça com a altura máxima outra vez. Acabámos por descartar esta hipótese porque é um pouco demorada para aplicar a um artefacto que reage ao som em tempo real.

```
int r=12;
int o=11;
int y=10;
int g=9;
int b=8;
int up=12;
int down=8;

void setup() {

      pinMode(0,INPUT);
      pinMode(r,OUTPUT);
      pinMode(o,OUTPUT);
      pinMode(y,OUTPUT);
      pinMode(g,OUTPUT);
      pinMode(b,OUTPUT);
}

void loop() {

      int val=analogRead(0);
      if(down==12) {
            down=8;
            up=12;
      }
      for(int x=up;x≧down;x--) {

            digitalWrite(x,HIGH);
            delay(val);
            digitalWrite(8,LOW);
            digitalWrite(9,LOW);
            digitalWrite(10,LOW);
            digitalWrite(11,LOW);
            digitalWrite(12,LOW);
      }

      for(int y=down;y≦12;y++) {

            digitalWrite(y,HIGH);
            delay(val);
            digitalWrite(8,LOW);
            digitalWrite(9,LOW);
            digitalWrite(10,LOW);
            digitalWrite(11,LOW);
            digitalWrite(12,LOW);
      }

      down+=1;
}
```

O código em si é relativamente simples. Começamos por introduzir variáveis para os LEDs, e definir o pinMode para output. Temos também as variáveis up e down que definem os limites do "bounce".
No loop, lemos o valor do da entrada analógica onde o potenciómetro está conectado, e utilizamos uma variável chamada 'val' para o armazenar. De seguida, temos dois ciclos for que vão iterando sob os LEDs, acendendo e apagando cada um consoante o efeito pretendido.

Até ao momento, como ainda não estamos na posse de todos os componentes, decidimos fazer experiências relevantes ao projeto no Tinkercad para ver que tipo de efeitos conseguiríamos obter. Há componentes que não estão disponíveis no Tinkercad, pelo que em alguns casos tivemos que utilizar um potenciómetro para simular o output dos mesmos (O sensor de som, por exemplo, é um dos que não estão presentes).

A nossa ideia é utilizar o valor captado pelo sensor de som para definir quantos LEDs estão acesos a cada momento. Para facilitar o desenvolvimento, queremos usar fitas de LED em vez de LEDs individuais. ainda não sabemos ao certo qual a dimensão do projeto final, mas provavelmente será, em parte, definido pelo tamanho das fitas que utilizarmos. Para além do sensor de som, queremos também utilizar um sensor de iluminação, que irá determinar a cor dos LEDs.

Queremos também implementar um código que permita calibrar o sensor de luz assim que se liga o artefacto, visto que as condições de iluminação nem sempre permitem a utilização do range completo de outputs de que o sensor é capaz.

CALIBRAGEM DO SENSOR DE ILUMINAÇÃO

O código para calibrar o LDR é o seguinte:

```
#define LRD_Pin A0

int LRD_Output = 0;
int LDR_Min = 1023;
int LDR_Max = 0;


void setup() {

      Serial.begin(9600);

      while (millis() < 6000) {

            LDR_Output = analogRead(LDR_Pin);

            if (LDR_Output > LDR_Max) {
                  LDR_Max = LDR_Output;
            }

            if (LDR_Output < LDR_Min) {
                  LDR_Min = LDR_Output;
            }
      }
}
```

O funcionamento da calibração é o seguinte: Quando ligamos o circuito, damos alguns segundos ao utilizador para expor o sensor aos níveis de iluminação que espera obter no local onde está a utilizar o artefacto, abrindo e fechando as cortinas, por exemplo.

Em termos de código, começamos por iniciar as variáveis do valor máximo e mínimo a 0 e 1023, respetivamente, e depois, à medida que o sensor capta a iluminação do espaço, vai atualizando estes valores, que depois podemos mapear para serem posteriormente utilizados pelos atuadores.

SIMULAÇÃO COM TODOS OS COMPONENTES

Para obtermos uma imagem mais completa daquilo que será o funcionamento de um protótipo físico deste artefacto, decidimos utilizar potenciómetros para simular o output dos dois sensores (LDR e sensor de som). Abaixo segue o código para esta experiência:

```
#include <Adafruit_NeoPixel.h>
#include <Servo.h>

#define POT1 A5 //Potenciómetro 1 (simula a o output do sensor de som)
#define POT2 A3 //Potenciómetro 2 (simula a o output do sensor de luz)
#define SER A4

int LED_Strip = 2, LEDs = 12;
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(LEDs, LED_Strip, NEO_GRB + NEO_KHZ800);

Servo myServo;

void setup() {

      Serial.begin(9600);
      pixels.begin();
      pixels.show();

      pinMode(LED_Strip,OUTPUT);
      pinMode(POT1,INPUT);
      pinMode(POT2, INPUT);

      myServo.attach(SER);
      myServo.write(0);

      delay(100);
}
```

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

```
void loop() {

      int amp = analogRead(POT1);
      int light = analogRead(POT2);

      for(int i = 0; i < map(amp, 0, 1023, 0, LEDs); i++) {
            pixels.setPixelColor(i, pixels.Color(map(light, 0, 1023, 0, 255), 0,
            map(light, 0, 1023, 255, 0)));
            pixels.show();
      }

      if(amp == 0) {
            pixels.clear();
      }

      delay(100);

      myServo.write(map(amp, 0, 1023, 0, 180));
      pixels.clear();
}
```

Neste código, começamos por incluir as bibliotecas que nos permitem controlar a fita de LEDs e o servo, e definir algumas variáveis relevantes, como as entradas em que cada um dos componentes está ligado e o número de LEDs da fita. Depois, no setup, escolhemos o pinMode dos componentes e inicializamos o servo e a fita de LEDs.

No loop, começamos por definir duas variáveis: amp, que simula o output que seria recebido do sensor de som, controlado pelo potenciómetro 1, e light, que simula o output do sensor de luz, controlado pelo potenciómetro 2,

De seguida, utilizando um ciclo "for", iteramos sob a fita de LEDs, desde o índice 0 até a um índice definido pelo output do sensor de som, este ciclo for é onde acendemos os leds.

Para definir a cor a ser mostrada pelos LEDs, utilizamos o output dado pelo sensor de luz. Neste exemplo estamos só a mapear uma luz mais azul para a iluminação elevada e uma mais vermelha para a iluminação mais baixa, mas posteriormente é provável que utilizemos mais cores.

A seguir ao ciclo for, usamos o if(amp == 0) { pixels.clear(); }, que serve para deixar os LEDs todos apagados caso o output do sensor de som seja 0, depois manipulamos o servo, também com o valor do output do sensor de som, e por último, fazemos um clear da fita de LEDs para estar pronta para o próximo "frame".

Como ainda não temos os componentes todos e tivemos que utilizar os potenciómetros no tinkercad como forma de simular os valores de output, por isso é possível que ao fazer a versão com os sensores pretendidos e testar o funcionamento, o efeito obtido seja algo diferente do que estes exemplos proporcionam, porque em todos os exemplos, temos um nível elevado de controlo sob os valores do output.

Com isto, é possível que tenhamos que efetuar algumas alterações na versão final, sejam elas em termos de código ou simplesmente de valores.

Outro problema que pode surgir é relativo aos Servos. Como nunca utilizámos um Servo não temos a certeza até que ponto é possível este acompanhar o output do sensor em tempo real, pelo que se verificarmos que não consegue, teremos que pensar noutro efeito para implementar que seja mais viável.

Adicionalmente, como vamos implementar o código por partes, pode dar-se o caso de, ao juntarmos os componentes e o código num artefacto completo, não acharmos o resultado tão bom como pretendíamos, e termos de ajustar novamente alguns valores até atingirmos o objetivo.

IMAGENS E CIRCUITOS

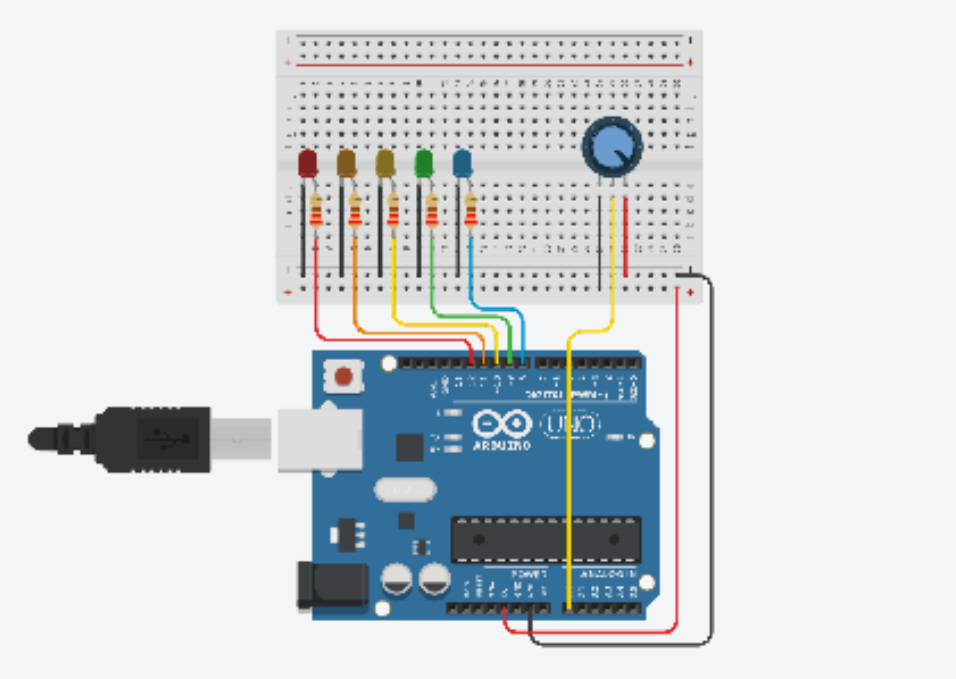

Fig.1. Montagem do “Bounce”

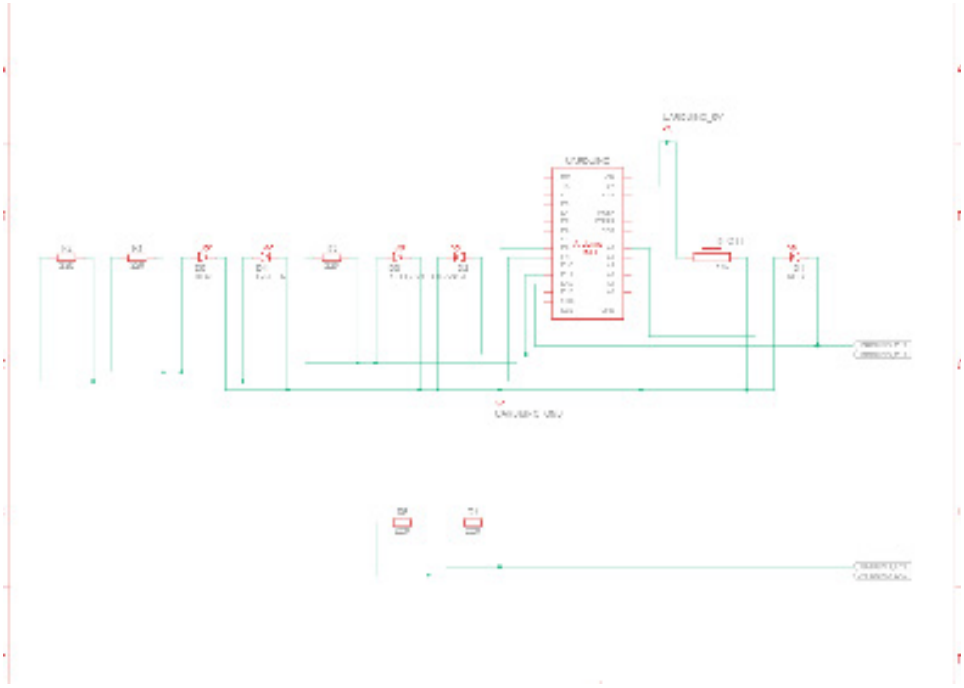

Fig.2. Circuito do “Bounce”

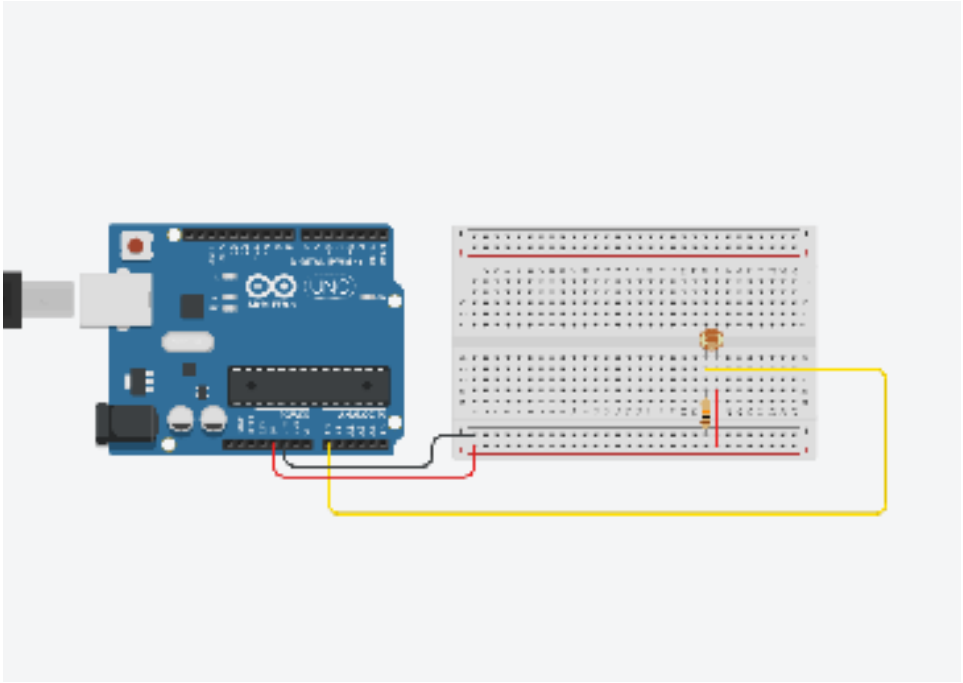

Fig.3. Montagem do LDR para a calibragem

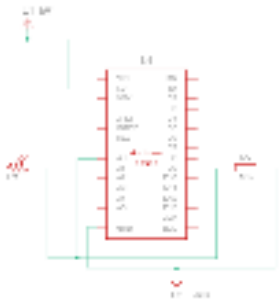

Fig.4. Circuito do LDR para a calibragem

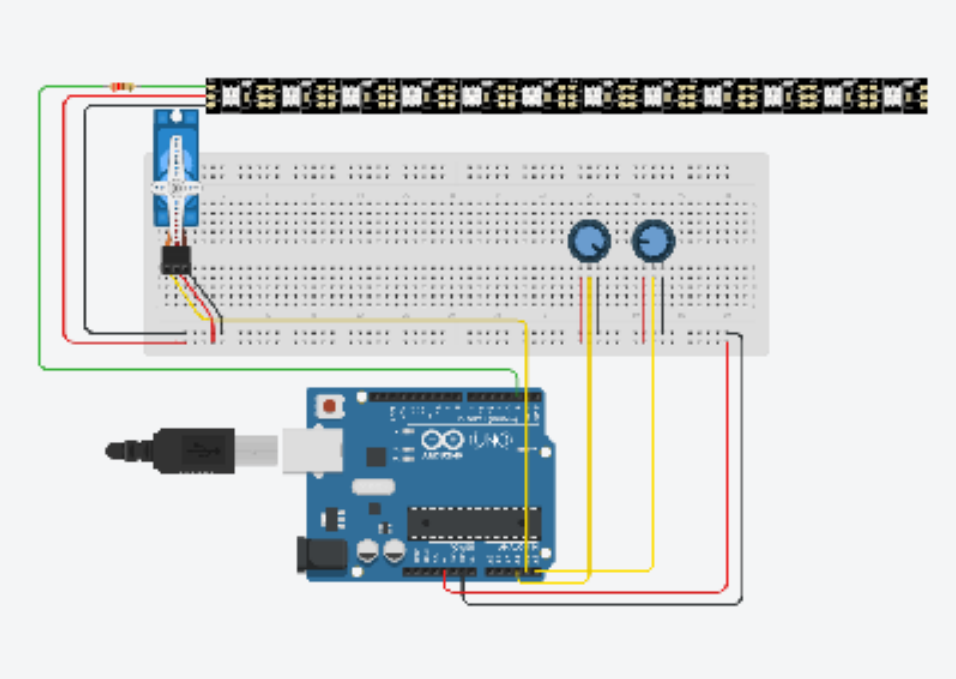

Fig.5. Montagem da simulação com os componentes todos

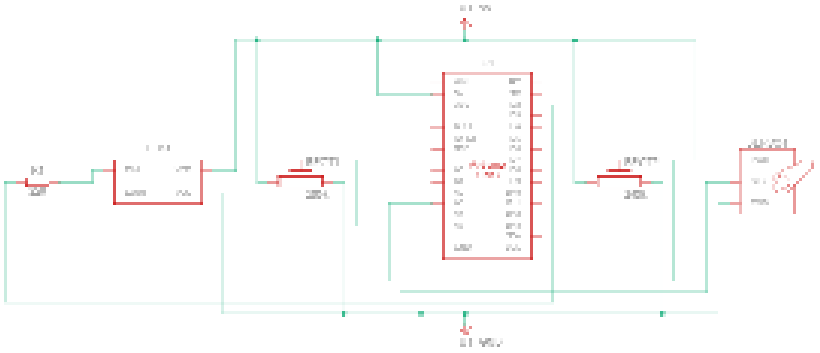

Fig.6. Circuito da simulação com os componentes todos

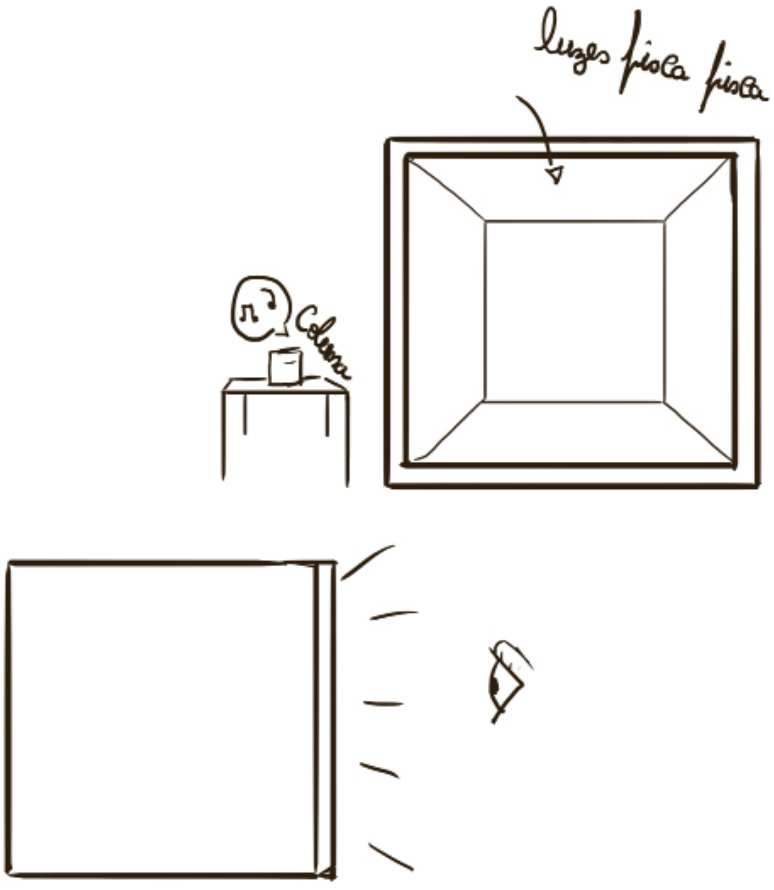

Fig.7. Sketch ilustrativo

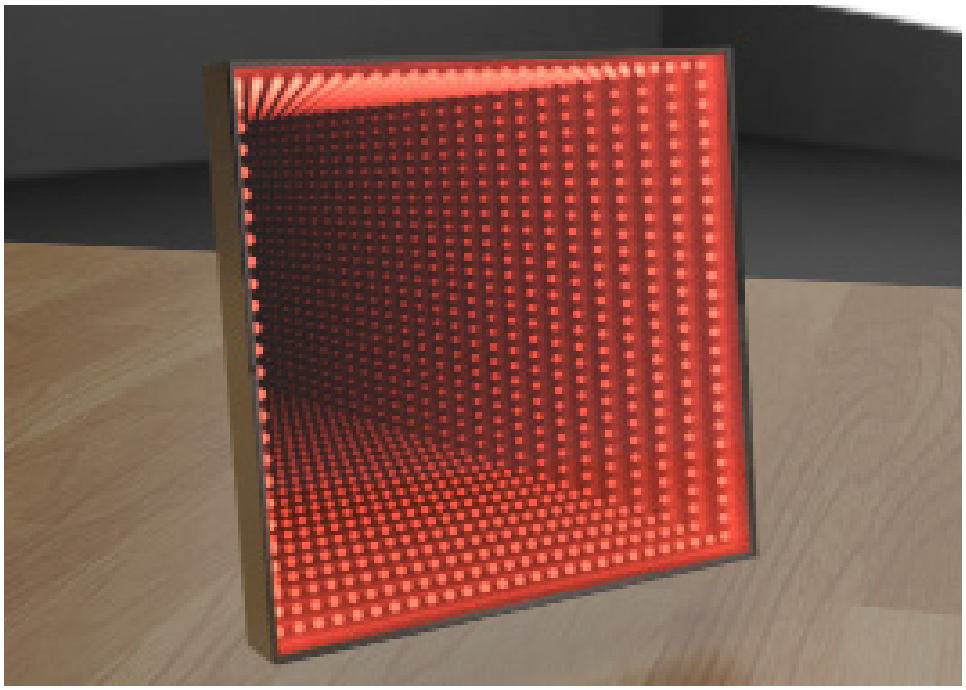

Fig.8. Render

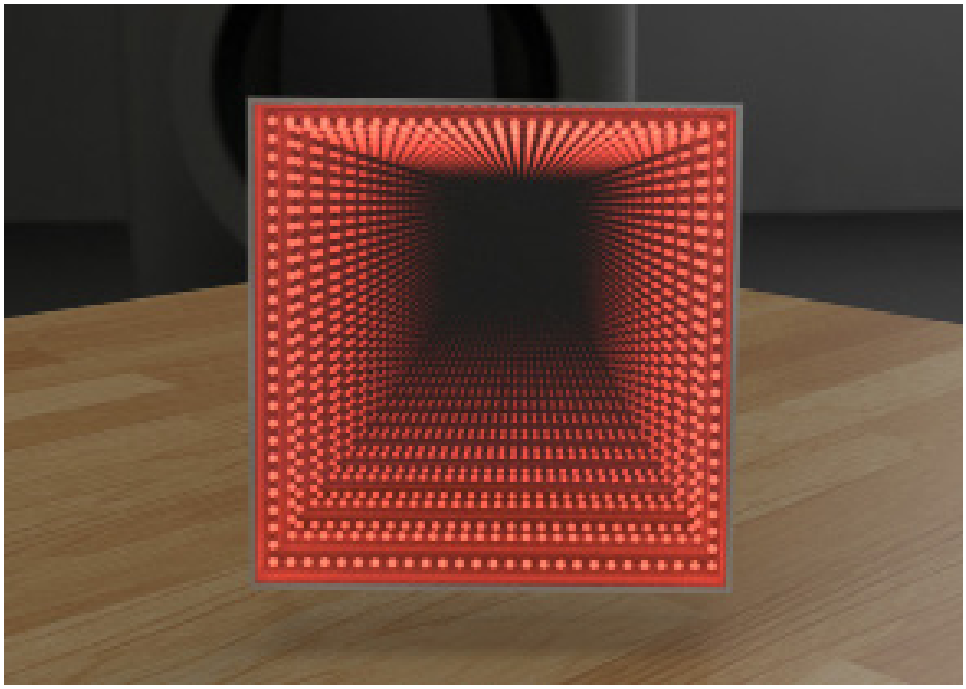

Fig.9. Render

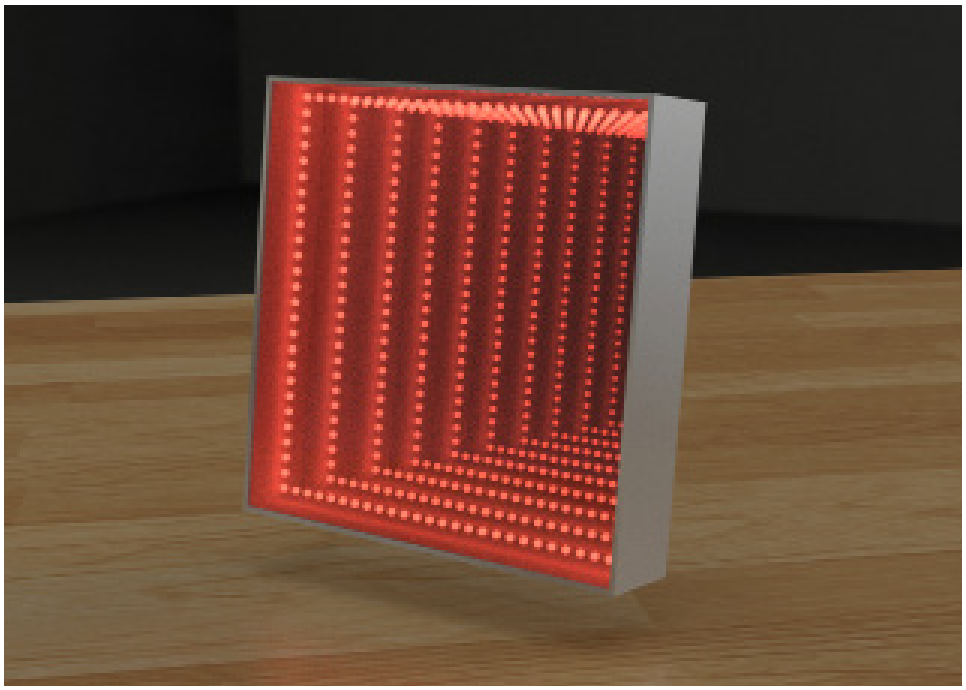

Fig.10. Render

CÓDIGO PROCESSING

BIBLIOTECAS E VARIÁVEIS

```
import processing.sound.*;
import processing.serial.*;

SoundFile song;
String[] songs;
String[] max_amps;
int curr_song = 0;
Amplitude amp;
BeatDetector beats;

float amp_value;
float min_amp = 0, max_amp = 0.75;

//————————————————————————————————————— flowfield
int scale = 50;
int cols, rows;

PVector[] flowfield;
float xy_inc = 0.05, z_inc = 0.0001;
float x_off, y_off, z_off = 0;

//————————————————————————————————————— partículas
ArrayList<particle> particles;

Serial port;
```

Começámos por importar as bibliotecas necessárias para o funcionamento do projeto, que são, neste caso, a biblioteca de som do processing, e a biblioteca Serial, que utilizamos para estabelecer a comunicação entre o processing e o arduino.

De seguida, introduzimos variáveis para armazenar informações pretinentes, como a música atual, dois arrays com informação retirada de ficheiros de texto relativamente à mesma, o input da amplitude, e os beats. Introduzimos também uma variável para o valor da amplitude, para evitar escrever “amp.analyze()” sempre que necessitamos do valor para algum cálculo, e as variáveis min_amp e max_amp, onde armazenamos os valores mínimos e máximos da amplitude de cada música.

Introduzimos variáveis para controlar a escala e as dimensões do flowfield, bem como para controlar o Perlin noise utilizado posteriormente para gerar o mesmo. Por fim, introduzimos um ArrayList para as partículas, e uma variável para o input da Port.

SETUP

```
fullScreen(P3D);
colorMode(HSB);
strokeCap(ROUND);

songs = loadStrings("songs/songs.txt");
max_amps = loadStrings("songs/amplitudes.txt");
song = new SoundFile(this, "songs/"+songs[0]+".wav");
max_amp = float(max_amps[0]);

amp = new Amplitude(this);
beats = new BeatDetector(this);

song.play();
amp.input(song);
beats.input(song);
beats.sensitivity(1);

//————————————————————————————————————— grelha de vetores
cols = floor(width/scale);
rows = floor(height/scale);
flowfield = new PVector[cols * rows];
particles = new ArrayList<particle>();

port = new Serial(this, "/dev/cu.usbserial-1110", 9600);
port.bufferUntil(10);

background(0);
```

No setup definimos as características gerais do programa, como o ColorMode e o tamanho do ecrã, depois carregamos os ficheiros necessários e iniciamos o playback da música.

```
amp_value = amp.analyze();

   fill(map(amp_value, min_amp, max_amp, 0, 10), map(amp_value, min_amp, max_amp,
60, 30));
  noStroke();
  rect(0, 0, width, height);

  //————————————————————————————————————————— preencher flowfield com vetores
  x_off = 0;
  for (int x = 0; x < cols; x++) {

    y_off = 0;
    for (int y = 0; y < rows; y++) {

      float ang = noise(x_off, y_off, z_off) * TWO_PI * 8;
      PVector vector = PVector.fromAngle(ang);
      vector.setMag(1);

      flowfield[x + y * cols] = vector;

      y_off += xy_inc;
    }

    x_off += xy_inc;
    z_off += map(amp_value, min_amp, max_amp, z_inc * 10, z_inc);
  }

  if ((beats.isBeat() || amp_value ≥ max_amp-0.2) && particles.size() < 5000) {
    for (int i = 0; i < int(map(amp_value, min_amp, max_amp, 1, 10)); i++) {
      particles.add(new particle(amp_value, new PVector(width/2, height/2)));
    }

    for (int i = 0; i < particles.size(); i++) {
      particle p = particles.get(i);

      if (random(1) ≤ 0.01) {
        strokeWeight(map(amp_value, min_amp, max_amp, 1, 2));
        stroke((map(amp_value, min_amp, max_amp, 0, 255) + frameCount) % 255, 255,
255, map(amp_value, min_amp, max_amp, 0, 30));

        particle p2 = particles.get(int(random(particles.size())));

        for (int j = 0; j < int(map(amp_value, min_amp, max_amp, 1, 4)); j++) {
           line(p.pos.x + random(-10, 10), p.pos.y + random(-10, 10), p2.pos.x +
random(-10, 10), p2.pos.y + random(-10, 10));
        }
      }
```

```
    }
  }

  if (particles.size() > 0) {

    for (int i = 0; i < particles.size(); i++) {
      particle p = particles.get(i);

      p.update(amp_value);
      p.follow(flowfield);
      p.display(amp_value);


      if (p.outsideScreen() || p.isDead()) particles.remove(i);
    }
  }
```

No Draw começámos por criar o background e armazenar o valor da amplitude na variável amp_value. De seguida, preenchemos o flowfield com vetores, sendo que estes estão constantemente a sofrer alterações ao longo do tempo.

Quando detetamos um beat ou algum momento de elevada amplitude na música, utilizamos o valor da amplitude para determinar quantas partículas novas vão ser colocadas no flowfield, e desenhamos linhas entre algumas partículas (probabilidade de 1%).

Após tudo estar calculado, utilizamos um ciclo for para iterar sobre o ArrayList, desenhar, atualizar, e remover as partículas se necessário.

```
String msg = song.isPlaying() ? “”+map(amp_value, 0, max_amp, 0, 550)+”\n” :
“”+0+”\n”;
  port.write(msg);

  if ( port.available() > 0) {  // If data is available,
    String val = port.readStringUntil(‘\n’);

    if (val ≠ null) {
      String[] pieces = val.split(“,”);
      String myString = pieces[0];
      int input = Integer.parseInt(pieces[1].trim());
      println(“myString: “ + myString);
      println(“input: “ + input);

      if (myString.equals(“Play”) && !song.isPlaying()) {
        println(“PLAY”);
        song.play();
      } else if (myString.equals(“Pause”) && song.isPlaying()) {
        song.pause();
```

```
    }
    if (input == 3) {
      song.stop();

      curr_song++;
      curr_song %= songs.length;

      song = new SoundFile(this, “songs/”+songs[curr_song]+”.wav”);
      max_amp = float(max_amps[curr_song]);

      amp.input(song);
      beats.input(song);

      song.play();
    } else if (input == 2) {
      song.stop();

      curr_song--;
      if (curr_song < 0) {
        curr_song = songs.length-1;
      }

      song = new SoundFile(this, “songs/”+songs[curr_song]+”.wav”);
      max_amp = float(max_amps[curr_song]);

      amp.input(song);
      beats.input(song);

      song.play();
    }
  }
}
```

Por último introduzimos uma variável que comunica com o Arduino para enviar o valor da amplitude, e verificamos se o Arduino comunicou com o processing, para podermos introduzir o código respetivo às interações com os botões.

```
PVector prevPos, pos, vel, acc;
  float vel_max;

  int size, lifespan;

//————————————————————————————————— construtor
  particle(float amp, PVector position) {

    //pos = new PVector(random(width), random(height));
    pos = new PVector(position.x + random(-20, 20), position.y + random(-20, 20));
    vel = new PVector(0, 0);
    acc = new PVector(0, 0);
    prevPos = pos.copy();
    size = int(map(amp, min_amp, max_amp, 1, 6));
    lifespan = 200;
  }
```

Para as partículas, criamos uma classe com diversos parâmetros, desde a posição da partícula, à velocidade, tamanho, e tempo de vida.

```
void display(float amp) {

     stroke((map(amp, min_amp, max_amp, 0, 255) + frameCount) % 255, 255, 255,
map(amp, min_amp, max_amp, 0, 100));
    strokeWeight(size);
    line(pos.x, pos.y, prevPos.x, prevPos.y);

    updatePrev();
  }

//————————————————————————————————— atualizar partícula
  void update(float amp) {

    vel.add(acc);
    vel.limit(vel_max);
    pos.add(vel);
    acc.mult(0);

    size = int(map(amp, min_amp, max_amp, 1, 6));
    vel_max = map(amp, min_amp, max_amp, 1, 15);
    lifespan--;
  }
```

```
//———————————————————————————————— atualizar posição anterior
  void updatePrev() {
    prevPos = pos.copy();
  }

//———————————————————————————————— vetores a seguir
  void follow(PVector[] vectors) {

    int x = (int)map(pos.x, 0, width, 0, cols - 1), y = (int)map(pos.y, 0, height,
0, rows - 1);
    int index = x + y * rows;

    addForce(vectors[index]);
  }

//———————————————————————————————— aplicar força
  void addForce(PVector force) {

    acc.add(force);
  }
```

Temos diversos métodos incluídos na classe. O display() é responsável por desenhar a partícula e, através de uma chamada ao updatePrev(), atualizar a variável que armazena a posição anterior da partícula.

O update() atualiza todos os outros parâmetros da partícula, e o follow() é o método que utilizamos para determinar que vetor é que está a afetar uma determinada partícula a cada momento.

```
  boolean outsideScreen() {
    return pos.x ≤ 0 || pos.x ≥ width || pos.y ≤ 0 || pos.y ≥ height;
  }

  boolean isDead() {
    return lifespan ≤ 0;
  }
```

Os últimos métodos são booleans que utilizamos para verificar as condições necessárias para remover as partículas, isto é, para saber se o seu lifespan chegou ao fim ou se saíram do ecrã, o que leva a que as partículas sejam removidas.

PROBLEMAS ENCONTRADOS E RESPETIVAS SOLUÇÕES

MÚSICAS E AMPLITUDES

Um dos primeiros problemas com que nos deparámos foi o facto de cada soundfile ter um volume diferente. Visto que não tínhamos nenhuma forma automatizada de alterar a amplitude das músicas para ficarem todas iguais, introduzimos as variáveis min_amp e max_amp para armazenar os valores respetivos à musica, que lemos a partir de um ficheiro de texto.

Esta estratégia foi necessária porque o valor da amplitude controla grande parte do funcionamento do programa, então as músicas com uma amplitude média mais baixa não mostravam tanto o efeito pretendido até aplicarmos esta alteração. Uma solução mais versátil seria dar ao utilizador uma maneira de importar músicas para o programa e ter alguma forma de analizar a música toda sem ter de a tocar para anotar os valores da mesma.

PERFORMANCE

Nos primeiros drafts que fizemos do visualizador, as partículas eram redondas em vez de rectangulares. Infelizmente, uma das desvantagens do processing é não possibilitar a utilização da placa gráfica para acelerar a execução do código sem alterar fundamentalmente o código.

Deparámo-nos com duas opções: Manter o P2D e as partículas redondas, ou passar para o P3D e conseguir mostrar um número muito mais elevado de partículas (nos primeiros drafts com P2D tínhamos 200 partículas em vez de 5000). Decidimos manter o P3D porque não considerámos a diferença de estética significante o suficiente para sacrificar a performance do artefacto.

COMUNICAÇÃO COM O ARDUINO

A comunicação que fazemos entre o processing e o Arduino é simples, temos apenas uma variável que utilizamos para enviar o valor da amplitude em cada frame. Caso a música esteja pausada, enviamos um 0 em vez do valor da amplitude, caso contrário a fita de LED continuaria a criar barras coloridas baseadas na última amplitude recebida.

CÓDIGO ARDUINO

Para o desenvolvimento deste código utilizamos a biblioteca FastLED, para controlar duas fitas de leds WS2812, onde cada led pode ser controlado individualmente. No código primeiro definimos os pins de onde vai sair a informação tanto para a fita de led como para os botões que utilizamos.

```
#include <FastLED.h>

int r = 152;
int g = 0;
int b = 80;

int bri = 0;

int max = 0;
int min = 1023;

int max2 = 0;
int min2 = 1023;

const int buttonPlay = 8;
const int buttonNext = 9;
const int buttonPrev = 10;

int state = 0;

int buttonState = 0;

String sv;

#define LED_PIN 7
#define NUM_LEDS 288

CRGB leds[NUM_LEDS];
CRGB led[NUM_LEDS];
int s = 0;
```

Na parte de inicialização do código, estamos apenas a iniciar os leds, e as características deles, bem como o número total de leds das fitas e o modo de cor usado pelas mesmas. Em seguida temos uma faze de inicialização do projeto, onde os leds se ligam um de cada vez a partir do meio até ao final respetivo, formando assim uma espécie de countdown para o início do visualizador. Inicializamos o serial e definimos os botões em INPUT_PULLUP, decidimos usar este tipo de input pois estávamos a ter alguns problemas de conexão dos cabos na montagem e com este tipo de input seria mais fácil.

```
void setup() {


  FastLED.addLeds<WS2812, LED_PIN, GRB>(leds, NUM_LEDS);
  for (int i = NUM_LEDS / 2; i ≥ 0; i--) {
    leds[i] = CRGB(r, g, b);
    leds[NUM_LEDS - i] = CRGB(r, g, b);
    FastLED.show();
  }
  Serial.begin(9600);


  pinMode(buttonPlay, INPUT_PULLUP);
  pinMode(buttonNext, INPUT_PULLUP);
  pinMode(buttonPrev, INPUT_PULLUP);
}
```

Passamos de seguida para o loop do programa, onde começamos por ter algumas condições para os estados dos botões. Na primeira condição, quando o botão é pressionado (neste caso utilizamos a confirmação quando o estado era LOW, pois os botões estão configurados em INPUT_PULLUP como mencionado anteriormente), e a variável buttonState é igual a 1 o valor dessa mesma variável fica igual a 2. Esta variável serve para definir qual é a string que vai ser enviada para o processing, que por sua vez define se a música está a tocar ou se está em pausa. As outras duas condições servem para alterar a variável state que por sua vez também seria enviada para o processing para controlar se a mudavam de música para a próxima ou para a anterior.

```
if (digitalRead(buttonPlay) == LOW) {
    if (buttonState == 1) {
      buttonState = 2;
    } else {
      buttonState = 1;
    }
  }

  if (buttonState == 1) {
    sv = “Play”;
  } else if (buttonState == 2) {
    sv = “Pause”;
  }

  if (digitalRead(buttonNext) == LOW) {
    state = 2;
  } else {
    state = 0;
  }

  if (digitalRead(buttonPrev) == LOW) {
    state = 3;
  }

  Serial.print(sv);
  Serial.print(“,”);
  Serial.println(state);
```

Para a leitura da luminosidade no local, utilizamos um sensor de luz, que foi declarado no seguinte código e onde estamos a definir dinamicamente os seus valores máximos e mínimos para que possa se calibrar automaticamente consoante o ambiente onde foi colocado. Em seguida utilizamos esses valores que são mapeados para uma variável que irá controlar o brilho das fitas.

```
int sensor = analogRead(A0);

    if (sensor > max2) {
      max2 = sensor;
    }

    if (sensor < min2) {
      min2 = sensor;
    }

    bri = map(sensor, min2, max2, 0, 180);
```

Aqui estamos a ler o sinal que está a ser enviado do processing, estamos a colocar esses valores numa String e a cortar os seus valores para que sejam lidos pelo arduino. De seguida transformamos esses valores de novo em inteiros. Estes valores que estão a ser recebidos, são a amplitude da música que está a ser tocada no processing que irá controlar as cores dos leds.

```
String str = Serial.readStringUntil('\n');
    str.trim();

    int ss = str.toInt();
```

Em seguida, estamos novamente a calibrar os valores recebidos do processing, pois não temos como saber quais os limites de cada música, assim o algoritmo calibra-se automaticamente a cada música.

```
if (ss > max) {
      max = ss;
    }

    if (ss < min) {
      min = ss;
    }

    s = ss;
```

O código a seguir é onde controlamos as cores dos leds em função da amplitude da música.

```
if ((s ≥ 450) && (s ≤ 550)) {
      leds[(NUM_LEDS / 2) - 1] = CRGB(0, 0, 255);
      leds[NUM_LEDS / 2] = CRGB(0, 0, 255);
    } else if ((s ≥ 400) && (s ≤ 450)) {
      leds[(NUM_LEDS / 2) - 1] = CRGB(153, 153, 0);
      leds[NUM_LEDS / 2] = CRGB(153, 153, 0);
    } else if ((s ≥ 350) && (s ≤ 400)) {
      leds[(NUM_LEDS / 2) - 1] = CRGB(255, 50, 255);
      leds[NUM_LEDS / 2] = CRGB(255, 50, 255);
    } else if ((s ≥ 300) && (s ≤ 350)) {
      leds[(NUM_LEDS / 2) - 1] = CRGB(10, 25, 217);
      leds[NUM_LEDS / 2] = CRGB(10, 25, 217);
    }


    else if ((s ≥ 276) && (s ≤ 300)) {
      leds[(NUM_LEDS / 2) - 1] = CRGB(50, 50, 150);
      leds[NUM_LEDS / 2] = CRGB(50, 50, 150);
    } else if ((s ≥ 250) && (s ≤ 275)) {
      leds[(NUM_LEDS / 2) - 1] = CRGB(230, 0, 10);
      leds[NUM_LEDS / 2] = CRGB(230, 0, 10);
    } else if ((s ≥ 235) && (s ≤ 250)) {
      leds[(NUM_LEDS / 2) - 1] = CRGB(0, 160, 0);
      leds[NUM_LEDS / 2] = CRGB(0, 160, 0);
    }
```

Aqui é onde estamos a dizer em que direção as cores se devem movimentar para que se consiga o efeito pretendido.

```
for (int i = 0; i ≤ ((NUM_LEDS / 2) - 2); i++) {
      leds[i] = leds[i + 1];
      leds[NUM_LEDS - 1 - i] = leds[(NUM_LEDS)-i - 2];
    }
```

PROBLEMAS ENCONTRADOS E RESPETIVAS SOLUÇÕES

O primeiro grande problema que encontramos foi no tipo de fitas led que iriamos usar para o projeto, pois nem todas são suportadas pelo arduino e muitas precisam de uma voltagem mais elevada do que aquilo que o arduino fornece. Após alguma pesquisa de modelos que funcionassem com as bibliotecas que iriamos utilizar, encontramos uma fita de led que apenas necessitava de 3 inputs, sendo eles, ground, data e 5V. Esta foi a melhor forma que podíamos ter encontrado para que o nosso objeto ficasse de acordo com aquilo que nós tínhamos visualizado.

Os botões do arduino causaram-nos alguns problemas de conectividade, ao início pensamos que fosse de os botões serem defeituosos, mas após algumas tentativas notamos que era a conectividade dos cabos que ligavam os botões ao arduino. A resolução acabou por ser muito simples, reduzimos o número de cabos conectados a cada botão utilizando-os em INPUT_PULLUP.

ARTEFACTO FÍSICO

MATERIAIS E PLANEAMENTO

No processo de montagem decidimos utilizar placas de MDF de 2mm para a caixa, com o material escolhido. Usámos o Adobe Illustrator para criar a caixa como nós queríamos, de modo a que esta tivesse os tamanhos adequados para não ser demasiado pequena ou demasiado grande, tomando também em conta o orçamento que tínhamos. Optámos por fazer uma caixa de 30x30. Por dentro, duas fitas de LEDs de 90 cm fazem uma espiral. Fizemos alguns compartimentos na caixa para manter o arduino e todos os componentes escondidos do visualizador para não perturbar a experiência. Além disso, fizemos três aberturas para colocar os botões de previous, play/pause e next, assim como uma abertura extra para o sensor de iluminação.
Tendo a caixa cortada, pintámos todas as placas com tinta acrílica preta para obter a estética pretendida.

MONTAGEM

Após serem adquiridos todos os materiais necessários para a montagem do artefacto, iniciámos a colagem das placas que viriam a compor a parte inferior da caixa, parte essa que incluiria todo o arduino. Dividimos a montagem em duas partes: inicialmente a parte inferior, previamente mencionada, e de seguida a parte superior, que incluiria os LEDs, assim como o espelho e a folha de plástico polipropileno transparente, acompanhada de uma película “one way mirror”.

Para a colagem das placas, apesar de apenas terem uma superfície de contacto de 2mm, utilizámos cola branca, especial para madeira, pois o peso total da caixa não justificava utilizar algo mais forte, que poderia vir a danificar as placas de MDF. De seguida, e agora com as quatro paredes erguidas, avançámos para a colocação do espelho, que devido a ter as dimensões exatas da caixa, não foi difícil de aplicar. Com isto feito, seguimos para a colagem dos LEDs, processo meticuloso que levou algum tempo a ser decidido pois a posição definitiva dos LEDs seria crucial para alcançar o objetivo pretendido.

Obtendo a posição desejada destes componentes, faltava-nos apenas montar a parte da frente, composta pela folha de plástico polipropileno transparente e pela película “one way mirror”, passo que mostraria ser mais complicado que o inicialmente esperado.

O corte da folha de plástico polipropileno não mostrou ser complexa, porque apenas com o auxilio de materiais básicos conseguimos obter as medidas desejadas, contudo a colagem da película "one way mirror" mostrou-se ser um dos maiores desafios de ser aplicada corretamente. Após diversas tentativas, chegámos a uma montagem que, apesar de não ter ficado perfeita, foi aquela que dentro das diversas iterações ficou mais apresentável. Com estes passos concluídos, bastou apenas certificarmo-nos que todos os componentes dentro da caixa estavam do nosso agrado, pois uma vez que a parte da frente fosse colada, não haveria uma maneira não destrutiva de aceder ao interior da mesma. Com isto feito, e agora com ambas as partes (inferior e superior) montadas, bastava apenas juntar as mesmas, então fizemos os últimos testes para ter a certeza que tudo estava montado e a funcionar conforme o esperado, e colámos ambas as partes.

PROBLEMAS ENCONTRADOS E RESPETIVAS SOLUÇÕES

Devido à relativa complexidade encontrada em todo o planeamento, criação e montagem deste projeto, seria inevitável encontrar diversos problemas ao longo do caminho. A primeira adversidade por nós encontrada começou logo por se mostrar nas dimensões das placas de MDF, pois como havia sido mencionado anteriormente, estas tinham uma grossura de apenas 2mm. Como seria de esperar, a colagem destas placas mostrou-se ser desafiante, devido à pequena área de contacto entre elas, e consequentemente pouca área de contacto entre a cola e as placas. Isto levou-nos a utilizar métodos criativos para encontrar soluções para estes problemas.

Como fomos colando as placas em partes, primeiro a estrutura inferior e só depois a superior, em cada uma levámos o nosso tempo para certificarmo-nos que tudo ficaria conforme esperado, arranjando formas estratégicas de posicionar objetos entre as placas para que estas ficassem perfeitamente perpendiculares entre si por mais de 20/30 minutos (tempo de secagem da cola) seguidos.

Outro problema também por nós encontrado foi a dificuldade da colagem de fita-cola e fita de dupla face na superfície pintada com acrílico, visto que esta resultava numa textura rugosa que dificultava a aderência das fitas. Para combater este problema, tentámos posicionar tudo de forma a criar a maior superfície de contacto possível entre a fita e a tinta, porém este viria a ser um problema que mostraria ser mais crítico que o inicialmente especulado, como foi visto na defesa presencial do artefacto.

Por último, outro problema que mostrou ser bastante evidente também foi a colagem da película “one way mirror” na folha de plástico polipropileno transparente. Fizemos no total três iterações desta colagem, decidindo posteriormente entre as três qual seria a mais adequada de utilizar. A primeira colagem foi a que nos colocou numa posição difícil pois mostrou ser muito desafiante de colocar. Após uma breve pesquisa sobre como colocar esta película corretamente, foram precisos três dos nossos membros trabalharem em simultâneo para tentar obter o melhor resultado possível, que mesmo assim mostrou oferecer as suas dificuldades. No entanto, acreditamos que a colagem que acabou por ser incluída na versão final do artefacto, ainda que não esteja aplicada perfeitamente, não afeta a componente visual do projeto.

IMAGENS

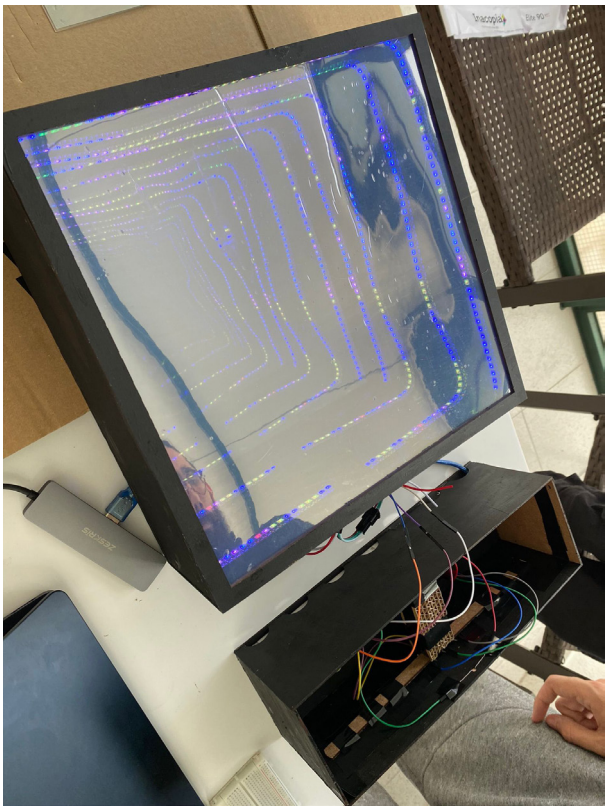

Fig.11. Caixa com as ligações visíveis

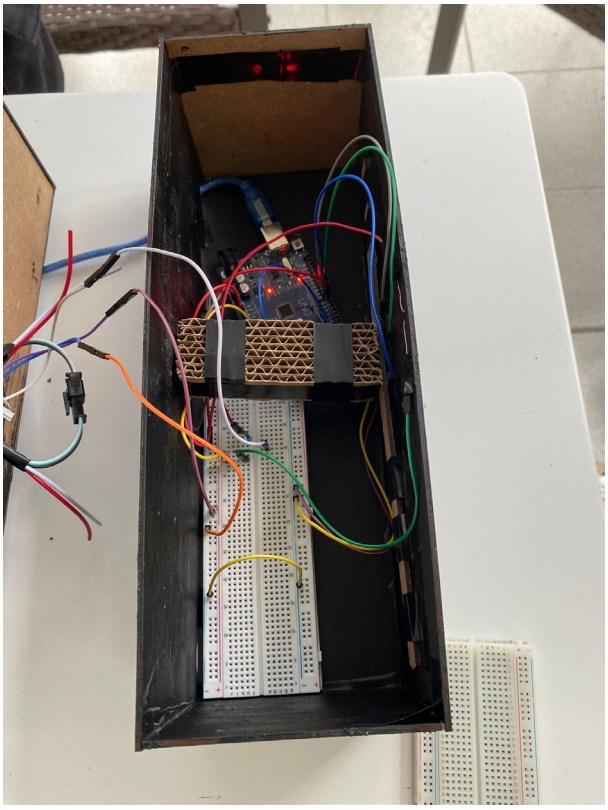

Fig.12. Circuito (close-up)

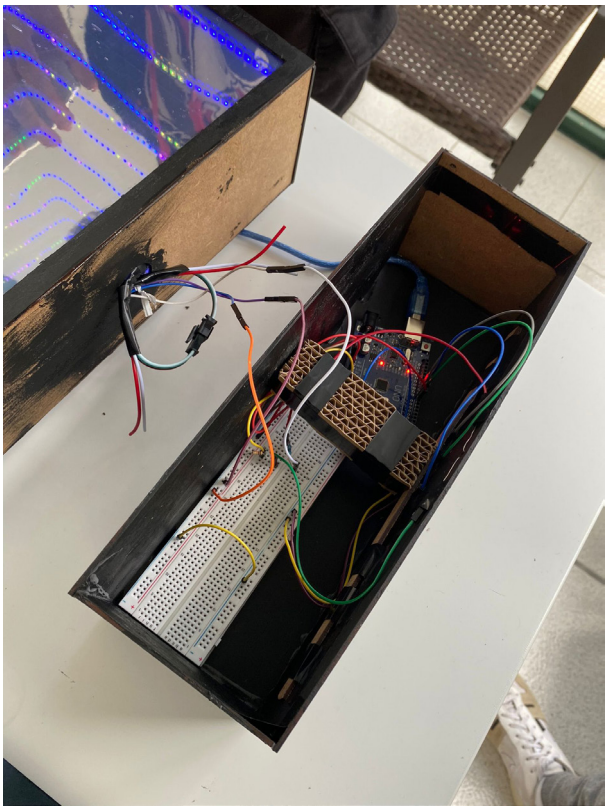

Fig.13. Circuito

Fig.14. Artefacto em funcionamento

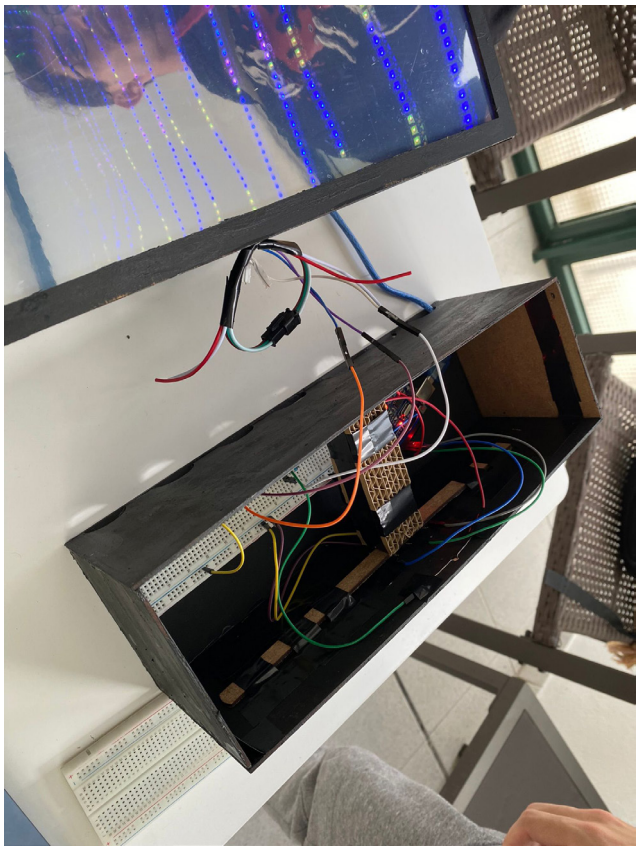

Fig.15. Artefacto em funcionamento

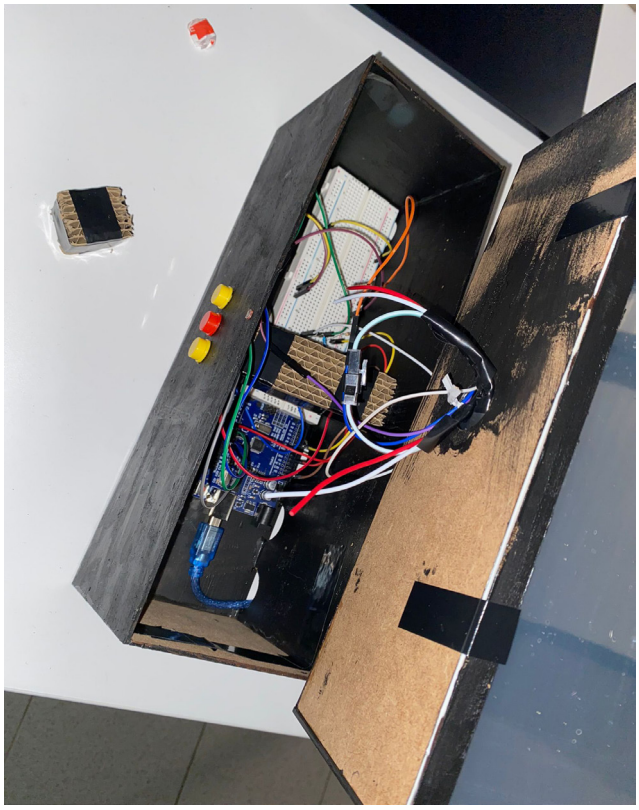

Fig.16. Circuito

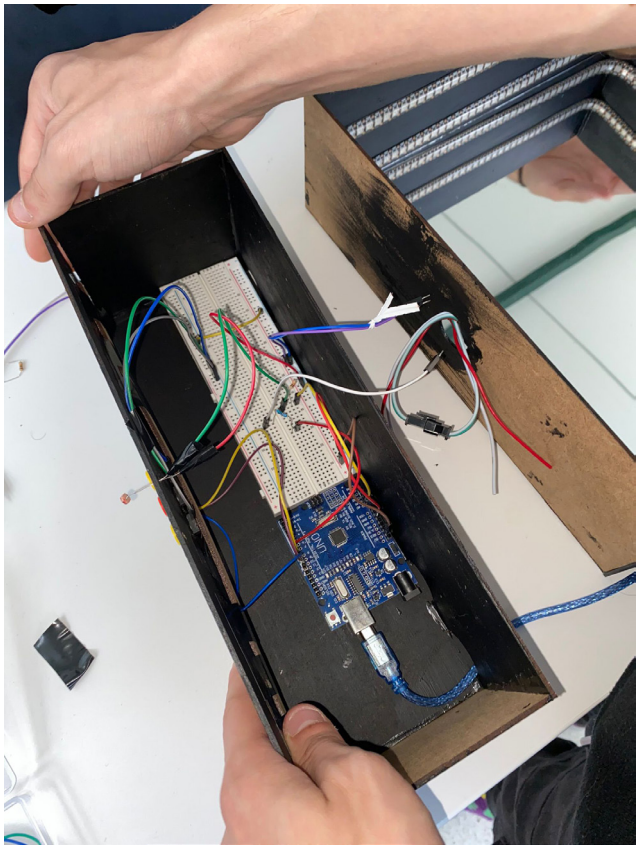

Fig.17. Circuito (não obstruído)

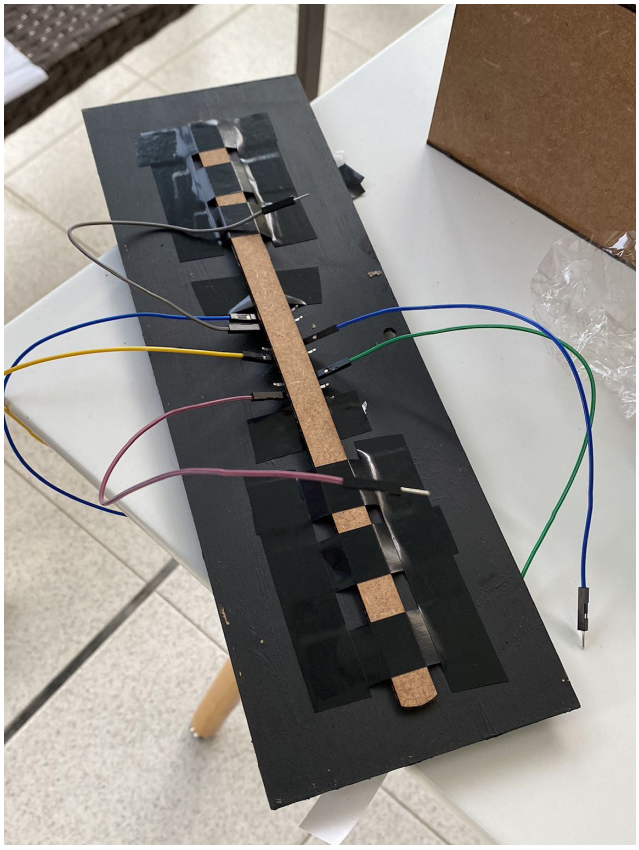

Fig.18. Ligação dos botões