EN SEUL MAJEUR

mise en scène et scénographie par
Chloé Thibaut Nguyen

avec la participation de
Judith Poillot
Marius Méchain
Nino Basset

dans le cadre des
EVENEMENTS SPECTACULAIRES
édition 2024

INTRODUCTION

Le chant d'un oiseau résonne dans le théâtre, celui du dernier Moho, petit passereau des îles d'Hawaii. Mélodie pour deux, parsemée de silences...

Emu, un, puis deux musiciens tentent de renouer le lien.

Le second effleure ses cordes muettes dans les silences du premier, dont les notes sonnent faux. Une pièce s'illumine. Puis deux. Puis une multitude. C'est un foyer miniature qui s'éclaire, les traces d'une vie qui se délivrent. À mesure que les cordes s'emballent, les intentions se mêlent, la mélodie est retrouvée. La bâtisse rayonne jusqu'à transpirer dans le réel.

L'espace d'un instant, ces deux présences se croisent et ravivent un souvenir. Fugace.

C'est un appel lancé au néant. Des notes adressées à un compagnon de vie ou de jeu, qui fut ou qui sera, peut-être jamais plus.

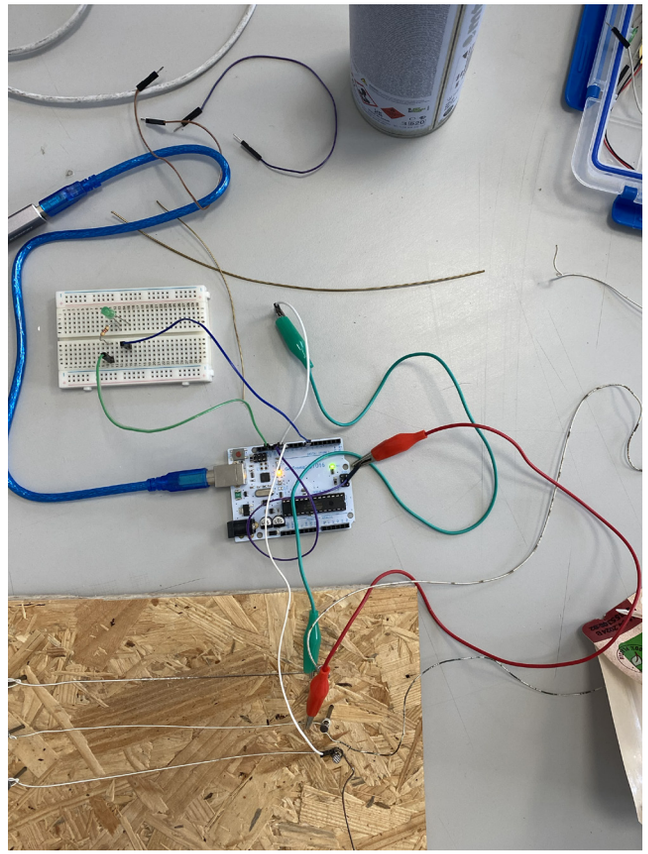
INTRODUCTION

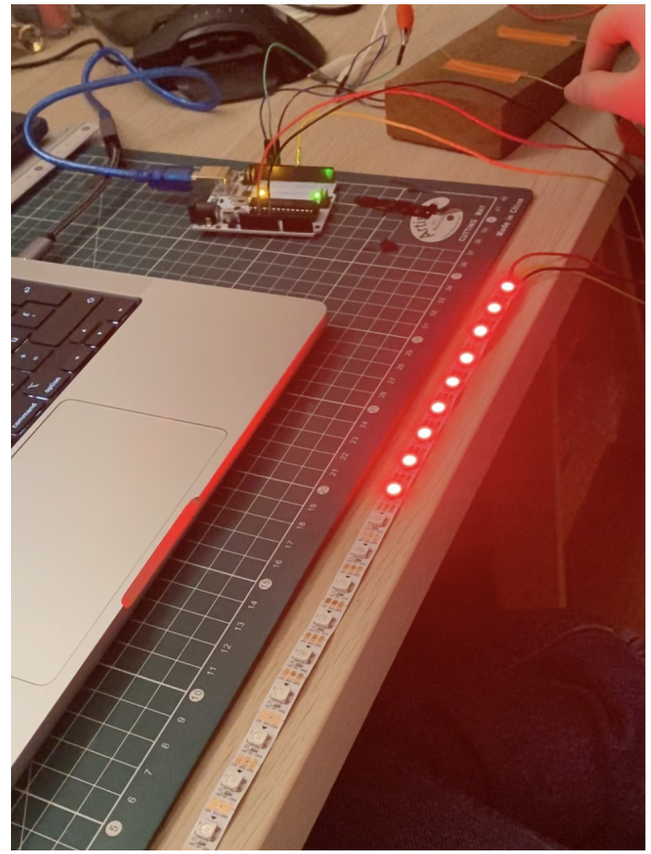
Lors du studio numérique artisanale, l'intention était d'utiliser des techniques de menuiserie pour construire une guitare puis l'augmenter à l'aide d'un système et d'un code pour en détourner l'usage.

Pour servir la poésie de ma performance, cette guitare, utilisée en miroir d'une autre "vraie" guitare, serait ratachée à une maison de poupée dont elle pourrait allumer les différentes pièces selon les cordes pincées.

Le principe fut de se reposer sur un capteur de toucher. En définissant chaque corde de ma guitare, en métal pour conduire le signal électrique, comme un capteur, je me retrouve à pouvoir donner la commande =
si corde touchée, alors (led allumée)
si corde non touchée, alors (led éteinte)

RECHERCHES ET TESTS





Premiers tests avec trois cordes, donc trois capteurs, reliées à une led puis à une bande led WS2812B (permettant de contrôler chaque led de la bande individuellement).

```
#include <SoftwareSerial.h>
#include "Arduino.h"
#include <Adafruit_NeoPixel.h>

int pinValues1;
int sensor1 = 12;
int pinValues2;
int sensor2 = 13;
int pinValues3;
int sensor3 = 11;

Adafruit_NeoPixel strip = Adafruit_NeoPixel(26, 7, NEO_GRB + NEO_KHZ800);

void setup() {
  pinMode(sensor1, INPUT);
  pinMode(sensor2, INPUT);
  pinMode(sensor3, INPUT);

  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
  Serial.begin(115200);
}

void loop() {
  pinValues1 = readCapacitivePin(sensor1);
  pinValues2 = readCapacitivePin(sensor2);
  pinValues3 = readCapacitivePin(sensor3);

  Serial.print("Sensor 1: ");
  Serial.print(pinValues1);
  Serial.print("\tSensor 2: ");
  Serial.print(pinValues2);
  Serial.print("\tSensor 3: ");
  Serial.println(pinValues3);

  // Control NeoPixel LEDs based on sensor readings
  if (pinValues1 > 2) {
    colorWipe(strip.Color(0, 0, 139), 0, 9); // Sensor 1 controls LEDs 0 to 8
  } else {
    colorWipe(strip.Color(0, 0, 0), 0, 9); // Turn off LEDs controlled by Sensor 1
  }
  if (pinValues2 > 2) {
    colorWipe(strip.Color(220, 198, 255), 9, 8); // Sensor 2 controls LEDs 9 to 16
  } else {
    colorWipe(strip.Color(0, 0, 0), 9, 8); // Turn off LEDs controlled by Sensor 2
  }
  if (pinValues3 > 2) {
    colorWipe(strip.Color(150, 204, 120), 17, 9); // Sensor 3 controls LEDs 17 to 25
  } else {
    colorWipe(strip.Color(0, 0, 0), 17, 9); // Turn off LEDs controlled by Sensor 3
  }
}

void colorWipe(uint32_t color, int startLED, int numLEDs) {
  for (int i = startLED; i < startLED + numLEDs; i++) {
    strip.setPixelColor(i, color);
  }
  strip.show();
}

uint8_t readCapacitivePin(int pinToMeasure) {
  volatile uint8_t* port;
  volatile uint8_t* ddr;
  volatile uint8_t* pin;

  byte bitmask;
  if ((pinToMeasure >= 0) && (pinToMeasure <= 7)) {
    port = &PORTD;
    ddr = &DDRD;
    bitmask = 1 << pinToMeasure;
    pin = &PIND;
  } else if ((pinToMeasure > 7) && (pinToMeasure <= 13)) {
    port = &PORTB;
    ddr = &DDRB;
    bitmask = 1 << (pinToMeasure - 8);
    pin = &PINB;
  } else if ((pinToMeasure > 13) && (pinToMeasure <= 19)) {
    port = &PORTC;
    ddr = &DDRC;
    bitmask = 1 << (pinToMeasure - 13);
    pin = &PINC;
  } else {
    return 0;
  }

  *port &= ~(bitmask);
  *ddr |= bitmask;
  delayMicroseconds(1);
  *ddr &= ~(bitmask);
  *port |= bitmask;
  int cycles = 17;
  for (int i = 0; i < 16; i++) {
    if (*pin & bitmask) {
      cycles = i;
      break;
    }
  }

  *port &= ~(bitmask);
  *ddr |= bitmask;

  return cycles;
}
```

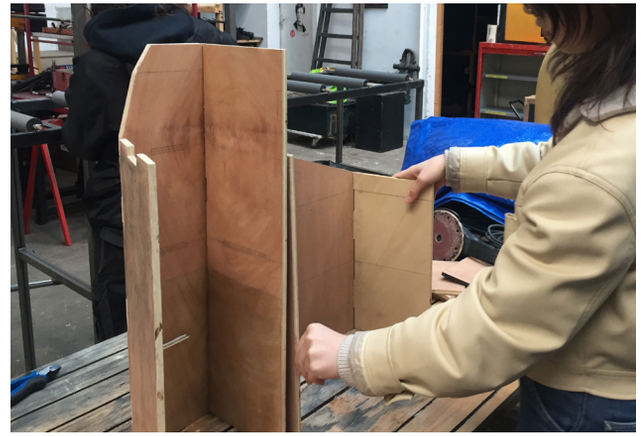
CONSTRUCTION - GUITARE





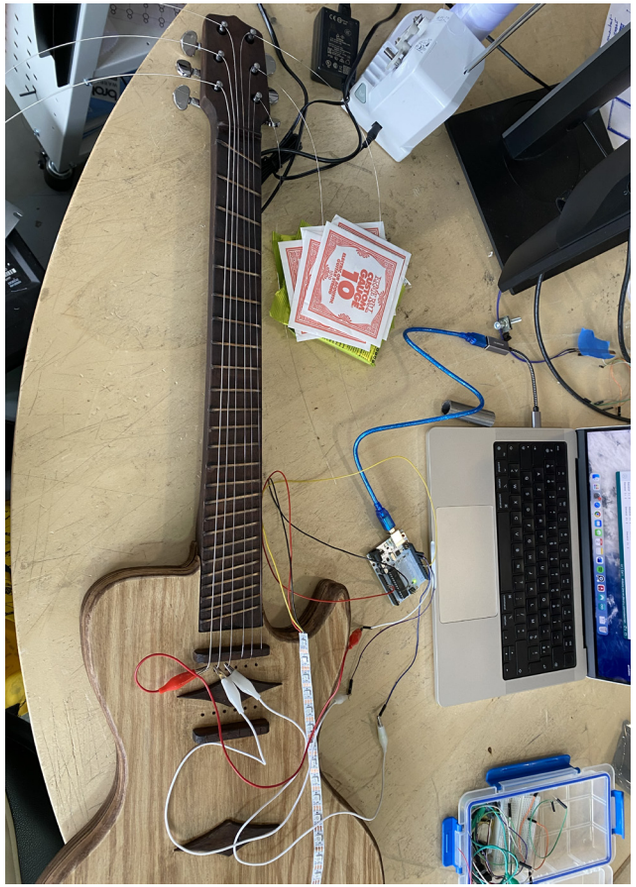
Contre-plaqué et tasseau récupérés au Théâtre de l'Aquarium.
Fraisage de la forme pour accueillir les connectiques. prise Jack et board arduino.

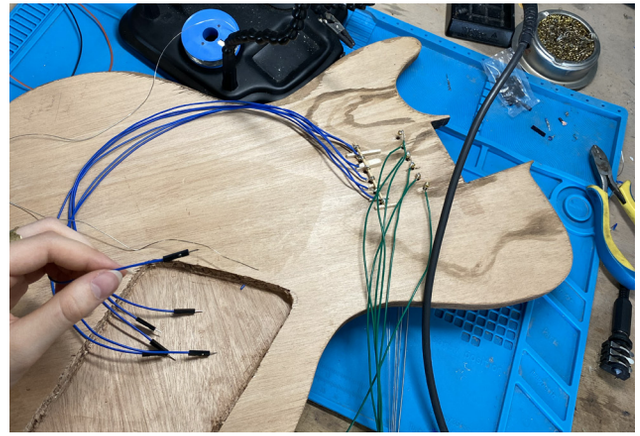
CONSTRUCTION - MAISON DE POUPÉE

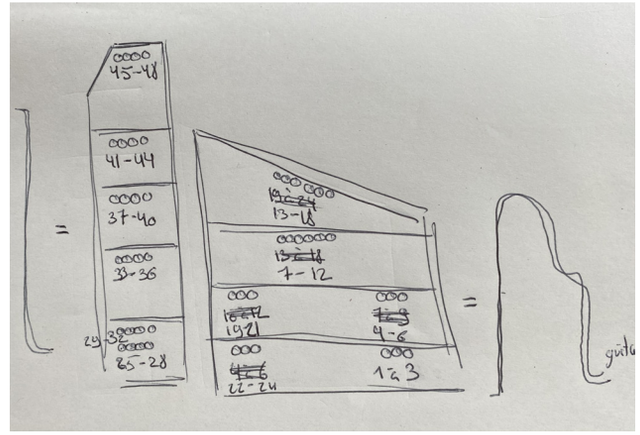


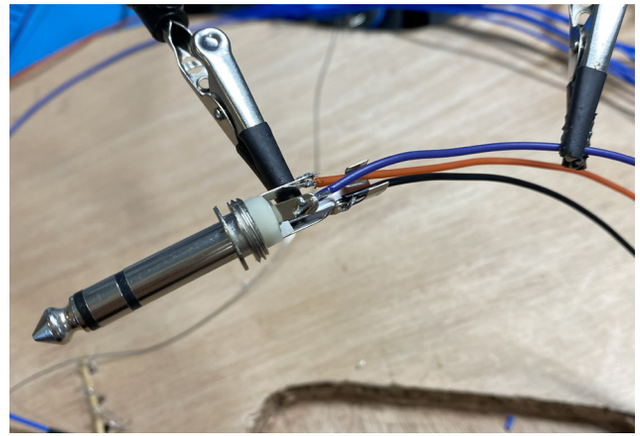
Encoches pour démonter aisément en vue de faire le passage de cable et le circuit plus tard.

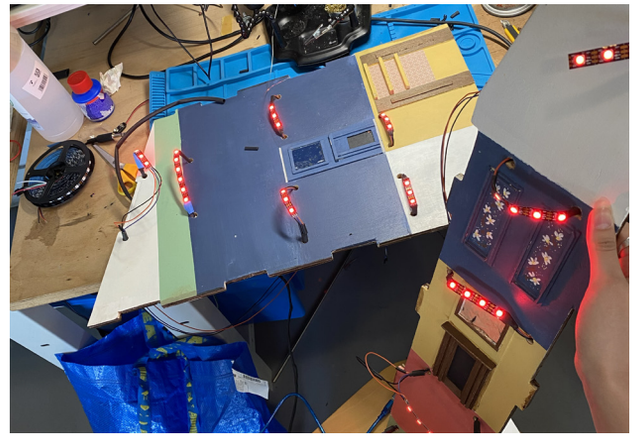
DERNIERS TESTS ET ASSEMBLAGE DU CIRCUIT











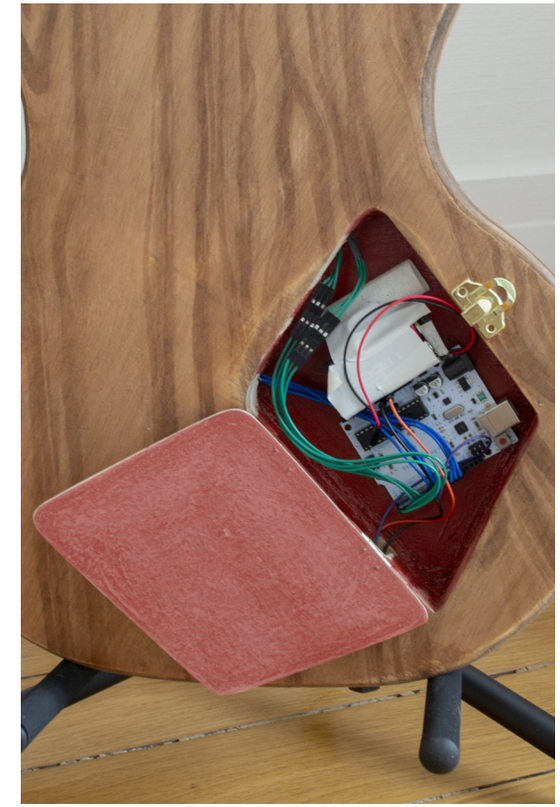
Test en duppliquant le code pour 6 cordes.
Soudure des cables et test de la bande led.

FINALISATION - CODE

```ino
#include <SoftwareSerial.h>
#include "Arduino.h"
#include <Adafruit_NeoPixel.h>

#define NUM_SENSORS 12
#define NUM_LEDS_TOTAL 47

int pinValues[NUM_SENSORS];
int sensorPins[NUM_SENSORS] = {1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13};

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LEDS_TOTAL, 7, NEO_GRB + NEO_KHZ800);

void setup() {
  for (int i = 0; i < NUM_SENSORS; i++) {
    pinMode(sensorPins[i], INPUT);
  }
  Serial.begin(115200);
  // Turn off all LEDs at the start
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}

void loop() {
  readSensorValues();
  printSensorValues();
  controlLEDs(); // Call the function to control LEDs based on sensor readings
  delay(100);
}

void readSensorValues() {
  for (int i = 0; i < NUM_SENSORS; i++) {
    pinValues[i] = readCapacitivePin(sensorPins[i]);
  }
}

void printSensorValues() {
  for (int i = 0; i < NUM_SENSORS; i++) {
    Serial.print("Sensor ");
    Serial.print(i + 1);
    Serial.print(": ");
    Serial.print(pinValues[i]);
    Serial.print("\t");
  }
  Serial.println();
}

// Function to control LEDs based on sensor readings
void controlLEDs() {
  for (int i = 0; i < NUM_SENSORS; i++) {
    if (pinValues[i] > 2) { // Assuming pin value greater than 2 indicates touch
      switch (i) {
        case 0:
          colorWipe(strip.Color(0, 0, 139), 0, 3); // Sensor 1 controls LEDs 0 to 2
          break;
        case 1:
          colorWipe(strip.Color(0, 128, 128), 3, 3); // Sensor 2 controls LEDs 3 to 5
          break;
        case 2:
          colorWipe(strip.Color(255, 105, 180), 6, 6); // Sensor 3 controls LEDs 6 to
          break;
        case 3:
          colorWipe(strip.Color(0, 100, 0), 12, 6); // Sensor 4 controls LEDs 12 to 17
          break;
        case 4:
          colorWipe(strip.Color(255, 165, 0), 18, 3); // Sensor 5 controls LEDs 18 to
          break;
        case 5:
          colorWipe(strip.Color(0, 0, 255), 21, 3); // Sensor 6 controls LEDs 21 to 23
          break;
        case 6:
          colorWipe(strip.Color(255, 0, 128), 24, 4); // Sensor 7 controls LEDs 24 to
          colorWipe(strip.Color(128, 0, 128), 0, 3); // Sensor 7 controls LEDs 0 to 2
          break;
        case 7:
          colorWipe(strip.Color(255, 255, 0), 28, 3); // Sensor 8 controls LEDs 28 to
          break;
        case 8:
          colorWipe(strip.Color(255, 69, 0), 31, 4); // Sensor 9 controls LEDs 31 to 3
          break;
        case 9:
          colorWipe(strip.Color(127, 255, 212), 35, 4); // Sensor 10 controls LEDs 35
          break;
        case 10:
          colorWipe(strip.Color(75, 0, 130), 39, 4); // Sensor 11 controls LEDs 39 to
          break;
        case 11:
          colorWipe(strip.Color(0, 0, 139), 43, 4); // Sensor 12 controls LEDs 43 to 4
          break;
      }
    } else {
      switch (i) {
        case 0:
          colorWipe(strip.Color(0, 0, 0), 0, 2); // Turn off LEDs controlled by Sensor
          break;
        case 1:
          colorWipe(strip.Color(0, 0, 0), 3, 3); // Turn off LEDs controlled by Sensor
          break;
        case 2:
          colorWipe(strip.Color(0, 0, 0), 6, 6); // Turn off LEDs controlled by Sensor
          break;
        case 3:
          colorWipe(strip.Color(0, 0, 0), 12, 6); // Turn off LEDs controlled by Senso
          break;
        case 4:
          colorWipe(strip.Color(0, 0, 0), 18, 3); // Turn off LEDs controlled by Senso
          break;
        case 5:
          colorWipe(strip.Color(0, 0, 0), 21, 3); // Turn off LEDs controlled by Senso
          break;
        case 6:
          colorWipe(strip.Color(0, 0, 0), 24, 4); // Turn off LEDs controlled by Senso
          colorWipe(strip.Color(0, 0, 0), 0, 3); // Turn off LEDs controlled by Sensor
          break;
        case 7:
          colorWipe(strip.Color(0, 0, 0), 28, 3); // Turn off LEDs controlled by Senso
          break;
        case 8:
          colorWipe(strip.Color(0, 0, 0), 31, 4); // Turn off LEDs controlled by Senso
          break;
        case 9:
          colorWipe(strip.Color(0, 0, 0), 35, 4); // Turn off LEDs controlled by Sen
          break;
        case 10:
          colorWipe(strip.Color(0, 0, 0), 39, 4); // Turn off LEDs controlled by Sen
          break;
        case 11:
          colorWipe(strip.Color(0, 0, 0), 43, 4); // Turn off LEDs controlled by Sen
      }
    }
  }
}

void colorWipe(uint32_t color, int startLED, int numLEDs) {
  for (int i = startLED; i < startLED + numLEDs; i++) {
    strip.setPixelColor(i, color);
  }
  strip.show();
}

uint8_t readCapacitivePin(int pinToMeasure) {
  volatile uint8_t* port;
  volatile uint8_t* ddr;
  volatile uint8_t* pin;

  byte bitmask;
  if ((pinToMeasure >= 0) && (pinToMeasure <= 7)) {
    port = &PORTD;
    ddr = &DDRD;
    bitmask = 1 << pinToMeasure;
    pin = &PIND;
  } else if ((pinToMeasure > 7) && (pinToMeasure <= 13)) {
    port = &PORTB;
    ddr = &DDRB;
    bitmask = 1 << (pinToMeasure - 8);
    pin = &PINB;
  } else if ((pinToMeasure > 13) && (pinToMeasure <= 19)) {
    port = &PORTC;
    ddr = &DDRC;
    bitmask = 1 << (pinToMeasure - 13);
    pin = &PINC;
  } else {
    return 0;
  }

  *port &= ~(bitmask);
  *ddr |= bitmask;
  delayMicroseconds(1);
  *ddr &= ~(bitmask);
  *port |= bitmask;
  int cycles = 17;
  for (int i = 0; i < 16; i++) {
    if (*pin & bitmask) {
      cycles = i;
      break;
    }
  }

  *port &= ~(bitmask);
  *ddr |= bitmask;

  return cycles;
}
```

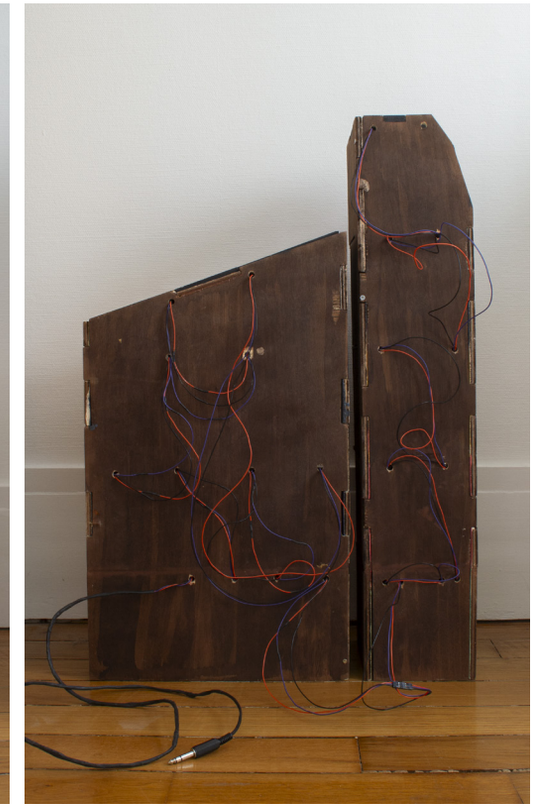
Utilisation d'array pour rendre la lecture du code plus claire.

FINALISATION - GUITARE



Si j'avais mieux réfléchis à la manière de souder les cables aux cordes, j'aurais sûrement pu les garder à l'extérieur du châssis et rendre le changement des cordes possibles sans démonter la guitare.
Faute de cela, une soudure pas très esthétique est survenue comme réparation d'urgence lors des 2 jours de performance.

FINALISATION - MAISON DE POUPÉE



Un branchement entre les systèmes de chaque maison permet au deux modules de rester distincts.
Une seule partie seulement peut-être branchée à la guitare.

PERFORMANCE

PERFORMANCE