

PROJET ARDUINO - PEIP 2

Année scolaire 2020-2021

TITRE : DJ Midi

Etudiants : BOUNEB Sami ; SORIN MACARACI

Encadrant : PASCAL MASSON

REMERCIEMENTS

Nous remercions Mr MASSON PASCAL pour nous avoir permis de faire un projet de cette envergure là.

Nous remercions également l'Université Nice Côte D'Azur de nous prêter le matériel nécessaire au bon déroulement du projet.

Nous remercions aussi les fournisseurs Atmel (pour la carte Arduino), de LED qui ont fabriqué du matériel de qualité.

Enfin, nous remercions nous-même pour avoir été fidèles à nos objectifs.

SOMMAIRE:

Introduction.....	page 4
Chapitre I: Le contrôleur MIDI	page 5
I.1. Qu'est-ce qu'un MIDI Controller?.....	page 5
I.2 Comment fonctionne le MIDI Controller?.....	page 5
I.3 Configurations MIDI.....	page 6
I.3.1. Partie tactile.....	page 6
I.3.2 Séquenceurs.....	page 7
Chapitre II : Matériaux utilisés.....	page 8
II.1. Quels matériaux pour notre projet ?.....	page 8
II.2. Le choix de la “Fast LED”.....	page 10
II.3. Schéma récapitulatif de notre projet	page 12
II.3.A. Commentaire	page 12
II.3.B. Schéma du projet.....	page 13
Chapitre III : La partie informatisée.....	page 15
III.1 MIDI <-> Arduino.....	page 15
III.1.1 Code Arduino.....	page 15
III.1.2 Hairless.....	page 19
Conclusion.....	page 20
Bibliographie.....	page 21

Introduction:

Nous vous présentons le DJ midi ! Une invention de Mr BOUNEB et Mr Macarici qui permettra de faire de la musique avec précision.

En effet ce projet est à base d'un support et du contrôleur MIDI avec Arduino. Qu'est qu'un contrôleur Midi ?

Le Musical Instrument Digital Interface ou **MIDI** est un protocole de communication et un format de fichier dédiés à la musique, et utilisés pour la communication entre instruments, électroniques, contrôleurs, séquenceurs, et logiciel de musique.

Le MIDI a été développé au début des années 80 pour normaliser la communication entre les matériels de musique informatisés.

Au fil des années, des représentants de tous les principaux fabricants ont travaillé ensemble pour créer, modifier et perfectionner le MIDI.

Des précisions seront apportées par la suite... (cf Chapitre 1)

Pour plus de détails sur notre DJ Midi, le rapport explique les différentes solutions techniques que nous avons trouvé durant nos 4 semaines.

Bonne lecture à Tous !



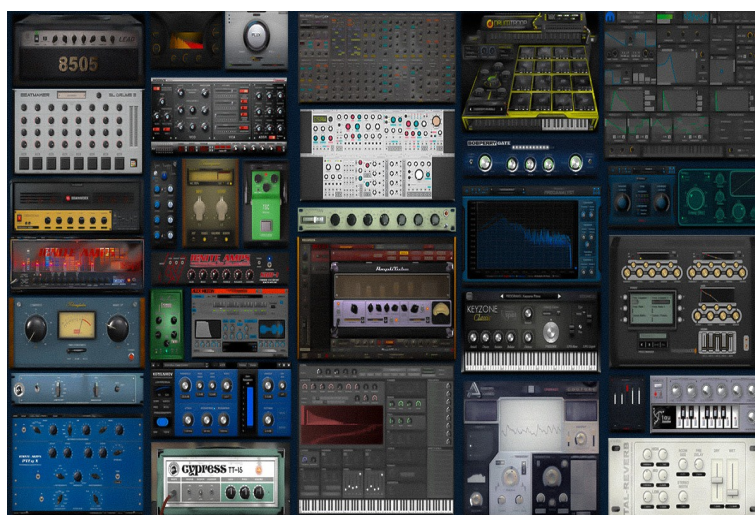
Chapitre I: Le contrôleur MIDI

I.1. Qu'est-ce qu'un MIDI Controller?

C'est un outil polyvalent qui te permet d'en faire énormément en studio.

Un contrôleur MIDI se transforme en n'importe quel instrument à partir des plugins VST. Les possibilités d'édition sont immenses et il transforme ton studio entier en une véritable usine musicale petite mais bien très efficace. Et ce n'est qu'une poignée de ses avantages.

Les plugins VST (Virtual Studio Technology) sont des outils dont les producteurs ont besoin pour composer et mixer de la musique dans un DAW. Un DAW ou Digital Audio Workstation est un périphérique ou un logiciel utilisé pour composer, produire, enregistrer, mixer et éditer de l'audio comme la musique, les paroles et les effets sonores.



I.2 - Comment fonctionne le MIDI Controller?

MIDI fait référence à un langage numérique partagé par les ordinateurs et certains instruments électroniques. Les ordinateurs lisent les données MIDI à l'aide de logiciels ou d'appareils, appelés séquenceurs, conçus pour lire les informations MIDI. Les données peuvent être composées sur l'ordinateur ou exécutées sur un instrument de musique MIDI, appelé contrôleur, puis envoyées au séquenceur.

Le MIDI a rendu possible l'utilisation d'instruments ou de systèmes de plusieurs fabricants, un langage partagé que tous les fabricants de musique électronique peuvent utiliser pour programmer leurs instruments et ordinateurs. Cela a permis aux musiciens de coordonner des instruments de différentes marques dans un seul système.

I.3. Configuration MIDI

I.3.1. Partie tactile

Lorsque on utilise un instrument MIDI, chaque fois qu'on appuie sur une touche, une note MIDI est créée (parfois appelée événement MIDI).

Chaque événement MIDI contient des instructions qui déterminent:

- La touche ON et OFF : Lorsque la touche est pressée/relâchée
- Les pitches ou notes jouées
- La vitesse : à quelle vitesse et à quel moment la touche est enfoncée
- L'aftertouch: la pression exercée sur la touche
- Le tempo (ou BPM)
- Le panning : réglage de la distribution panoramique dans l'espace stéréo ou multi canaux.
- Les modulations
- Le volume

MIDI transmet également des données d'horloge MIDI pour permettre une synchronisation parfaite entre 2 instruments et même plus.

Les données d'horloge MIDI sont dépendantes du tempo du périphérique principal qui est généralement un séquenceur. Ainsi, si on change notre tempo général, le MIDI veille à ce que toute la configuration reste synchronisée. C'est comme un petit leader numérique qui commande toutes les machines !

I.3.2. Séquenceurs

La configuration MIDI la plus courante est d'utiliser un séquenceur comme hub principal. Les séquenceurs sont utilisés pour enregistrer, éditer, envoyer et lire les données MIDI qui composent un projet. Ils peuvent être matériels comme une station de son, ou un ordinateur exécutant un séquenceur DAW ou un autre programme de séquençage.

Le séquenceur est l'axe principal de la composition. Il envoie des instructions à toutes les différentes parties de l'installation, enregistre les performances, et conserve la trace des arrangements globaux. C'est le langage MIDI qui rend tout ça possible.







Chapitre II : Les matériaux utilisés

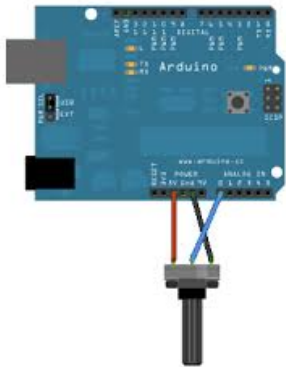
II.1. Quels matériaux pour notre projet ?

Le choix des matériaux est crucial pour notre DJ Midi.

Nous proposons de résumer les matériaux nécessaires dans un tableau.

<p>Carton</p> 	<p>En effet, le carton sera le support principal de notre projet. Il sera au-dessus de la plaque électrique et de la carte arduino.</p>
<p>Métal</p> 	<p>Le métal est une possibilité qui rendrait notre projet plus solide et plus professionnel.</p>
<p>Plastique</p> 	<p>Cette matière sera très utile pour fabriquer les boutons et également pour gérer la tension via le volume (autre bouton)</p>
<p>Joint</p> 	<p>Le joint rendra plus esthétique et solide le bouton qui génère le son.</p>

Potentiomètre



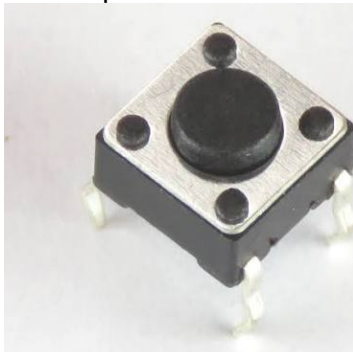
Le potentiomètre convertit les valeurs numériques en valeur analogique. Il permettra d'augmenter/diminuer le volume ainsi que la vitesse du son créer.

Cable (fil)



Ce qui assurera la communication entre la carte , multiplexer , fast led , bouton , ... La base de notre projet !

bouton poussoir


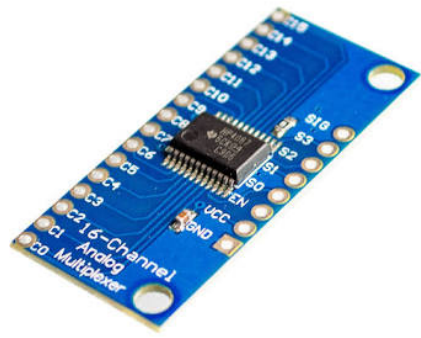


Simple mais efficace il sera en dessous du bouton utilisateur pour générer un son.

Résistance n°1 (330 ohm)



Résistance qui accompagneront les LED Obligatoire pour ne pas "cramer" la LED

<p>Résistance n°2 (2200 ohm)</p> 	<p>Résistance accompagnant le tensiomètre afin de réguler d'éventuels bruits parasites.</p>
<p>Multiplexer</p> 	<p>Composant électronique servant à gérer et à contrôler les Fast LED. Le principe : concentrer sur la carte arduino les informations arrivant des différentes Fast LED.</p>

II.2. Le choix de la “Fast LED”

Depuis peu on voit des nouveaux modèles sur le marché, comme la *SK6812*, l'*APA104* ou encore l'*APA106*. Tous ces modèles sont en réalité des variantes du *WS2812B*.

Tout d'abord la *SK6812*. A la différence des *WS2812B*, il en existe plusieurs versions :

1. RGB,
2. RGBCW (Blanc Froid)
3. RGBWW (Blanc Chaud)
4. RGBNW (Blanc Neutre)
5. RGB Mini (4 ou 5 mm)

Ces 2 dernières versions possèdent en plus du Rouge, Vert, Bleu traditionnel, la couleur Blanche en 3 versions (Blanc Froid comparable à du Blanc très Blanc, le Blanc Chaud à du Blanc un peu Jaune et le Blanc Neutre qui est entre les 2).

PS : Nous pensions, à tort, que la présence du blanc aurait permis une consommation moindre car avec des bandes traditionnelles RGB, pour faire du blanc, il faut que les 3 couleurs soit actives. Et bien suite aux dernières infos recueillis, chaque LED aurait besoin de 80 mA contre 60 mA pour un affichage de la bande en blanc donc faites attention d'avoir assez de puissance d'alimentation si vous la choisissez (ça fait une différence).

Sachez qu'il existe même des versions SK6812 Mini en 4 ou 5 mm mais disponibles qu'en version RGB. Elles feront très bien l'affaire pour ceux qui veulent quelque chose de discret. Le résultat est le même qu'avec des bandes de 10 ou 12 mm.

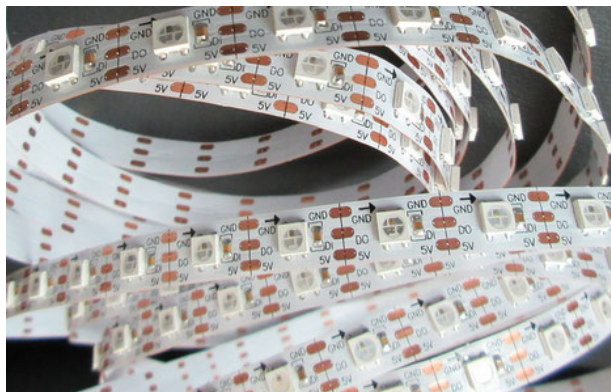
Elles sont à privilégier sur les *WS812B* pour toute installation en 3 Fil. Voici une vidéo montrant la différence entre le *RGB* et le *RGBW* :

<https://youtu.be/v8cNMAFmriI>

Ensuite vient l'*APA104*/*APA106*. Elles sont quasi-identiques à la *WS2812B*.

C'est en quelque sorte un clone de ce modèle (à quelques variantes près), elles sont moins chères à produire.

Image d'une bande d'*APA104* (on croirait voir une *WS2812B*):



II.3. Schéma récapitulatif de notre projet

A) Commentaire

Il s'agit d'une visualisation du DJ Midi. Comme vous pouvez le voir sur les schémas ci-dessous beaucoup de composants entrent en jeu pour la réalisation du DJ Midi.

1) La partie technique du DJ Midi

Tout d'abord, les Fast LED vont accompagner la musique créée par le logiciel. Il donne un rythme et une ambiance festive pour ne pas avoir que de la musique.

Egalement, le controller Midi Arduino est le chef d'orchestre du projet. C'est la pièce maîtresse qui à la fois reçoit, traite et donne les informations à l'ensemble des composants. Vous l'aurez compris elle est indispensable pour le DJ Midi

Enfin, le multiplexer ainsi que les boutons poussoirs permettront à l'utilisateur de mieux profiter du DJ Midi (volume , choix du son , ...)

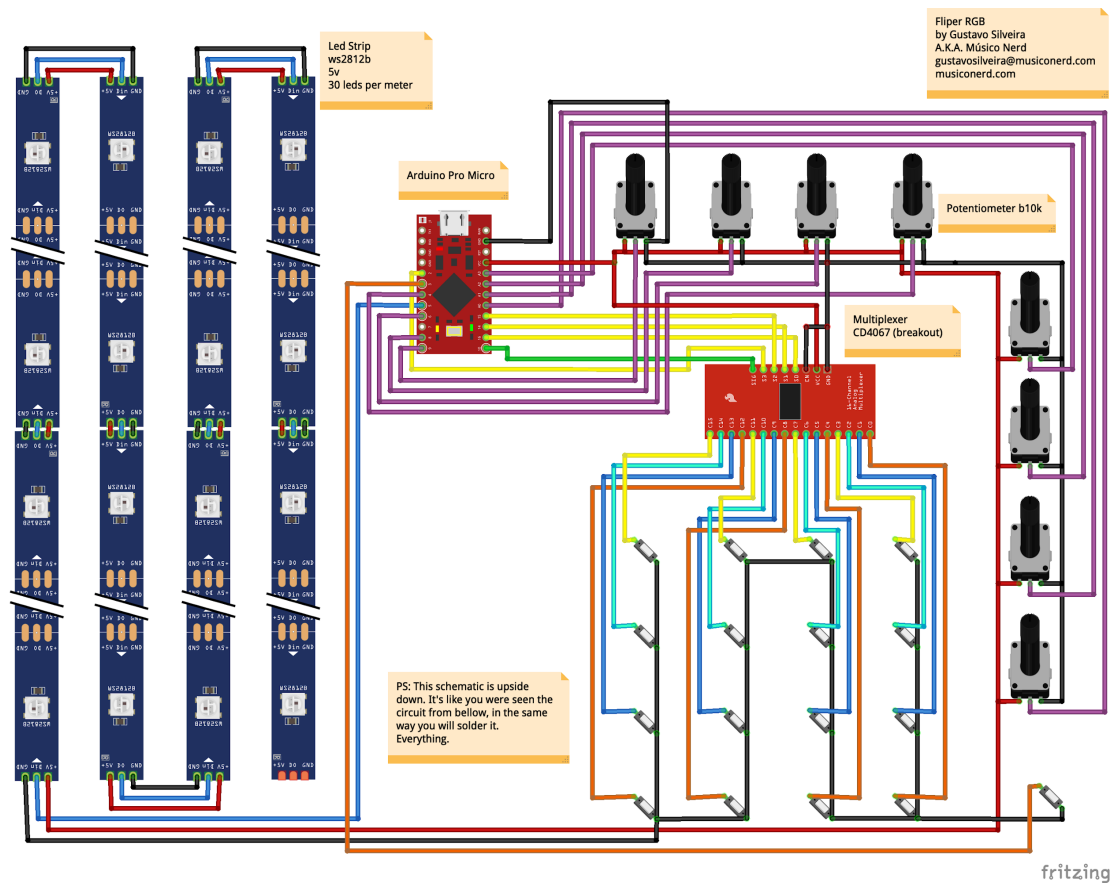
2) La partie utilisateur du DJ Midi

La partie utilisateur est très simple. un support cartonné ou métallique dans lequel on creuse des petits trous circulaires et 2 trous rectangulaires.

Ces trous nous permettent de mettre les boutons pour générer le son (trous circulaire) et de varier la rapidité et le volume du son (trou rectangulaire)

B) Schéma

Le dessous du DJ Midi



A 5x5 grid of circles. The top row has 5 circles, the second row has 5 circles, the third row has 5 circles, the fourth row has 5 circles, and the bottom row has 5 circles. A small circle is located in the center of the grid, between the second and third rows and between the third and fourth columns. The grid is enclosed in a rectangular border with rounded corners. The border consists of a thin line on the left and right sides, and a thicker line on the top and bottom sides. There are small circles at the corners of the border.

Chapitre III - La partie informatisé

III.1 - MIDI <-> Arduino

III.1.1 - Code Arduino

```
#include "FastLED.h"

FASTLED_USING_NAMESPACE
/* à définir
DATA_PIN /* entrée ou sortie
LED_TYPE /* type de LED
COLOR_ORDER /*l'ordre de couleur des LED
NUM_LEDS /*nombre de LED
HUE_OFFSET /*decalage de teinte
CRGB leds[NUM_LEDS];
byte ledIndex[NUM_LEDS] = {15, 8, 7, 0, 14, 9, 6, 1, 13, 10, 5, 2, 12, 11, 4, 3};/* les chaînes reliés à chaque Fastled

BRIGHTNESS /* brillance
FRAMES_PER_SECOND /*

byte ch1Hue = 135;
byte maxHue = 240;

include <MIDI.h>

using MUX
#include <Multiplexer4067.h>

include <Thread.h>
#include <ThreadController.h>

Nbmux 1 /* nombre de multiplexeurs
/* Définir s0, s1, s2, s3, et x en entrée/sortie
s0 /* (IO) numerique
s1 /* (IO) numerique
s2 /* (IO) numerique
s3 /* (IO) numerique
x1 /* (IO) analogique du premier multiplexeur, on definit x si on a besoin
Multiplexer4067 mux[N_MUX] = {
    Multiplexer4067(s0, s1, s2, s3, x1)
};

const int N_Boutons = 16; /* nombre total de boutons connectés à la carte Arduino + multiplexeur
const int N_Boutons_ARDUINO = 0; /* nombre de boutons connectés directement à la carte Arduino
const int Boutons_ARDUINO_PIN[N_BUTTONS] = {}; /* (IO) de chaque bouton connecté directement à la carte Arduino

const int CHANNEL_BUTTON_PIN = 3;
int BUTTON_CC[N_BUTTONS] = {12, 16}; /* ajouter le nombre de bouton poussoir

const int N_BUTTONS_PER_MUX[N_MUX] = {16}; /* nombre de boutons dans le multiplexeur
const int BUTTON_MUX_PIN[N_MUX][16] = { 0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15};/* (IO) de chaque bouton dans le multiplexeur dans l'ordre

int buttonMuxThreshold = 300;

int buttonCState[N_BUTTONS] = {0}; // stocke la valeur du bouton
int buttonPState[N_BUTTONS] = {0}; // stocke la valeur precedente du bouton

unsigned long lastDebounceTime[N_BUTTONS] = {0}; /* la derniere fois que la sortie de l'IO a été alternée
unsigned long debounceDelay = 5;

// velocity
byte velocity[N_BUTTONS] = {127};

// POTENTIOMETERS
const int N_POTS = 8; /* nombre total de potentiometre
const int N_POTS_ARDUINO = 8; /* nombre total de potentiometre connectés directement à Arduino
const int POT_ARDUINO_PIN[N_POTS_ARDUINO] = {A0, A1, A2, A3, A6, A7, A8, A9}; /* (IO) de chaque potentiometre

int POT_CC[N_POTS] = {1, 2, 3, 4, 5, 6, 7, 8}; /* ajoute le numero CC à chaque potentiometre

const int N_POTS_PER_MUX[N_MUX] = {0}; /* nombre de potentiometre dans chaque potentiometre dans l'ordre
const int POT_MUX_PIN[N_MUX][16] = {}; /* (IO) de chaque potentiometre dans le multiplexeur dans l'ordre qu'on veut

int potCState[N_POTS] = {0}; // l'etat actuel du potentiometre
int potPState[N_POTS] = {0}; // l'etat precedent du potentiometre
int potVar = 0; // Difference entre l'etat actuel et l'etat precedent

int potMidiCState[N_POTS] = {0}; // la valeur du MIDI à l'etat actuel
int potMidiPState[N_POTS] = {0}; // la valeur du MIDI à l'etat precedent

const int TIMEOUT = 300; /* durée pendant laquelle le potetnionetre sera lu après avoir dépassé le varThreshold
const int varThreshold = 10; /* Le seuil pour la variation du signal du potentiometre
boolean potMoving = true; // si le potentiometre est en marche
unsigned long PTime[N_POTS] = {0}; // le temps stocké précédemment
unsigned long timer[N_POTS] = {0}; // stocke le temps écoulé depuis que le temps a été reinitialisé

float filterA = 0.3;
float filterB = 0.7;

// CHANNEL
byte MIDI_CH = 0; /* La chaîne MIDI à utiliser
byte BUTTON_MIDI_CH = 0; /* La chaîne MIDI à utiliser
byte NOTE = 36; /* La note la plus basse à utiliser
byte CC = 1; /* Le CC le plus bas à utiliser
boolean channelMenuOn = false;
byte midiChMenuColor = 200;

// THREADS

ThreadController cpu;
Thread threadPotentiometers;
Thread threadChannelMenu;

void setup() {
    Serial.begin(115200);
    //FAST LED
    FastLED.addLeds<LED_TYPE, DATA_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalLEDStrip);

    // la brillance de la fastled
    FastLED.setBrightness(BRIGHTNESS);
    setAllLeds(ch1Hue, 30);

    FastLED.show();
    // Boutons
    /* Initialise les boutons avec la resistance tirage vers le haut (pull up)
    for (int i = 0; i < N_BUTTONS; i++) {
        pinMode(Boutons_ARDUINO_PIN[i], INPUT_PULLUP);
    }
}
```

```

FastLED.show();
// Boutons
/** Initialise les boutons avec la resistance tirage vers le haut (pull up)
for (int i = 0; i < N_BUTTONS_ARDUINO; i++) {
    pinMode(BUTTON_ARDUINO_PIN[i], INPUT_PULLUP);
}

pinMode(CHANNEL_BUTTON_PIN, INPUT_PULLUP);
// Potentiometere
// Initialise les potentiometres avec les resitances pull up
for (int i = 0; i < N_POTS_ARDUINO; i++) {
    pinMode(POT_ARDUINO_PIN[i], INPUT_PULLUP);
}

// Multiplexeurs
// Initialise les multiplexeurs
for (int i = 0; i < N_MUX; i++) {
    mux[i].begin();
}
/** definir chaque IO X comme
pinMode(x1, INPUT_PULLUP);

// Threads
// Potentiometers
threadPotentiometers.setInterval(15); // combien de millisecondes
threadPotentiometers.onRun(potentiometers); // la fonction va etre ajouter au thread
// Channel Menu
threadChannelMenu.setInterval(40); // combien de millisecondes
threadChannelMenu.onRun(channelMenu); // la fonction va etre ajouter au thread
cpu.add(&threadPotentiometers); // ajouter chaque thread ici
cpu.add(&threadChannelMenu); // ajouter chaque thread ici

}

void loop() {
    MIDIread();

    buttons();
    cpu.run(); // pour les threads

    FastLED.show();
    // insérer un délai pour maintenir la fréquence d'images
    FastLED.delay(1000 / FRAMES_PER_SECOND);
}

////////////////////////////////////////
// BOUTONS
void buttons() {
    // Lire les IO de la carte Arduino
    for (int i = 0; i < N_BUTTONS_ARDUINO; i++) {
        buttonCState[i] = digitalRead(BUTTON_ARDUINO_PIN[i]);
    }

    int nButtonsPerMuxSum = N_BUTTONS_ARDUINO; // decalage de buttonCState à chaque lecture du multiplexeur

    // Lit l'IO de chaque multiplexeur
    for (int j = 0; j < N_MUX; j++) {
        for (int i = 0; i < N_BUTTONS_PER_MUX[j]; i++) {
            buttonCState[i + nButtonsPerMuxSum] = mux[j].readChannel(BUTTON_MUX_PIN[j][i]);

            if (buttonCState[i + nButtonsPerMuxSum] > buttonMuxThreshold) {
                buttonCState[i + nButtonsPerMuxSum] = HIGH;
            }
            else {
                buttonCState[i + nButtonsPerMuxSum] = LOW;
            }
        }
        nButtonsPerMuxSum += N_BUTTONS_PER_MUX[j];
    }

    for (int i = 0; i < N_BUTTONS; i++) { // Lit les boutons connectés à Arduino
        if ((millis() - lastDebounceTime[i]) > debounceDelay) {

            if (buttonPState[i] != buttonCState[i]) {
                lastDebounceTime[i] = millis();

                if (buttonCState[i] == LOW) {

                    velocity[i] = 127; // si on presse le bouton la velocity est de 127
                }
                else {

                    velocity[i] = 0; // si le bouton est lâché la velocity est nulle
                }

                if (buttonCState[i] == LOW) { // envoie seulement quand le bouton est pressé
                    MIDI.sendControlChange(BUTTON_CC_N[i], velocity[i], BUTTON_MIDI_CH); // note, velocity, chaîne
                }
            }
        }
    }

    // POTENTIOMETRES
    void potentiometers() {

        // Lit les IO de la carte d'Arduino
        for (int i = 0; i < N_POTS_ARDUINO; i++) {
            potCState[i] = analogRead(POT_ARDUINO_PIN[i]);

            int reading = analogRead(POT_ARDUINO_PIN[i]);
            float filteredVal = filterA * reading + filterB * potPState[i]; // valeur du filtre
            potCState[i] = filteredVal;
        }
    }
}

```



```

int nPotsPerMuxSum = N_POTS_ARDUINO; // decalage de buttonCState à chaque lecture du multiplexeur

// Lit l'IO de chaque multiplexeur
for (int j = 0; j < N_MUX; j++) {
    for (int i = 0; i < N_POTS_PER_MUX[j]; i++) {
        potCState[i + nPotsPerMuxSum] = mux[j].readChannel(POT_MUX_PIN[j][i]);
    }
    nPotsPerMuxSum += N_POTS_PER_MUX[j];
}

for (int i = 0; i < N_POTS; i++) { // le circuit à travers tous les potentiometres

    potMidiCState[i] = map(potCState[i], 0, 1023, 0, 127); // la lecture du potCState à une valeur utilisable en MIDI

    potVar = abs(potCState[i] - potPState[i]); // calcule la valeur absolue entre l'etat actuel et l'etat precedent du potentiometre

    if (potVar > varThreshold) { // si la variation du potentiometre est supérieure au seuil
        PTime[i] = millis(); // stocke le temps precedent
    }

    timer[i] = millis() - PTime[i]; // reinitialiser le temps

    if (timer[i] < TIMEOUT) { //Si le temps est superieur au temps autorisé le potentiometre a le droit de marcher
        potMoving = true;
    }
    else {
        potMoving = false;
    }

    if (potMoving == true) { // si le potentiometre est toujours en mouvement , envoyez le CC
        if (potMidiPState[i] != potMidiCState[i]) {

            MIDI.sendControlChange(POT_CC_N[i], potMidiCState[i], MIDI_CH); // nombre CC, valeur CC, chaine midi
            potPState[i] = potCState[i]; // stocke la lecture actuelle du potentiometre pour comparer avec la suivante
            potMidiPState[i] = potMidiCState[i];
        }
    }
}

}

// Channel Menu
void channelMenu() {

    while (digitalRead(CHANNEL_BUTTON_PIN) == LOW) {

        setAllLeds(midiChannelMenuColor, 0); // allumes toutes les lumieres
        leds[ledIndex[BUTTON_MIDI_CH]].setHue(midiChannelMenuColor + 60); // allume une chaine specifique
        channelMenuOn = true;

        // Lire les IO de la carte Arduino
        for (int i = 0; i < N_BUTTONS_ARDUINO; i++) {
            buttonCState[i] = digitalRead(BUTTON_ARDUINO_PIN[i]);
        }

        int nButtonsPerMuxSum = N_BUTTONS_ARDUINO; // decalage de buttonCState à chaque lecture du multiplexeur

        // Lit l'IO de chaque multiplexeur
        for (int j = 0; j < N_MUX; j++) {
            for (int i = 0; i < N_BUTTONS_PER_MUX[j]; i++) {
                buttonCState[i + nButtonsPerMuxSum] = mux[j].readChannel(BUTTON_MUX_PIN[j][i]);

                if (buttonCState[i + nButtonsPerMuxSum] > buttonMuxThreshold) {
                    buttonCState[i + nButtonsPerMuxSum] = HIGH;
                }
                else {
                    buttonCState[i + nButtonsPerMuxSum] = LOW;
                }
            }
            nButtonsPerMuxSum += N_BUTTONS_PER_MUX[j];
        }

        for (int i = 0; i < N_BUTTONS; i++) { // lit les boutons connectés à la carte Arduino

            if ((millis() - lastDebounceTime[i]) > debounceDelay) {

                if (buttonPState[i] != buttonCState[i]) {
                    lastDebounceTime[i] = millis();

                    if (buttonCState[i] == LOW) {

                        BUTTON_MIDI_CH = i;
                        //Serial.print("Channel ");
                        //Serial.println(BUTTON_MIDI_CH);

                    }
                    buttonPState[i] = buttonCState[i];
                }
            }
        }
        FastLED.show();

        FastLED.delay(1000 / FRAMES_PER_SECOND);
    }

    if (channelMenuOn == true) {
        setAllLeds(ch1Hue, 30);
        //Serial.println("menu lights off");
    }
    channelMenuOn = false;

    FastLED.show();

    FastLED.delay(1000 / FRAMES_PER_SECOND);
}

void setAllLeds(byte hue_, byte randomness_) {

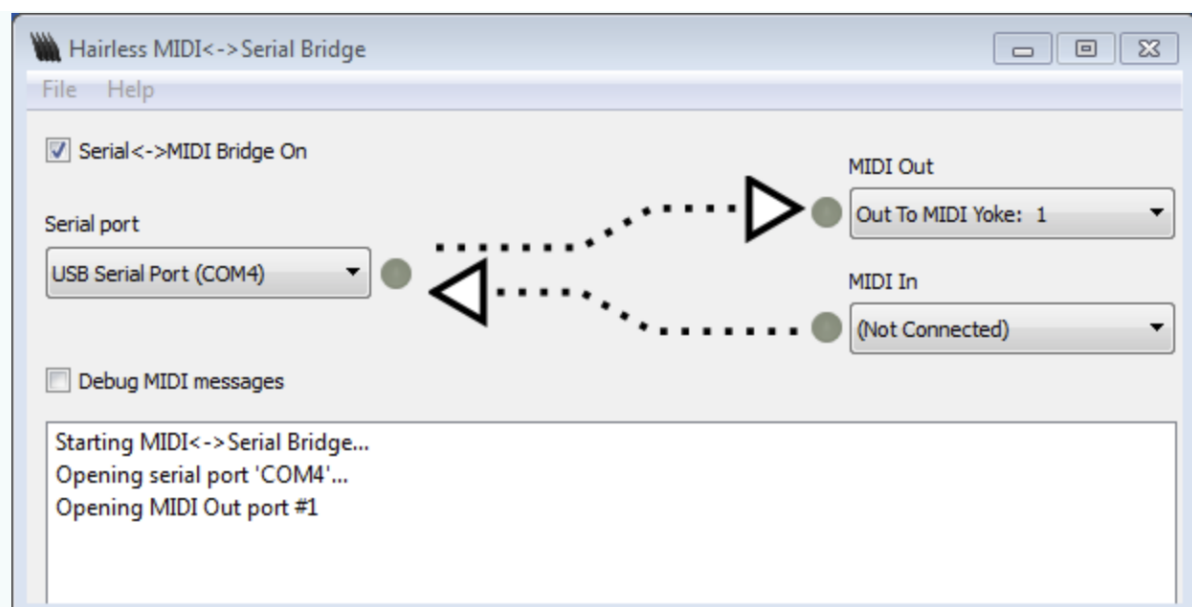
    for (int i = 0; i < NUM_LEDS; i++) {
        byte offset = random(0, randomness_);
        leds[i].setHue(hue_ + offset);
    }
}

```

Le message "Control Change" permet en effet de modifier les paramètres d'un appareil MIDI comme le volume, le panoramique, le taux de réverbération et d'autres détaillés dans ce dossier. On dispose potentiellement de 128 contrôleurs possibles (0 à 127) (certains ne sont pas définis par la norme).

III.2.2 - Hairless

On utilisera Hairless MIDI pour connecter un port de série de notre périphérique (comme Arduino) qui permet d'envoyer et de recevoir des signaux MIDI et ensuite. Grâce à Hairless on va pouvoir gérer le volume ou utiliser un instrument de musique avec nos potentiomètres et boutons



Conclusion:

Ce projet nous a permis d'élargir notre culture musicale notamment et d'acquérir des compétences techniques et d'organisation dans le binôme.

Le rapport bibliographique est seulement une partie de notre projet car il peut y avoir des imprévus ou des soucis techniques auxquels nous n'aurions pas pensé.

De plus, les difficultés rencontrées sont diverses : la coordination du binôme, les recherches sur un sujet inconnu, la complexité du projet,...

Enfin, nous avons beaucoup appris de ce projet et en particulier la diversité qu'offre Arduino en termes de fonctionnalité notamment avec le Midi controller qui permet plusieurs possibilités de projet.

Bibliographie

<https://hackaday.io/project/164546-arduino-midi-controller-diy/details>

<https://study.com/academy/lesson/what-is-midi-controller-interface-songs.html>

<https://blog.landr.com/fr/quest-ce-que-le-midi%E2%80%AF-guide-pour-debutant-sur-loutil-le-plus-puissant-de-la-musique/>

<http://lesporteslogiques.net/wiki/ressource/electronique/bibliotheque> fastled

<https://ambimod.jimdofree.com/2017/01/19/bien-choisir-sa-bande-de-leds-pour-faire-de-l-ambilight/>

<https://www.youtube.com/watch?v=98rNZtrdpBM>

<https://fr.audiofanzine.com/mao/editorial/dossiers/le-midi-les-midi-control-change.html>