Compte rendu séance 7 :

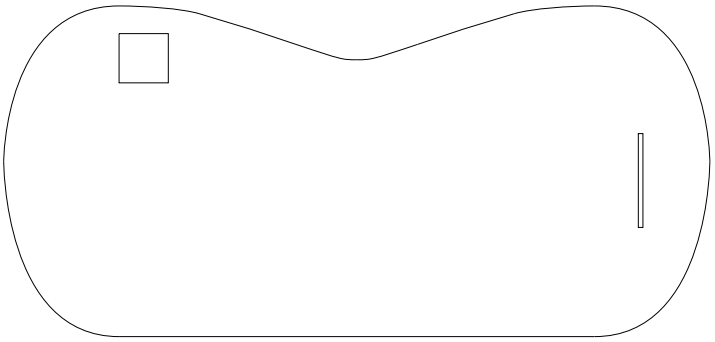
Pour cette séance, j'étais chargé de créer un fichier *.svg* pour que l'on puisse, suite à l'utilisation de la découpe laser au FabLab (site des Templiers), obtenir le dessus du boîtier pour le *Variophuino*.

Ce n'est pas une image classique (telle qu'un *.png* ou *.jpeg*) généralement codée sous forme matricielle, c'est-à-dire que l'image est marquée et coloriée pixel par pixel (il y a donc des pertes de qualité après un rétrécissement puis ré-agrandissement de l'image).

SVG signifie *Scalable Vector Graphic*. Le SVG stocke l'image sous **forme vectorielle** : l'image est explicitée par un texte structuré qui décrit de quoi est composée l'image (points, segments, cercles, couleurs...). Le fichier énonce comment sont organisés les éléments entre eux dans un format XML.

[*https://admaker.fr/blog/svg-vs-png-quel-format-image-seo/*]

Voici ci-contre l'image (au format *.png* ici car suffisant) envoyée à la découpeuse laser :



Les futures ajustements appliqués à cette planche de bois de 5 millimètres d'épaisseur avec la forme ci-dessus concerne beaucoup plus mon binôme Léna que moi… donc passons à la suite !

On connaît maintenant l'usage définitif des 7 leds.
6 d'entre elles gardent le même usage que précédemment (indiquer à l'utilisateur du *Variophuino* « ce qu'il se passe » si je puis dire sans trop détailler, par exemple si le *Variophuino* est allumé ou non), cependant la led restante change de fonctionnalité :

code_fred_testons | aled.h | fonctions_basiques.h | general_play.h | melodies.h | notes.h | other_left_piano.h | principal_right_piano.h

```
/*****************************************************************************
 *                  On retrouve ici tout ce qui gère
 *                le système de LED (écran compris).
*****************************************************************************/


const int where_led_on_off = 13;    //pas besoin de coder cet led
```

car directement branchée au 5V

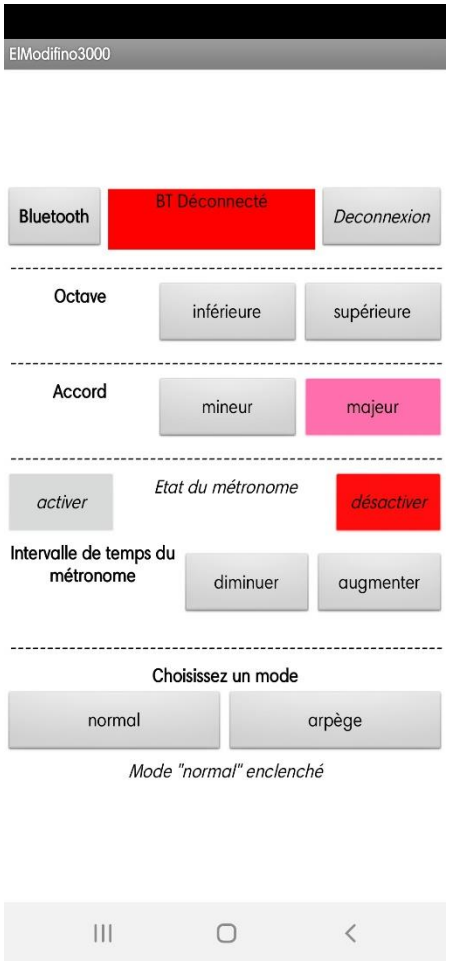
```
const int where_led_B_b = 12;
const int where_led_G_b = 11;
const int where_led_R_b = 10;
const int where_led_B_m = 9;
const int where_led_G_m = 8;
const int where_led_R_m = 7;
const int where_led_bend = 6;
const int where_led_tempo = 5;
const int where_led_recorded = 4;
const int where_led_for_bluetooth = 3;
```

la fameuse 7ème led restante, anciennement nommée *where_led_rhythm*

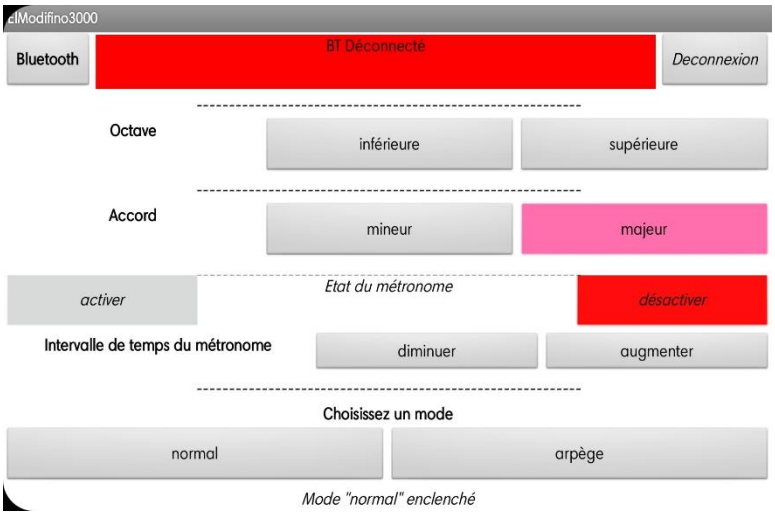
Cette led sera allumé si et seulement si un appareil est connecté via bluetooth au *Variophuino*.

Parlons maintenant de l'application installée sur un téléphone (le mien tout d'abord, celui de Léna pourra aussi faire l'affaire mais pas tout de suite) pour permettre la communication d'informations via bluetooth avec le *Variophuino*.

Voici à quoi cela ressemble une fois lancée sur mon téléphone à la verticale :



Voici ci-dessous ce que ça donne avec le téléphone à l'horizontal (ce n'est pas encore « joli » pour moi, l'utilisation à l'horizontale est un bonus) :



Le fait d'appuyer sur un bouton via cette application permet d'envoyer un caractère à la carte bluetooth du *Variophuino*, je rappelle (si je ne

l'avais pas déjà dit) que c'est la procédure *commands()* se trouvant dans le fichier *fonctions_basiques.h* :

```
void commands() {
  if (Serial.available() || Serial1.available()) {
    int ancien_mode = mode;
    if (Serial1.available()) {charblu = Serial1.read();}
    else if (Serial.available()) {charblu = Serial.read();}
    switch (charblu) {
      case 'S':
        print_current_set();
        break;
      case 'V':
        ...
      ...
    }
  }
  checking_tempo();
};
```

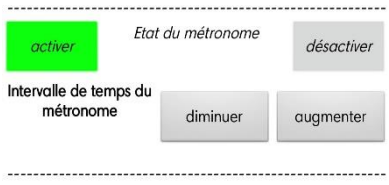
On a défini un *Serial1* au cas où la carte Arduino recevrait des informations liées aux pins 14 et 15, soit *RX1* et *TX1* (pas forcément dans cet ordre d'ailleurs) mais il ne servira pas au final, le laisser ne change donc rien au code.

L'entier *ancien_mode* permet de savoir s'il y a eu un changement de mode suite au caractère *charblu* envoyé depuis un des boutons de l'application sur téléphone.

La procédure *checking_tempo()* permet de voir s'il est temps de changer l'état de la led qui sert de métronome. Pour éviter de couper le programme juste pour une led, on ne se sert pas de la procédure *delay()* mais de la fonction *millis()* qui renvoie le nombre de millisecondes écoulées depuis l'allumage de la carte Arduino :

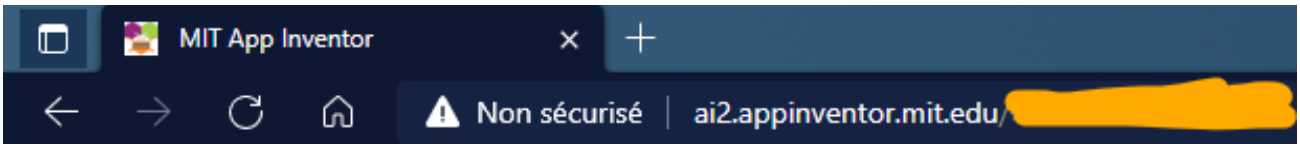
```
void checking_tempo() {
  if (metronome) {
    current_time = millis();
    if (current_time - last_time >= metrovalle) {
      led_tempo_state = !led_tempo_state;
      digitalWrite(where_led_tempo, led_tempo_state);
      last_time = current_time;
    }
  }
};
```

La variable booléenne *metronome* est activé par le bouton « *activer* » sur l'application, bouton qui une fois appuyé est d'ailleurs vert :



Je pense avoir le juste milieu entre pas assez de couleurs et trop de couleurs (du moins dans ma tête, tout n'est pas encore définitif).

Je précise aussi un détail important: je code cette application mobile depuis un site du MIT (non sécurisé malheureusement…).



On jongle entre deux sous-interfaces: une pour l’affichage finale et une autre pour coder tout cela (qui fait d’ailleurs beaucoup penser au logiciel de codage *Scratch*… inutile de le présenter).