Compte rendu séance 6 :

De son côté mon binôme Léna a fait beaucoup de soudures, on a donc passé pas mal de temps durant la séance à vérifier que tout remarchait bien, comme avant les soudures.

Il y a donc eu des mises en évidence d'erreurs dans le code, comme celle-ci-dessous :

```
void actual_frequencies_update() {
  an0 = analogRead(where_bend);
  if (mode == 1) {
    for (int i = 0; i < 20; i++) {
      special_actual_r_frequencies[i] = special_r_set[i] + constrain(map(an0, 0, 1023, 0,
special_r_gaps[i]), 0, special_r_gaps[i]);
    }
  }
  else {
    for (int i = 0; i < 13; i++) {
      actual_frequencies[i] = the_set[i] + constrain(map(an0, 0, 1023, 0,
pitchbend_variations_set[i]), 0, pitchbend_variations_set[i]);
    }
  }
};
```

Ce qu'il y a dans le *else{}* est le code présent avant d'intégrer le module bluetooth et les différents modes (ici toujours que 2 modes différents: *normal* et *arpège*).

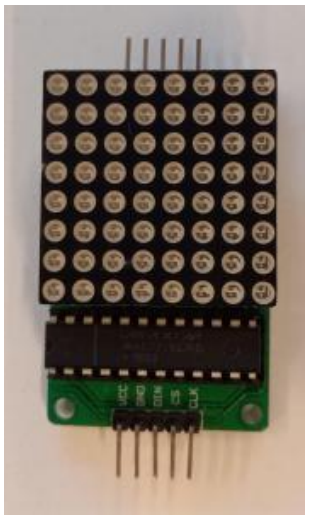
Nommons x_k la fréquence envoyée à l'un des deux buzzers (changeant suivant le mode) avec k la valeur de la variable *mode* dans le code (*0* -> *normal* ; *1* -> *arpège*).
Nommons y_k la fréquence de base et z_k celle dépendant du potentiomètre responsable de l'effet *pitchbend*.
On a donc: $x_0 = y_0 + z_0$ et $x_1 = y_1 + z_1$

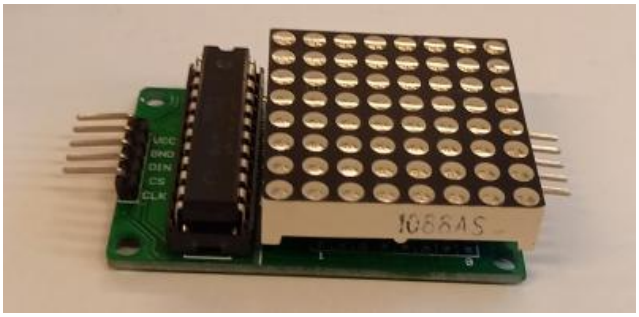
Il se trouve que l'erreur était une erreur de codage pour x_0.
Au lieu d'avoir « $x_0 = y_0 + z_0$ », on avait « $x_0 = y_0 + z_1$ ».
mode étant de base à 0, x_1 n'était donc pas encore défini, de même pour y_1 et surtout z_1, donc z_1 vaut par défaut 0.

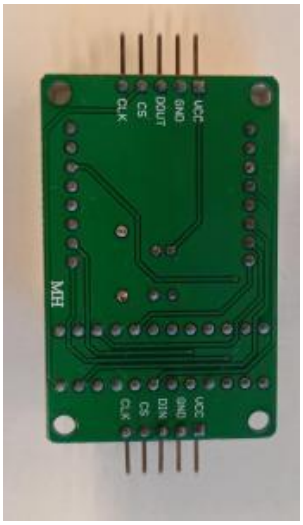
Avant de me rendre compte du problème, Léna et moi avions cru à un soudain non-fonctionnement du potentiomètre, enfin de l'effet *pitchbend* en général.

Il y a eu d'autres erreurs toutes bêtes de ce type, toutes étant réglées maintenant.

Aussi, durant cette séance, j'ai commencé à me familiariser avec un écran de 8 leds de largeur et de 8 leds de hauteur (cela donc de « gros » pixels si je puis dire). Voici à quoi il ressemble:







Je n'ai pas trouvé de bibliothèque simple regroupant toutes les fonctionnalités telles que celles des pins *DIN*, *CS* et *CLK*. Je vais donc pour l'instant fortement m'inspirer d'un programme trouvé en ligne, en espérant tout comprendre au final.

Voici la variable que j'ai créée qui servira à afficher le chiffre 2:

```
int two[8] = {0x00, 0x00, 0x18, 0x24, 0x08, 0x10, 0x3C, 0x00};
```

Chaque valeur de la liste est un nombre en base 16 (hexadécimal). L'ordinateur sait que c'est de l'hexadécimal grâce au *0x* devant le nombre, qui est d'ailleurs codé sur un octet (8 bits). Voici un exemple ci-dessous avec la 4ième valeur de la liste *two[]* ci-avant:

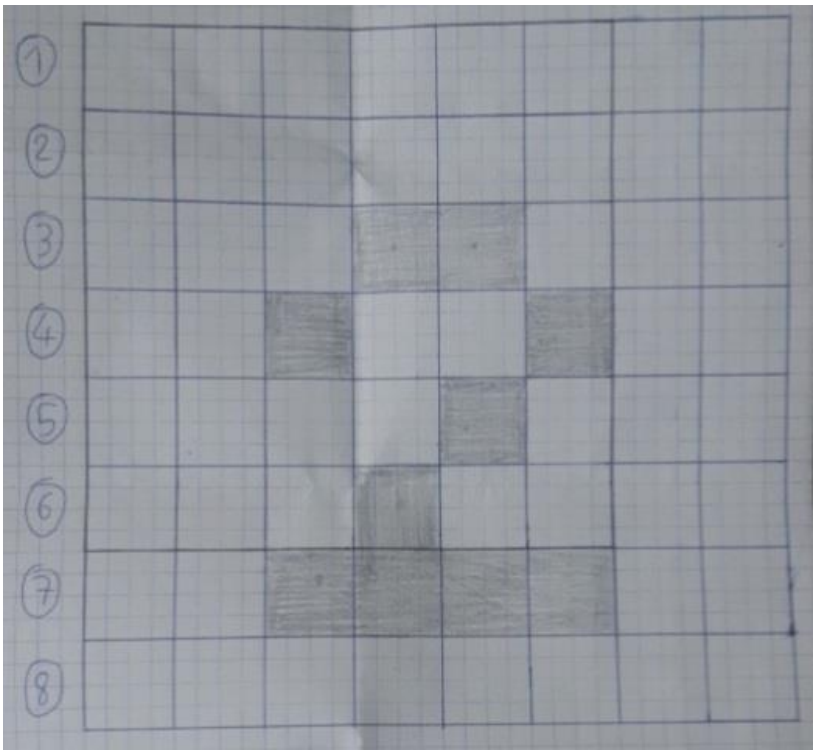
« 0x 24 » ----> « 0 00100100 »

encodage arduino

encodage arduino

Chaque bit de l'octet indique à une led de la ligne (ici la 4ième ligne) de l'écran 64 leds si elle doit s'allumer ou non (*1* -> *allumée* ; *0* -> *éteinte*)

Voici ce que cela rendrait une fois l'écran allumée, sachant que la led serait allumée si et seulement si la case du schéma est grisée:



Pour la fois prochaine, je vais voir comment coder d’autres effets applicables via bluetooth, sachant que le mode *arpège* m’a pris pas mal de temps…

On notera que, le volume du son n’étant pas « codé » (c’est-à-dire que le potentiomètre modifie directement la valeur de la tension délivrée au buzzer associé), je ne pourrais pas faire d’effets bluetooth modifiant le volume.