Compte rendu séance 2 :

J'ai réorganisé les deux fichiers *pitches.h* et *notes.h* :

buzzer_test | notes.h | pitches.h

```
/*************************************************
 * Absolument toutes les notes d'un piano classique !
 * (le tri est fait dans notes.h)
*************************************************/



#define NOTE_AS0 27
#define NOTE_AS0 29
#define NOTE_B0  31


/*-------------------------------------------------------*/

/*    début octave n°1    */
#define NOTE_C1  33   //première note jouable -> Do0
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
/*     fin octave n°1     */

/*    début octave n°2    */
#define NOTE_C2  65    //Do1
#define NOTE CS2 69
```

```
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
/*     fin octave n°7     */

#define NOTE_C8  4186   //dernière note jouable -> Do7

/*-------------------------------------------------------*/

#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

buzzer_test | notes.h | pitches.h

```
#include "pitches.h"

int notes[85] = {
  NOTE_C1,
  NOTE_CS1,
  NOTE_D1,
```

```
  NOTE_AS7,
  NOTE_B7,
  NOTE_C8
};
```

On a convenu avec mon binôme Léna que nous n'utiliserons toutes les octaves que si nous en avions le temps, nous nous contenterons des octaves numéros 2, 3 et 4.

Je n'ai pas encore testé ce répertoriage de données sur la carte Arduino Uno, on en est encore à la phase théorique où on (enfin Léna) contrôle quand le buzzer doit émettre un son, et donc moins axé sur quel son.

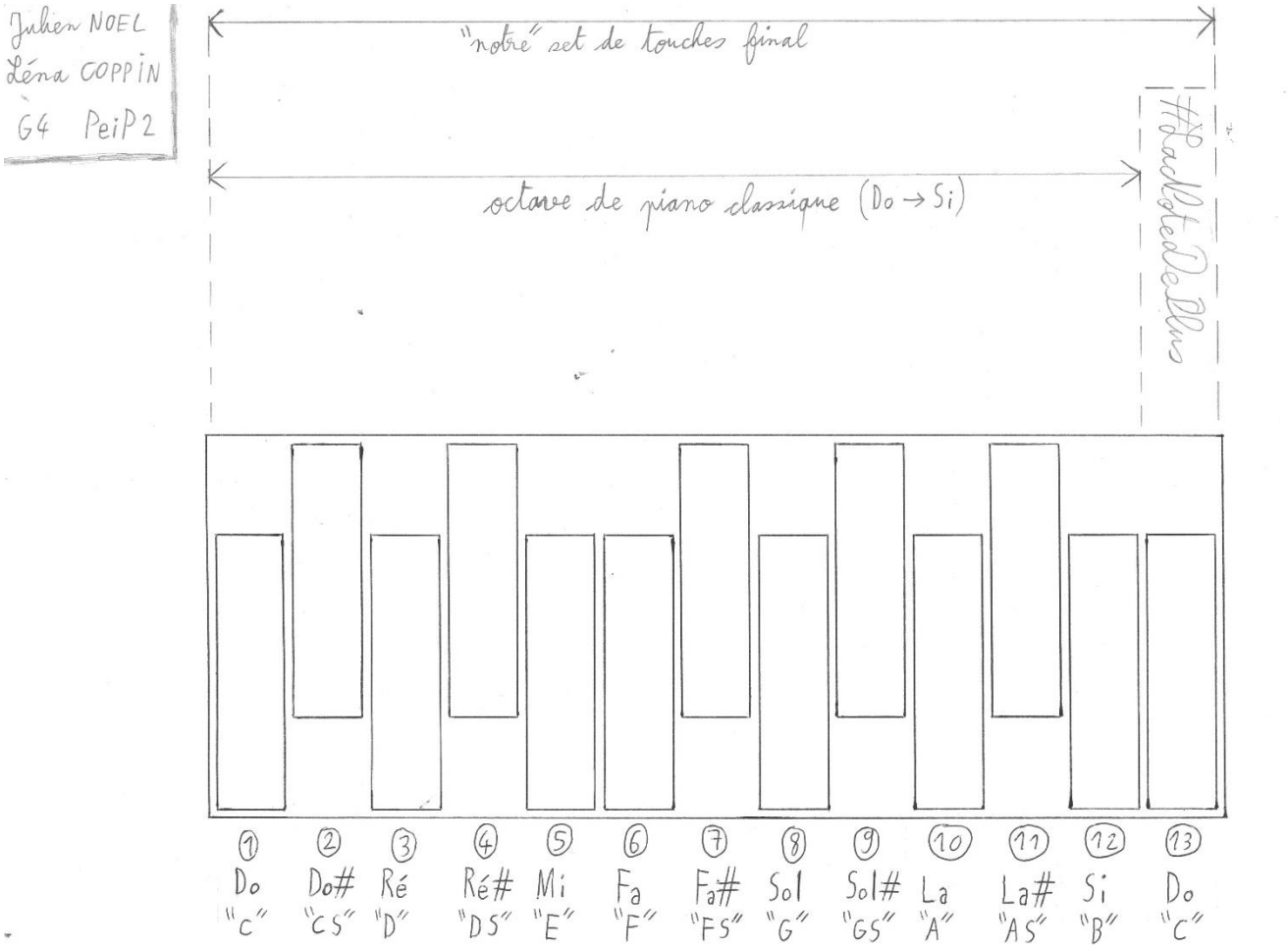
Pour éviter de prendre trop de mémoire (encore une fois), on va définir une liste d'entiers qui représente un seul set de notes, c'est-à-dire uniquement les notes qui vont nous servir :

```
int octave[8] = {};

void choosen_octave(int i){
  /*L'entier i est compris entre 1 et 7 inclus.*/
  int index = 0;
  for (int n = 0; n < 13; n++){
    index = n + (i-1) * 12;
    /*L'entier index est compris entre 0 et 84 inclus.
    Il parcourt 13 valeurs consécutives,
    en commençant et terminant par un multiple de 12.*/
    octave[n] = notes[index];
    }
};
```

Ce n'est pas une fonction car premièrement retourner une liste d'entiers n'est pas si simple que cela (après quelques recherches) et deuxièmement cela ne sert à rien de sauvegarder les sets de notes qui ne servent pas immédiatement. C'est donc une procédure qui modifie la liste *octave[]*.

Je l'ai d'ailleurs appelé *octave[]* et non *set[]* pour que ce soit plus compréhensible mais ce n'est pas exactement la même chose :



Un set est composé de 13 touches alors qu'une octave n'en est composée que de 12 (la 13ème touche est la 1ère touche de l'octave d'au-dessus).

On notera que les touches numéros 1, 3, 5, 6, 8, 10, 12 et 13 sont des touches blanches et que celles numéros 2, 4, 7, 9 et 11 sont des touches noires si l'on veut comparer ce futur (et j'espère magnifique) variophone à un piano classique.

Voici comment j'ai décidé de coder la fonction qui, en lui envoyant l'indice de la note (de 0 à 84), renvoie une chaîne de caractère (de longueur 2 ou 3) :

```
char dico_notes_str[] = {'A','B','C','D','E','F','G','S'};

char print_note (int i) {
  /*i est compris entre 0 et 84 car c'est l'indice de la note de la liste notes[]*/
  char type_note;
  char numero_note;
  int i_bis = i % 12;            //i_bis est compris entre 0 et 11 inclus
  int i_ter = floor(i / 12) + 1;  //i_ter est compris entre 1 et 8 inclus
  //floor(5.7) = 5; floor(-6.9) = -7; floor(1.0) = 1

  switch (i_bis) {
    ...
    }

  switch (i_ter) {
    ...
    }

  return type_note + numero_note;
  }
```

On «pioche» dans la liste *dico_notes_str[]* les caractères qu'il nous faut pour afficher la note d'indice *i*.

Mettre un *switch* au lieu d'un *if* permet une syntaxe plus claire (12 cas pour *i_bis* et 8 cas pour *i_ter*).