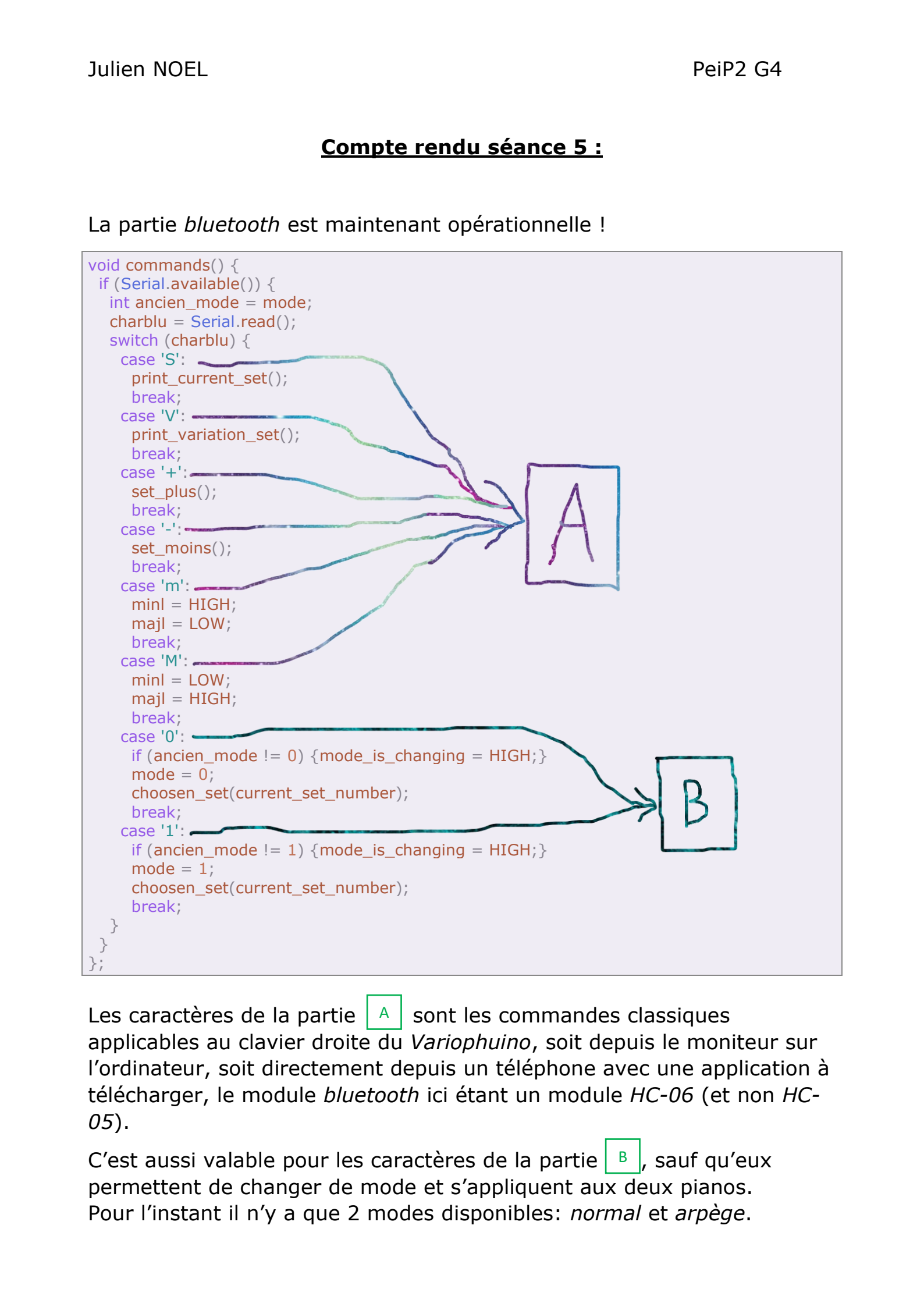
Compte rendu séance 5 :

La partie *bluetooth* est maintenant opérationnelle !

```
void commands() {
  if (Serial.available()) {
    int ancien_mode = mode;
    charblu = Serial.read();
    switch (charblu) {
      case 'S':
        print_current_set();
        break;
      case 'V':
        print_variation_set();
        break;
      case '+':
        set_plus();
        break;
      case '-':
        set_moins();
        break;
      case 'm':
        minl = HIGH;
        majl = LOW;
        break;
      case 'M':
        minl = LOW;
        majl = HIGH;
        break;
      case '0':
        if (ancien_mode != 0) {mode_is_changing = HIGH;}
        mode = 0;
        choosen_set(current_set_number);
        break;
      case '1':
        if (ancien_mode != 1) {mode_is_changing = HIGH;}
        mode = 1;
        choosen_set(current_set_number);
        break;
    }
  }
};
```



Les caractères de la partie [A] sont les commandes classiques applicables au clavier droite du *Variophuino*, soit depuis le moniteur sur l'ordinateur, soit directement depuis un téléphone avec une application à télécharger, le module *bluetooth* ici étant un module *HC-06* (et non *HC-05*).

C'est aussi valable pour les caractères de la partie [B], sauf qu'eux permettent de changer de mode et s'appliquent aux deux pianos.
Pour l'instant il n'y a que 2 modes disponibles: *normal* et *arpège*.

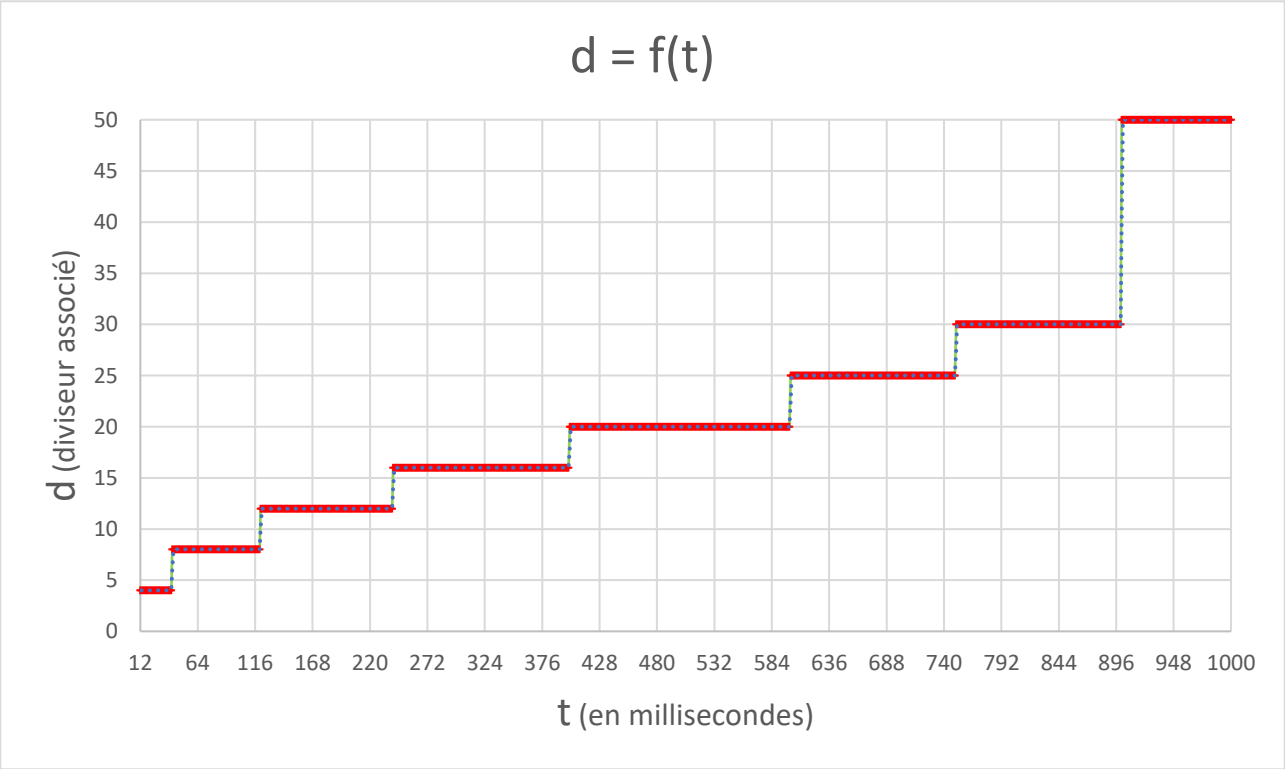
Le "*super_3_times_200_updater()*" vous rappelle quelque chose ?
Eh bien maintenant il a un remplaçant qui ne sert que pour le mode *normal*: *super_3_times_delta_temps_updater()* !

La variable *delta_temps* varie en fonction d'un autre potentiomètre à glissière qui, après un *map()* et un *constrain()*, revoie une valeur contenue dans la variable *an1*.
La valeur de la variable *diviseur* dépend de celle de *an1*.

```
delta_temps = an1 - (an1 % diviseur);
```

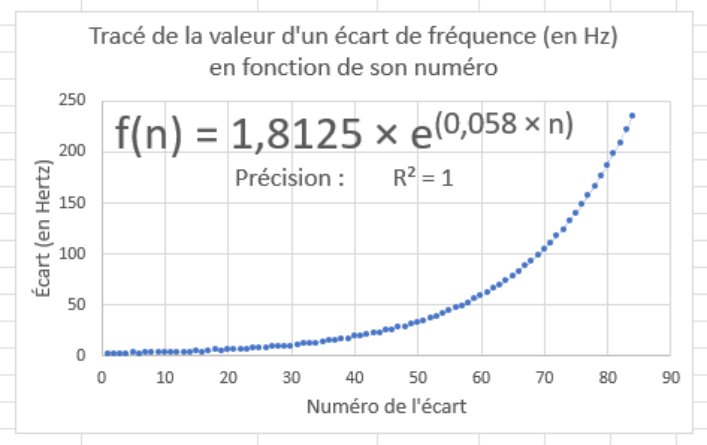
delta_temps est donc un multiple de *diviseur*.

Voici ci-dessous le graphe (en rouge) de *diviseur* (-> d) en fonction de *an1* (-> t), en précisant qu'aucune équation de courbe n'est possible ici car ce qu'il y a en rouge est en réalité 1000-12+1=989 points :



Voici ci-contre ce que j'appelle un **vrai** graphe avec une belle équation de courbe !

(c'était pour l'effet *pitchbend*)



Voici à quoi ressemble la fonction *pianos_checking()* qui est exécutée dans *loop()* :

```
void pianos_checking() {
 commands();

 if (mode == 1) {special_arpege_updater();}
 else {basic_updater();}

 left_t_update();
 right_t_update();
 mode_is_changing = LOW;
 //print_variables_truth();
 while (tl_played || tr_played) {
  if (mode == 1) {special_arpege_updater();}
  else {basic_updater();}

  if (correct_left_playing() || correct_right_playing()) {

   if (mode == 1) {arpege_mode();}
   else {normal_mode();}

  }
  left_t_update();
  right_t_update();
  commands();
  charblu = '\0';
  mode_is_changing = LOW;
  //print_variables_truth();
 }
 left_tone.stop();
 right_tone.stop();
};
```

LES MODES

PAS IMPORTANT ici

Pour l'instant, si *mode* vaut 0, le *Variophuino* est en mode *normal*, et si *mode* vaut 1, celui-là est en mode *arpège*.
Voici un extrait d'organisation du code :

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

void commands()

```
/*=======================[PARTIE EFFETS]=======================*/


//  effet: aucun                    ->      mode = 0
void normal_mode() {
  super_3_times_delta_temps_updater();
  basic_updater();
  if (right_all_ok()) {right_play(1000);}
  else {right_tone.stop();}
};
```

J'allais oublier: *aled.h* sera dédier aux LEDs, ainsi on pourra appeler les procédures/fonctions créées directement dans le code originel.
Il remplace le fichier *bluetooth.h* qui s'est avéré être inutile…