Séance 4 : chauffage d'appoint pour les abeilles "BeeWarm"

Introduction :

Lors de cette quatrième séance, j'ai poursuivi le développement de l'application mobile et consolidé la communication Bluetooth avec l'ESP32. L'objectif principal était d'améliorer la stabilité de l'interface, d'optimiser la gestion du Bluetooth lors des changements d'écran, et d'implémenter l'affichage des données de température issues du système embarqué. Cette séance a également permis de corriger un bug observé auparavant, ainsi que de restructurer l'application pour obtenir un fonctionnement plus propre, plus fluide et plus professionnel.

1) Amélioration de l'application mobile :

D'abord, j'ai entièrement refait l'application BeeWarm. Jusqu'ici, elle reposait sur un seul écran dont les différentes sections étaient montrées ou cachées selon l'état de la connexion. Cette méthode fonctionnait, mais elle devenait complexe à maintenir et générait certains bugs visuels.
J'ai donc adopté une architecture structurée autour de plusieurs écrans indépendants, chacun ayant une fonction précise. pour rappel, l'application comporte plusieurs écrans distincts : un écran de connexion(Screen 1), un écran de menu (Menu), un écran de monitoring des températures (TempMonitor) et un écran dédié au réglage de la température cible (TempSet)

2) Gestion du Bluetooth entre les différents écrans :

Pendant mes essais, j'ai découvert que la connexion Bluetooth de MIT App Inventor ne se conserve pas automatiquement lorsqu'on change d'écran. Cela entraînait une perte immédiate de communication dès que l'on accédait au monitoring ou au réglage de la consigne.

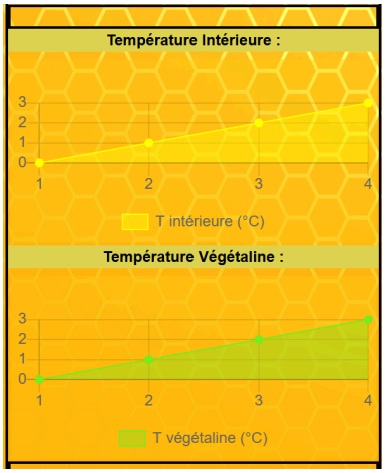
Pour résoudre ce problème, j'ai mis en place une transmission du nom ou de l'adresse du device Bluetooth via le paramètre *startValue* lors de l'ouverture d'un nouvel écran. Ainsi, l'écran suivant peut récupérer cette information et rétablir la connexion Bluetooth dès son initialisation.
Grâce à cette méthode, l'écran *TempMonitoring* peut maintenant recevoir correctement les données envoyées par l'ESP32 sans nécessiter une reconnexion manuelle de l'utilisateur.

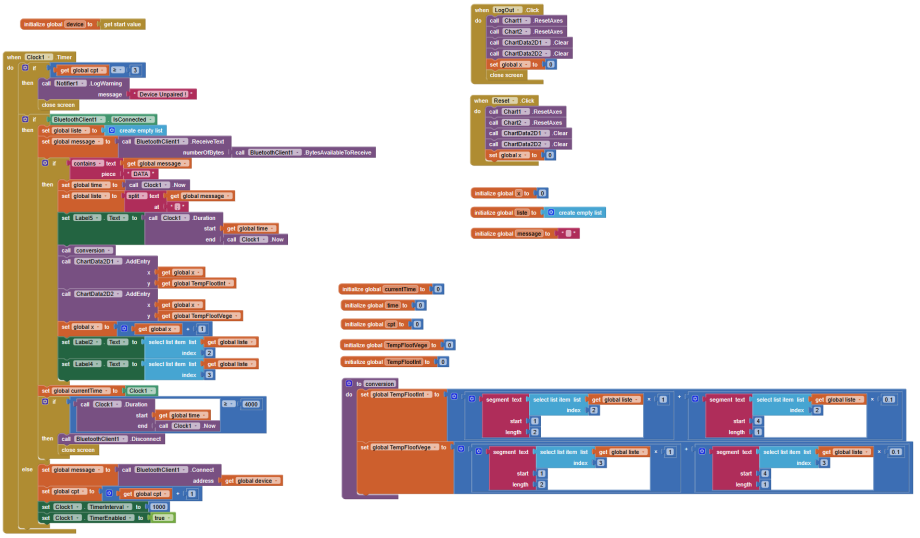
Je n'ai pas encore appliqué ce mécanisme à la partie "Set Température", mais il suffira d'utiliser exactement le même principe.

4) Correction d'un bug des graphes dans "TempMonitoring" :

Un bug important existait auparavant : lorsque l'utilisateur accédait à l'écran de monitoring, l'application affichait parfois des symboles "??" dans les graphiques, et aucune courbe n'apparaissait tant que l'on ne quittait pas puis ne revenait pas sur l'écran.
J'ai identifié l'origine du problème, qui était liée au moment où les données arrivaient par Bluetooth. Le graphique tentait de se mettre à jour alors que la connexion n'était pas encore prête.

Pour corriger cela, j'ai ajouté un délai d'environ trois à quatre secondes avant le premier rafraîchissement, ce qui laisse le temps à l'écran de s'initialiser et à la connexion Bluetooth de s'établir correctement.





5) Mise à jour et description du code Arduino :

Le code Arduino fonctionne toujours sur le même principe d'envoi régulier d'une trame contenant les données de température. Le programme génère des valeurs simulées pour tVeg et tRuche, puis les contraint dans la plage utile pour le suivi de la ruche.
et l'adresse MAC de l'ESP32.

La communication est réalisée sous la forme :DATA;Tveg;Truche;Consigne;Chauffage;MAC

L'ESP32 peut également recevoir des commandes sous forme de texte, notamment celles permettant de modifier la température cible. Lorsque l'application envoie une commande du type `SET=xx.x`, l'ESP32 extrait la nouvelle valeur, l'applique à la consigne interne, puis renvoie un message de confirmation.

```ino
#include <BluetoothSerial.h>
BluetoothSerial SerialBT;

float tVeg = 30.0;
float tRuche = 30.0;
float consigne = 33.0;

String macStr;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("BeeWarmESP32"); // Nom Bluetooth

  // Récupérer l'adresse MAC
  uint8_t mac[6];
  esp_efuse_mac_get_default(mac);
  char buffer[18];
  sprintf(buffer, "%02X:%02X:%02X:%02X:%02X:%02X",
  mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
  macStr = String(buffer);

  Serial.println("MAC = " + macStr);
}

void loop() {
  tVeg   += random(-5, 6) * 0.1;
  tRuche += random(-5, 6) * 0.1;

  tVeg   = constrain(tVeg,   25, 45);
  tRuche = constrain(tRuche, 25, 45);

  int chauffage = (tRuche < consigne) ? 1 : 0;

  // données enviyées
  String msg = "DATA;" + String(tVeg,1) + ";" + String(tRuche,1) + ";" + String(consigne,1) + ";" + String(chauffage) + ";" + macStr;
  SerialBT.println(msg);
  Serial.println(msg);

  // Lire commande Bluetooth
  if (SerialBT.available()) {
    String cmd = SerialBT.readStringUntil('\n');
    cmd.trim();
    if (cmd.startsWith("SET:")) {
      consigne = cmd.substring(4).toFloat();
      SerialBT.println("OK;Consigne=" + String(consigne,1));
    }
  }

  delay(3000);
}
```

Conclusion :

Cette quatrième séance a représenté une étape très importante dans la consolidation du projet BeeWarm. On a restructuré l'application pour la rendre plus efficace et plus simple à utiliser. On a également implémenté l'affichage des températures en temps réel, corrigé les bugs liés aux graphiques et amélioré la gestion de la connexion Bluetooth entre les différents écrans.

Lors des prochaines séances, je prévois :

- d'intégrer le code Arduino final avec le système réel de chauffage,

- de travailler sur l'ajout d'une base de données d'identifiants pour sécuriser l'accès,
- et de participer à la mise en place du moteur pas-à-pas pour le module mécanique.