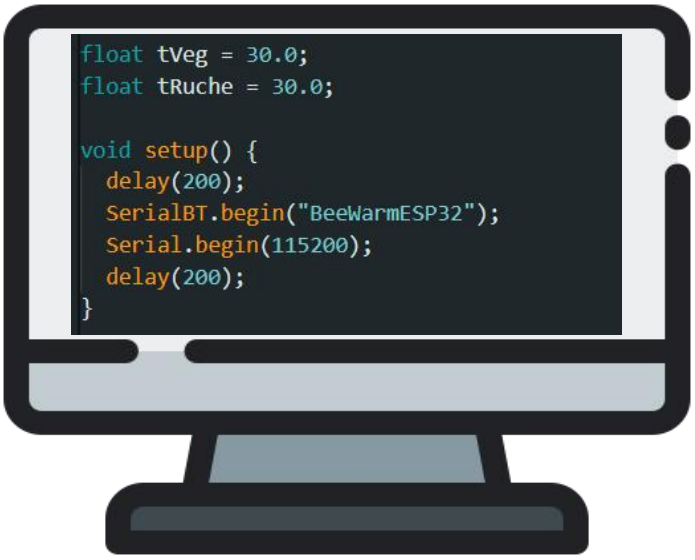
RAPPORT DE SÉANCE 3 :

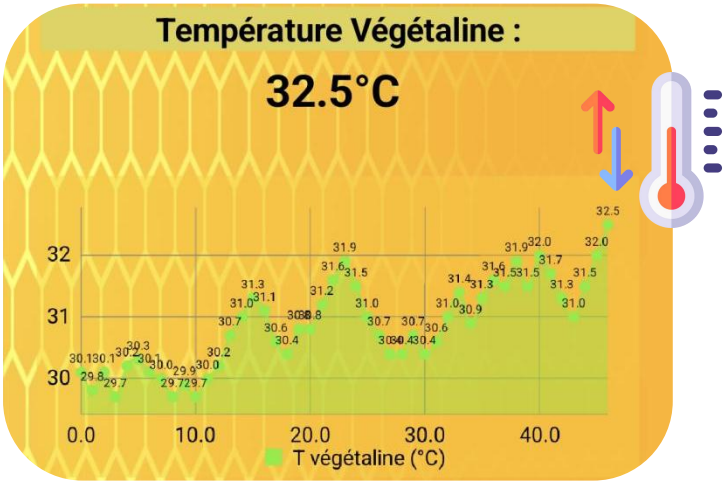
Durant la **3ième séance** nous avons commencé le **code Arduino**.
Dans un premier temps, nous avons **généré des températures de la ruche** et de la **végétaline aléatoirement** afin de pouvoir **tester l'envoi** et l'**affichage** de celles-ci **sur l'application**.

Nous nous sommes servis de la **bibliothèque « BluetoothSerial.h » afin d'utiliser le Bluetooth de l'ESP32.**



Ensuite, nous avons initialisé les variables de températures moyennes de la végétaline et de la ruche ainsi que le Bluetooth de l'ESP32.
Celui-ci s'appellera *« BeeWarmESP32 »* et ce nom s'affichera lors du choix de l'appareil dans l'application.

Pour arriver à cet affichage de températures aléatoires allant de 25°C à 45°C, nous avons utilisé « random(-5,6) » qui donne aléatoirement des nombres entiers compris entre [-5 ; 6 [.
On multiplie ensuite par 0,1 pour obtenir une variation de température comprise entre [- 0,5°C ; 0,5°C].
De même pour « tRuche ».



```
void loop() {
  tVeg += random(-5, 6)*0.1;
  tRuche += random(-5, 6)*0.1; // [-5;6[ x 0,1 => simuler des variations entre [-0,5°C;0,5°C]
  if (tVeg < 25) tVeg = 25; // T min végétaline
  if (tVeg > 45) tVeg = 45; // T max végétaline
  if (tRuche < 25) tRuche = 25; // T min ruche
  if (tRuche > 35) tRuche = 35; // T max ruche

  // conversion float à string avec ##,#
  String message = "DATA;" + String(tVeg, 1) + "°C;" + String(tRuche, 1) + "°C";
  SerialBT.println(message); // envoi du message à l'ESP32
  Serial.println(message); // affichage dans la console arduino
  delay(3000);
}
```

Ensuite, le **message doit être envoyé** sous la forme **« DATA;tVeg°C;tRuche°C »**.
Cependant, dans le programme, les **températures tVeg et tRuche** sont des **variables de type float**. Avant de les **envoyer par Bluetooth**, il faut les **transformer en texte**.
L'expression **« String(tVeg, 1) » convertit la valeur flottante tVeg en une chaîne de caractères en conservant un seul chiffre après la virgule**. Le **second paramètre (1) indique la précision souhaitée**. Cette conversion permet d'**intégrer proprement la température dans le message texte transmis via Bluetooth**.
L'instruction **« SerialBT.println(message) » envoie ce message par Bluetooth** et **ajoute automatiquement un retour à la ligne**. Enfin, le message est transmis **toutes les trois secondes** grâce à l'instruction **« delay(3000) »** placée à la fin de la boucle.

Puis, nous avons ajouté la **gestion d'un capteur de température DS18B20** afin de **mesurer réellement la température de la végétaline**. Le capteur utilise un **bus OneWire** connecté sur la **broche 14**, ce qui nécessite l'utilisation des **bibliothèques OneWire et DallasTemperature**. L'**objet sensors** permet ensuite d'**interroger facilement tous les capteurs présents sur le bus**.
Dans le **setup()**, l'instruction **« sensors.begin() » initialise la communication**, puis **« sensors.getDeviceCount() »** permet de **connaître le nombre total de capteurs détectés**. Une boucle affiche ensuite l'**adresse de chaque capteur trouvé**, ce qui permet de **vérifier qu'ils sont correctement reconnus par l'ESP32**.

VOID SETUP () :

```
// Start up the library for temp sensor
sensors.begin();
// Grab a count of devices on the wire
numberOfDevices = sensors.getDeviceCount();
// Loop through each device, print out address
for(int i=0;i<numberOfDevices; i++) {
  // Search the wire for address
  if(sensors.getAddress(tempDeviceAddress, i)) {
    Serial.print("SENSOR found n° ");
    Serial.print(i, DEC);
    Serial.println();
  }
  else {
    Serial.print("Ghost device at ");
    Serial.print(i, DEC);
    Serial.println(" but could not detect address. Check power and cabling...");
  }
}
//end temp config
delay(200);
```

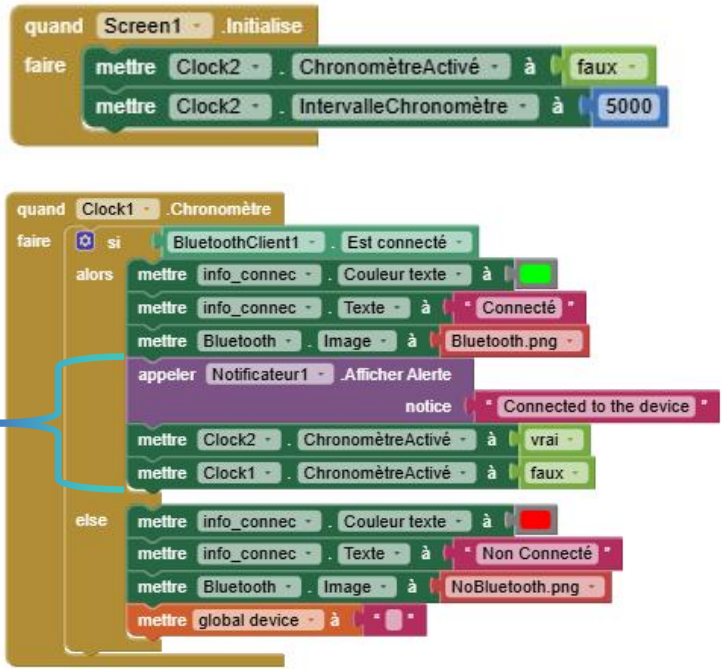
VOID LOOP () :

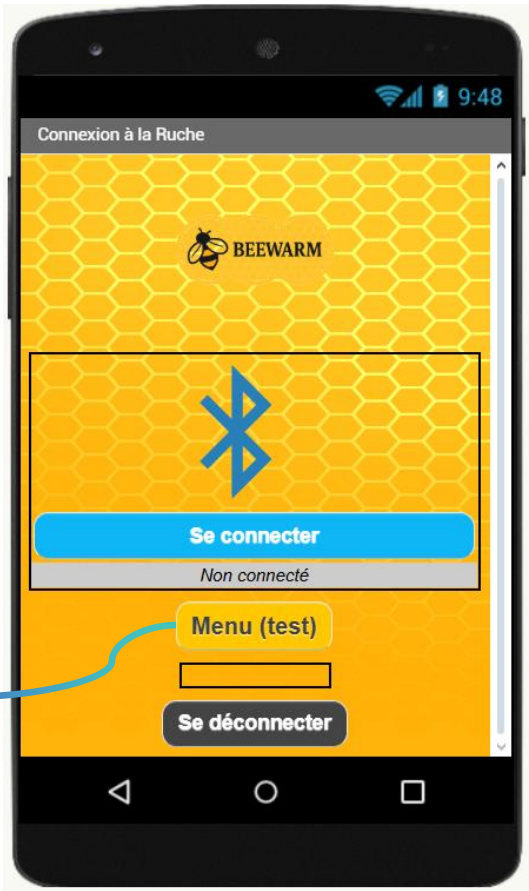
```
// Lire la valeur du capteur de temperature
sensors.requestTemperatures(); // Send the command to get temperatures
// Loop through each device, print out temperature data
tVeg = 0.0;
for(int i=0;i<numberOfDevices; i++) {
  // Search the wire for address
  if(sensors.getAddress(tempDeviceAddress, i)){
    Serial.print("Capteur: ");
    Serial.println(i,DEC);
    float tempC = sensors.getTempC(tempDeviceAddress);
    if (tempC == DEVICE_DISCONNECTED_C) {
      Serial.println("Deconnecte !");
      return;
    }
    tVeg = tVeg + tempC;
    Serial.print(tempC);
    Serial.println("°C");
  }
}
```

Dans la boucle principale (loop), le programme commence par mettre à jour les mesures grâce à la fonction sensors.requestTemperatures(). La variable tVeg est alors remise à zéro, puis une boucle parcourt tous les capteurs détectés. Pour chacun d'eux, sensors.getTempC(tempDeviceAddress) retourne la température en degrés Celsius.
Si une valeur correspond à un capteur déconnecté, un message d'erreur est affiché. Sinon, la température lue est ajoutée à tVeg. À la fin de la boucle, la moyenne des capteurs est calculée grâce à : « tVeg = tVeg / numberOfDevices ».

Cette moyenne constitue la température réelle de la végétaline.
La température de la ruche (*tRuche*) n'est pas encore mesurée par un capteur.

Enfin, pour **améliorer l'ergonomie de l'application**, nous avons prévu qu'une **fois la connexion Bluetooth établie**, l'**utilisateur soit automatiquement redirigé vers le menu principal**. Pour cela, un délai de cinq secondes est utilisé afin d'**afficher brièvement une confirmation de connexion** (*accompagné d'un notifier*), puis d'**ouvrir automatiquement le menu**. (Le bouton *Menu (test)* est donc **retiré**.)

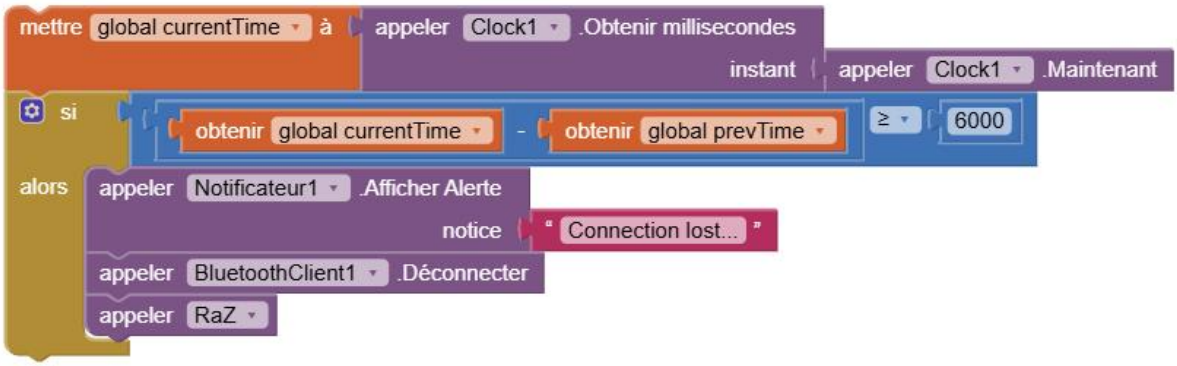




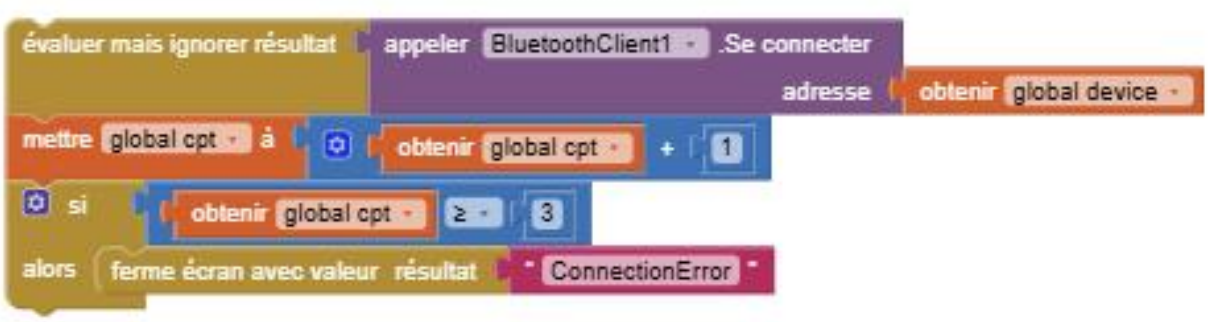
Cette logique repose sur une deuxième horloge (Clock2), programmée avec un intervalle de 5 secondes et activée uniquement lorsque la connexion est confirmée. Clock2 déclenche alors l'ouverture du menu.

Cependant, un **problème est apparu** : l'**écran précédent** (***Screen1***) **continuait de fonctionner en arrière-plan** et **ouvrait le menu de manière répétée**.
Pour éviter cela, nous avons introduit une **variable de contrôle,**
« notConnectedBefore », qui **garantit que l'ouverture automatique du menu** ne se produit qu'**une seule fois**.

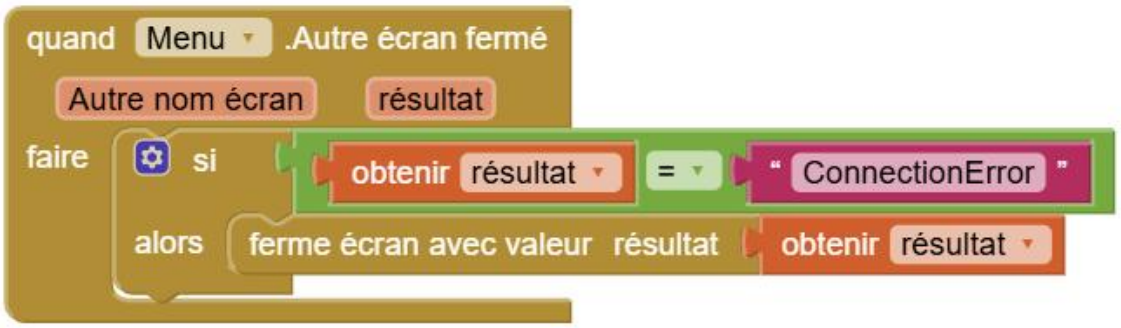
En ce qui concerne le **monitoring de la température**, nous avons mis en place un **mécanisme de détection de perte de communication :**
Si aucune donnée n'est reçue pendant plus de 6 secondes (***currentTime - prevTime***), l'application considère que la **connexion Bluetooth** est **perdue**.



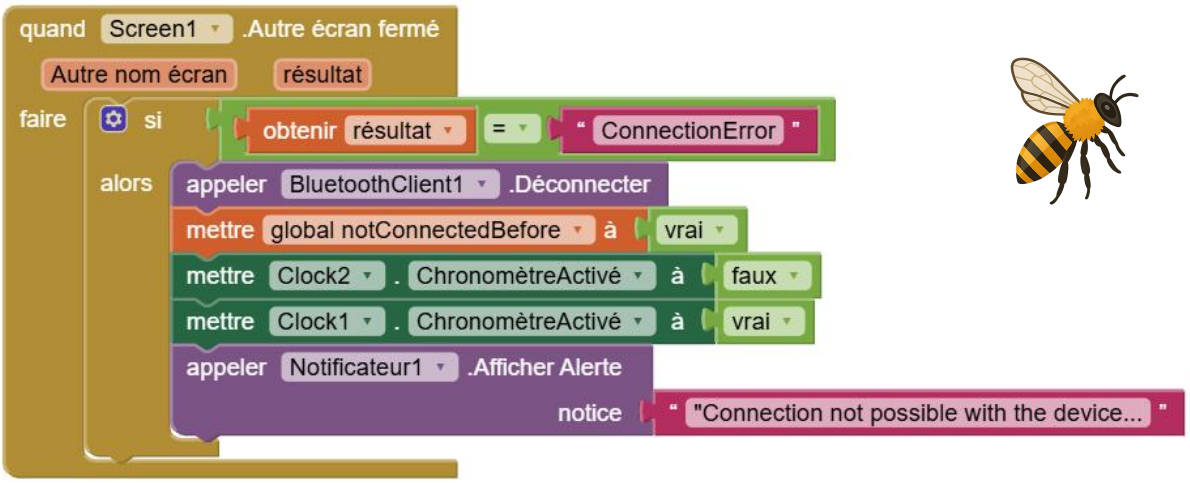
Dans ce cas, elle **tente automatiquement** de se **reconnecter** jusqu'à **3 fois**.
Si les **3 tentatives échouent**, l'**écran de surveillance se ferme** et **renvoie une valeur au menu**, sous forme de **chaîne de caractères (« ConnectionError »)**.



Dans l'**écran Menu**, un **bloc** ***"Quand Menu.Autre écran fermé…"*** permet de **récupérer la valeur de retour**. Si celle-ci **correspond** bien à **« ConnectionError »**, l'application **revient alors à Screen1** en lui **passant** cette **même valeur de retour**.



Enfin, lors de ce **retour à Screen1**, même logique qu'avant puis, une **déconnexion Bluetooth est effectuée** afin de **préparer l'application** à une **éventuelle nouvelle tentative de connexion par l'utilisateur**.



Le « notificateur » sert à informer l'utilisateur d'un problème de connexion avec l'appareil.