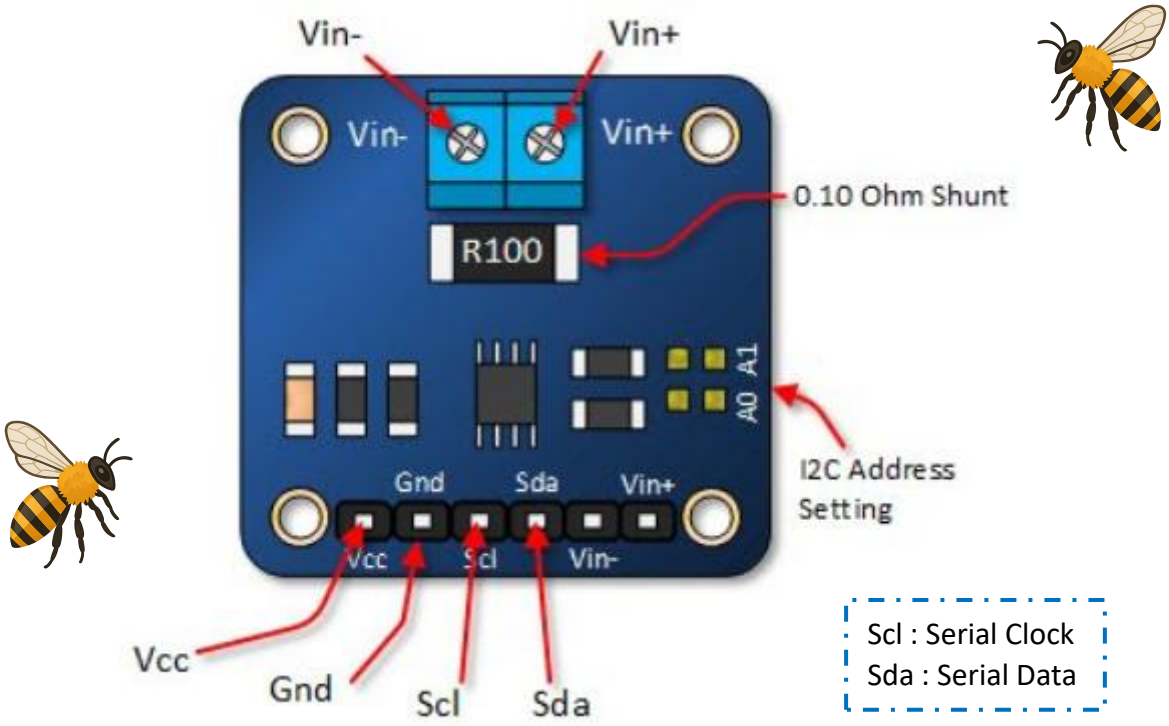
RAPPORT DE SÉANCE 5 :

Durant la **5ième séance** nous avons continué le **code Arduino**.
Dans un premier temps, nous avons utilisé l'**INA219** pour **mesurer tension/courant** et **adapter la puissance du panneau solaire**.

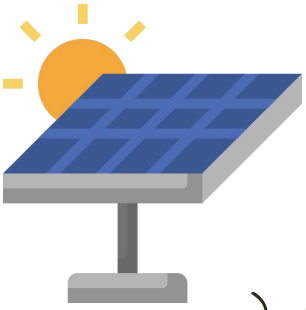


Pour assurer le **maintien de la ruche** à une **température stable**, **deux résistances chauffantes** sont **contrôlées** depuis l'**ESP32**. L'objectif est de **fournir une puissance proportionnelle à celle réellement disponible**, car la source d'alimentation (un **panneau solaire**) **varie continuellement** en **fonction de l'ensoleillement**.
Le **contrôle des résistances** se fait via les **broches SIGNAL1 et SIGNAL2**, pilotées en **PWM** à l'aide de la **fonction analogWrite()**.
Deux seuils sont utilisés autour de la **température de consigne** :
si la **température de la ruche** dépasse la consigne de **plus de 2 °C**, les **résistances sont activées** ;
si elle est inférieure à la consigne **moins 2 °C**, elles sont **coupées**.
Ce **comportement par hystérésis** évite des **commutations trop fréquentes** et **stabilise la régulation**.

```
Adafruit_INA219 ina219;
#define SIGNAL1 2
#define SIGNAL2 5 // resistances
bool heating = false;

#define SDA_PIN 21 // pin ina219 serial data
#define SCL_PIN 22 // pin ina219 serial clock
```

Pour adapter la **puissance fournie aux résistances** en **fonction de l'énergie réellement disponible**, le système utilise un **capteur INA219**.
Celui-ci mesure en temps réel :

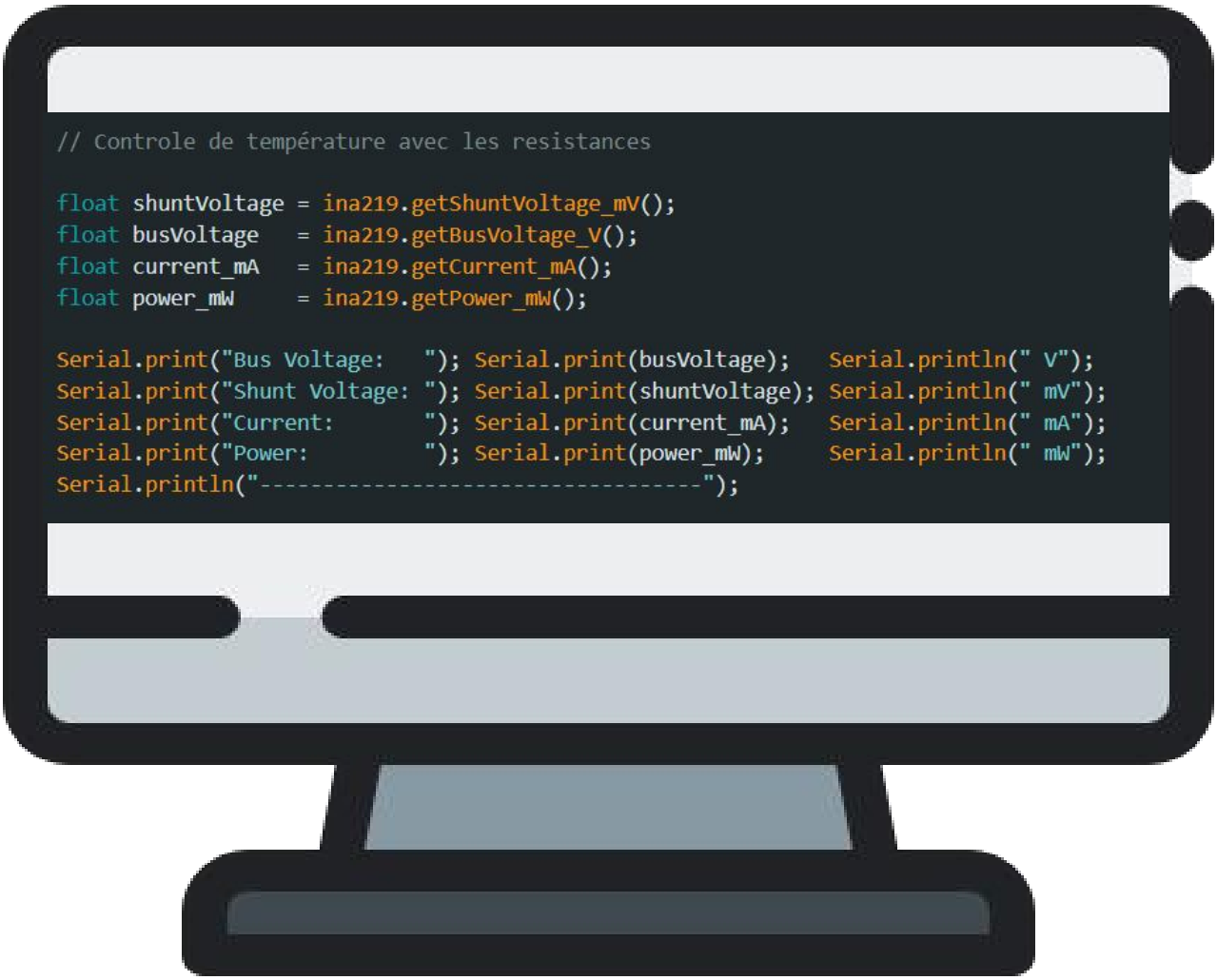
- la **tension du bus** (*sortie du panneau*),
- le **courant** fourni,
- la **puissance instantanée** (*busVoltage × current*).



Ces valeurs sont obtenues grâce aux fonctions :

ina219.getBusVoltage_V();
ina219.getShuntVoltage_mV();
ina219.getCurrent_mA();
ina219.getPower_mW();

La **puissance mesurée power_mW** est ensuite **convertie en valeur de PWM** (entre **0 et 255**) grâce à une **règle de proportion** :
int puissanceDeChauffe = (power_mW * 255) / MPPT;
où **MPPT** représente approximativement la **puissance maximale attendue du panneau solaire** (*à mesurer*).
Ainsi, plus la **puissance disponible** est **élevée**, plus les **résistances peuvent chauffer**, et à l'inverse, lorsque le **panneau produit peu**, la **chauffe est naturellement limitée pour ne pas provoquer d'effondrement de tension**.
Ce principe permet une **adaptation automatique** et **continue** de la **puissance de chauffe**, s'intégrant parfaitement à un système alimenté exclusivement par **énergie solaire**.

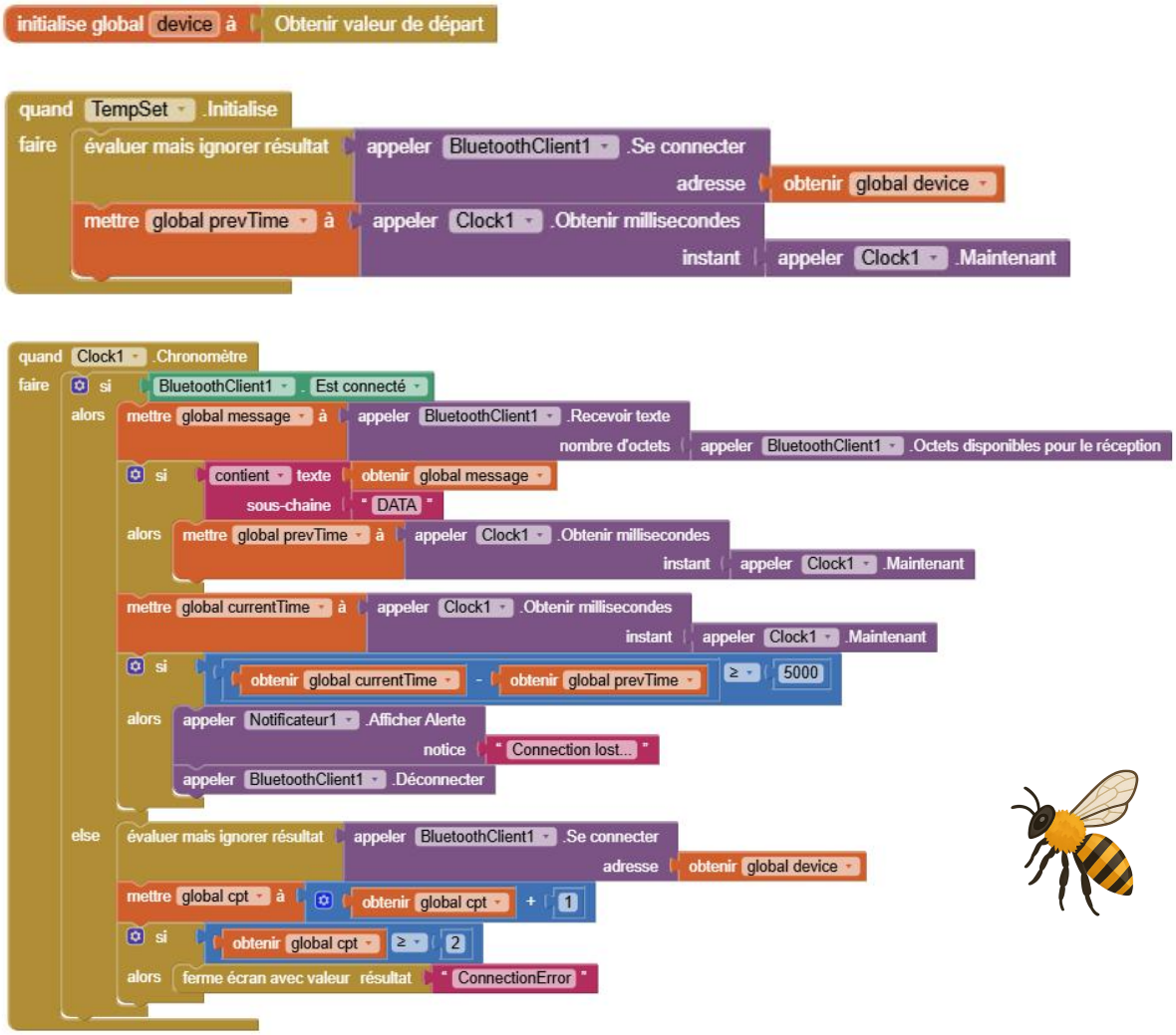


```
// Controle de température avec les resistances

float shuntVoltage = ina219.getShuntVoltage_mV();
float busVoltage   = ina219.getBusVoltage_V();
float current_mA   = ina219.getCurrent_mA();
float power_mW     = ina219.getPower_mW();

Serial.print("Bus Voltage:   "); Serial.print(busVoltage);   Serial.println(" V");
Serial.print("Shunt Voltage: "); Serial.print(shuntVoltage); Serial.println(" mV");
Serial.print("Current:       "); Serial.print(current_mA);   Serial.println(" mA");
Serial.print("Power:         "); Serial.print(power_mW);     Serial.println(" mW");
Serial.println("---------------------------------");
```

VOID LOOP () :

```
if(SerialBT.available()){
  String messageRecu = SerialBT.readString(); // On recupere le message sous forme de caractere
  messageRecu.trim();   // Enlève espaces, \r, \n
  delay(6);
  Serial.println(messageRecu);
  temperatureConsigne = messageRecu.substring(messageRecu.lastIndexOf('=') + 1).toInt();
  Serial.print("Température de consigne modifiée : ");
  Serial.print(temperatureConsigne);
  Serial.print(" °C");
  Serial.println();
}

int MPPT = 4000;
int puissanceDeChauffe = (power_mW*255)/MPPT; // produit en croix
if (tRuche > temperatureConsigne + 2) {
  analogWrite(SIGNAL1, puissanceDeChauffe);
  analogWrite(SIGNAL2, puissanceDeChauffe);
}
else if (tRuche > temperatureConsigne - 2) {
  analogWrite(SIGNAL1, 0);
  analogWrite(SIGNAL2, 0);
}
```

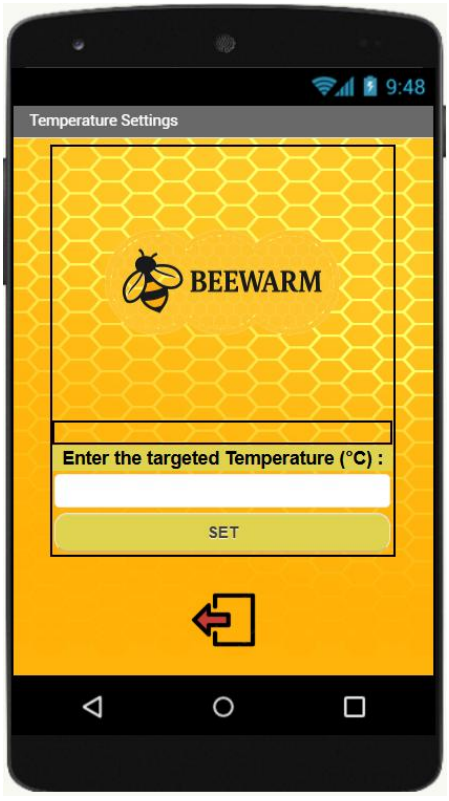
Le système reçoit les **commandes envoyées** depuis l'application mobile via **Bluetooth**.
Lorsqu'un message arrive, l'**ESP32** le lit et met à jour la **température de consigne**.
Le **message envoyé par l'application** est de la forme : «**SET:SP=32**»
On détecte un potentiel **message Bluetooth** avec **SerialBT.available()** qui vérifie si des **données** ont été **reçues** depuis l'application.
Puis, on **lit le message complet** avec **readString()** qui lit tous les caractères envoyés par le smartphone.
Ensuite, on **nettoie le message** avec **trim()** qui **supprime automatiquement** :
retour à la ligne (\n), **retour chariot** (\r), **espaces parasites**.
Cela garantit ainsi un message **propre** et **exploitable**.
Enfin, on **extrait la valeur de consigne**. Le programme récupère la partie après le dernier **«=»** du message, puis la **convertit en entier** avec **temperatureConsigne = messageRecu.substring(messageRecu.lastIndexOf('=') + 1).toInt();**
Par exemple, dans **SET:SP=32**, cela isole "**32**".

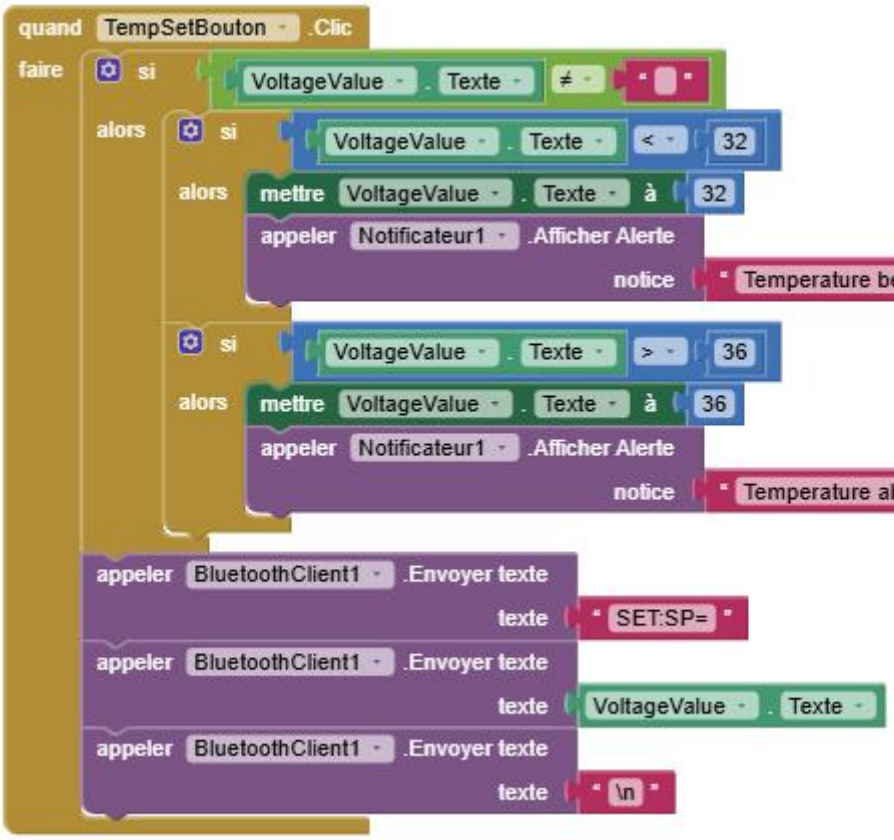


Pour la page de «SET», nous avons réutilisé le même principe de réception Bluetooth que dans la page de «Monitoring», mais ici :

- les données reçues ne sont plus analysées / traitées ;
- elles servent uniquement à détecter si la connexion Bluetooth est toujours active.

En pratique :
si l'application ne reçoit plus aucun message de l'ESP32 pendant un certain temps, elle considère que la connexion est perdue et tente automatiquement de se reconnecter.





Enfin, ce bloc est exécuté lors du clic sur «TempSetBouton».
Le programme vérifie la valeur de température saisie : si elle est inférieure à 32°C, elle est automatiquement corrigée à 32°C ; si elle dépasse 36°C, elle est ramenée à 36°C.
Dans les deux cas, une alerte s'affiche pour prévenir l'utilisateur.
Une fois la valeur validée, le programme envoie la commande Bluetooth "SET:SP=", suivie de la température corrigée, puis d'un retour à la ligne, afin de transmettre correctement la consigne à l'ESP32.