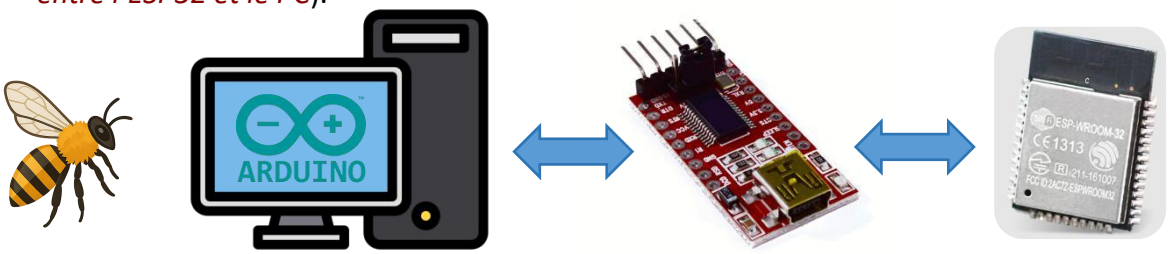
RAPPORT DE SÉANCE 2 :

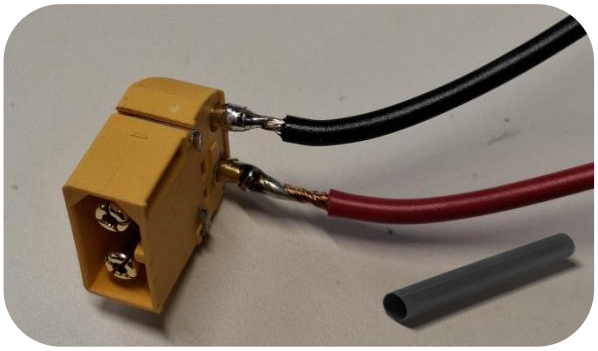
Durant la **2ième séance** nous avons **réglé** pas mal de **problèmes**. Pour commencer, nous avons voulu tester si la **carte ESP32** présente sur le **circuit imprimé fonctionnait bien**. Nous avons donc tenté de **connecter la carte ESP32 à l'ordinateur** à l'aide d'une **carte FTDI** (*une carte intermédiaire permettant la communication série entre l'ESP32 et le PC*).



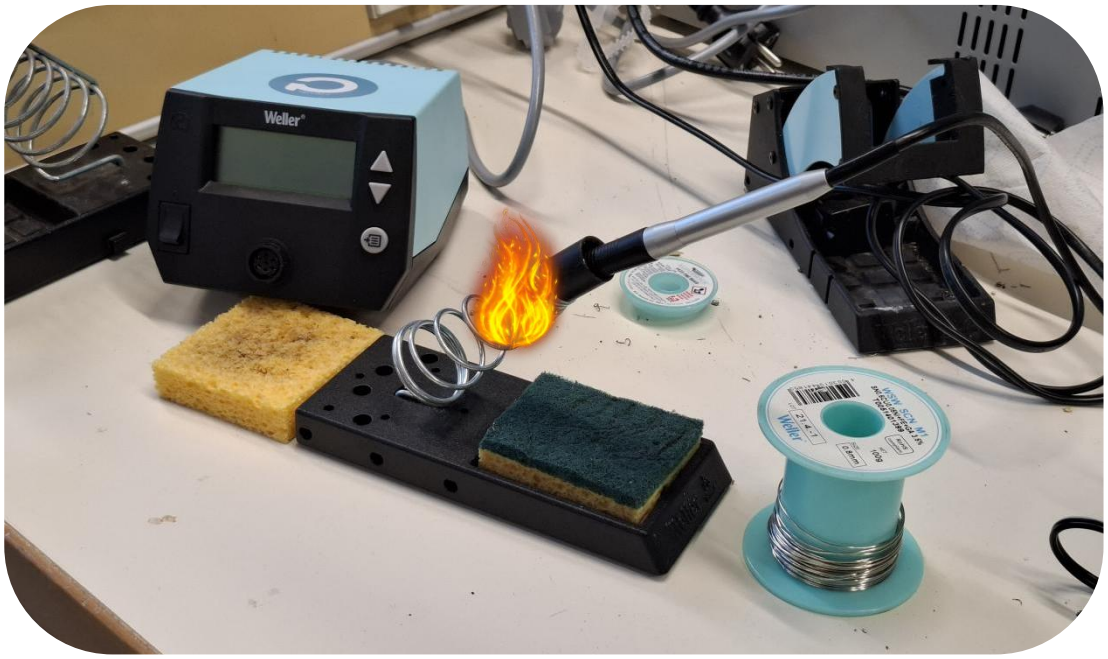
Nous avons donc **téléchargé et installé les pilotes FTDI** adaptés pour permettre la détection correcte de la carte.
Cependant, après l'**installation des pilotes**, un autre problème est survenu : le logiciel **Arduino IDE** ne détectait plus les **ports** de communication.
Pour résoudre ce souci, il a fallu réinstaller la **dernière version** d'**Arduino IDE 2.0**.
Une fois cette mise à jour effectuée, la **carte ESP32** était enfin détectée dans la liste des ports disponibles.

Nous avons ensuite tenté de téléverser le code dans l'ESP32.
Le téléversement s'est effectué, mais le moniteur série d'Arduino affichait un message récurrent d'erreur.
Après analyse, nous avons constaté que l'ESP32 ne recevait pas assez de courant pour démarrer le module Bluetooth, celui-ci nécessitant une poussée de courant supplémentaire au démarrage.

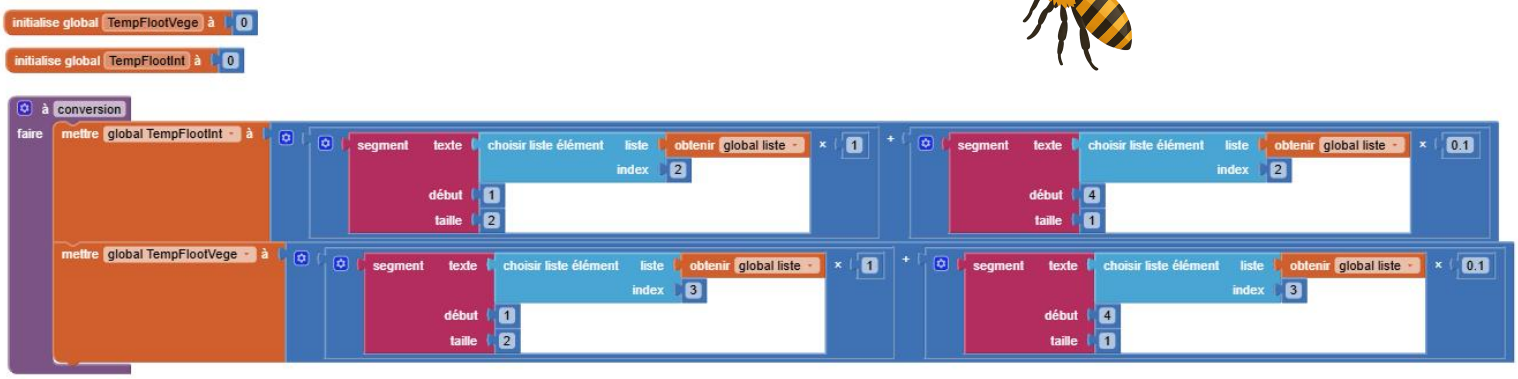
Pour pallier ce problème, il fallait **alimenter la carte du circuit imprimé** en **12 V** via la **prise prévue** pour le panneau solaire.
Cependant, cette prise s'est révélée **défectueuse**.
Nous avons donc décidé de **souder de nouveaux câbles d'alimentation**, **plus longs** et **plus résistants**, et de les protéger avec une **gaine thermorétractable** afin d'assurer une meilleure **fiabilité** et **sécurité**.



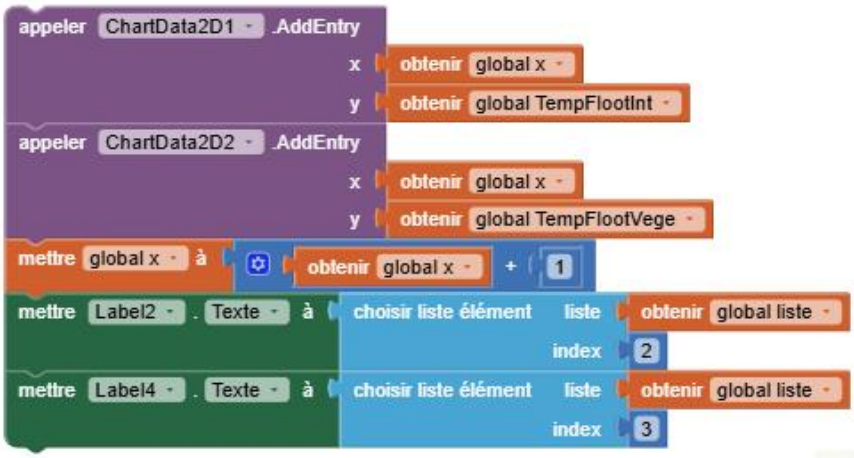
Après ces modifications, la carte a été correctement alimentée.
Le téléversement du code dans l'ESP32 s'est déroulé sans problème, et nous avons pu activer et tester le Bluetooth, confirmant le bon fonctionnement de la carte.



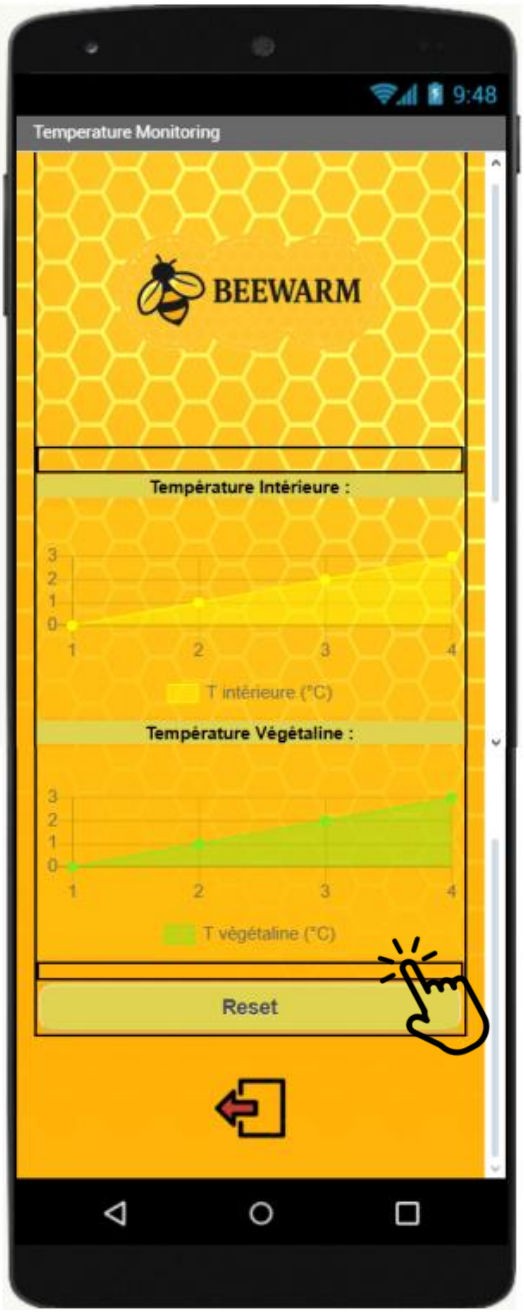
Ensuite, nous avons continué l'**application sur AppInventor**.
Pour rappel, lorsqu'on reçoit le **message**, celui-ci est sous la forme « ***DATA;T_ruche°C;T_végétaline°C*** ».
On récupère seulement les messages commençant par « ***DATA*** » (pour montrer que la **réception des données peut débuter**). Puis, on place les éléments du message dans une **liste de string** (ici, il y a donc **3 éléments séparés** par « **;** »).
Afin d'**afficher correctement les températures** sur les **deux graphiques**, on doit **convertir les températures sous forme de chaînes de caractères en nombres à virgule** (***float***). Pour ce faire, nous avons implémenté une procédure nommée ***'conversion'***. Celle-ci fonctionne de la manière suivante : par exemple nous avons « ***26,5°C*** » dans **global liste** à l'**index 2**, on récupère les **deux premiers caractères** (ici, « ***26*** ») qu'on **multiplie par 1** pour **récupérer les dizaines**.
Puis, on récupère en parallèle le **4ième élément de taille 1** correspondant ici à « ***5*** » et on le **multiplie par 0.1** pour le **convertir en dixième**. On **additionne les deux** ce qui nous donne ***26.5*** que l'on peut **stocker dans une variable**.



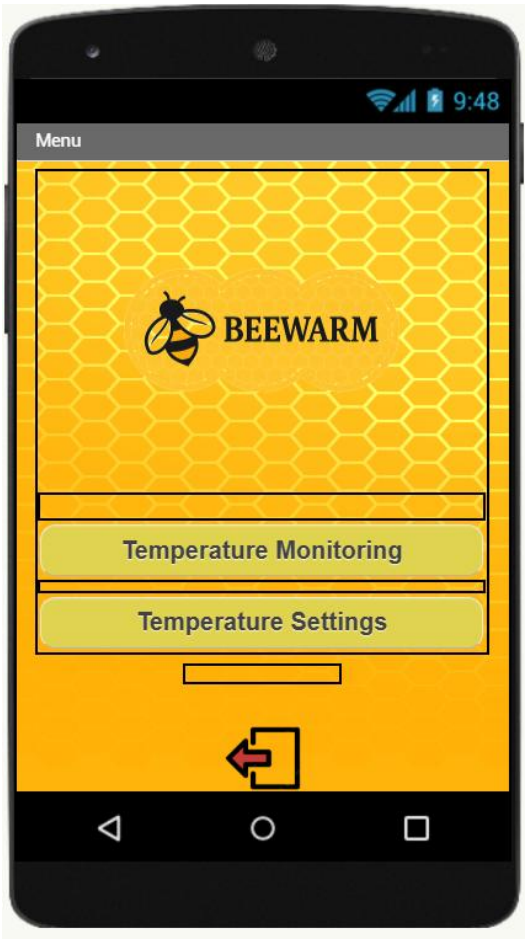
Pour continuer, on peut **afficher les données de températures** sur les **deux graphiques**. Le **x** est **incrémenté** à chaque nouvelle valeur afin de **progresser sur l'axe des abscisses**.
On peut aussi **afficher la température** sur les **labels au-dessus des graphiques** (*donc directement en chaîne de caractère avec les éléments de la liste*).



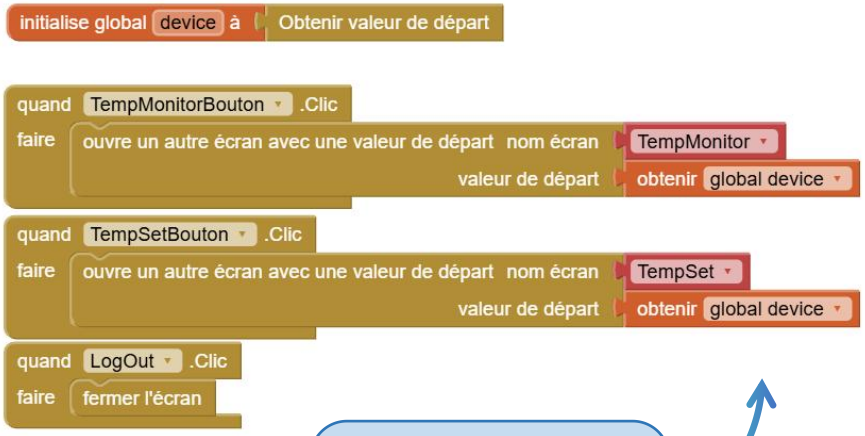
Le bouton « Reset » sert à remettre à 0 l'affichage sur les deux graphiques (*effacer les données de températures affichées*).



Pour avoir **différentes parties dans notre application**, nous avons utilisé **différents écrans**, celui-ci correspond donc à la **surveillance de la température** (***monitoring***).
Pour aller sur cet écran il faut d'abord passer sur l'écran du **menu** qui sert à **choisir** si on veut aller **regarder la température** ou bien **la modifier** (***set***).

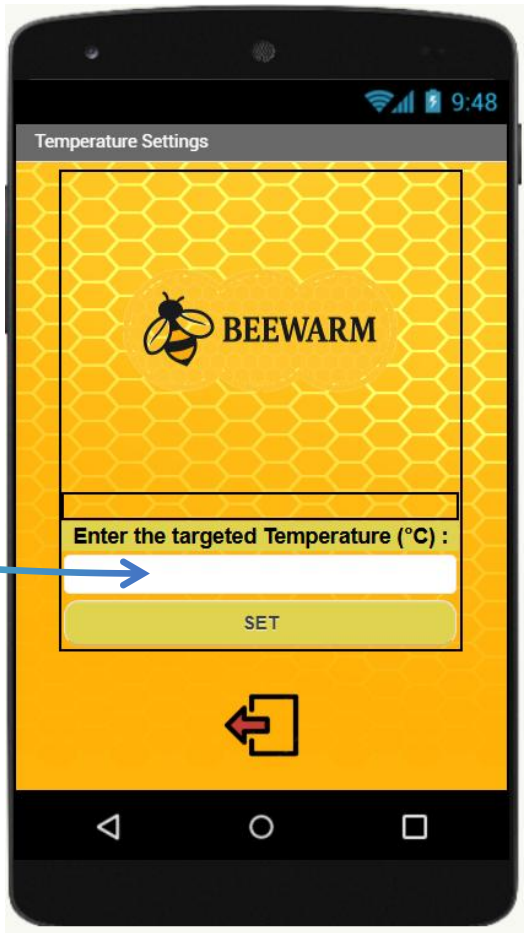


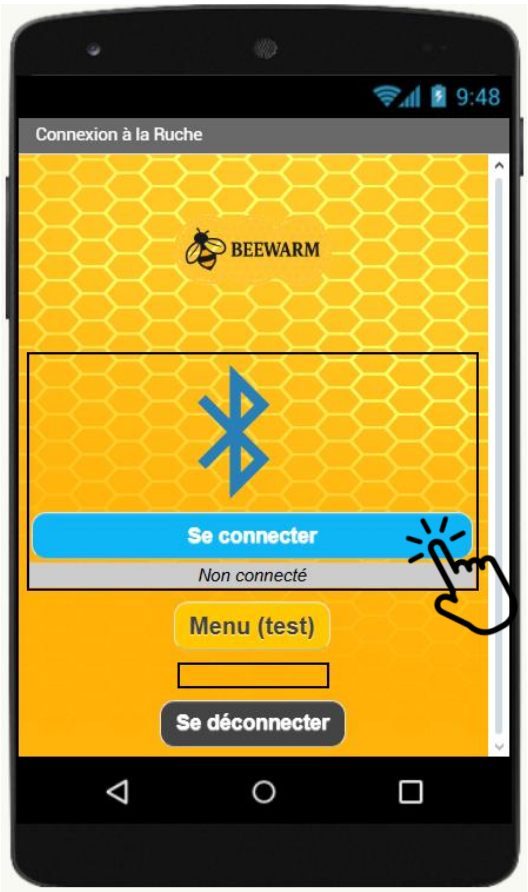
Ici nous avons donc le menu qui sert à décider si on veut 'monitorer' ou 'set' la température.



Le rôle de la valeur de départ (variable device) est expliqué plus tard...

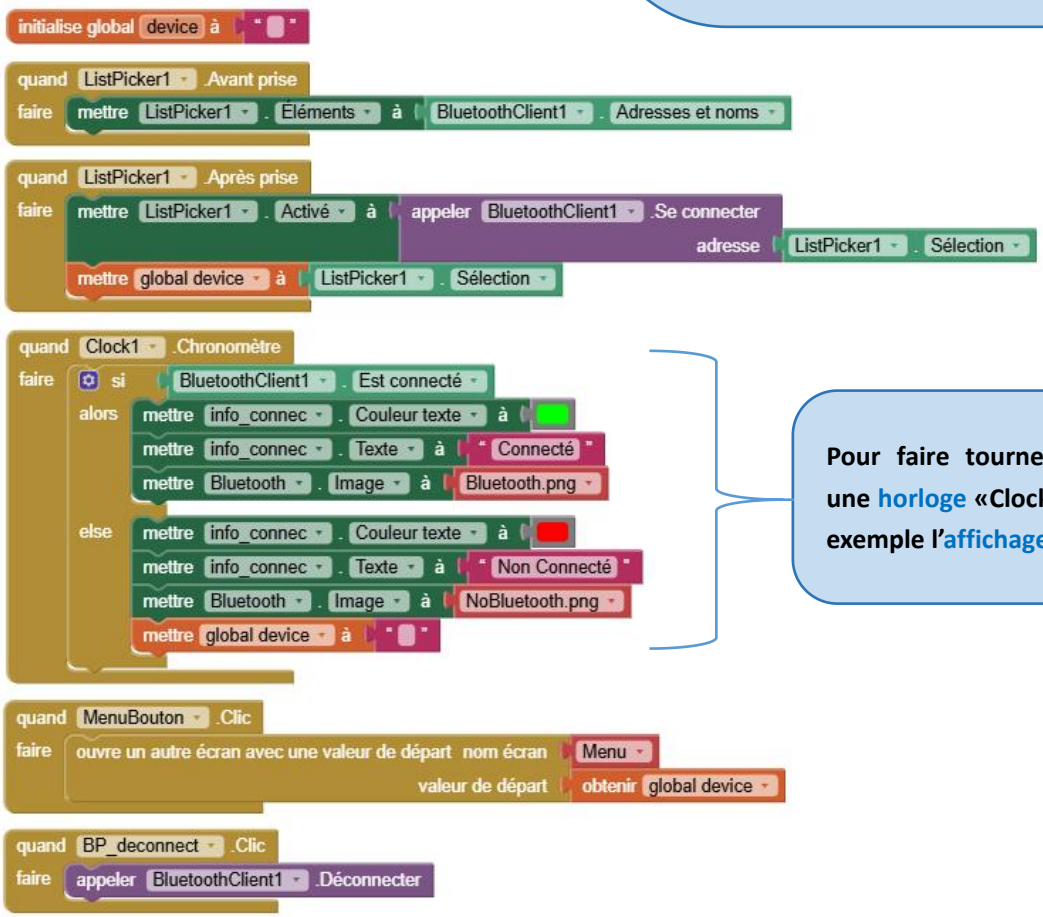
Ensuite, si on clique sur Temperature Settings, nous arrivons sur cette page (la partie code scratch n'est pas encore implémentée). Elle devra récupérer la valeur correctement entrée dans la zone de texte et l'envoyer en Bluetooth à l'ESP32.





Enfin, nous arrivons sur la première page. Celle-ci sert à connecter l'application à l'ESP32.
Lorsqu'on clique sur « Se connecter », une liste s'ouvre avec tous les appareils appareillés au smartphone.
On choisit le bon : « BeeWarmESP32 ».

Si nous sommes connectés à l'ESP32 et que nous changeons ensuite d'écran pour, par exemple, aller sur l'écran de monitoring ; la connexion n'est pas maintenue.
Donc nous avons initialisé une variable « device » qui va récupérer l'adresse de l'ESP32 lors de la connexion.
Puis, lorsqu'on ouvre un autre écran, on transmet une valeur de départ (*la variable « device »*) afin de maintenir la connexion.
La valeur de départ permet ainsi de transmettre des données entre les écrans. Celle-ci est ensuite récupérée lors de l'initialisation du nouvel écran.



Pour faire tourner l'application, on utilise une horloge «Clock1» pour mettre à jour par exemple l'affichage de la connexion réussie.