Rapport de séance du Vendredi 17 Janvier 2020 (séance 4):

Entre les 2 séances, j'ai écrit la classe Wifi pour encapsuler l'envoi de points via Wifi. Apparemment, il est impossible d'utiliser la fonction *delay()* en dehors des contextes d'appel de *setup()* et *loop()*. Je me suis arrangé pour mettre toutes les méthodes "lourdes" d'initialisation (ex: envoi des premières commandes AT + delay) dans le contexte d'appel de *setup()*. Pour rappel, les classes sont documentées dans la section **Wiki** de mon dépôt Github.

Pour la méthode listenForPoints(), il a fallu être un peu stratégique. En effet, comme expliqué dans un rapport précédent, l'Arduino n'est pas multithread et on ne peut pas vraiment faire de callbacks "à la JavaScript". Quel est le problème si un fait une méthode qui se contente d'attendre les points puis de renvoyer le tableau de points correspondant ? Le message qu'envoie le programme Java pour dire qu'il a fini d'envoyer les points est "**point:done**". Donc il faudrait une méthode qui remplisse un tableau de points puis qui renvoie le tableau à la réception du message de fin. Sauf qu'on n'a aucun moyen de savoir quand est-ce que l'utilisateur va envoyer les points ni si le message de fin va bien arriver (la liaison avec le module wifi n'étant pas très optimisée). Il y a donc un risque de tomber dans une boucle infinie qui attendrait vainement le message de fin, et le programme Arduino se retrouverait totalement bloqué.

Quelle est donc la solution ? Il faut faire en sorte que notre méthode *listenForPoints()* rende régulièrement la main à la fonction d'appel (*loop() par exemple). Un développeur Java penserait naturellement à jeter une exception (TimeOutException par exemple), mais comme expliqué dans un précédent rapport nous ne pouvons pas utiliser les exceptions en Arduino et de plus ce n'est pas le consensus en C++. La méthode *listenForPoints()* va donc renvoyer un **booléen**: vrai si elle a récupéré tous les points et le message de fin dans les temps, faux sinon. Elle va prendre 2 paramètres en argument: une référence vers le Vector de Point (je pense que les références sont plus faciles à appréhender que les pointeurs pour certains) et le délai maximum pour récupérer les points. Si on met un délai de 0, alors il n'y a pas de délai maximum. Si le Vector contient déjà des points, notre méthode ne fait pas le ménage elle se contente d'ajouter des points. De cette façon la fonction d'appel reprend la main régulièrement et on peut détecter les incohérences: normalement le travail de *listenForPoints()* est terminé lorsqu'elle renvoie **true** cependant si elle renvoie **false** et que la taille du Vector n'a pas évolué c'est qu'on est face à une incohérence. Pour voir un exemple de code en détail, voir l'onglet **Wiki**.

Toujours entre les 2 séances, j'ai essayé de rendre le programme Java (**pancakeDrawer**) un peu plus ergonomique. Maintenant lorsqu'on envoie des données à l'Arduino il y a une petite jauge de progression en bas du

programme. De plus il est possible de sauvegarder et de ré-ouvrir ses croquis (ça fonctionne avec l'interface Serializable).

Il reste cependant quelques problèmes d'instabilité du programme Arduino avec le transfert: Java semble dire que le transfert se déroule correctement mais rien ne s'affiche sur le moniteur série. Ca arrive surtout lorsqu'il y a beaucoup de points. C'est assez curieux. Pour l'instant mes "suspects" sont: la liaison série, la faible quantité de RAM de la carte Arduino et ma classe Vector (quoique à mon sens il n'y a que mon destructeur qui soit vraiment "mal programmé", pour le reste j'ai fait de mon mieux pour avoir un code propre).

Comme expliqué dans un précédent rapport de séance la liaison wifi ou bien la liaison série Arduino-module wifi n'est pas très fiable. Le **Thread.sleep()** dans le code Java a permis d'arranger ça mais il reste très ponctuellement des points aux valeurs aberrantes. Pour le coup ce sera très facile à corriger.