Nathan
DJIAN-MARTIN
G2

POLYTECH NICE-SOPHIA

Rapport de séance n°3 06/01/2020

L'Arduino possède une mémoire appelée EEPROM qui permet de faire de la persistance de données et de les garder en mémoire même après un redémarrage de la carte (contrairement aux variables que l'on a l'habitude de déclarer qui sont stockées dans la SRAM). Sur le site officiel microchip on peut voir que les puces ATmega328P ont 1024 octets de mémoire EEPROM. Il faut faire attention lors du stockage des valeurs analogiques dans la mémoire EEPROM car chaque octet peut prendre des valeurs allant de 0 à 255 alors que les valeurs analogiques vont de 0 à 1023.
J'ai donc créé un programme (*eprom_test* sur GitHub) qui possède deux fonctions :
- *writeInts* qui écrit des entiers aléatoires dans la mémoire
- *readInts* qui lit les entiers de la mémoire et les écrit dans le tableau eeprom_readout

J'ai comparé ce que la fonction *writeInts* écrivait et ce que la fonction *readInts* lisait après un redémarrage de l'Arduino à l'aide du moniteur série et les deux sorties sont identiques.
On pourra donc stocker les combinaisons secrètes même après un rédémarrage de l'Arduino !
Pour l'instant le code se limite aux fonctions *EEPROM.write* et *EEPROM.read* qui ne permettent de stocker que des octets. Il existe les méthodes put et get qui permettent de stocker des structures de données plus complexes (float, double) voire même nos propres structures de données (objets). En fonction du modèle de stockage de la combinaison secrète nous aurons peut être besoin de ces fonctions.

J'ai ensuite recommencé à réfléchir au programme principal en essayant de détecter des coups (avec un bouton pour simplifier les tests) et de stocker les combinaisons dans un tableau dans un premier temps. La combinaison n'est rien d'autre qu'un tableau d'entiers. Chaque entier modélise l'écart entre deux temps obtenus avec la fonction *millis*. On stocke les écarts de temps entre chaque coup (chaque fois que le bouton est pressé dans le cas de mes tests). Il y a un nombre de coup maximal que l'on peut donner et si on ne frappe pas pendant 2 secondes le programme considère que l'on a terminé la combinaison. Le principe de détection repose sur une boucle *do … while* qui permet d'exécuter au moins une fois la boucle (car initialement on n'a pas de temps de référence auquel se comparer donc la condition du while n'est pas valide).
Pour l'instant je ne suis pas parvenu à récupérer une combinaison. J'ai terminé d'écrire le programme en fin de séance et je n'ai eu que quelques minutes pour le tester. Il n'affichait que des 0 dans la console (c'est ce avec quoi je remplis mon tableau

initialement). Donc j'en ai déduis qu'il ne détectait pas les coups. Le bouton fonctionne car je l'ai testé avec un montage basique juste avant. Le problème vient très probablement de l'algorithme de détection qui n'est pas bon. Le code est tout de même disponible sur le GitHub sous le nom de *combinaison_detect_test*.