UPEC
Connaissance - Action

UNIVERSITÉ
PARIS-EST CRÉTEIL
VAL DE MARNE

Licence Professionnelles
LPMIMR
Année 2018 - 2019

Projet Tutoré : Paintball

Création et programmation de nouveaux jeux

Robles *Jérémie*
Mayeur *Jean-Noël*
De Meyer *Magaly*

Université Paris-Est Créteil
61 Avenue du Général de
Gaulle, 94010Créteil

Avec la participation de Mr **Lemaire** Bruno

SOMMAIRE

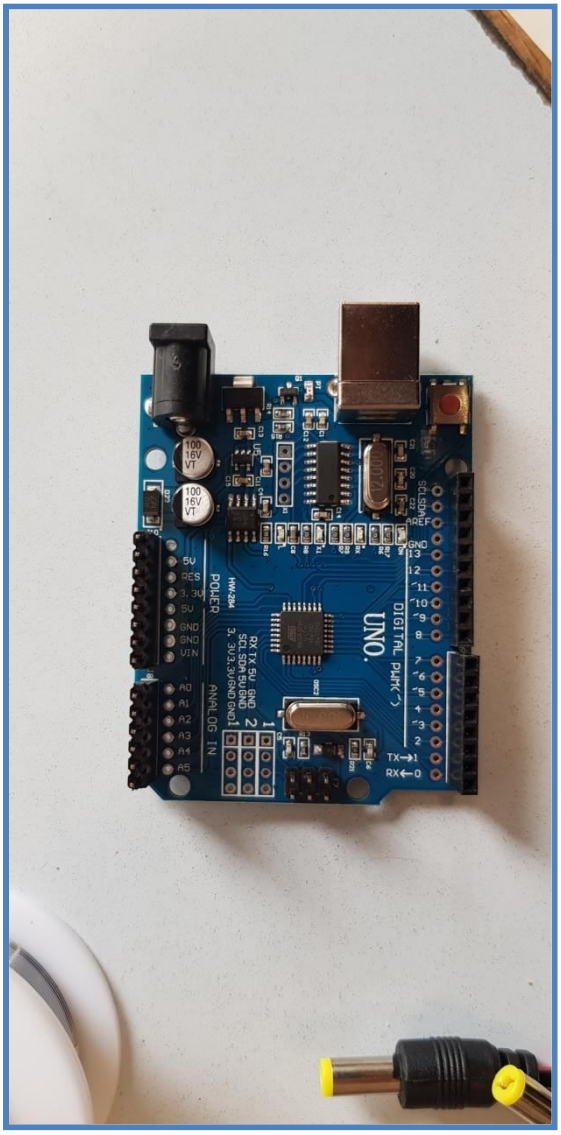
Introduction

Durant notre année universitaire en Licence Professionnelle MIMR, nous devions choisir un projet à effectuer en plus de nos activités scolaires et en entreprise.
Jérémie, Jean-Noël et Magaly avons choisi le projet "paintball" proposé par Monsieur Lemaire Bruno qui, en autodidacte avait réussi à nous présenter un début de prototype . Il s'agit de créer de nouveaux jeux qui seront utilisés lors de diverses parties de paintball. Ces différents jeux doivent être disponible à partir d'une valise dite "maître" qui devra par la suite faire parti des valises joueurs.
Plusieurs points de réflexions sont à prendre en compte tel que la solidité des différents supports par rapport au choc que les différents joueurs peuvent leur assigner, leur résistances face aux intempéries car cela peut se pratiquer sous la pluie, la distance de communication entre les différents émetteurs avec ou sans perturbateur car la communication s'effectue grâce à une antenne.

I.

Réalisation des différents boitiers

On utilisera pour la partie électronique une carte arduino UNO puis trois cartes électroniques vierges sur lesquelles on effectuera les différents montages. Les quatre cartes électroniques seront identique car elles seront attribuées à chaque boitier. La seule différence sera au niveau de la programmation.

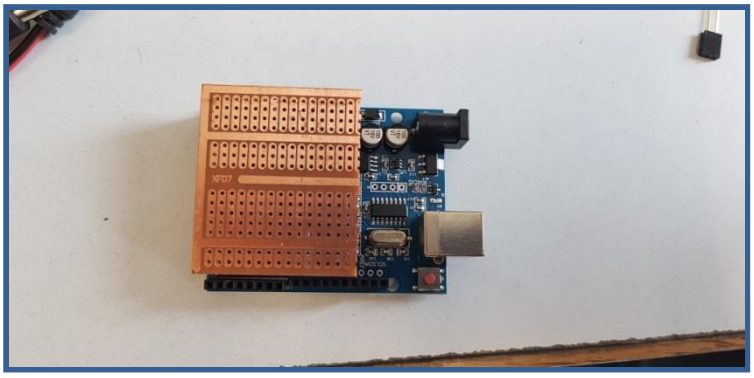


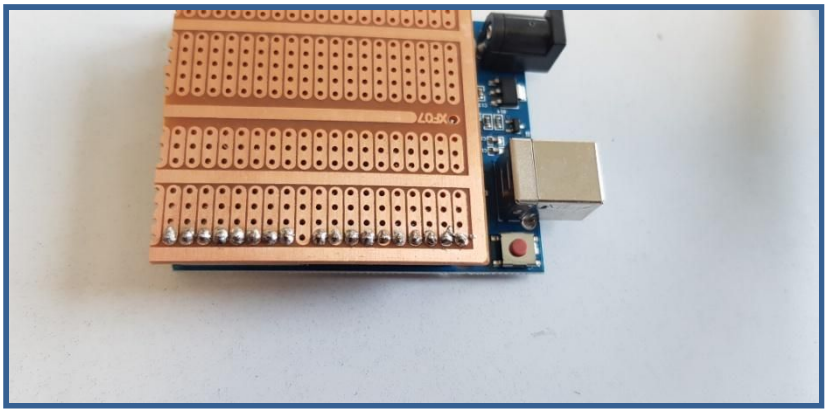
Association des divers ports :

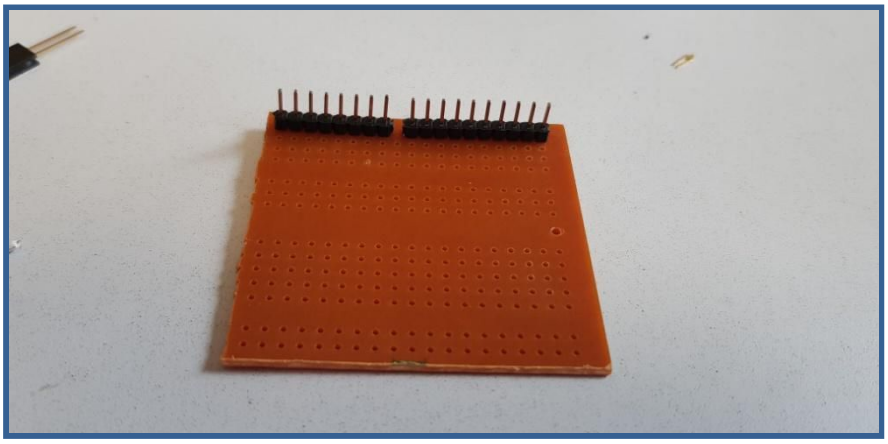
13 → SCK
12 → MISO
11 → MOSI
10
9 → BTN (3)
8 → CSN
7 → CE
6 → BTN (2)
5 → BTN (1)
4 → LED (R)
3 → LED ()
2 → LED (B)
1 → Tx
0 → Rx

Après l'association, on passe au montage et aux soudures des quatre autres cartes. On aura donc besoin de divers matériaux tel que boutons poussoirs, fil d'éteins, résistances, câbles, 4 écrans I2C, des antennes..

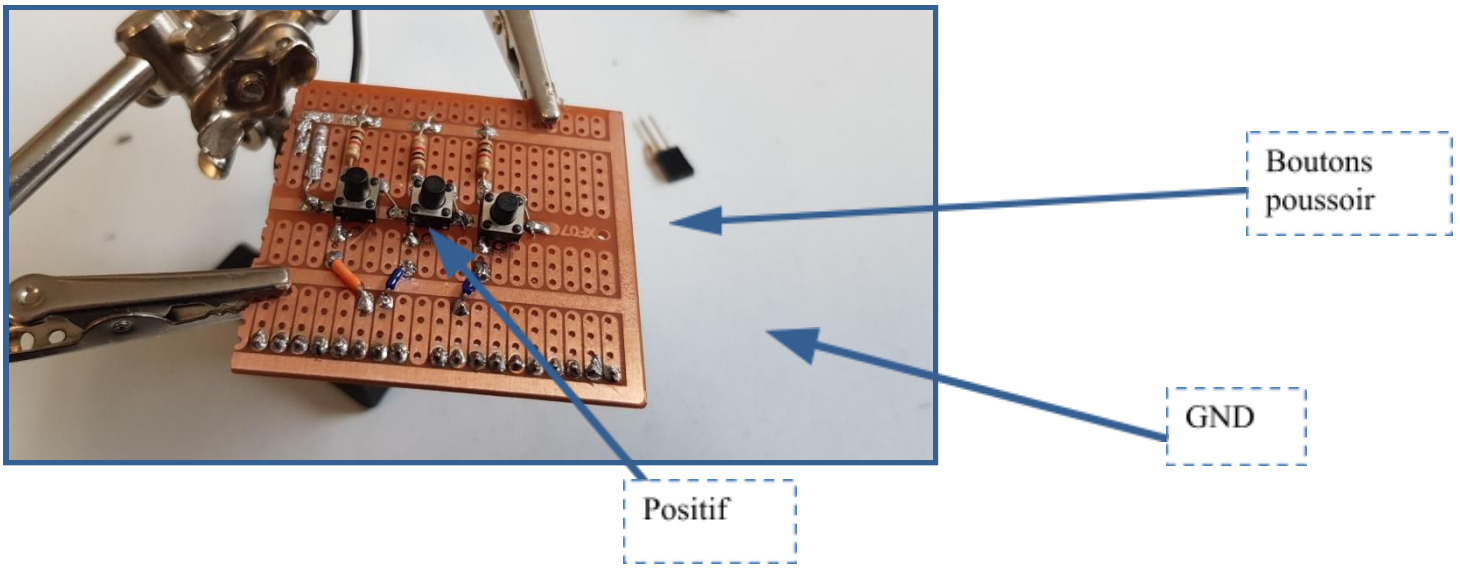
On commencera par la découpe la carte électronique et on lui ajoutera des pins afin de pouvoir la connecter à la carte arduino.



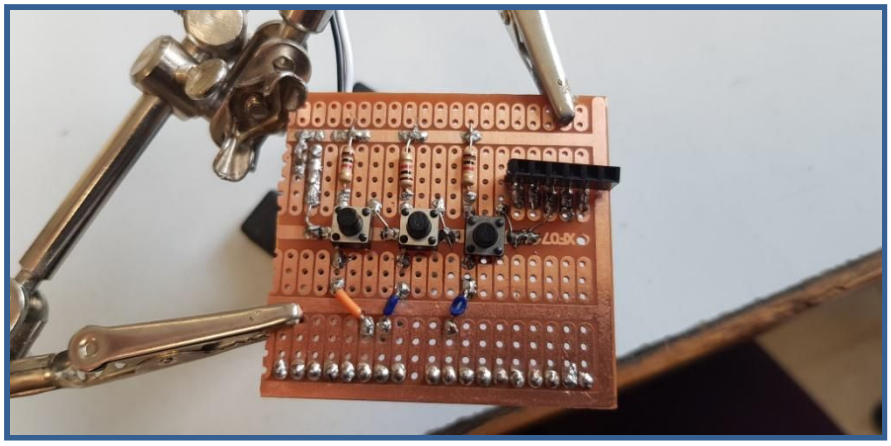


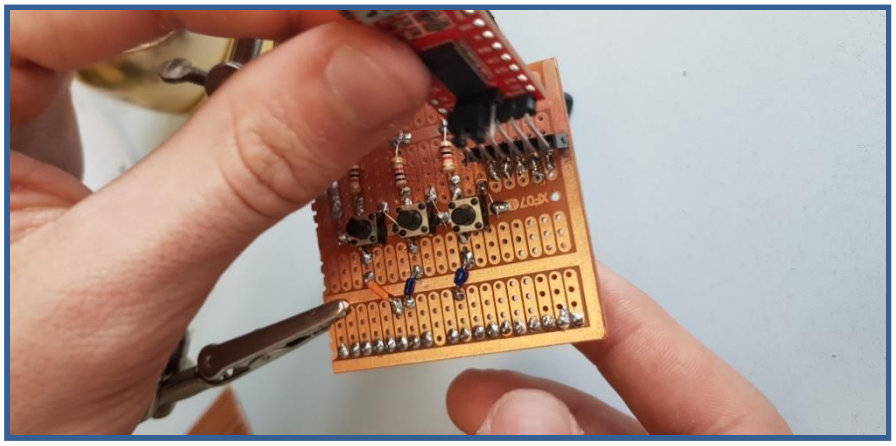


Ce suit alors une succession de soudures des différents composants.

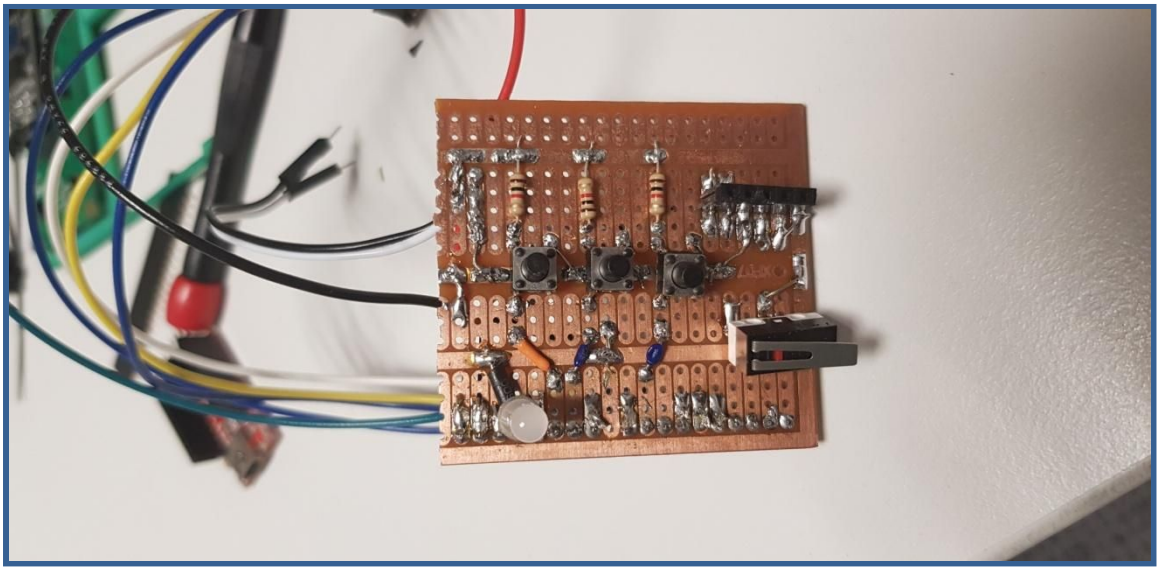


Placement du FTDI :

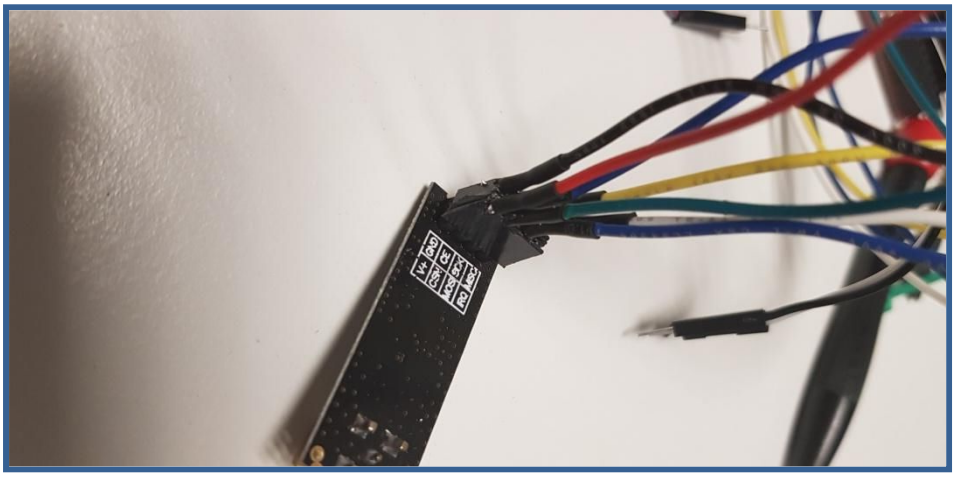


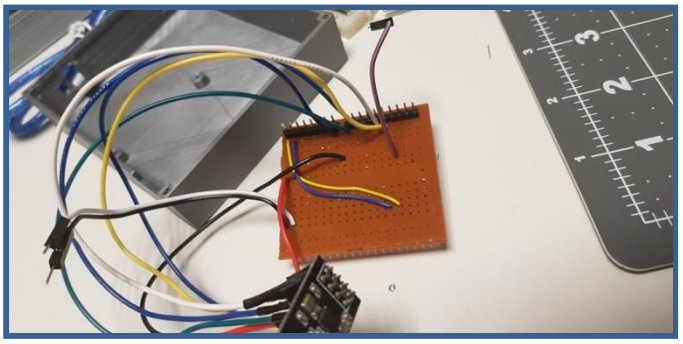


Carte complète :

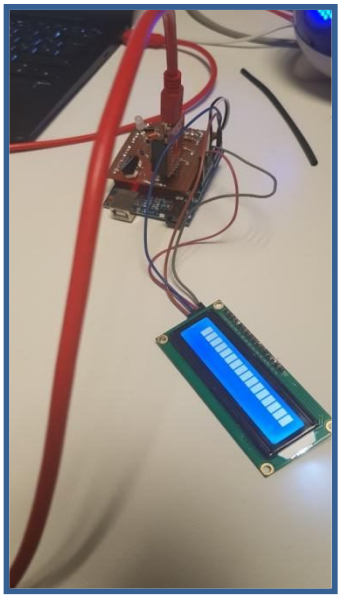


Soudure et branchement de l'antenne :



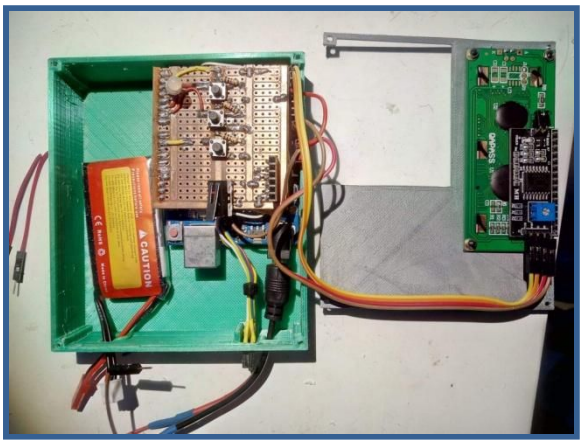


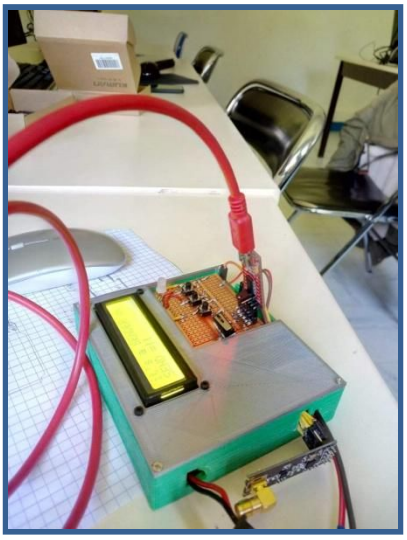
1er test :



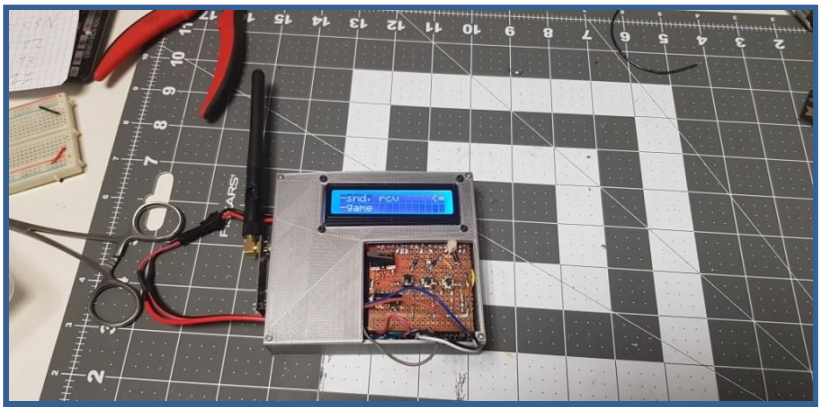
Test avec l'écran I2C

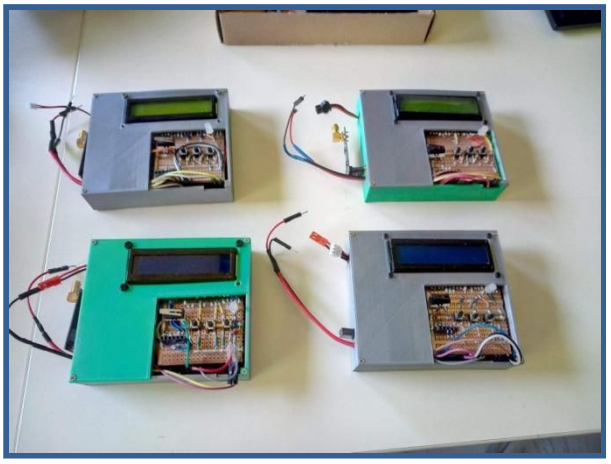
1er boitier terminé :





Puis les autres boitiers :

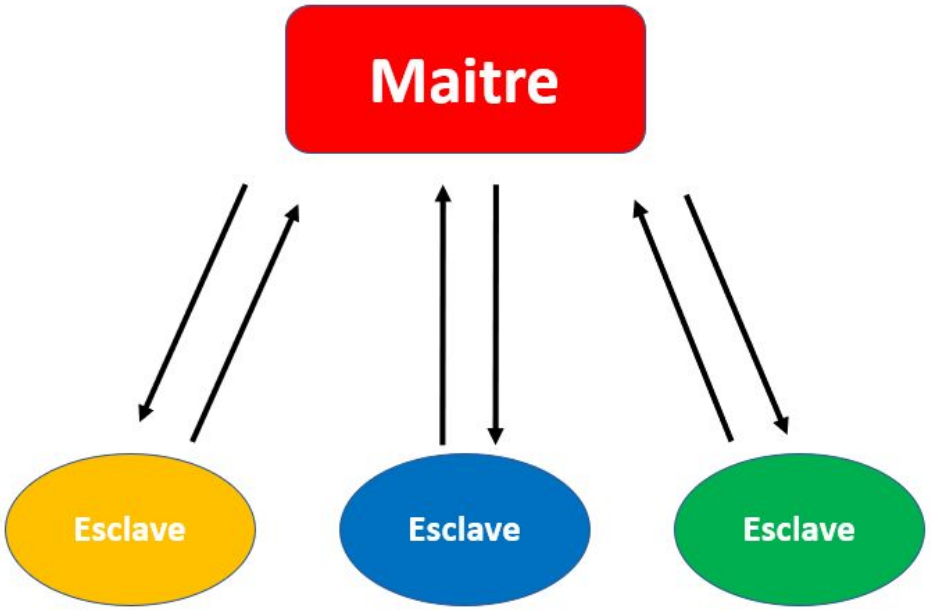




Communication entre boîtiers

Lors de la communication entre les boîtiers nous avons choisi d'instaurer une hiérarchie pour faciliter les communications, en effet nous avons, après avoir réfléchi à la question et la solution la plus simple et fiable que nous avons trouvée à été de restreindre au maximum les communications entre le boitiers. Les boîtiers peuvent donc uniquement communiquer avec le boitier "maitre". Ce boitier est celui qui va gérer tous les autres en leur disant par exemple quand commencer la partie, quand l'arrêter, en regroupant les scores de tous les boitiers esclaves pour calculer le vainqueur de la partie…

Le boitier maitre est le chef d'orchestre de notre système, c'est exclusivement avec lui que tout le monde communique et c'est exclusivement lui qui communique avec tout le monde.



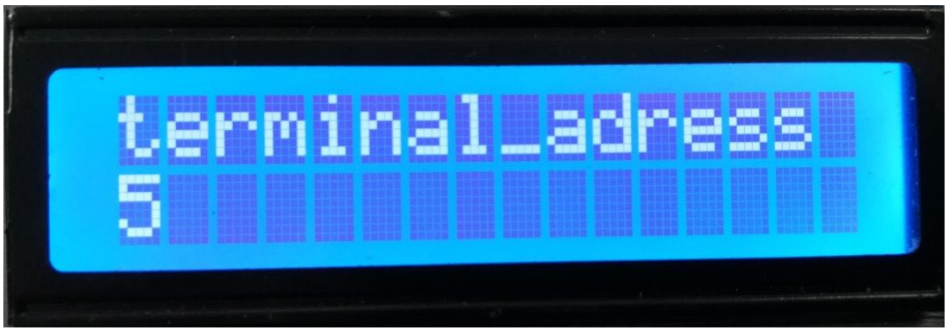
Les données ne transitent pas d'esclaves à esclaves mais uniquement de maitre à esclaves ou d'esclaves à maitre

Réseau sans fil

La contrainte principale de la communication entre les différents boîtiers est bien sûr leur mode de communication. En effet ici nous parlons bien de moyen de communication sans fil entre différents boîtiers (et donc entre différents microprocesseurs qui devront tous coordonner leurs actions pour pouvoir agir de manière cohérente entre eux). Dans ce but, les 3 principales solutions techniques pour permettre cette communication à distance étaient : la communication WiFi, la communication Bluetooth, ou la communication radio. La communication Bluetooth a été rapidement écarté pour la simple raison qu'elle ne permet que de faire communiquer 2 équipements différents entre eux. Dans notre cas on parle d'un réseau de boîtier qui se composera d'au moins deux équipements mais très probablement plus que cela. Le Bluetooth ne répondait donc pas du tout à cette exigence. Le WiFi constituait une solution bien plus avantageuse sur le papier mais un problème se pose rapidement. La portée de la communication WiFi. Généralement on parle de quelques dizaines de mètres dans un environnement dégradé (comme dans une maison). Or cette portée semble trop faible pour une utilisation dans le cadre d'une partie de paintball qui peut s'étendre sur des distances de plusieurs dizaines voire centaines de mètres. Le choix le plus pertinent c'est donc alors porté sur la communication par radio, qui permet des communications sur de grandes portées et (théoriquement) sans limitation de boîtiers utilisés puisqu'il faudra ici écrire notre propre protocole de communication.

L'équipement radio utilisé pour émettre et recevoir les données sera donc les NRF24, ces antennes permettent d'émettre et de recevoir les données sous la forme de tableau de char (en langage C++). Il a donc fallu créer un ensemble de fonctions permettant de mettre en forme ces tableaux de char et de les déchiffrer pour en extraire les informations utiles.

Dans un premier temps il est primordial de pouvoir identifier chaque boîtier, en effet les données émises par une antenne se retrouvent toutes en transit "sur les ondes" et sont accessibles pour n'importe quelle antenne qui capte cette bande de fréquences. Il faut donc pouvoir déterminer le destinataire de chaque message envoyé, pour cela nous avons mis un point un système d’adressage pour chaque boîtier, de cette manière chaque boitier pourra adresser précisément un autre en évitant que les autres interprètent le message reçu comme étant un message leur étant adressé. Les adresses pour les boîtiers esclaves allants de 5 à 253 inclus, en effet certaines adresses sont réservées pour des usages précis. L’adresse 254 sera par exemple l’adresse mise automatiquement à un boitier qui se revendique maître (qui lance une partie), l’adresse 255 est l’adresse qui permet d’envoyer un message à tout le monde (tous les boitiers sont réglés pour interpréter leur adresse mais aussi l’adresse broadcast (255)). Et enfin les adresse de 0 à 4 sont réservé à potentiel usage futur.



Les messages transmis par les antennes sont comme dits précédemment des tableaux de char (des tableaux d'octets). Pour faciliter la création et l'interprétation de ces tableaux nous avons mis au point une structure nommée "trame". Dans une trame on stocke toutes les informations d'un tableau de char mais sous une forme qui facilite grandement leur utilisation, cela évite de devoir constamment aller chercher dans le tableau la valeur souhaitée, ici on a plus qu'à accéder directement à l'information voulut.

```
typedef struct    trame_s
{
  unsigned char   adress;
  unsigned char   adress_to;

  unsigned char   secure;

  unsigned char   size_trame;
  unsigned char   number_command;

  unsigned char     data[10][4];

}                 trame_t;
```

La trame se décompose ainsi :

- adress : C'est l'adresse du boitier qui a émis ou qui veut émettre la trame
- adress_to : C'est l'adresse du boitier qui doit recevoir la trame
- secure : C'est un paramètre qui indique si le boitier qui reçoit la trame doit envoyer un accusé de réception (fonctionnalité pas encore au point)
- size_trame : C'est le nombre de char qui compose la trame, on peut l'utiliser comme une sécurité pour vérifier l'intégrité de la trame reçues
- number_command : Le nombre de commandes dont la trame est constituée (cette information est généré dans la fonction create_trame mais ne transite pas sur les ondes)
- data[10][4] : C'est les commandes dont est constituée la trame. Chaque tableau de taille 4 est une command, le nombre de commandes maximales est arbitrairement limité à 10

Afin de créer une trame nous avons fait une fonction à paramètres variables ce qui nous permet de créer des trames avec un nombre de commandes sur mesure. Pour cela il faut envoyer la trame à configurer, puis un tableau contenant le protocole (mon adresse, l'adresse du destinataire, le mode secure et la dimension de la trame seront automatiquement saisis par la fonction), puis un liste de commandes de 0 à 10 (voir plus) de taille 3 unsigned char.

```
int  create_trame(trame_t *t, unsigned char network[4], ...)    // work in progress...
{
  int8_t             i;
  va_list            arg;                // create list arg
  char*              tmp_data;

  va_start(arg, network);           // init start pointer arg_list to *p pointer

  t->adress     = network[0];
  t->adress_to  = network[1];
  i             = 0;

  while (va_arg(arg, unsigned char*)[0] != END_COMMAND)                                  // count number of arguments
    i++;

  if (i > 10)
    return (1);   // bugs

  va_end(arg);

  t->number_command = i;
  t->size_trame     = 4 + (t->number_command * 3);
  t->secure         = 1;

  va_start(arg, network);
  for( i = 0; i < t->number_command; i++)
  {
      //strncpy(t->data[i], va_arg(arg, unsigned char*), 4);

      tmp_data = va_arg(arg, unsigned char*);

      t->data[i][0] = tmp_data[0];
      t->data[i][1] = tmp_data[1];      // FUCK YOU STRCPY AND STRNCPY, YOU CAN'T WRITE unsigned char with 0 decimal value
      t->data[i][2] = tmp_data[2];              // FUUUUUUCKK !!!!!!!!!!
      t->data[i][3] = '\0';       // check '\0' is present or not

      //Serial.println(decompress_char(t->data[i][2]));
  }
  va_end(arg);
  return (0);       // no bugs
}
```

Une fois créer la trame est prête à être envoyé. Pour cela il faut savoir que les antennes RF24 fonctionnent à la fois en émission et en réception, il faut donc manuellement changer le mode de l'antenne grâce certaines fonctions (ces initialisations peuvent prendre plusieurs ms).

```
void  radio_init_sender(const byte address[6])
{
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}

void  radio_init_receive(const byte address[6])
{
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
```

Déroulement d'une partie

Avant de pouvoir lancer une partie il y a tout d'abord une phase d'attente lors de laquelle le maître va patienter pendant que tous les joueurs vont rejoindre la partie. Pendant cette phase les boîtiers esclaves doivent se connecter au boitier maitre. Ces deux cas de figure sont présentés ci-dessous

: 

Esclave essayant de se connecter au maitre

Ici les esclaves cherchent à se connecter à une partie, pour cela il envoie là une trame à l'adresse 254 (celle du maitre) quand l'utilisateur appuie sur le bouton du milieu, dans cette trame se trouve une seule commande c'est celle de la demande d'intégration à la partie en cours. Si cette trame est bien reçue par le maitre, celui-ci est censé envoyer un accusé de réception à l'émetteur pour confirmer son ajout à la partie.

Esclave ayant rejoint une partie

Une fois que le boitier est connecté il attend de recevoir l'ordre de commencer une partie. Cet ordre viendra du maitre quand il aura décidé de lancer une partie.

Maitre attendant que les esclaves se connectent

Ici on voit l'interface côté maitre, il attend que les esclaves lui envoient des trames de demande de connexion, lorsqu'il en reçoit une il ajoute l'adresse de l'esclave à sa liste des joueurs et lui envoie un accusé de réception pour lui confirmer qu'il l'a bien ajouté à sa liste de joueurs.

Une fois que le maitre souhaite lancer une partie il presse le bouton du milieu et décide à quel jeu il veut jouer. Il va ensuite envoyer une trame qui indique le début de la partie a tous les esclaves (grâce à l'adresse 255 qui est un broadcast). Puis il lance sa partie localement. Tout comme les esclaves qui vont lancer leur partie localement quand ils recevront la trame de début de partie.

Durant la partie chaque boitier qu'il soit maitre ou esclave va agir de la même manière. Dans le cas du jeu Flag les joueurs doivent appuyer sur les boutons pendant 10 secondes pour capturer un drapeau, une fois les 10 secondes écoulées le boitier appartient à l'équipe représentée par le bouton en question et le boitier va localement compter les scores en fonction de qui possède le boîtier.

A la fin des parties tous les esclaves envoient au maitre leurs scores qu'ils ont calculés dans une trame et le maitre va alors faire une addition de tous les scores reçus. Il a alors les scores globaux de la partie et est capable d'afficher le gagnant de la partie. Il lui suffit donc d'afficher l'équipe gagnante ainsi que le score qu'elle a fait.

1/ Structure de l'interface :

Pour naviguer dans nos boitiers nous avons fait un système de frame (de fonction d'affichage de deux propositions de sous programmes, tel que "LED" afin de tester ce composant), et grâce à la variable "POS" nous savons quelle frame afficher et/ou quelle sous programme lancer.

exemple type d'une frame:

```
3
4 void  frame1 (int8_t  *frame )
5 {
6   lcd.clear ();
7
8   char* line1  = "-send data" ;
9   char* line2  = "-receive data"  ;
10
11  lcd.set_Cursor (0, 0);
12  lcd.printstr (line1 );
13  lcd.set_Cursor (0, 1);
14  lcd.printstr (line2 );
15
16  *frame  = 1;
17 }
18
```

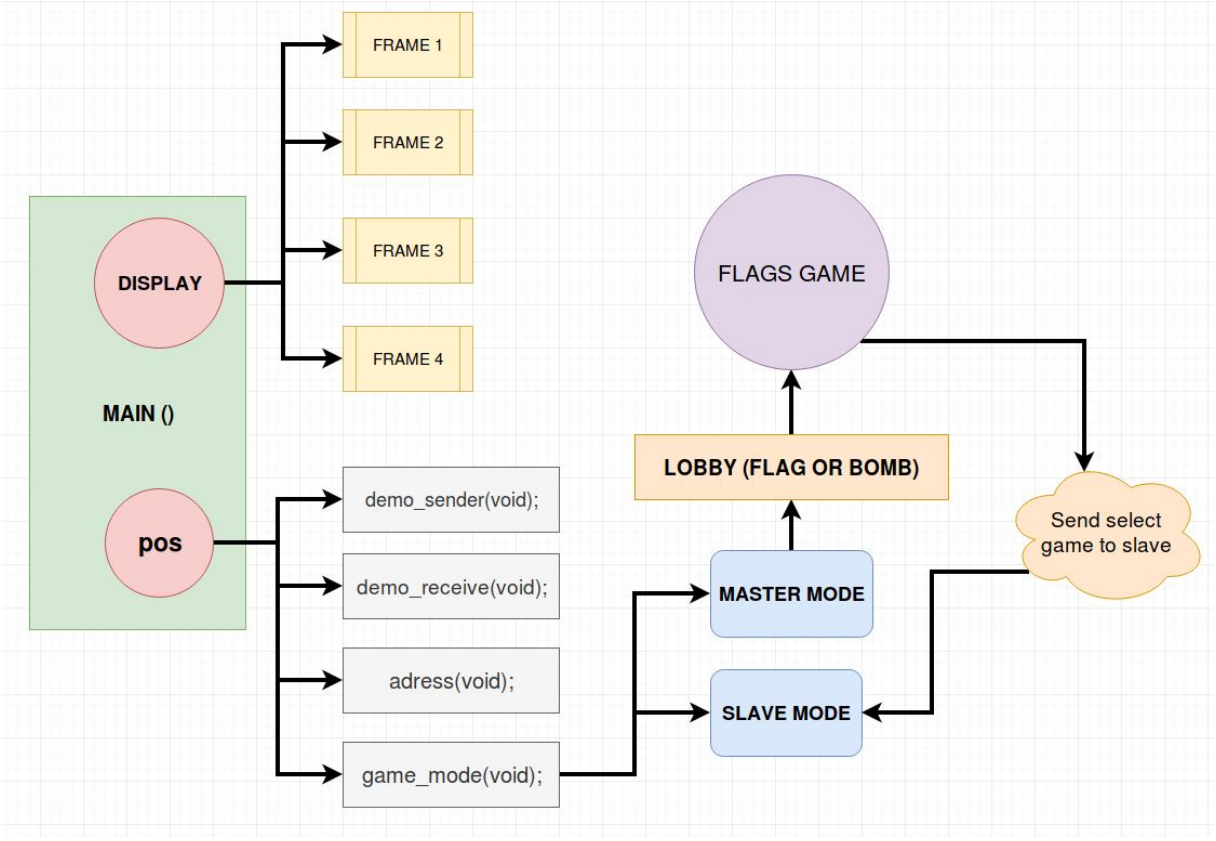
exemple type d'un item :

```
5  void   opt_demo (void )
6  {
7         int8_t  bt1, bt2, bt3;
8
9         lcd.clear ();
10        lcd.print (" it's a demo "  );
11        do
12        {
13            key_loop (&bt1, &bt2, &bt3);
14
15
16            lcd.backlight ();     // set li
17        } while  (!bt3);
18        lcd.clear ();
19 }
```

Ici la fonction key_loop() revoie dans les boutons 1, 2 et 3 un 1 si et seulement si un bouton est relâché, sinon les boutons seront mis à 0 automatiquement.

Si le bouton 3 est à 1 alors on casse la boucle infinie et on retourne au menu du MAIN().

Voici un aperçus de l'ensemble des interaction des fonction dans le programme :



Conclusion

Notre projet aura donc vu l'aboutissement d'un système mécatronique complet, en effet dans celui-ci les trois domaines de cette discipline ont été représentés (mécanique, électronique, mécanique). En effet nous avons dans ce projet eu l'occasion de réaliser un boitier plastique pour y placer notre système électronique qui lui même permettait d'exécuter le programme informatique. Le but initial du projet qui était de faire des jeux interactif entre différents boitier qui communique par liaison sans fil est rempli bien que de nombreuses amélioration restent à ce jour possible. D'un point de vu informatique le programme pourrait être amélioré en ajoutant de nouveau jeux ou bien en instaurant un système d'accusé de réception automatique après la réception d'une trame (comme le protocole TCP/IP). D'un point de vu électronique les boîtiers actuel sont uniquement des prototypes qui peuvent provoquer des soucis notamment lors de la communication avec les antennes RF24 (comme des problèmes de connectique) et une bonne piste d'amélioration serait par exemple de créer par usinage des nouveaux circuit imprimé.

Ces pistes d'amélioration devront sûrement être prises en compte pour la suite du projet.