M2M IoT : Biathlon Laser

Prototype

GAUCHY Anthony

Enseignant : Didier Donsez

Master 2 - Génie informatique

Année universitaire 2016 - 2017

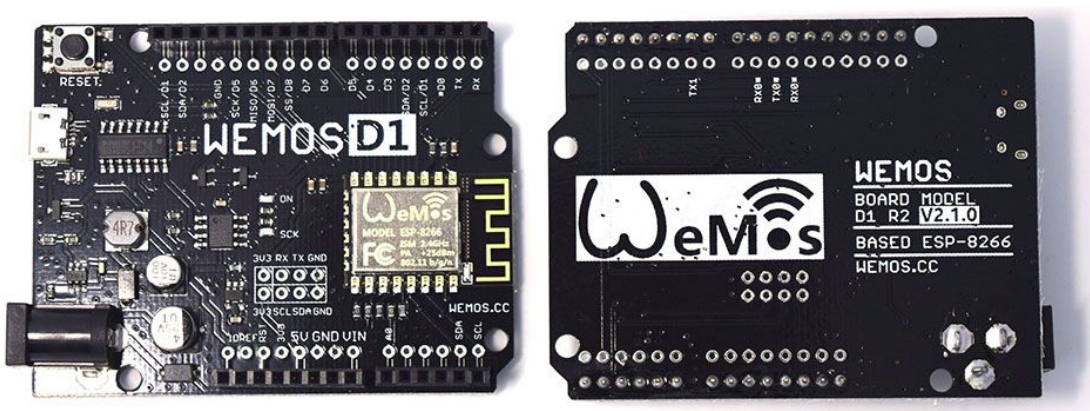
Introduction :

Ce mini-projet avait pour but la réalisation d'un jeu laser, pouvant par la suite être adapté pour réaliser un jeu de biathlon laser par exemple. Dans ce cas là, le but de ce projet restera la mise en place d'un prototype pour observer la faisabilité du projet ainsi que la prise en main d'outils IoT.

L'objectif optimal serait donc un jeu de biathlon laser connecté qui permettrait en plus de simplement jouer, de pouvoir transmettre un nom pour chaque joueur afin de pouvoir stocker les scores de chacun indépendamment.

1 Le matériel :

Wemos D1 :



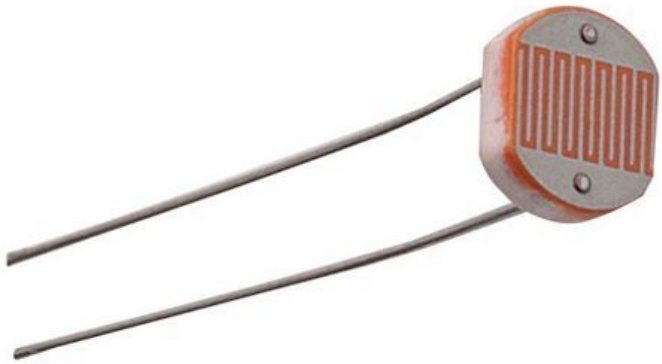
https://www.wemos.cc/product/d1.html

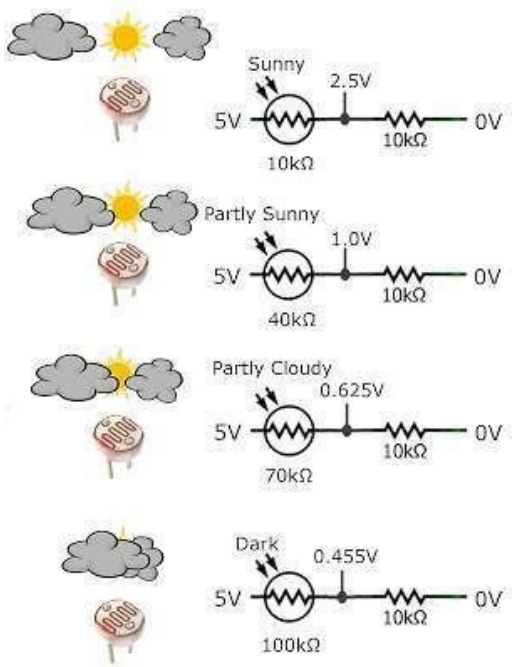
La carte Wemos D1 est une carte compatible Arduino basée sur l'ESP-8266. L'un des points forts de cette carte est le fait qu'elle dispose d'une compatibilité wifi. Elle ne dispose nativement par contre que d'une entrée analogique.

Servo 9g :

Ce petit moteur de type servo permet une rotation à 180°. Il fonctionne très bien avec les cartes Arduino grâce à sa compatibilité 5v. Nous nous servirons de ce moteur pour faire monter et descendre notre cible.

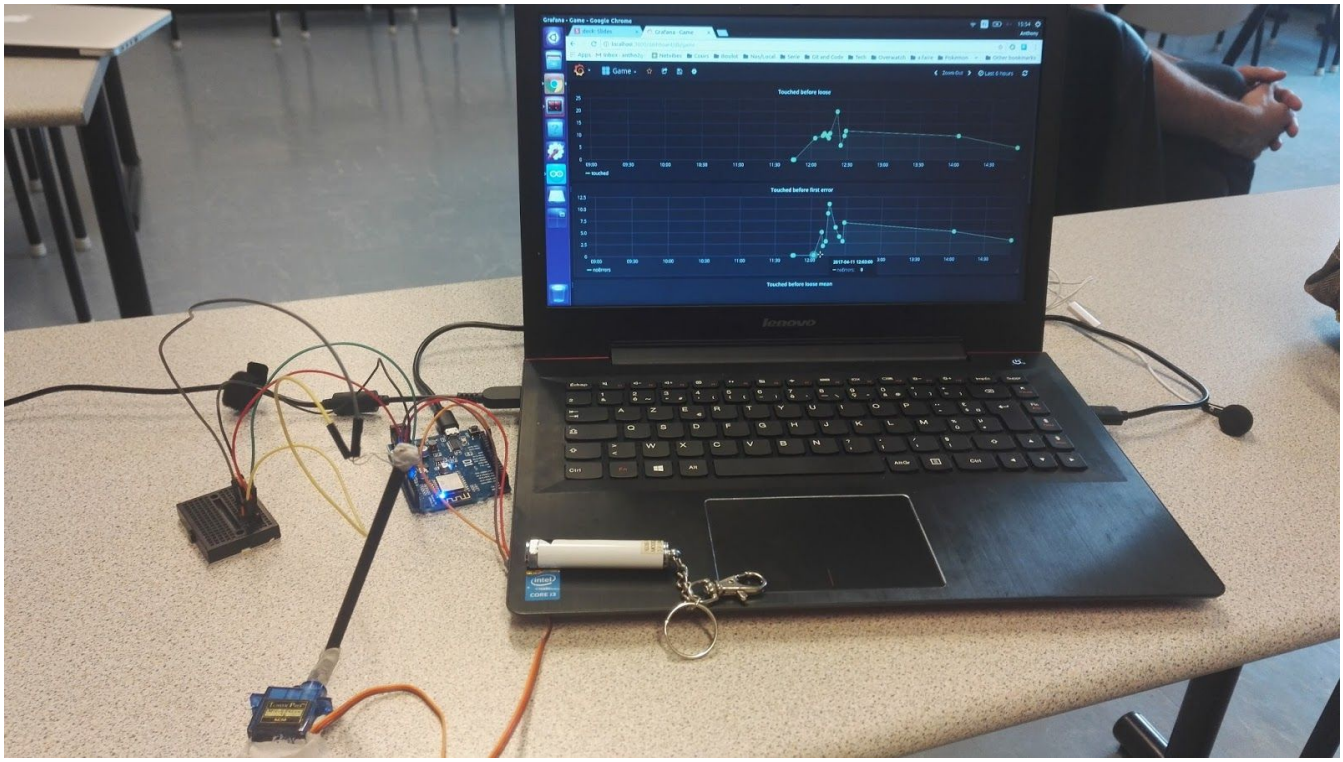
Photoresistance :





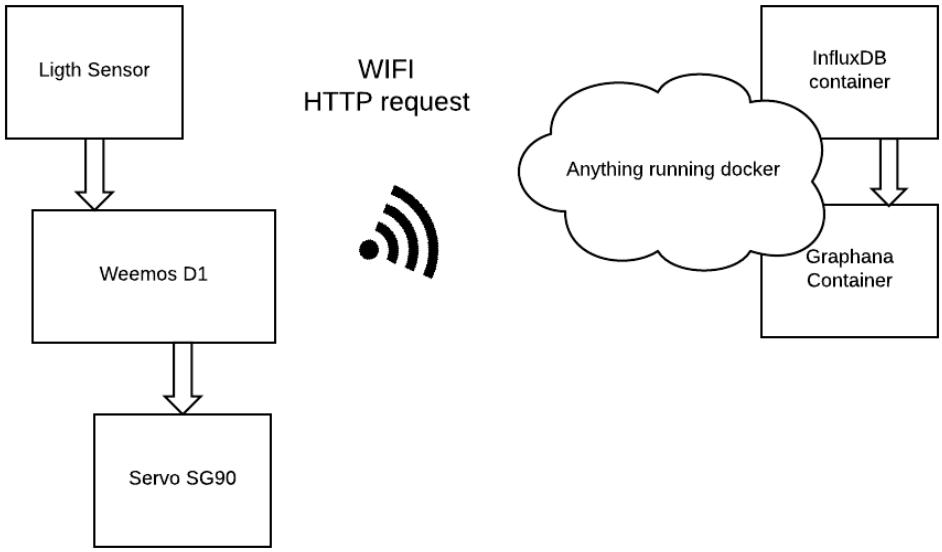
Cette photorésistance fera office de cible, en effet c'est elle qui détectera si le laser la touche. Pour ce faire, la résistivité de la photorésistance varie selon l'intensité lumineuse qu'elle reçoit.
Comme le montre ce schéma, il faut réaliser un pont diviseur de tension avec une autre résistance classique. Il faut choisir la bonne résistance pour obtenir des valeurs dans la zone qui nous intéresse. Ici, nous ne voulons pas voir la différence entre la nuit et le jour mais entre le jour et le laser. J'ai donc pour cela utilisé une résistance de 300 ohm. Le choix de cette résistance fait partie des difficultés que j'ai rencontré dans la réalisation de ce projet. En effet, si l'on joue en plein jour avec une résistance de 10k ohm on ne peut pas faire la différence entre la cible au soleil et la cible touchée par le laser.

2 Montage :



Voici une photo du montage final réalisé (une vidéo est disponible en cliquant sur ce lien : https://youtu.be/nmw8IzrO6qw). On peut voir sur la gauche, le pont diviseur de tension réalisé à l'aide d'une photorésistance (relié grâce aux fils), d'une plaque d'essai et de fils de prototypage. Au premier plan, on observe la cible réalisée à l'aide du moteur servo, d'une chute de découpeuse laser et de la photorésistance reliée à la plaque d'essai, le tout tenu avec de la pâte à fixe. Un pointeur laser classique a été utilisé car la carabine laser intelligente n'a pas été développée. La carte Wemos est reliée au pont diviseur de tension (input) et au moteur servo (output) et est alimentée par le pc portable. Le pc portable héberge aussi les conteneurs docker InfluxDB et Graphana (détails plus loin).

3 Architecture et outils :



Voici le schéma de mon architecture. La carte Wemos communique directement avec la base InfluxDB en HTTP. Ensuite, le conteneur Graphana récupère les informations pour les afficher de manière plus claire.

InfluxDB :

Ce container docker (https://github.com/tutumcloud/influxdb) a été utilisé durant le projet, il est aujourd'hui déprécié en faveur du container officiel.

Voici comment a été lancé le container :

```
docker run -d -p 8083:8083 -p 8086:8086 \
       -e PRE_CREATE_DB="wadus" -e ADMIN_USER="root" -e INFLUXDB_INIT_PWD="somepassword"
       \
       --name influxdb \
       tutum/influxdb
```

Cela m'a permis de lancer mon container avec une base de créée (nommée "wadus"), un utilisateur et un mot de passe. Pour vérifier que mon docker était fonctionnel, je me suis connecté en ligne de commande à ma base de données :

```
docker exec -ti influxdb /usr/bin/influx
```

Et, j'ai ajouté des données de test dans ma base :

```
Connected to http://localhost:8086 version 1.0.0
InfluxDB shell version: 1.0.0
> use wadus
```

```
Using database wadus
> INSERT cpu,host=serverB,region=us_west value=0.05
> INSERT cpu,host=serverA,region=us_west value=0.63
> INSERT cpu,host=serverA,region=us_west value=0.5
```

Enfin, j'ai vérifié que ces données soient correctement ajoutées.

```
> select * from cpu
name: cpu
---------
time                  host   region value
1489502821817283769   serverA us_west 0.64
1489502949813150613   serverA us_west 0.8
1489502952702119197   serverA us_west 0.9
1489502953721946387   serverA us_west 0.9
1489502956760710962   serverA us_west 0.3
1489503202518273042   serverA us_west 0.4
1489503204728894930   serverA us_west 0.2
1489503207033608966   serverA us_west 0.1
1489503209653382996   serverA us_west 0.9
1489503484664419631   serverB us_west 0.9
...                   ...
...                   ...
...                   ...
...                   ...
...                   ...
```

Grafana :

Pour Grafana, j'ai utilisé le container docker officiel (https://github.com/grafana/grafana-docker) que j'ai lancé en le connectant directement avec mon container InfluxDB lancé au préalable :

```
docker run -d -p 3000:3000 \
       --link influxdb:influxdb \
       --name grafana \
       grafana/grafana
```

Ceci m'a permis de n'avoir aucune manipulation à faire pour relier les containers docker InfluxDB et Grafana.

J'ai ensuite créé un Dashboard à l'aide des données d'exemples que j'avais ajouté dans ma base (voir ci dessus).

Node-red :

J'ai ensuite créé et visualisé un serveur node-red avec le container officiel :

```
docker run -it -p 1880:1880 --name mynodered nodered/node-red-docker
```

Par la suite, j'ai pris le parti de ne pas utiliser node-red car j'ai utilisé une librairie qui permettait de relier directement la carte à une base Influx via HTTP. Le défaut de mon implémentation est par contre, l'obligation d'utiliser une base InfluxDB.

4 Code et résultats :

Le code de mon projet au complet est disponible dans le dépôt github suivant : https://github.com/agauchy/m2mIoTLaserGame.

En pratique, la fonction setup() de ma carte la connecte au réseau wifi, à la base de données et au servo.

Ensuite, chaque itération de la fonction *loop()* correspond à une partie du jeu. Pour lancer la partie, il faut toucher une fois la cible. Puis, la cible se lève pendant 3 secondes (ce temps diminue à chaque tour et tend vers 1 seconde), si la personne touche la cible il marque un point sinon c'est une erreur. Le jeu continue tant que moins de 3 erreurs ont été commises. A la fin de la partie, la carte envoie via https deux entrées dans la base InfluxDB, une pour le nombre de cibles atteintes, et une pour le nombre de cibles atteintes avant la première erreur.

Code pour tester si la cible est touchée :

```
bool targetIsTouched() {
  int sensorValue = analogRead(A0);
  if(sensorValue > 300){
    return true;
  }
  return false;
}
```

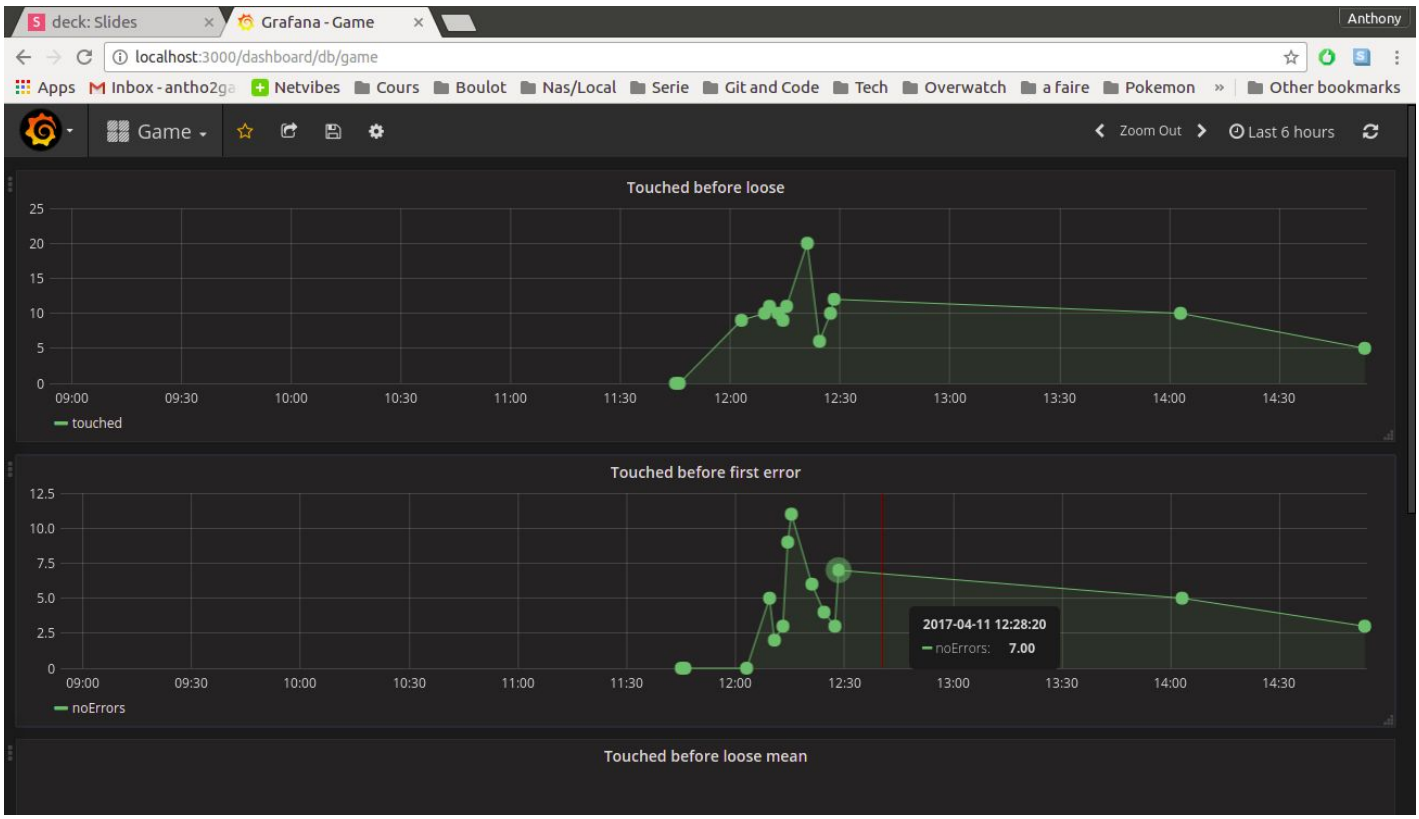
Code pour tester 3 secondes ou moins :

```
bool testLigth3sec(int nbTour) {
  float i;
  for (i = 0; i <= 1000.0 + (2000.0/nbTour); i++) {
    if (targetIsTouched()) {
      return true;
    }
    delay(1);
  }
  return false;
}
```

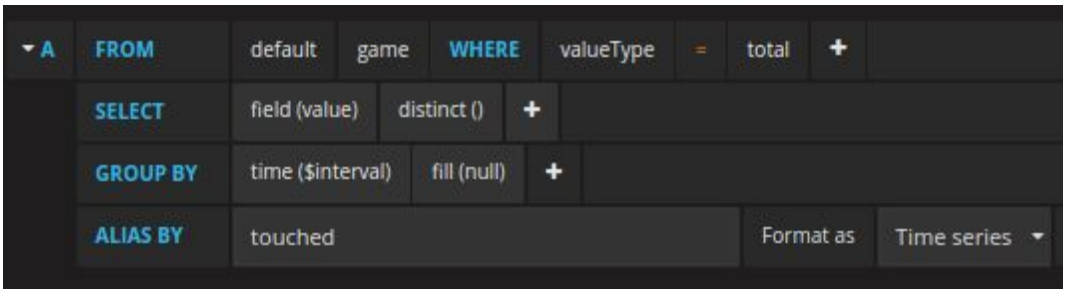
Code pour animer le moteur servo :

```
void makeServoUp(Servo myservo) {
  int pos;
  for (pos = 0; pos <= 75; pos += 1) // goes from 0 degrees to 180 degrees
  { // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
...
```

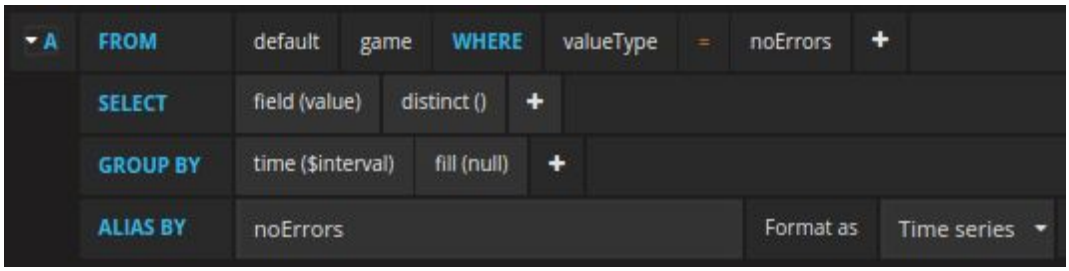
Suite à plusieurs parties, voici le résultat obtenu sur Grafana :

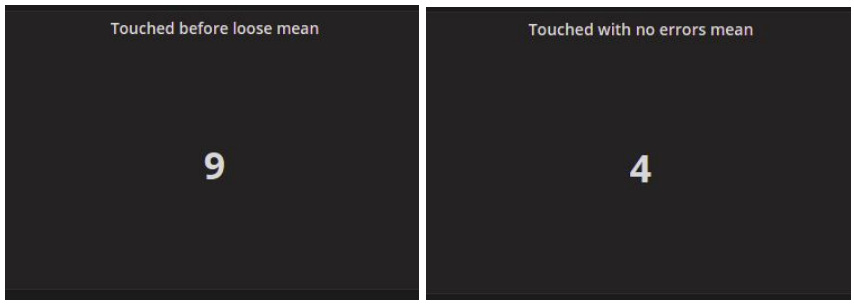


Voici la configuration de la requête pour le diagramme “Touched before loose” (correspond au nombre total de cible touchées avant d’avoir fait **3** erreurs) :



Voici la configuration de la requête pour le diagramme “Touched before first error” (correspond au nombre total de cible touchées avant d’avoir fait **1** erreur) :





Une vidéo du système en fonctionnement et de la mise à jour de Grafana est disponible ici : https://youtu.be/nmw8IzrO6qw.

5 Amélioration envisageables et conclusion :

La principale faiblesse de ce prototype est le fait que le score reste anonyme. Pour changer cela, la solution imaginée serait de créer une carabine laser qui serait elle aussi connectée. Grâce à son téléphone, on rentrerait son nom dans la carabine. La carabine communiquerait ensuite avec le système de cible pour envoyer le nom et lancer la partie. Ainsi, on pourrait ajouter un tag “name” dans notre requête à InfluxDB et utiliser ce tag pour réaliser un “Group By name” dans les requêtes Grafana afin d’obtenir une courbe par joueur.

Ce projet m’aura permis d’en apprendre beaucoup sur les principes et les outils liés à l’IoT malgré qu’il ne soit pas complètement achevé.