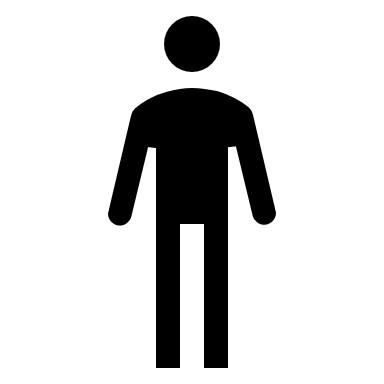
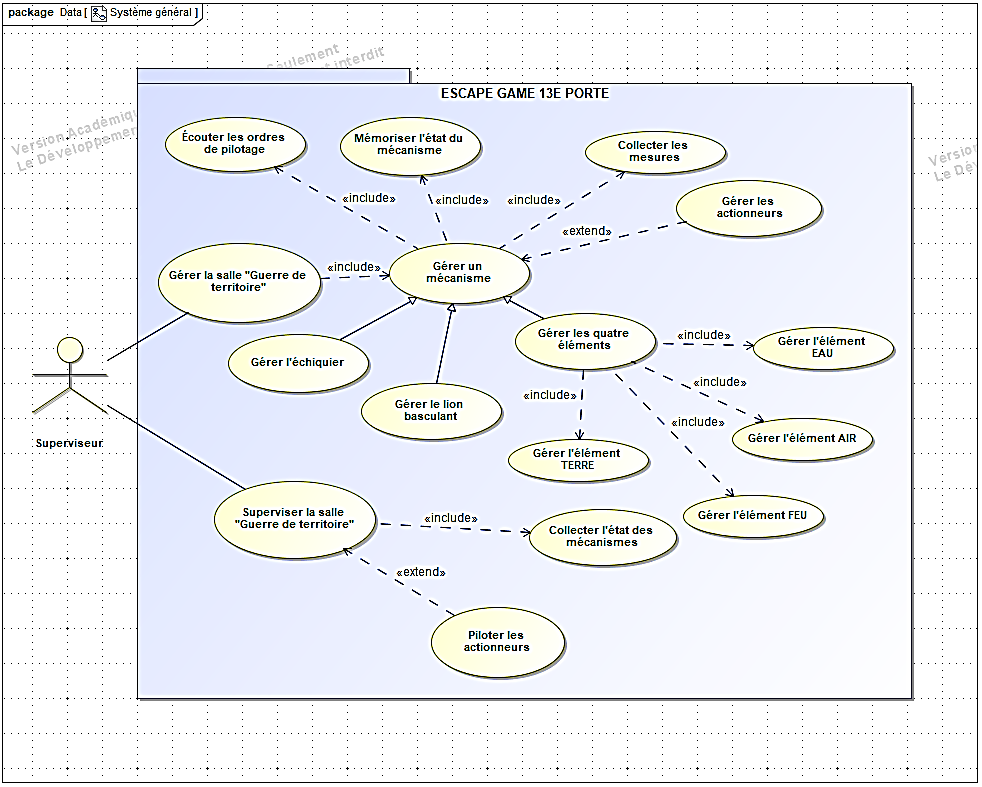
Revue de Projet

***Présentation du projet***

La société *La 13EME PORTE* est une enseigne d’Escape Game. Les gérants de la société souhaitent que le système technique d’une salle actuellement en place soit recréé entièrement afin de corriger les erreurs de conception et les différents bugs existants. De plus il est demandé d’ajouter plusieurs fonctionnalités permettant d’améliorer le travail du superviseur.  
  
Le système de la salle s’appuie sur 9 mécanismes et il devra associer à chacun des mécanismes un Arduino dédié.

Un mécanisme pourra collecter des mesures par l’intermédiaire d’actionneur (capteurs, boutons, etc…) placé sur la carte Arduino et devra écouter les ordres de pilotage.  
De plus chaque mécanisme transmettra régulièrement par liaison I2C à la Raspberry les valeurs de ses différentes entrées.   
Chaque information reçue par la Raspberry sera enregistrée en base de données et installée sur un serveur positionné dans le bâtiment par l’intermédiaire du poste de supervision.  
Sur ce poste de supervision on pourra visualiser l’état de la salle et piloter les actionneurs.

***Diagramme de cas d’utilisation***

*********Répartition des tâches***

* Etudiant 1 : Constantin
* Gérer le Mécanisme 4
* Gérer le Mécanisme 8
* Gestion du bus I2c entre Arduino et Raspberry :

Envoie des ordres de pilotage de la Raspberry à l’Arduino et les traiter

Envoi des mesures et des états des actionneurs sur la Raspberry et les traiter

* Etudiant 2 : Corentin
* Créer la base de données
* Installer les serveurs Apache et MySql
* Gérer le Mécanisme 6
* Gérer le Mécanisme 7
* Récupérer la valeur des capteurs et l’état des actionneurs de la Raspberry au serveur
* Etudiant 3 : Joshua
* Configuration réseau du matériel
* Création d’une application Web sur l’affichage de l’état de la salle
* Gérer le Mécanisme 1
* Gérer le Mécanisme 2
* Gérer le Mécanisme 3
* Etudiant 4 : Thomas
* Création d’une application Web de pilotage des actionneurs
* Gérer le Mécanisme 5
* Gérer le Mécanisme 9
* Envoie des ordres de pilotage du PC de supervision à la Raspberry

Thomas

Joshua

Corentin

******

Constantin

Constantin

Thomas

Joshua

Corentin

**Application Web de pilotage des actionneurs**

**Application Web sur l’affichage de l’état de la salle**

***Matériels utilisés***



* Raspberry Pi 3B :

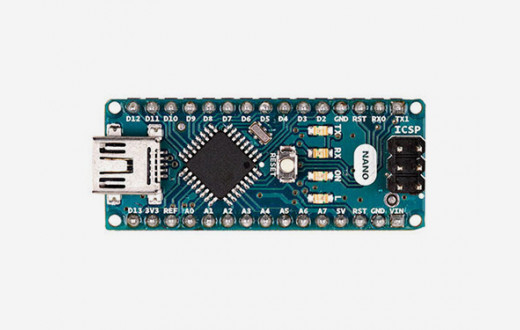
**Programmation sur Raspberry :**

Langage : Python

Nous utiliserons l’émulateur de Terminal PuTTY afin de pouvoir programmer directement depuis notre PC sur la Raspberry

* Arduino :

**Matériel Arduino pour la salle de jeu**

* Neuf Arduino Nano
* Quatre capteurs à effet hall A3144
* Module de commutateur de clé 3 pin 1NO1NC
* Afficheur 7 segments
* Capteur d’humidité
* Interrupteur à bascule
* Sept capteurs photosensibles
* Boutons poussoir
* Leds
* Capteur de fin de course
* Capteur de poids HX711

**Programmation sur Arduino :**

Langage : Arduino

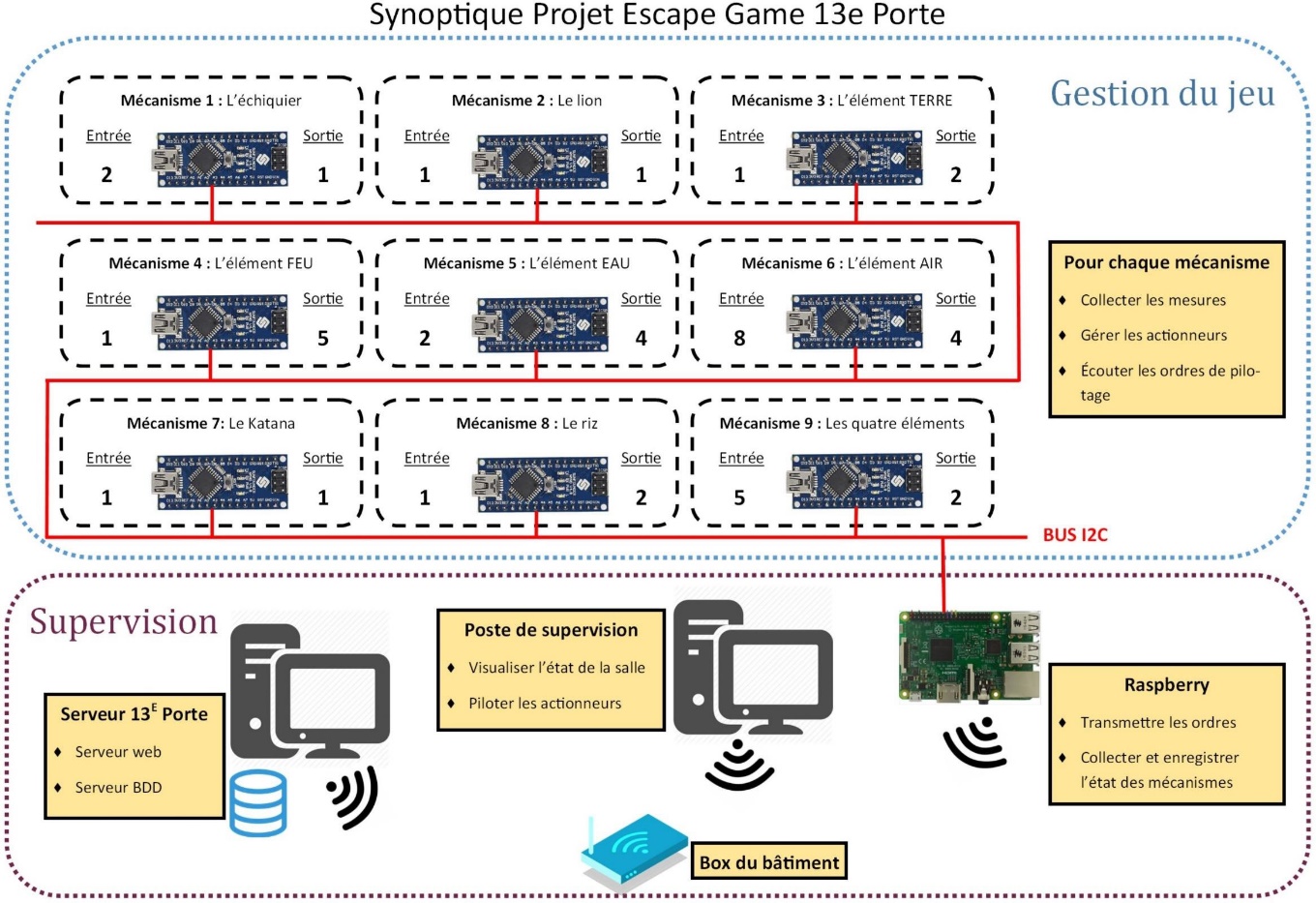
IDE : Arduino

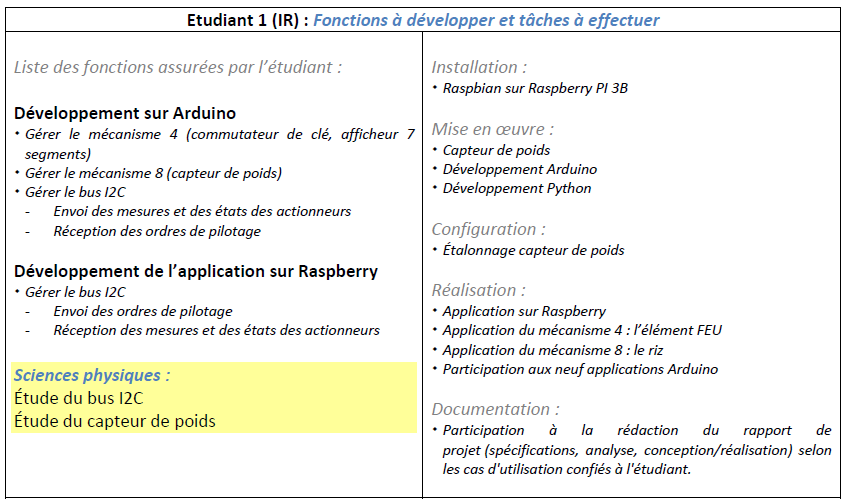
• Un poste informatique pour le serveur *13e Porte*

• Un poste informatique de supervision (éventuellement le même poste que le serveur)

***Ma tâche***

Ma tâche





***Liaison I2c***

Pour lier les Arduino à la Raspberry nous allons passer par le protocole i2C :   
- pour envoyer l’état des actionneurs et les valeurs des capteurs des Arduino à la Raspberry   
- pour envoyer les ordres de pilotage du PC de supervision de la Raspberry aux Arduino.

Le protocole I2C permet à plusieurs composants de dialoguer entre eux de manière bidirectionnelle mais en half-duplex uniquement. Et grâce à seulement trois fils : Un signal de donnée (SDA), un signal d’horloge (SCL), et un signal de référence électrique (Masse).

Les données sont transmises en série à 100Kbits/s en mode standard et jusqu’à 400Kbits/s en mode rapide.

Chaque périphérique sur le bus I2C est adressable, avec une adresse unique pour chaque périphérique du bus

Il y a un principe de maitre et esclave. Ici nous utiliseront le Raspberry Pi en tant que maitre et les Arduino en tant qu’esclave.   
Le maitre (la Raspberry) est le composant qui initialise un transfert, génère le signal d’horloge et termine le transfert. Dans notre cas il sera récepteur et émetteur.  
Les esclaves (Les Arduinos) sont les composants adressés par un maître. Dans notre cas ils seront récepteurs et émetteurs.



***Description des mécanismes***

* Mécanisme 4 :

Gestion de la clé (Elément FEU)

Description du Sous-système : Un interrupteur à clef est caché dans le décor, les joueurs doivent trouver la clef et la tourner dans l’interrupteur

Résultat :

1. Une LED témoin s’allume sur le tableau de contrôle
2. Une « Tête de Dragon » sort du plafond (S2). En effet, la désactivation de l’électroaimant a pour effet de libérer une trappe au plafond
3. Une machine à fumée (220 Volt) est allumée via un relais. Cette machine à fumée est dissimilée dans la « tête de dragon » afin que ce dernier crache de la fumée lorsqu’il apparait aux joueurs
4. 5 LED blanches s’allument afin au-dessus afin d’éclairer cette dernière
5. L’élément FEU (LED) s’allume sur la tablette à destination des joueurs

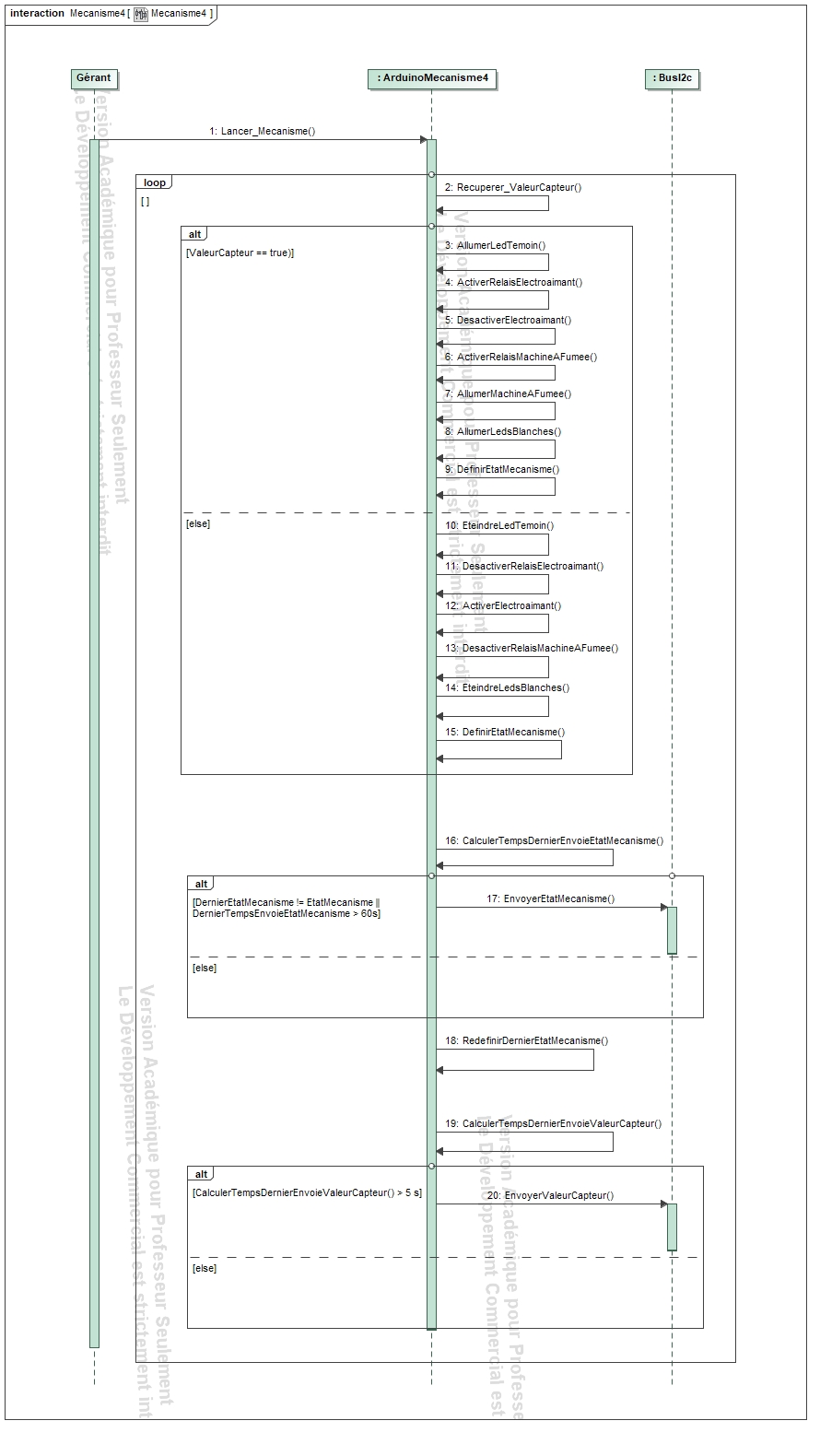
## Entrées :

* E1
  + Type de capteur : Interrupteur à clef
  + Référence : Commutateur de clé 3 pin 1NO1NC
  + Signal : numérique
  + Voltage : 0-5V

## Sorties :

* m\_dragon
  + Rôle : Activer/désactiver un électroaimant via un relais + LED de contrôle
  + Condition : Si E1 est à 1 alors activer la sortie, sinon, désactiver la sortie
  + Voltage : 0-5V
* m\_fumee
  + Rôle : Activer/désactiver une machine à fumée (220Volt) via un relais
  + Condition : Si E1 est à 1 alors activer la sortie, sinon, désactiver la sortie
  + Voltage : 0-5V
* m\_led
  + Rôle : Activer/désactiver 5 LED blanche câblées en parallèles
  + Condition : Si E1 est à 1 alors activer la sortie, sinon, désactiver la sortie
  + Voltage : 0-5V (5 Volts + résistance de 220 ohms)

* m\_feu
  + Rôle : entrée de la gestion des quatre éléments



* Mécanisme 8 :

# Gestion de la balance de riz et du tableau

Description du Sous-système : les joueurs doivent faire « une offrande » de 50 grammes de riz sur coupelle posée sur un capteur de poids. La condition est validée avec une précision de +/- 2 grammes.

Résultat :

1. La LED rouge s’éteint et la LED verte s’allume
2. Une LED témoin s’allume au panneau de contrôle.
3. Un électroaimant (12V) est désactivé via un relais (5V) libérant ainsi la chute d’un tableau (indice)

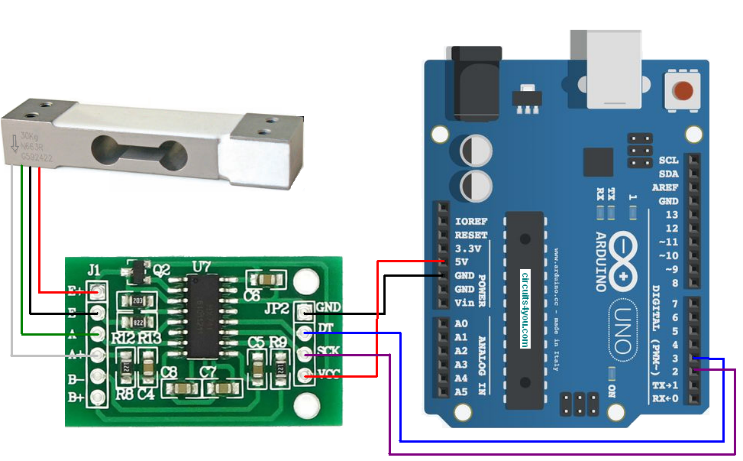
## Entrées :

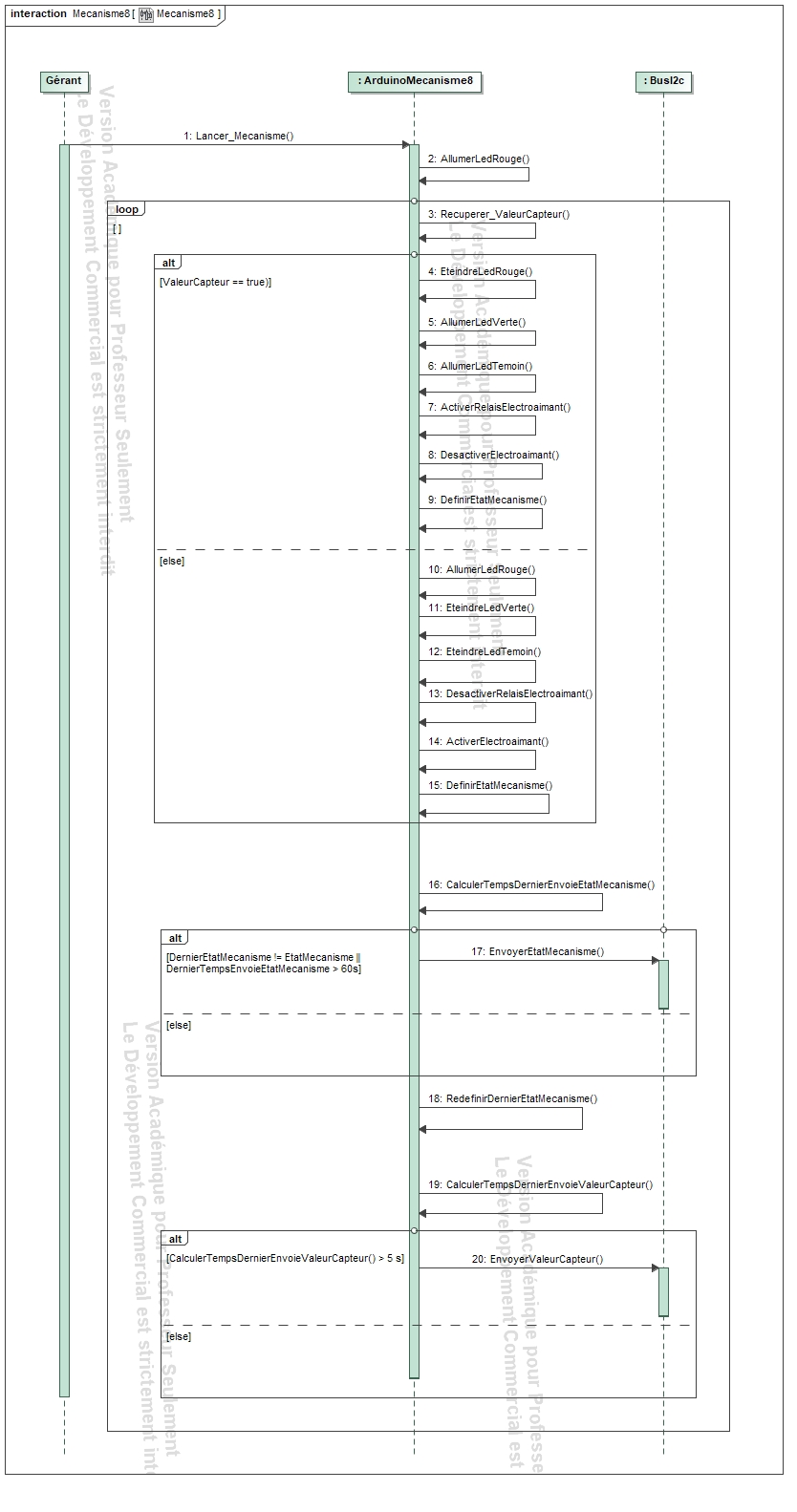
* E1
  + Type de capteur : capteur de poids
  + Référence : HX711
  + Signal : numérique
  + Voltage :  0-5V

## Sorties :

* m\_tableau
  + Rôle : activer/désactiver une électro aimant via un relais (tableau)
  + Condition : Si E1 est compris entre 48 grammes et 52 grammes alors activer la sortie, sinon, désactiver la sortie
  + Voltage : 0-5V
* m\_led
  + Rôle : Désactivation LED rouge et activation LED verte (Indicateur de réussite)
  + Condition : Si E1 est compris entre 48 grammes et 52 grammes alors éteindre la LED rouge et allumer la LED verte, sinon c’est l’inverse
* Voltage : 0-5V

## Remarques :

  
Schéma de branchement du capteur de poids HX711



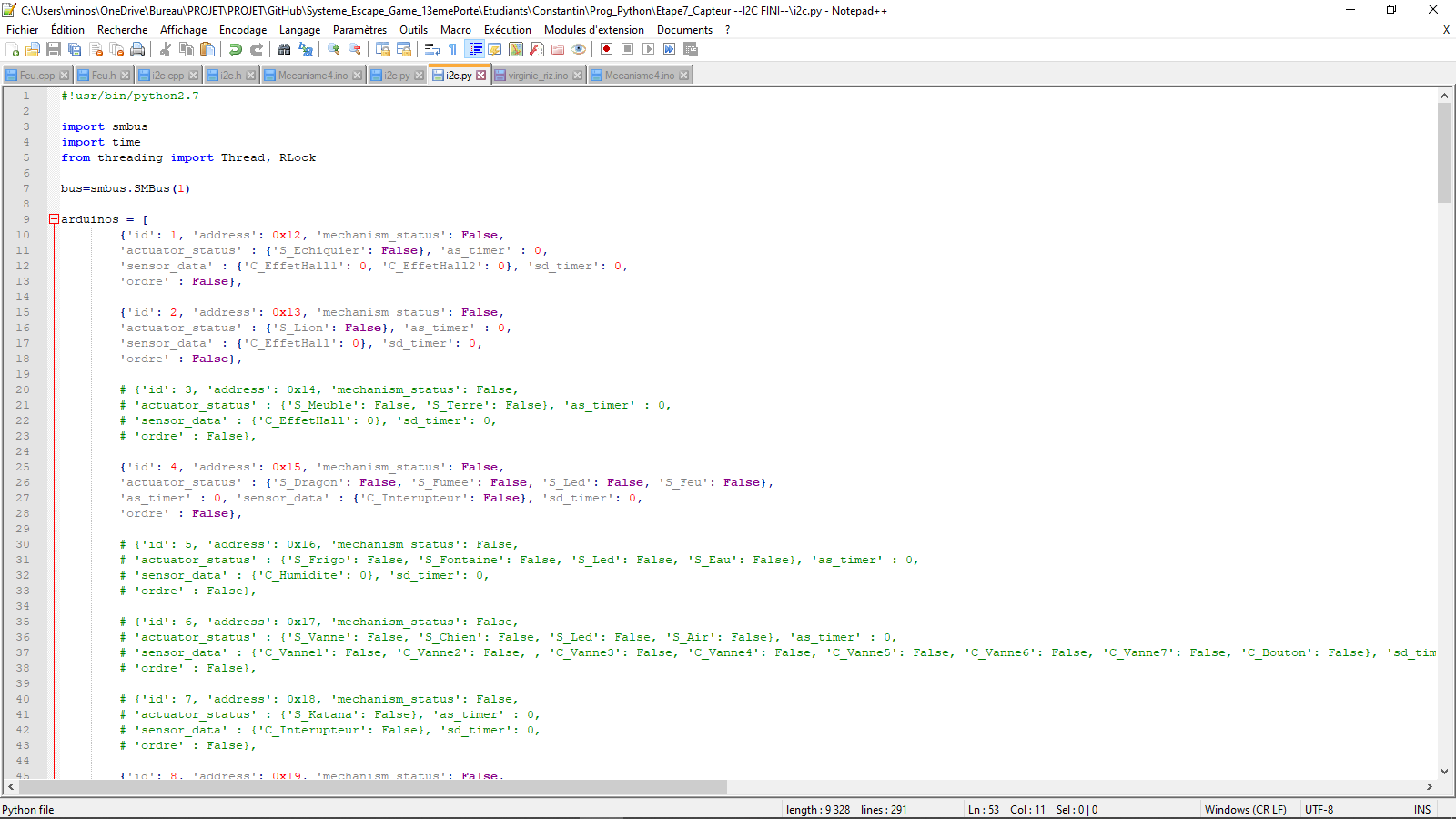
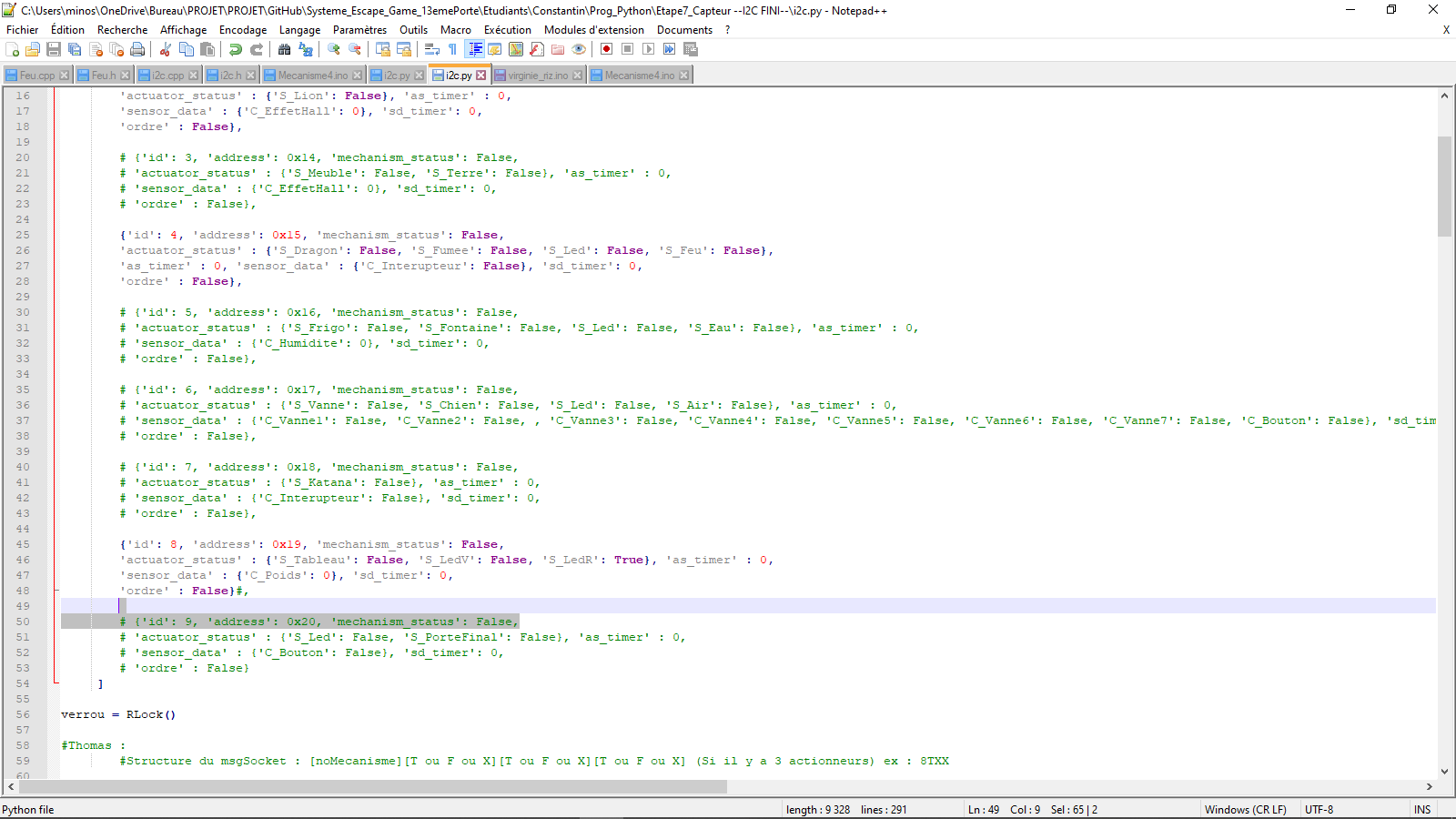
Script Python sur Raspberry :

Ce script python consiste à :

* Recevoir la valeur des capteurs de chaque mécanisme à chaque changement ou sinon toutes les 5s max. Et ensuite envoyer les valeurs à la BDD.
* Recevoir la valeur des actionneurs de chaque mécanisme à chaque changement ou sinon toutes les 60s. Et ensuite envoyer les valeurs à la BDD.
* Analyser l’état des mécanismes et les envoyer en BDD à chaque nouvel envoie des actionneurs.
* Envoyer les ordres reçus depuis l’application Web au mécanisme correspondant

Création d’une liste de 9 dictionnaires correspondant aux 9 Arduinos. Les clés de ces dictionnaires sont identiques.

Voici un fragment de la liste de dictionnaires :

  
 (…) // 7 autres dictionnaires  


Chaque dictionnaire Arduino possède les clés suivantes :

* « **id** » : correspond au numéro du mécanisme assigné à l’Arduino
* « **address** » : correspond à l’adresse esclave assigné à l’Arduino pour l’i2c
* « **mechanism\_status** » : correspond à l’état du mécanisme (valeur True/False)
* « **actuator\_status** » : correspond à l’état des actionneurs du mécanisme

C’est une liste des différents actionneurs (ayant chacun une valeur True/False).

* « **as\_timer**» : correspond à une valeur entière (<= 60)

Nécessaire à l’exécution du thread ASTimer pour l’envoie des valeurs actionneurs toutes les 60s

* « **sensor\_data** » : correspond à l’état des capteurs du mécanisme

C’est une liste des différents capteur (ayant chacun une valeur entière ou True/False).

* « **sd\_timer**» : correspond à une valeur entière (<= 5)

Nécessaire à l’exécution du thread SDTimer pour l’envoie des valeurs capteurs toute les 5s max.

* « **ordre** » : correspond à une valeur Boolean (True/False) si un ordre a été envoyé ou non.

Ce sont ces dictionnaires qui sont modifiés à chaque changement dans les mécanismes et c’est depuis eux que les informations sont envoyées à la BDD.

Afin de recevoir les informations des mécanismes en temps réel il a fallu utiliser des Threads. De sorte que toutes les Arduinos communique avec la Raspberry en même temps.  
Pour cela il a d’abord fallu créer le thread « **ArduinoCom** » qui correspond à la communication de la Raspberry avec une Arduino ; ensuite il a fallu assigner à chaque Arduino un thread ArduinoCom.

Le thread ArduinoCom consiste à effectuer en boucle les instructions suivantes :

Une image contenant capture d’écran

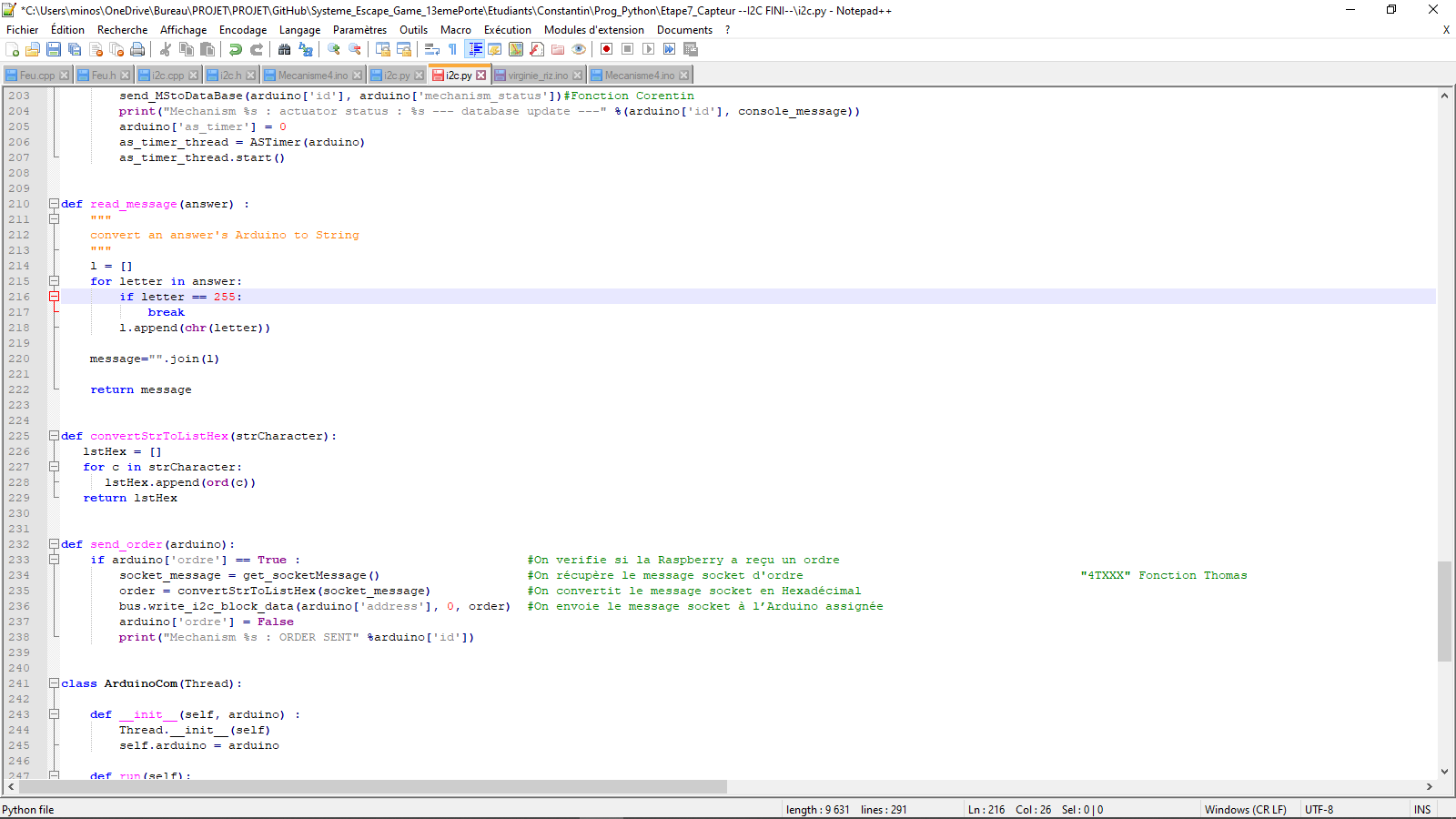
Description générée automatiquement

1. **Si un ordre a été donné depuis l’application Web l’envoyer à l’Arduino**
2. **Récupérer le message i2c depuis l’Arduino**
3. **Suivant le message i2c, gérer l’état des actionneurs**
4. **Suivant le message i2c, gérer la valeur des capteurs**

**1. Si un ordre a été donné depuis l’application Web l’envoyer à l’Arduino**

On appelle la fonction send\_order().

Elle exécute principalement les instructions suivantes :



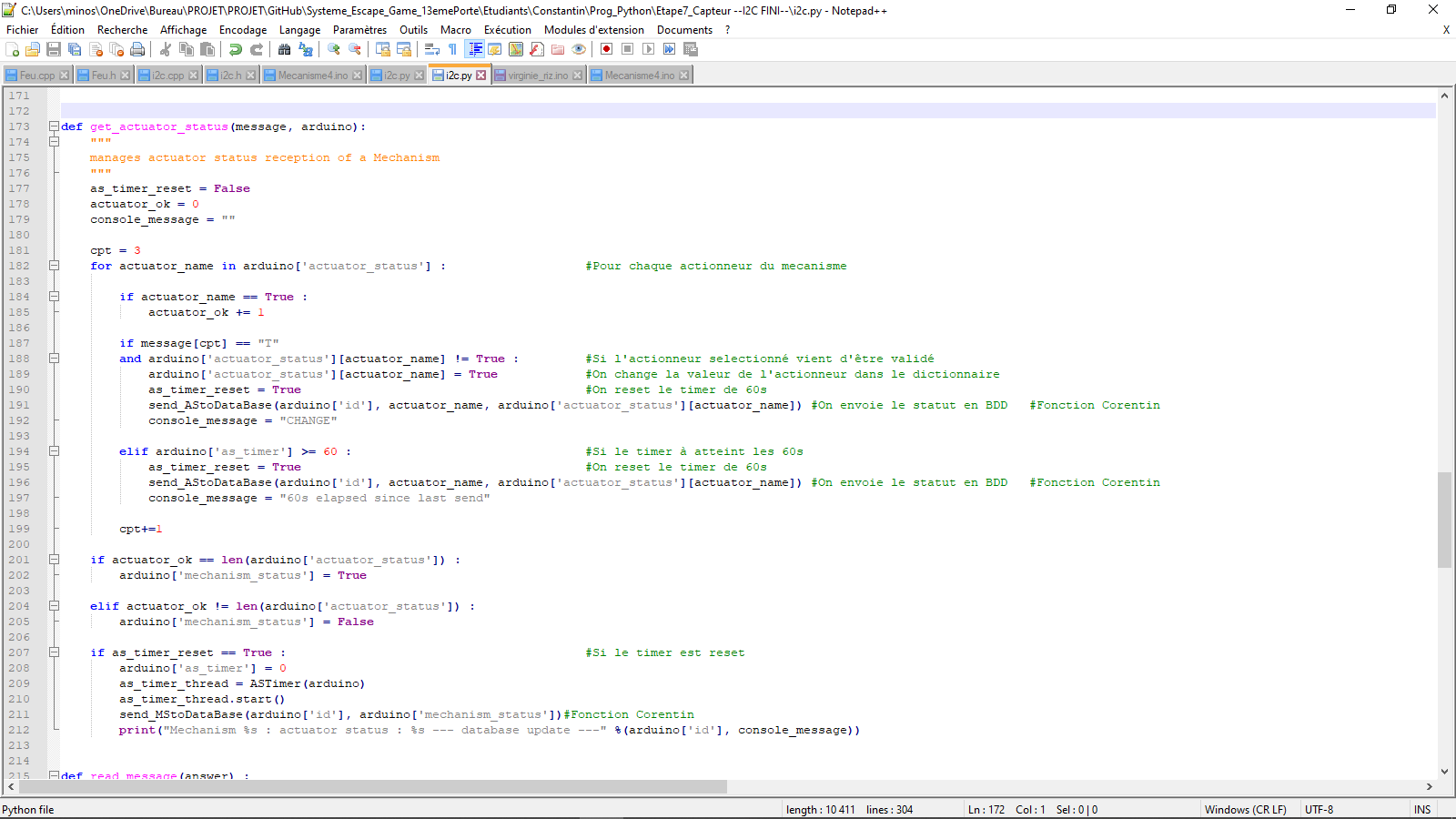
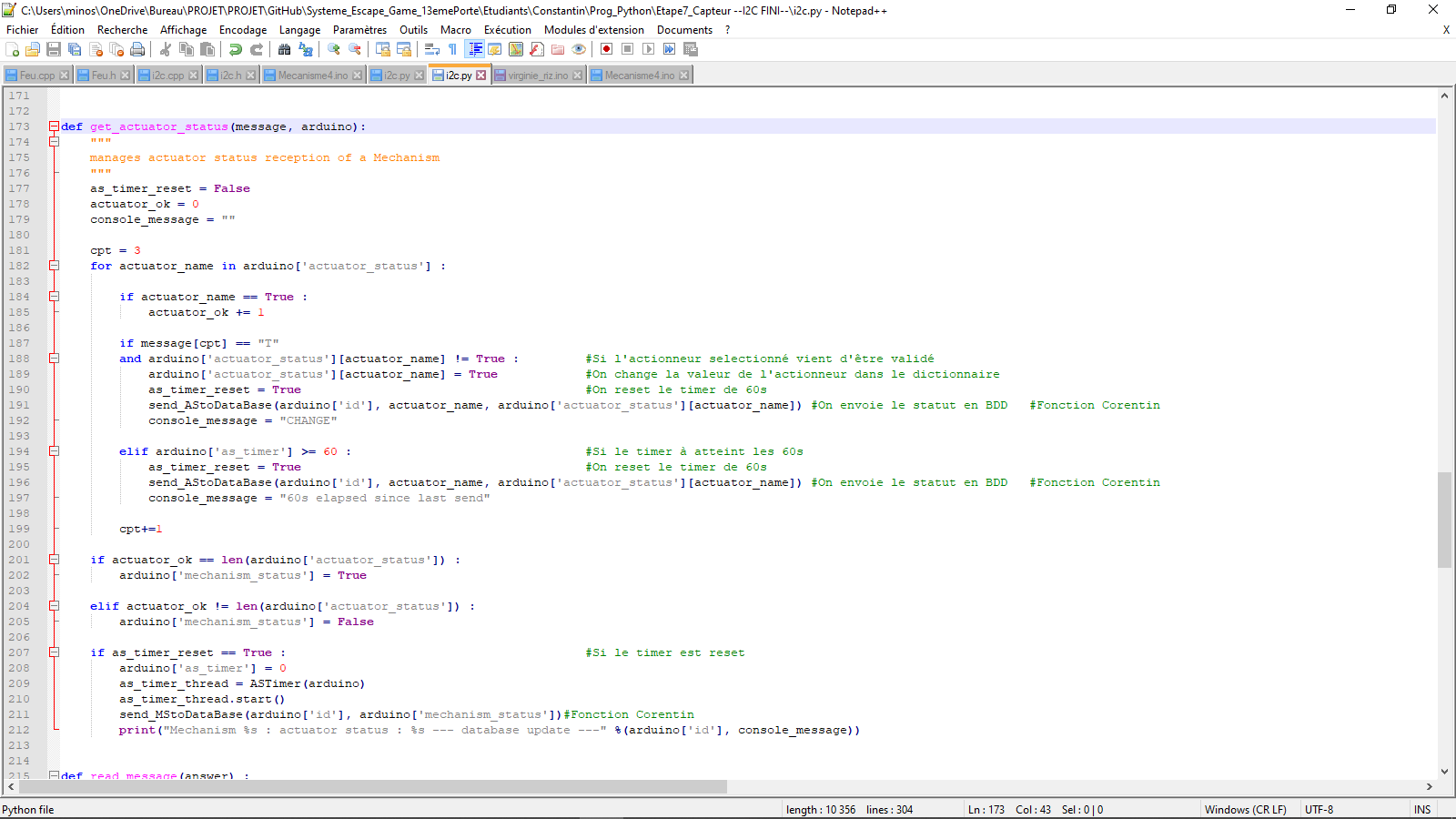
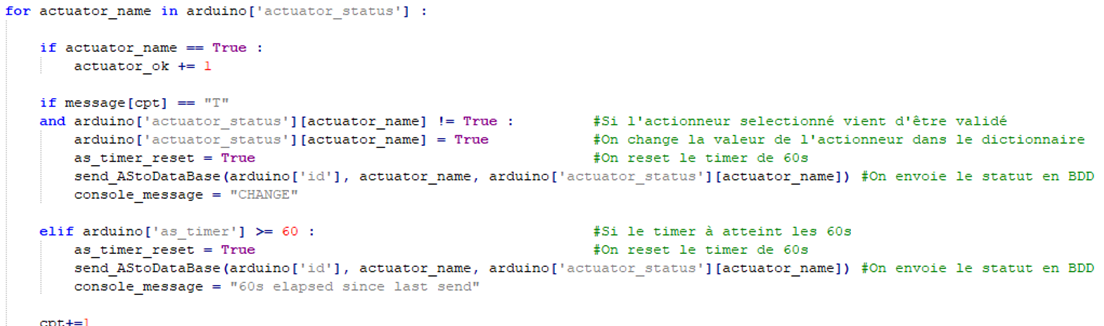
**2. Récupérer le message i2c depuis l’Arduino**

On récupère le message i2c envoyé par l’Arduino et on convertit le message (ASCII décimal) en une chaine de caractère (String) grâce à la fonction read\_message().

**3. Suivant le message i2c, gérer l’état des actionneurs**

On appelle la fonction get\_actuator\_status().

Elle exécute principalement les instructions suivantes :

* Récupérer l’état des actionneurs

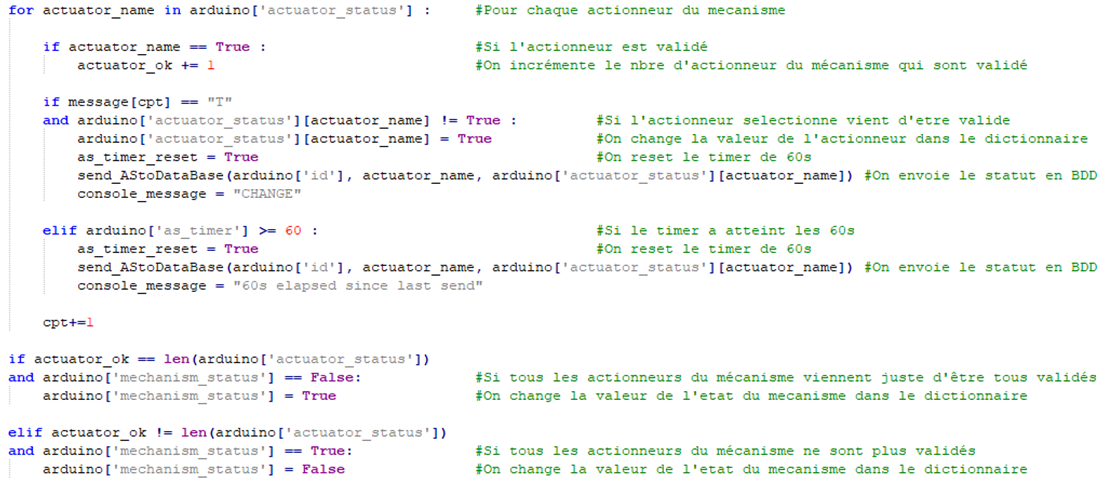
Afin de faire fonctionner le timer de 60s en temps réel il a fallu utiliser des Threads. De sorte que toutes les Arduinos possède leur propre timer de 60s.  
Pour cela il a d’abord fallu créer le thread «**ASTimer** » qui correspond à un timer de 60s ; ensuite il a fallu assigner à chaque Arduino un thread ASTimer.

Le thread ASTimer consiste à effectuer les instructions suivantes :

Une image contenant capture d’écran, moniteur, ordinateur

Description générée automatiquement

* Définir l’état du mécanisme

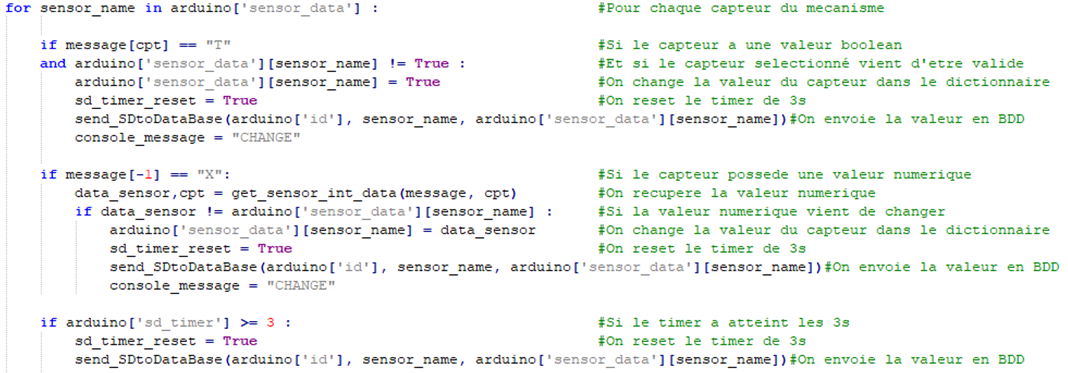
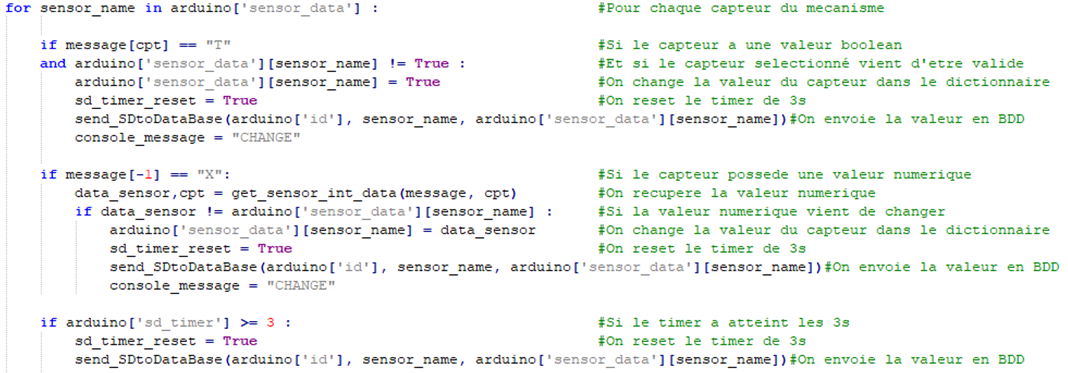
  
Une image contenant capture d’écran, moniteur, écran, ordinateur

Description générée automatiquement

**4. Suivant le message i2c, gérer la valeur des capteurs**

On appelle la fonction get\_sensor\_data().

Elle exécute principalement les instructions suivantes :

Une image contenant capture d’écran, portable, ordinateur

Description générée automatiquement

De même que pour récupérer l’état des actionneurs, afin de faire fonctionner le timer de 3s en temps réel il a fallu utiliser des Threads. De sorte que toutes les Arduinos possède leur propre timer de 3s.  
Pour cela il a d’abord fallu créer le thread «**SDTimer** » qui correspond à un timer de 3s ; ensuite il a fallu assigner à chaque Arduino un thread SDTimer.

Le thread SDTimer consiste à effectuer les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

Programme i2c sur Arduino :

Programme Arduino Mécanisme 4 Feu :

Le programme exécute principalement :

Une image contenant capture d’écran, ordinateur

Description générée automatiquement  
Une image contenant capture d’écran, ordinateur

Description générée automatiquement  
Une image contenant capture d’écran, ordinateur

Description générée automatiquement

On instancie d’abord un objet Feu qu’on nomme « mecanism ». On lui donne une configuration de base dans setupMechanism() puis on appelle en boucle la méthode « **execute()** » de la classe Feu.

Cette méthode exécute principalement les instructions suivantes :

Une image contenant capture d’écran, portable, ordinateur, moniteur

Description générée automatiquement

Programme Arduino Mécanisme 8 Riz :

Le programme exécute principalement :

Une image contenant capture d’écran

Description générée automatiquement  
Une image contenant capture d’écran, ordinateur

Description générée automatiquement  
Une image contenant capture d’écran, ordinateur

Description générée automatiquement

On instancie d’abord un objet Riz qu’on nomme « mecanism ». On lui donne une configuration de base dans setupMechanism() puis on appelle en boucle la méthode « **execute()** » de la classe Riz.

Cette méthode exécute principalement les instructions suivantes :

Une image contenant capture d’écran, portable, ordinateur

Description générée automatiquement