**Sommaire**

[I. Introduction 2](#_Toc39604369)

[II. Analyse 2](#_Toc39604370)

[A. Mécanisme 4 : Feu 2](#_Toc39604371)

[B. Mécanisme 8 : Riz 4](#_Toc39604372)

[C. Communication entre les Arduinos et la Raspberry 6](#_Toc39604373)

[III. Réalisation du Projet 8](#_Toc39604374)

[A. Rappel de la tâche de l’étudiant 8](#_Toc39604375)

[B. Mécanisme 4 : Feu 9](#_Toc39604376)

[C. Mécanisme 8 : Riz 10](#_Toc39604377)

[D. Liaison i2c 12](#_Toc39604378)

[IV. Réalisation du programme de régulation 16](#_Toc39604379)

[A. Carte Raspberry 16](#_Toc39604380)

[B. Carte Arduino 17](#_Toc39604381)

[C. Journalisation 17](#_Toc39604382)

[V. Réalisation de l’application « Journalisation » 17](#_Toc39604383)

[VI. Test unitaire 17](#_Toc39604384)

[VII. Fiches recettes 17](#_Toc39604385)

[A. Connexion Raspberry 17](#_Toc39604386)

[B. Lancer le système de régulation 17](#_Toc39604387)

[C. Lecture des événement journalisés 17](#_Toc39604388)

[VIII. Conclusion 17](#_Toc39604389)

[A. Communication de groupe 17](#_Toc39604390)

[B. Regard critique du projet 17](#_Toc39604391)

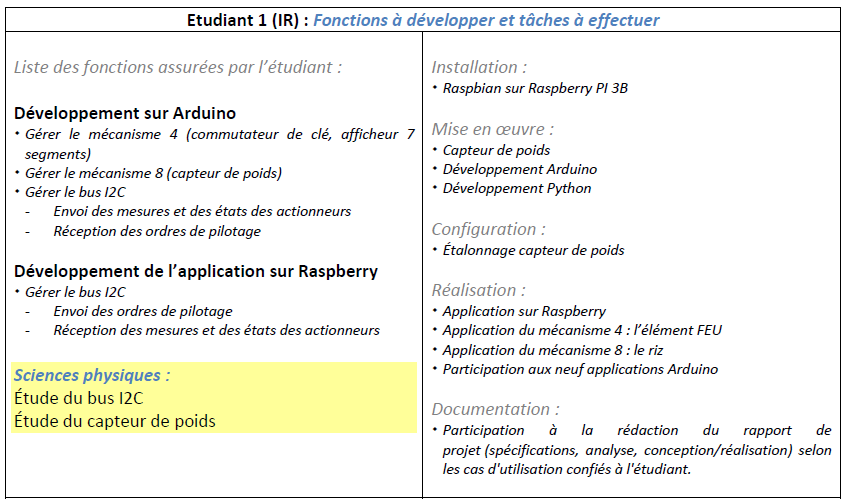
[C. Connaissances apportées 18](#_Toc39604392)

[D. Ce qui me reste à faire 18](#_Toc39604393)

[E. Poursuite d’étude 18](#_Toc39604394)

[Annexe 18](#_Toc39604395)

# Introduction



# Analyse

## Mécanisme 4 : Feu

### Synopsis du mécanisme

Le mécanisme 4 repose sur un interrupteur à clef.

S’il y a un changement de position positif :

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.



### Diagramme de classe

Une image contenant texte, capture d’écran

Description générée automatiquement

### Diagramme de séquence



## Mécanisme 8 : Riz

### Synopsis du mécanisme

Le mécanisme 4 repose sur un capteur de poids.

Si le capteur détecte une mesure comprise entre 48 et 52g:

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.



### Diagramme de classe

Une image contenant capture d’écran

Description générée automatiquement

### Diagramme de séquence



## Communication entre les Arduinos et la Raspberry

### Synopsis de la 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.

### Schéma Câblage liaison i2c



### Diagramme de séquence

Une image contenant texte, carte

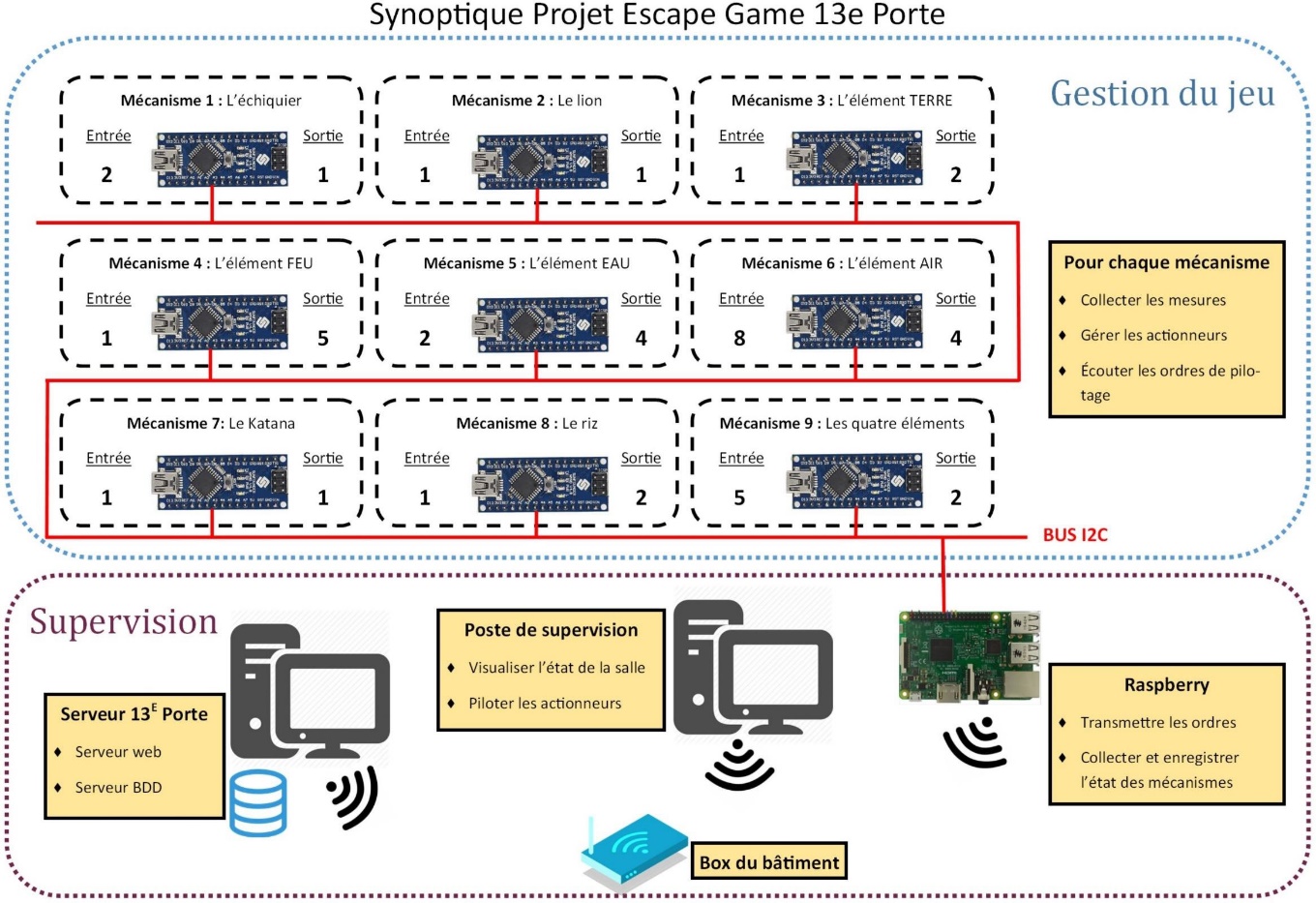
Description générée automatiquement

# Réalisation du Projet

## Rappel de la tâche de l’étudiant

Dans le système Escape Game, ma partie de développement consiste à gérer :

* Le Mécanisme 4 : le Feu
* Le Mécanisme 8 : le Riz
* Le bus I2c pour une liaison Arduino et Raspberry afin de :
  + - Recevoir sur la Raspberry l’état des actionneurs et la valeur des capteurs des Arduinos
    - Exécuter sur les Arduinos les ordres envoyer depuis l’application de supervision



## Mécanisme 4 : Feu

### Le main

Le programme exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

### La classe Feu

La classe Feu se présente sous cette forme :

Une image contenant capture d’écran

Description générée automatiquement

### La méthode execute

La méthode **« exécute »** est la méthode qui fait fonctionner le mécanisme. Elle est appelée dans la fonction loop du programme et exécute en boucle les instructions suivantes :

Une image contenant capture d’écran, ordinateur

Description générée automatiquement

## Mécanisme 8 : Riz

### Le main

Le programme exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

### La classe Riz

La classe Riz se présente sous cette forme :

Une image contenant capture d’écran

Description générée automatiquement

### La méthode execute

La méthode **« exécute »** est la méthode qui fait fonctionner le mécanisme. Elle est appelée dans la fonction loop du programme et exécute en boucle les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

## Liaison i2c

### Script Python sur Raspberry

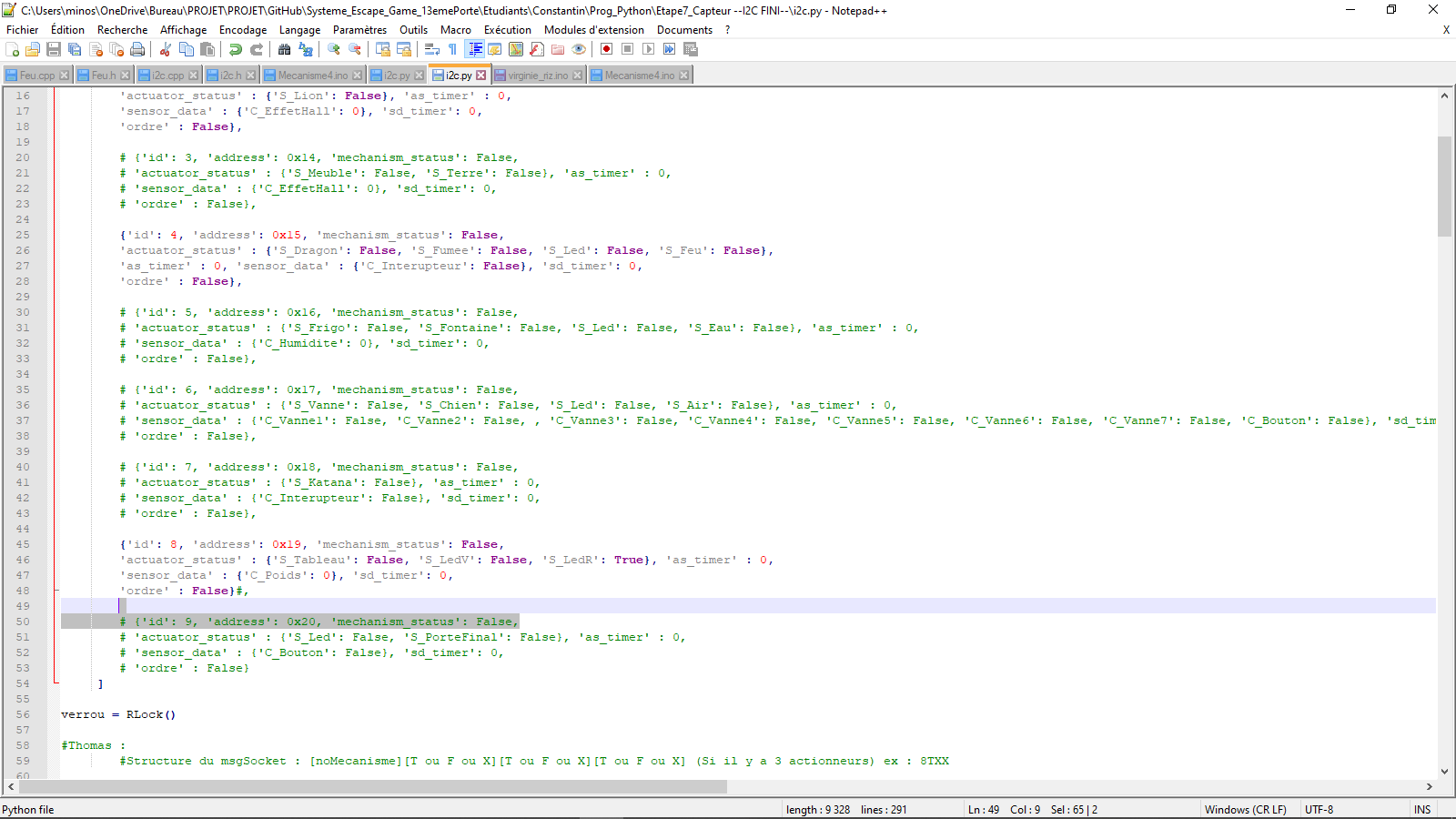
Ce script python consiste à :

* 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 l’état des actionneurs de chaque mécanisme à chaque changement ou sinon toutes les 60s. Et ensuite envoyer les valeurs à la BDD.
* Recevoir l’état de chaque mécanisme à chaque changement ou sinon toutes les 60s. Et ensuite envoyer les valeurs à la BDD.
* Envoyer les ordres reçus depuis l’application Web au mécanisme correspondant

Il a fallu créer une liste de 9 dictionnaires correspondant aux 9 Arduinos. Les clés de ces dictionnaires sont identiques.

Voici un fragment de la liste de dictionnaires :

Une image contenant capture d’écran

Description générée automatiquement  
 (…) // 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)
* « **ms\_noTimer**» : correspond à une valeur entière

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

* « **actuator\_status** » : correspond à l’état des actionneurs du mécanisme

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

* « **as\_noTimer**» : correspond à une valeur entière

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\_noTimer**» : correspond à une valeur entière

Nécessaire à l’exécution du thread SDTimer pour l’envoie des valeurs capteurs toute les 4s.

* « **order** » : correspond à une valeur entière, à l’ordre envoyé par le PC de supervision.

Ce sont ces dictionnaires qui sont modifiés à chaque changement dans les mécanismes.

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. Envoyer un ordre à l’Arduino**

On appelle la fonction send\_order().

Elle exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

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

On appelle la fonction get\_message().

Elle exécute les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

**3. Gérer l’état des mécanismes**

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

Elle exécute principalement les instructions suivantes :

**Une image contenant capture d’écran

Description générée automatiquement**

D’après le diagramme d’exigence, s’il n’ y a pas de changement d’état pendant 60 secondes il faut envoyer l’état en BDD.   
Pour ce faire il est donc judicieux de faire fonctionner un timer de 60s en parallèle. Il a donc fallu utiliser une nouvelle fois des threads.  
Ainsi chaque Arduino possède son propre thread qui correspond au timer de l’état mécanisme. Ce thread est appelé «**MSTimer** »

Le thread MSTimer exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

**4. Gérer l’état des actionneurs**

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

Elle exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

De même que pour l’état d’un mécanisme, d’après le diagramme d’exigence s’il n’ y a pas de changement d’état pendant 60 secondes il faut envoyer l’état en BDD.   
Pour ce faire il est judicieux de faire fonctionner un timer de 60s en parallèle. Il a donc fallu utiliser une nouvelle fois des threads.  
Ainsi chaque Arduino possède son propre thread qui correspond au timer de l’état des actionneurs. Ce thread est appelé «**ASTimer** »

Le thread ASTimer exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

**5. Gérer la valeur des capteurs**

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

Elle exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

D’après le diagramme d’exigence s’il n’ y a pas de changement d’état pendant 5 secondes max il faut envoyer l’état en BDD.   
Pour ce faire il est judicieux de faire fonctionner un timer de 4s en parallèle. Il a donc fallu utiliser une nouvelle fois des threads.  
Ainsi chaque Arduino possède son propre thread qui correspond au timer de la valeur des capteurs. Ce thread est appelé «**SDTimer** »

Le thread SDTimer exécute principalement les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

### Partie i2c sur Arduino

La partie i2c du programme de chaque Arduino consiste à :

* Envoyer les informations du mécanisme à la Raspberry
* Exécuter les ordres reçus

Afin de communiquer avec la Raspberry il a d’abord fallu assigner une adresse esclave unique à chaque Arduino. De telle sorte que la Raspberry identifie les Arduinos suivant leur adresse.

Une adresse est définie de cette manière en tête de tous les programmes Arduino :

Une image contenant capture d’écran

Description générée automatiquement

De plus la communication i2c est établie grâce à la bibliothèque **« Wire.h » :**

Une image contenant capture d’écran

Description générée automatiquement

**1. Envoyer les informations du mécanisme à la Raspberry**

Pour ce faire on construit un message i2c possédant une structure définie :

ms**[F ou T]**as**[F ou T][F ou T ou X][F ou T ou X] [F ou T ou X]**sd**[F ou T ou {Valeur Numérique}X]…**

12eme caractère et +

6eme – 9eme caractère

3eme caractère

Le 3ème caractère : correspond à l’état du mécanisme :

* Si le mécanisme est validé on envoie **T**
* Si le mécanisme n’est pas validé on envoie **F**

Le 6ème - 9ème caractère : correspond à l’état des actionneurs du mécanisme :

* Si l’état de l’actionneur est validé on envoie **T**
* Si l’état de l’actionneur n’est pas validé on envoie **F**
* S’il n’y a plus d’actionneur on envoie **X**

Le 12ème caractère et + : correspond à la valeur / état des capteurs du mécanisme

* Si le capteur possède un état booléen :
  + - * Si l’état du capteur est validé on envoie **T**
      * Si l’état du capteur n’est pas validé on envoie **F**
* Si le capteur possède une valeur numérique on envoie la **valeur numérique + X**

Pour envoyer le message i2c on exécute les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

**2. Exécuter les ordres reçus**

Pour exécuter les ordres envoyés depuis le PC de supervision le programme nécéssite les instructions suivantes :

Une image contenant capture d’écran

Description générée automatiquement

De la même manière que le message i2c envoyé à la Raspberry depuis les Arduinos , le message d’ordre à lui aussi une structure bien définie :

**[1- 9] [0 ou 1 ou 2] [0 ou 1 ou 2]…**

3eme caractère et +

1er caractère

2eme caractère

Le 1er caractère : correspond au numéro du mécanisme

Le 2ème caractère : correspond à l’état du mécanisme  :

* Si l’état de l’actionneur doit être validé il y a un **1**
* Si l’état de l’actionneur doit être invalidé il y a un **0**
* Si on ne touche pas à l’état du mécanisme il y a un **2**

Le 3ème caractère et + : correspond à l’état des actionneurs du mécanisme :

* Si l’état de l’actionneur doit être validé il y a un **1**
* Si l’état de l’actionneur doit être invalidé il y a un **0**
* Si on ne touche pas à l’actionneur il y a un **2**

# Réalisation du programme de régulation

## Carte Raspberry

## Carte Arduino

## Journalisation

# Réalisation de l’application « Journalisation »

# Test unitaire

# Fiches recettes

## Connexion Raspberry

## Lancer le système de régulation

## Lecture des événement journalisés

# Conclusion

## Communication de groupe

## Regard critique du projet

## Connaissances apportées

## Ce qui me reste à faire

## Poursuite d’étude

# Annexe