



Sommaire

I.	Introduction.....	3
II.	Etudiant 1 (Constantin Minos)	4
A.	Rappel de la tâche de l'étudiant.....	4
B.	Communication i2c entre Arduinos et Raspberry	5
C.	Structure des messages.....	6
D.	Réalisation du programme Mécanisme 4 : le Feu.....	7
E.	Réalisation du programme Mécanisme 8 : le Riz	10
F.	Réalisation du programme i2c	13
G.	Test unitaire.....	19
H.	Fiches recettes.....	20
I.	Conclusion	24
III.	Etudiant 2 (Corentin BRENY)	26
A.	Rappel de la tâche de l'étudiant	26
B.	Réalisation du programme Mécanisme 6 : Air	27
C.	Réalisation du programme Mécanisme 7 : Katana	33
D.	Création de la base de données	37
E.	La photorésistance en Physique-Appliquée	42
F.	Observer le fonctionnement du mécanisme 6 : Air	43
G.	Observer le fonctionnement du mécanisme 7 : Katana.....	44
H.	Regard critique du projet (Corentin Breny).....	45
I.	Connaissances apportées (Corentin Breny)	45
J.	Ce qui me reste à faire (Corentin Breny).....	45
K.	Poursuite d'étude (Corentin Breny)	45
IV.	Étudiant 3 (Joshua PINNEAU)	46
A.	Rappel des tâches de l'étudiant	46
B.	Partie physique : l'effet Hall	47
C.	Mécanisme N°1 : L'Échiquier.....	48
D.	Mécanisme N°2 : Le lion basculant	51
E.	Mécanisme N°3 : L'élément Terre.....	54
F.	Visualiser l'état de la salle	57
G.	Partie réseau Matériel.....	62



H.	Fiches recettes.....	63
I.	Conclusion	67
V.	Etudiant 4 (Thomas CADEAU).....	69
A.	Rappel du cahier des charges.....	69
B.	Réalisation du programme Mécanisme 5 : l'Eau.....	71
C.	Présentation du matériel et du mécanisme.....	71
D.	Programme.....	74
E.	Réalisation du programme Mécanisme 9 : les 4 Éléments	77
F.	Présentation du matériels et du mécanisme	78
G.	Programme.....	79
H.	Développement sur la Raspberry	81
I.	Raspberry (Définition)	82
J.	Programmation par socket.....	82
K.	Putty (Définition)	82
L.	Développement de l'application WEB de supervision	83
M.	Programmation PHP/HTML/CSS.....	83
N.	Création de PopUp	84
O.	Fiches recettes.....	85
P.	Conclusion	87
VI.	Annexe.....	89
A.	Annexe 1(Constantin MINOS) :	89
B.	Annexe 2 :.....	90
C.	Annexe 3 :.....	91
D.	Annexe 4 :.....	92
E.	Annexe 5 :.....	93
F.	Annexe 6 :.....	94
G.	Annexe 7 :.....	95
H.	Annexe 8 :.....	96
I.	Annexe 9 :.....	97
J.	Annexe 10 (Joshua PINNEAU) :.....	100
K.	Annexe 11 :.....	103
L.	Annexe 12 :.....	104



I. Introduction

Rappel du cahier des charges

Le client souhaite que le système technique actuellement en place soit recréé entièrement. La finalité du projet est la suivante :

- Chaque mécanisme du système doit pouvoir fonctionner de manière indépendante sur une carte Arduino nano.
- Une carte Raspberry doit pouvoir :
 - Récupérer les informations mécanismes via un bus i2c et les envoyer en base de données.
Les informations mécanismes sont les suivantes :
 - ✚ Pour l'état des mécanismes : A chaque changement et toutes les 60s
 - ✚ Pour l'état des actionneurs : A chaque changement et toutes les 60s
 - ✚ Pour l'état/la valeur des capteurs : A chaque changement et toutes les 5s max
 - Récupérer des messages d'ordre via un serveur socket et les envoyer aux Arduinos via un bus i2c.
- Une base de données doit pouvoir stocker les informations mécanismes
- Une application Web doit pouvoir visualiser les informations mécanismes en temps réel.
- Une application Web doit pouvoir activer/désactiver l'état de chaque actionneur et chaque mécanisme.



II. Etudiant 1 (Constantin Minos)

A. Rappel de la tâche de l'étudiant

Aperçu des tâches réalisées

Dans le système Escape Game, ma partie de développement consiste à créer plusieurs programmes :

- Deux programmes pour deux mécanismes sur Arduino :
 - Le mécanisme 4 : le Feu, avec un interrupteur à clef (Langage Arduino ~C++)
 - Le Mécanisme 8 : le Riz, avec un capteur de poids (Langage Arduino ~C++)
- Deux programmes pour une communication Arduino-Raspberry via i2c afin de :
 - Recevoir les informations mécanismes sur la Raspberry (Langage Python et Arduino)
 - Envoyer les ordres aux Arduinos et les exécuter (Langage Python et Arduino)

Pour la partie i2c le langage python est utilisé avec différentes librairies, comme smbus pour la communication i2c.

De même le langage Arduino est utilisé avec Wire pour l'i2c et avec HX711 pour le capteur de poids.

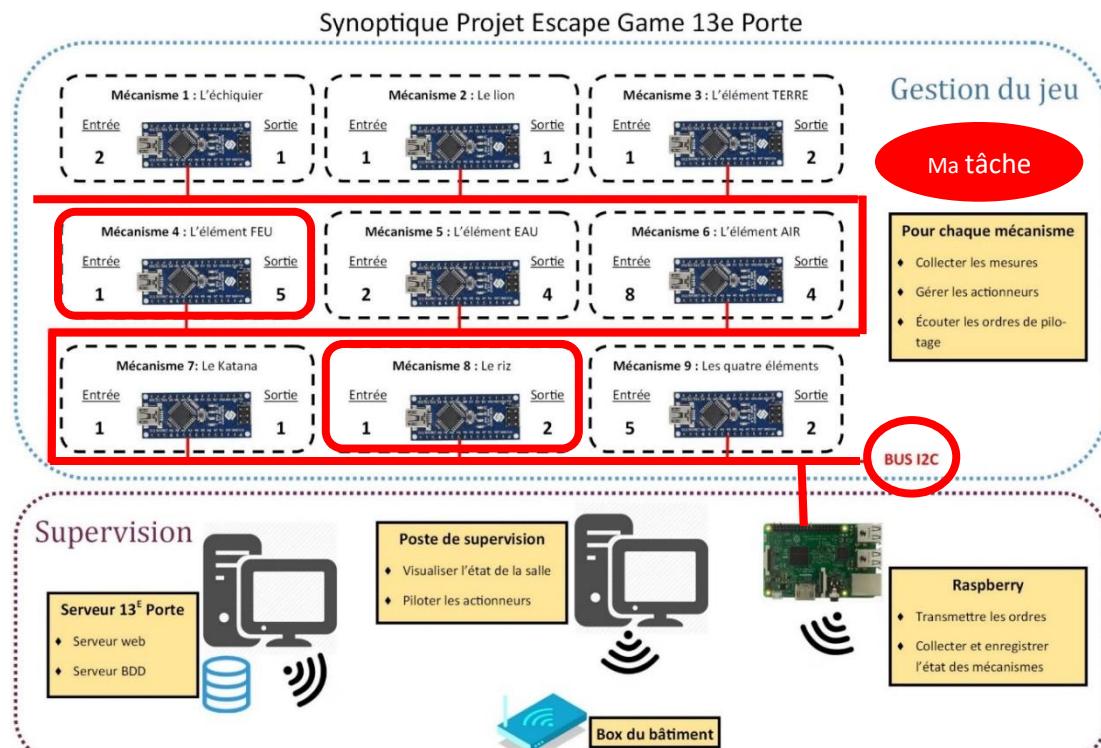


Image 1 : Ma tâche personnelle

Contraintes liées au développement

A cause du confinement le programme concernant le mécanisme 4 et le mécanisme 8 n'a pas pu être testé par manque de relais et de capteur de poids.



B. Communication i2c entre Arduinos et Raspberry

Synoptique

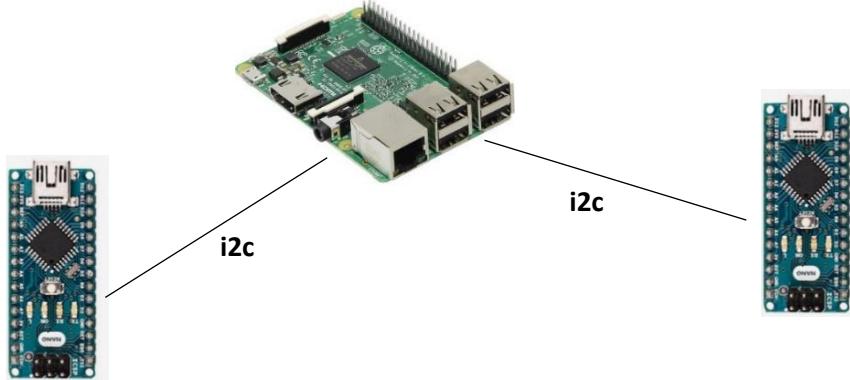


Image 2 : Communication Raspberry - Arduinos

Schéma Câblage

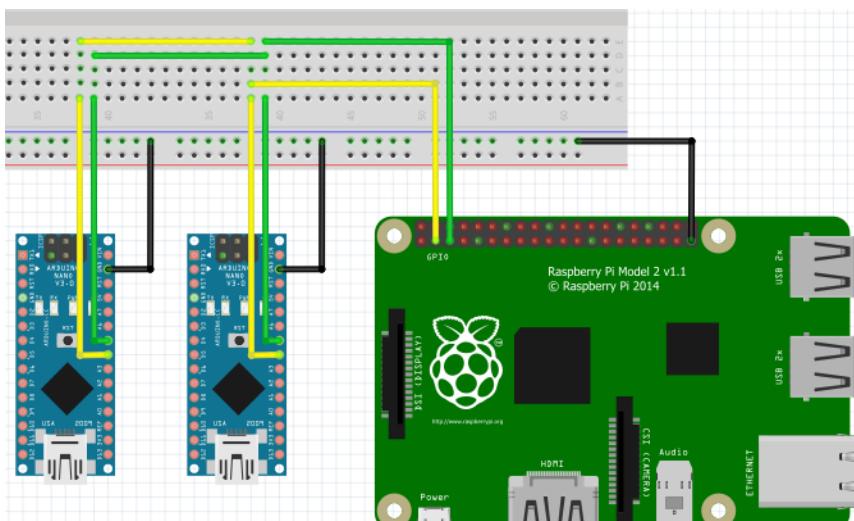


Image 3 : Schéma câblage avec 1 Raspberry et 2 Arduinos

Communication

Avec l'i2c il y a un principe de maître et esclave. Ici nous utiliserons le Raspberry Pi en tant que maître et les Arduino en tant qu'esclave.

Le maître (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.



C. Structure des messages

Message i2c des informations mécanismes

Afin d'envoyer les informations mécanismes à la Raspberry on construit un message bien défini :

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]...

3eme caractère

6eme – 9eme caractère

12eme caractère et +

Le 3^{ème} 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 6^{ème} - 9^{ème} 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 12^{ème} 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**

Message Socket/i2c d'ordre

Afin d'envoyer un ordre à une Arduino on construit un message avec une structure bien définie :

[1- 9] [0 ou 1 ou 2] [0 ou 1 ou 2]...

1er caractère 2eme caractère 3eme caractère et +

Le 1^{er} caractère : correspond au numéro du mécanisme

Le 2^{ème} 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 3^{ème} 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**



D. Réalisation du programme Mécanisme 4 : le Feu

Schéma câblage

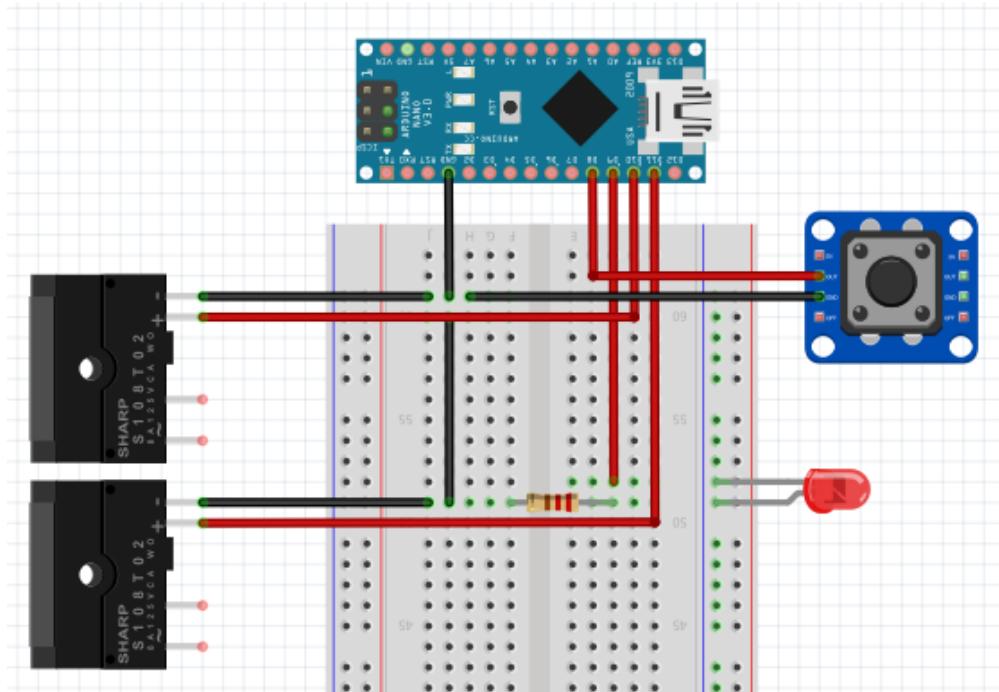


Image 4 : Schéma câblage du mécanisme 4

Programme

Initialisation

On définit des variables pour les pins utilisées de l'Arduino :

```
#define CInterupteur_PIN 8          //interrupteur a clef sur pin 8
#define SLed_PIN 9                 //led controle sur pin 9
#define SDragon_PIN 10             //relais de l'électroaimant de la ventouse dragon sur pin10
#define SFumee_PIN 11              //relais de la machine à fumée sur pin 11
```

Image 5 : Code en tête de feu.ino

On initialise le matériel utilisé pour le mécanisme :

```
pinMode(CInterupteur_PIN , INPUT_PULLUP);           //On initialise le pin de l'interupteur à clef

pinMode(SLed_PIN, OUTPUT);                         //On initialise le pin de la led
digitalWrite(SLed_PIN , LOW);                      //On éteint la led par défaut

pinMode(SDragon_PIN, OUTPUT);                      //On initialise le pin du relais de l'électroaimant de la ventouse dragon
digitalWrite(SDragon_PIN, LOW);                    //On active le relais de l'électroaimant par défaut

pinMode(SFumee_PIN, OUTPUT);                       //On initialise le pin du relais de la machine à fumée
digitalWrite(SFumee_PIN, HIGH);                     //On désactive le relais machine à fumée par défaut
```

Image 6 : Code fonction setupMechanism() dans feu.ino



Le main

Le programme exécute principalement les instructions suivantes :

```

Feu mechanism = Feu();                                //On instancie un objet de type Feu

void setup() {
    mechanism.setupMechanism();                      //On donne une configuration de base au mécanisme
}

void loop() {
    delay(100);                                     //On attend 0.1 seconde
    mechanism.execute();                            //On exécute le mécanisme
}

```

Image 7 : Code main dans feu.ino

La classe Feu

Principaux attributs

Les principaux attributs de la classe sont les suivants :

```

bool S_Dragon;                                         //actionneur qui active/désactive l'électroaimant de la ventouse dragon
bool S_Fumee;                                         //actionneur qui active/désactive la fumée
bool S_Led;                                            //actionneur qui active/désactive la led de contrôle
bool S_Feu;                                             //actionneur de l'élément FEU sur la tablette des 4 éléments
bool C_Interupteur;                                    //état de la position détecter par l'interupteur à clef
bool mechanism_status;                                //indique si le mécanisme est activé ou non

```

Image 8 : Code classe feu dans feu.ino

Principales méthodes

Les principales méthodes sont les suivantes :

```

public :
    Feu();                                              //constructeur de la classe
    void setupMechanism();                            //configuration de base du mécanisme
    void execute();                                     //méthode qui fait fonctionner le mécanisme

```

Image 9 : Code classe feu dans feu.ino

La méthode execute

Synopsis

La méthode « **execute** » est la méthode qui fait fonctionner le mécanisme. Elle est appelée dans la fonction loop du programme et donc exécutée en boucle.

Récupérer l'état de l'interrupteur à clef

Pour récupérer l'état de l'interrupteur à clef on utilise l'instruction suivante :

```

sd_reading = digitalRead(CInterupteur_PIN);           //On récupère la valeur du capteur intérupiteur à clef

```

Image 10 : Code fonction execute() dans feu.ino

Valider le mécanisme

Pour valider le mécanisme on exécute les instructions suivantes :



```
if (mechanism_status == false) {                                //Si il y a eu le premier changement de position
    S_Dragon = true;                                         //On active l'électroaimant de la ventouse dragon
    S_Fumee = true;                                         //On active la fumée
    S_Led = true;                                           //On allume la led verte
    S_Feu = true;                                            //On allume l'element FEU des 4 éléments
    mechanism_status = true;                                 //On valide le mécanisme
}
```

Image 11 : Code fonction execute() dans feu.ino



E. Réalisation du programme Mécanisme 8 : le Riz

Montage

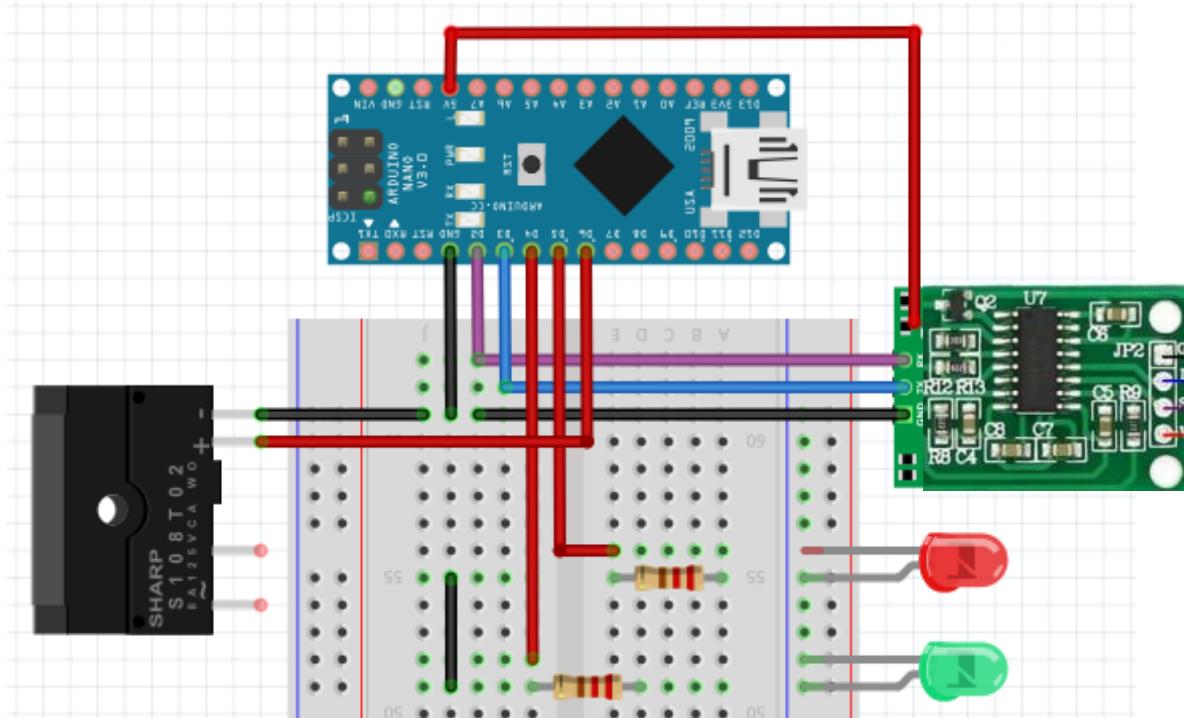


Image 12 : Schéma montage du mécanisme 8

Programme

Initialisation

Afin d'utiliser le capteur de poids il a fallu utiliser la librairie HX711 :

```
#include "HX711.h" //librairie nécessaire au capteur de poids
```

Image 13 : Code en tête de riz.ino

On définit des variables pour les pins utilisées de l'Arduino :

```
#define CPoids_CLK 2 //clk du capteur de poids sur pin 2
#define CPoids_DOUT 3 //dout du capteur de poids sur pin 3
#define SLedV_PIN 4 //led verte sur pin 4
#define SLedR_PIN 5 //led rouge sur pin 5
#define STableau_PIN 6 //relais de l'électroaimant de la chute de tableau sur pin 6
```

Image 14 : Code en tête de riz.ino



On initialise le matériel utilisé pour le mécanisme :

```
HX711 scale(CPoids_DOUT, CPoids_CLK); //On initialise les pins du capteur de poids
scale.set_scale(calibration_factor); //On ajuste le capteur de poids au facteur d'étalonnage

pinMode(SLedV_PIN, OUTPUT); //On initialise le pin de la led verte
digitalWrite(SLedV_PIN, LOW); //On éteint la led verte par défaut

pinMode(SLedR_PIN, OUTPUT); //On initialise le pin de la led rouge
digitalWrite(SLedR_PIN, HIGH); //On éteint la led rouge par défaut

pinMode(STableau_PIN, OUTPUT); //On initialise le pin du relais de l'électroaimant de la chute de tableau
digitalWrite(STableau_PIN, HIGH); //On désactive le relais de l'électroaimant par défaut
```

Image 15 : Code fonction setupMechanism() dans riz.ino

Le main

Le programme exécute principalement les instructions suivantes :

```
Riz mechanism = Riz(); //On instancie un objet de type Riz

void setup() {
    mechanism.setupMechanism(); //On donne une configuration de base au mécanisme
}

void loop() {
    delay(100); //On attend 0.1 seconde
    mechanism.execute(); //On exécute le mécanisme
}
```

Image 16 : Code fonction main dans riz.ino

La classe Riz

Principaux attributs

Les principaux attributs de la classe sont les suivants :

```
bool S_Tableau; //actionneur qui active/désactive l'électroaimant de la chute de tableau
bool S_Led; //actionneur qui active/désactive la led de contrôle
int C_Poids; //valeur mesuré par le capteur de poids
bool mechanism_status; //indique si le mécanisme est activé ou non
```

Image 17 : Code classe riz dans riz.ino

Principales méthodes

Les principales méthodes sont les suivantes :

```
Riz(); //constructeur de la classe
void setupMechanism(); //configuration de base du mécanisme
```

Image 18 : Code classe riz dans riz.ino

La méthode execute

Synopsis

La méthode « **execute** » est la méthode qui fait fonctionner le mécanisme. Elle est appelée dans la fonction loop du programme et donc exécutée en boucle.



Récupérer la valeur du capteur de poids

Pour récupérer la valeur mesurée par le capteur de poids on utilise l'instruction suivante :

```
sd_reading = (scale.get_units() / 2.0); //On récupère une lere mesure du capteur de poids
```

Image 19 : Code fonction execute() dans riz.ino

Valider le mécanisme

Pour valider le mécanisme on exécute les instructions suivantes :

```
if ( (sd_reading >= 0.48)
&& (sd_reading <= 0.52 )
&& mechanism_status == false ) {
    ...
    ...
    ...
    ...
    S_Led = true; //On allume la led verte et on éteint la rouge
    S_Tableau = true; //On libère la chute du tableau
    mechanism_status = true; //On valide le mécanisme
}
```

Image 20 : Code fonction execute() dans riz.ino



F. Réalisation du programme i2c

Script Python sur Raspberry

Synoptique

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

Programme

La Raspberry est maître dans la communication i2c. Et la communication i2c est établie grâce à la bibliothèque **smbus** :

```
import smbus #librairie nécessaire pour la communication i2c
```

Image 21 : Code en tête de i2c.py

Les dictionnaires

Pour stocker les informations relatives aux Arduinos dans le programme il a été judicieux de créer une liste de 9 dictionnaires correspondant aux 9 Arduinos. Les clés de ces dictionnaires sont identiques.

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'envoi 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'envoi des valeurs capteurs toute les 4s.
- « **order** » : correspond à une valeur entière, à l'ordre envoyé par le PC de supervision.



Voici un fragment de la liste de dictionnaires :

```
arduinoss = [
    {'id': 1, 'address': 0x12,
     'mechanism_status': False, 'ms_noTimer': 0,
     'actuator_status' : {'S_Echiquier': False}, 'as_noTimer' : 0,
     'sensor_data' : {'C_EffetHalll': 0, 'C_EffetHall2': 0}, 'sd_noTimer': 0,
     'order' : 0},

    {'id': 2, 'address': 0x13,
     'mechanism_status': False, 'ms_noTimer': 0,
     'actuator_status' : {'S_Lion': False}, 'as_noTimer' : 0,
     'sensor_data' : {'C_EffetHall': 0}, 'sd_noTimer': 0,
     'order' : 0},
```

Image 22 : Code définition dictionnaire arduinos[] dans i2c.py

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

Communication Arduino

Afin de recevoir les informations des mécanismes en temps réel il a fallu utiliser des threads. De sorte que toutes les Arduinos communiquent avec la Raspberry en même temps.

Un thread est une séquence d'instructions qui s'exécute parallèlement aux autres.

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 :

```
while self.running:

    try :
        print("Arduino %s : communication" %self.arduino['id'])

        send_order(self.arduino)                                #On envoie le message d'order a l'Arduino s'il y en a un

        message = get_message(self.arduino)                      #On recuper le message i2c venant de l'Arduino

        get_mechanism_status(message, self.arduino)             #On verifie l'etat du mecanisme

        get_actuator_status(message, self.arduino)              #On verifie l'etat des actionneurs

        get_sensor_data(message, self.arduino)                  #On verifie la valeur des capteurs

        time.sleep(2)                                         #On attend 2 secondes

    except :
        self.running = False
        thread = ArduinoCom(self.arduino)
        thread.start()                                         #Si il y a une erreur quelconque dans l'execution du thread
                                                               #On arrete le thread
                                                               #On cree un nouveau thread
                                                               #On lance le nouveau thread
```

Image 23 : Code thread ArduinoCom dans i2c.py



Les principales instructions du thread ArduinoCom sont les suivantes :

Envoyer un ordre à une Arduino

On appelle la fonction **send_order()**.

Elle exécute principalement les instructions suivantes :

```
if arduino['order'] != 0 : #On verifie si l'Arduino a recu un order
    order = convertStrToListHex(str(arduino['order']))
    bus.write_i2c_block_data(arduino['address'], 0, order) #On convertit le message socket en Hexadecimal
    print("Mechanism %s : ORDER SENT : %s" %(arduino['id'],arduino['order'])) #On envoie le message socket a l'Arduino assignee
    arduino['order'] = 0 #On reset l'ordre
```

Image 24 : Code fonction send_order() dans i2c.py

Récupérer le message i2c des informations mécanismes

On appelle la fonction **get_message()**.

Elle exécute les instructions suivantes :

```
answer = bus.read_i2c_block_data(arduino['address'],0x32) #On recuper le message i2c
l = []
for letter in answer:
    if letter == 255:
        break
    l.append(chr(letter))

message="".join(l) #On converti le message i2c en une chaine de caractere

return message #msFasFFFFsdF/msFasFFXXsd0X
```

Image 25 : Code fonction get_message() dans i2c.py

Gérer l'état des mécanismes

On appelle la fonction **get_mechanism_status()**.

Elle exécute principalement les instructions suivantes :

```
cpt = 2 #La partie capteur du message commence au 3eme caractere
if message[cpt] == "T" :
    if arduino['mechanism_status'] == False :
        arduino['mechanism_status'] = True
    ResetTimer = True #Si le mecanisme vient d'etre valide
                      #On change la valeur du mecanisme dans le dictionnaire
                      #On reset le timer

elif message[cpt] == "F" :
    if arduino['mechanism_status'] == True :
        arduino['mechanism_status'] = False
    ResetTimer = True #Si le mecanisme vient d'etre invalide
                      #On change la valeur du mecanisme dans le dictionnaire
                      #On reset le timer

if ResetTimer == True :
    arduino['ms_noTimer'] += 1 #Pour reset le timer
    thread = MSTimer(arduino) #On incremente la valeur "_noTimer" dans le dictionnaire
    #On cree un nouveau thread
    thread.start() #On lance le nouveau thread

send_MStoDataBase(arduino, console_message) #On envoie l'etat du mecanisme en BDD
```

Image 26 : Code fonction get_mechanism_status() dans i2c.py



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 :

```

no_thread = selfarduino['ms_noTimer']                                #On definit le numero du thread

while self.running:
    time.sleep(60)                                                 #On attend 60 secondes

    if selfarduino['ms_noTimer'] != no_thread :                      #Si le thread n'est plus le timer actuel
        self.running = False                                         #On arrete le thread
    else :
        send_MStoDataBase(selfarduino, console_message)             #Sinon
                                                                #On envoie l'état des mecanismes en BDD

```

Image 27 : Code thread MSTimer dans i2c.py

Gérer l'état des actionneurs

On appelle la fonction **get_actuator_status()**.

Elle exécute principalement les instructions suivantes :

```

cpt = 5
for actuator_name in arduino['actuator_status'] :                      #La partie capteur du message commence au 6ème caractere
    #Pour chaque actionneur du mecanisme

    if message[cpt] == "T" :
        if arduino['actuator_status'][actuator_name] != True :
            arduino['actuator_status'][actuator_name] = True
            ResetTimer = True

    elif message[cpt] == "F" :
        if arduino['actuator_status'][actuator_name] != False :
            arduino['actuator_status'][actuator_name] = False
            ResetTimer = True

    cpt+=1

if ResetTimer == True :
    arduino['as_noTimer'] += 1
    thread = ASTimer(arduino)
    thread.start()

    send_AStoDataBase(arduino, console_message)                            #On envoie la valeur des actionneurs en BDD

```

Image 28 : Code fonction get_actuator_status() dans i2c.py

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 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 est presque le même que MSTimer.



Gérer la valeur des capteurs

On appelle la fonction `get_sensor_data()`.

Elle exécute principalement les instructions suivantes :

```

cpt = 11
for sensor_name in arduino['sensor_data'] :
    #La partie capteur du message commence au 12eme caractere
    #Pour chaque capteur du mecanisme

    if message[cpt] == "T" :
        if arduino['sensor_data'][sensor_name] != True :
            arduino['sensor_data'][sensor_name] = True
            ResetTimer = True

    elif message[cpt] == "F" :
        if arduino['sensor_data'][sensor_name] != False :
            arduino['sensor_data'][sensor_name] = False
            ResetTimer = True

    elif message[-1] == "X":
        data_sensor,cpt = get_sensor_int_data(message, cpt)
        if data_sensor != arduino['sensor_data'][sensor_name] :
            arduino['sensor_data'][sensor_name] = data_sensor
            ResetTimer = True

    cpt += 1

if ResetTimer == True :
    arduino['sd_noTimer'] += 1
    thread = SDTimer(arduino)
    thread.start()

send_SDtoDataBase(arduino, console_message)
    #Pour reset le timer
    #On incremente la valeur "_noTimer" dans le dictionnaire
    #On cree un nouveau thread
    #On lance le nouveau thread

    #On envoie la valeur des actionneurs en BDD

```

Image 29 : Code fonction `get_sensor_data()` dans `i2c.py`

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 :

```

no_thread = selfarduino['sd_noTimer']                                #On definit le numero du thread

while self.running:
    time.sleep(4)                                                 #On attend 4 secondes

    if selfarduino['sd_noTimer'] != no_thread :                  #Si le thread n'est plus le timer actuel
        self.running = False                                     #On arrete le thread
    else :
        send_SDtoDataBase(selfarduino, console_message)          #Sinon
                                                                #On envoie la valeur des capteurs en BDD

```

Image 30 : Code thread SDTimer dans `i2c.py`



Programme Arduino

La partie i2c du programme de chaque Arduino consiste à :

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

Programme

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 :

```
#define SLAVE_ADDRESS 0x15 //initialisation de l'Arduino avec l'adresse 0x15
```

Image 31 : Code en tête de tous les programmes Arduinos

De plus la communication i2c est établie grâce à la bibliothèque **Wire** :

```
#include "Wire.h"           //librairie nécessaire pour la communication i2c
```

Image 32 : Code en tête de tous les programmes Arduinos

Envoyer les informations du mécanisme à la Raspberry

Pour envoyer le message i2c on exécute les instructions suivantes :

```
void send_status() {
    Wire.write(getMessageI2C().c_str());      //On écrit le message i2c dans l'objet Wire
}

void setup() {
    Serial.begin(9600);
    Wire.begin(SLAVE_ADDRESS);                //On indique à l'objet Wire l'adresse esclave utilisée par l'Arduino
    Wire.onRequest(send_status);              //On envoie le messageI2C sur le bus i2c
}
```

Image 33 : Code dans tous les programmes Arduinos

Exécuter les ordres reçus depuis la Raspberry

Pour exécuter les ordres envoyés depuis la Raspberry et a posteriori depuis le PC de supervision, le programme nécessite les instructions suivantes :

```
void receive_order(int numBytes) {
    String data_received;

    while(Wire.available() > 0) {          //Tant que le message i2c reçu n'est pas fini
        char c = Wire.read();             //On lit le caractère suivant du message sur le bus i2c
        data_received += String(c);       //On ajoute le caractère du message aux données reçus
    }

    if(data_received != "2") {           //Si les données reçues sont bien un message d'ordre
        execute_order(data_received);   //On exécute les ordres du message d'ordre
    }
}

void setup() {
    Serial.begin(9600);
    Wire.begin(SLAVE_ADDRESS);          //On indique à l'objet Wire l'adresse esclave utilisée par l'Arduino
    Wire.onReceive(receive_order);     //On récupère le message s'ordre reçu sur le bus i2c via la fonction receive_order
}
```

Image 34 : Code dans tous les programmes Arduinos



G. Test unitaire

Pour garantir la fiabilité du programme, un test unitaire a été mis en place. En cette période de confinement il repose uniquement sur la partie i2c puisqu'il n'a pas été possible de tester les 2 mécanismes par manque de matériels.

Eléments testés :			Le script I2c.py et la partie i2c des programmes Arduinos					
Objectif du test :			Récupérer sur la Raspberry les informations mécanismes suivant le diagramme d'exigence + Exécuter les ordres sur les Arduinos					
Nom du testeur :			Constantin MINOS			Date :	16/05/2020	
Moyens mis en œuvre :			Logiciel : Putty/Arduino			Matériel : une Raspberry et deux Arduinos	Outil de développement : Python/Arduino	
-----TEST -----				Longueur du câble de la liaison i2c :				20 cm 1m
Etape	Arduino 1			Arduino 2				Résultat
	Etat mécanisme	Etat d'1 actionneur	Etat capteur interrupteur à clef	Etat mécanisme	Etat d'1 actionneur	Valeur Capteur de Poids		
1	Validé L'Arduino 1 a validé le statut	Invalidé	Invalidé	Invalidé	Validé La Raspberry a donné un ordre	0g	OK Annexe 1	OK Annexe 2
2	Validé	Validé L'Arduino 1 a validé le statut	Invalidé	Invalidé	Validé	0g	OK	OK
3	Invalidé La Raspberry a donné un ordre	Validé	Validé L'Arduino 1 a validé le statut	Invalidé	Validé	0g	OK	OK
4	Invalidé	Validé	Invalidé L'Arduino 1 a invalidé le statut	Validé L'Arduino 2 a validé le statut	Validé	0g	OK	OK
5	Validé La Raspberry a donné un ordre	Invalidé L'Arduino 1 a invalidé le statut	Invalidé	Validé	Invalidé La Raspberry a donné un ordre	50g L'Arduino 2 a détecté 50g	OK Annexe 3	OK Annexe 4
6	Invalidé La Raspberry a donné un ordre	Validé La Raspberry a donné un ordre	Validé L'Arduino 1 a validé le statut	Invalidé La Raspberry a donné un ordre	Validé L'Arduino 2 a validé le statut	20g L'Arduino 2 a détecté 20g	OK	OK
7	Attendre 4 secondes (les états/valeurs capteurs sont envoyés en BDD)						OK Annexe 5	OK Annexe 6
	Invalidé	Validé	Validé	Invalidé	Validé	20g		
8	Attendre 56 secondes (les états mécanismes/actionneurs sont envoyés en BDD)						OK Annexe 7	OK Annexe 8
	Invalidé	Validé	Validé	Invalidé	Validé	20g		



H. Fiches recettes

Dans cette partie, les quatre fiches recettes sont destinées au client. Elles permettent de valider le fonctionnement de ma tâche dans le système.

Observer le bon fonctionnement du mécanisme 4 : le Feu

Nom :	Observer le fonctionnement du mécanisme 4 : le Feu
Recette :	Technique
Objectif	Lancer le mécanisme afin qu'il s'exécute correctement
Elément à tester	Feu.ino
Pré requis	Ouvrir Feu.ino dans l'IDE Arduino et réaliser les câblages

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		Un message indique « téléchargement terminé »	<input type="checkbox"/>
2	Changer la position de l'interrupteur à clef sur "on"		Le mécanisme s'exécute correctement	<input type="checkbox"/>

Rapport de test	Testé par : L'Etudiant 1	
Conformité		
<input type="checkbox"/> Excellente		
<input type="checkbox"/> Moyenne		
<input type="checkbox"/> Faible		
Commentaire :		Approbation :



Observer le bon fonctionnement du mécanisme 8 : le Riz

Nom :	Observer le fonctionnement du mécanisme 8 : le Riz
Recette :	Technique
Objectif	Lancer le mécanisme afin qu'il s'exécute correctement
Elément à tester	Riz.ino
Pré requis	Ouvrir Riz.ino dans l'IDE Arduino et réaliser les câblages

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		Un message indique « téléchargement terminé ».	<input type="checkbox"/>
2	Placer entre 48g et 52g sur la balance		Le mécanisme s'exécute correctement.	<input type="checkbox"/>

Rapport de test	Testé par : L'Etudiant 1	
Conformité		
<input type="checkbox"/> Excellente		
<input type="checkbox"/> Moyenne		
<input type="checkbox"/> Faible		
Commentaire :		Approbation :



Observer l'envoi des informations mécanismes en BDD

Nom :	Observer l'envoie des informations mécanisme en BDD
Recette :	Technique
Objectif	Lancer le script python afin que les infos mécanismes s'envoient en BDD
Elément à tester	i2c.py
Pré requis	Configurer la Raspberry, réaliser les câblages et avoir BDD fonctionnel

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Lancer le fichier i2c.py	Entrer la commande « python i2c.py ».	Le système est maintenant lancé.	<input type="checkbox"/>
2	Changer l'état/valeur d'un capteur et d'un actionneur		Le nouvel état/valeur du capteur et de l'actionneur a été envoyé en BDD.	<input type="checkbox"/>
3	Attendre 2 minutes		L'état/valeur des capteurs du mécanisme est envoyé en BDD toutes les 4s, celui des actionneurs toutes les minutes.	<input type="checkbox"/>

Rapport de test	Testé par : L'Etudiant 1	
Conformité		
<input type="checkbox"/> Excellente		
<input type="checkbox"/> Moyenne		
<input type="checkbox"/> Faible		
Commentaire :		Approbation :



Observer l'envoi des ordres sur les Arduinos

Nom :	Observer l'envoie des ordres sur les Arduinos
Recette :	Technique
Objectif	Lancer le script python afin que les ordres s'envoient aux Arduinos
Elément à tester	i2c.py
Pré requis	Configurer la Raspberry, réaliser les câblages et Appli Web fonctionnel

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Lancer le fichier i2c.py	Entrer la commande « python i2c.py ».	Le système est maintenant lancé.	<input type="checkbox"/>
2	Changer l'état d'un actionneur sur l'appli Web		L'état de l'actionneur a changé.	<input type="checkbox"/>

Rapport de test	Testé par : L'Etudiant 1	
Conformité		
<input type="checkbox"/> Excellente		
<input type="checkbox"/> Moyenne		
<input type="checkbox"/> Faible		
Commentaire :		Approbation :



I. Conclusion

Communication de groupe

Durant tout le projet nous avons mis en place une cohésion de groupe. Pour cela, plusieurs mesures ont été prises avec notamment la réalisation de :

- Diagramme de Gantt pour se partager le travail et s'organiser dans la réalisation des tâches du projet
- Répertoire sur la plateforme GitHub pour partager son travail et échanger documents et programmes
- Chartes graphiques (Word, PowerPoint) identiques pour tous les étudiants du groupe
- Réunions régulières sur Discord (moyen de communication) pour travailler, rendre compte de son travail et s'aider

Ces mesures efficaces nous ont permis de réaliser un travail achevé et rigoureux en équipe.

D'autant plus qu'ayant dû effectuer notre projet en confinement nous avons pu faire l'expérience du travail de groupe à distance, une expérience enrichissante.

Je remercie les étudiants Joshua PINNEAU, Thomas CADEAU et Corentin BRENNY pour leur participation au projet.

Regard critique du projet (Constantin Minos)

J'ai choisi ce projet parce que je voulais découvrir l'i2c et l'univers Arduino que je n'avais encore jamais utilisé concrètement et j'ai beaucoup apprécié ma tâche. J'ai découvert et expérimenté plusieurs notions qui m'ont beaucoup intéressé. J'ai donc un avis extrêmement positif sur le projet.

Connaissances apportées (Constantin Minos)

Ce projet m'a apporté bien des connaissances.

Au niveau Hardware j'ai appris à :

- Monter des modules électroniques sur une Arduino
- Établir une connexion i2c entre Arduinos-Raspberry via breadboard et relais
- Réaliser des schémas-montages électroniques

Et au niveau Software dans mes programmes j'ai appris à :



- Configurer (pleinement) une Raspberry
- Réaliser des threads en Python
- Programmer une Arduino
- Faire communiquer Arduino-Raspberry via i2c

Des connaissances importantes qui me seront sûrement utiles plus tard.

De plus j'avais déjà de bonnes bases en python et en C++ (~ langage Arduino) donc j'ai pu aider mes camarades dans la réalisation de leurs tâches quand ils avaient un problème.

Poursuite d'étude (Constantin Minos)

À la suite d'un entretien concluant, ma candidature a été retenue par l'ESAIP une école d'ingénieurs située à Angers.

Là-bas je ne sais pas encore vers quelle spécialité m'orienter mais mon choix porterait plutôt vers la cybersécurité ou pourquoi pas l'IOT(objets connectés) dans la continuité de ce projet.



III. Etudiant 2 (Corentin BRENY)

A. Rappel de la tâche de l'étudiant

Etudiant 2 (IR) : Fonctions à développer et tâches à effectuer	
<p>Liste des fonctions assurées par l'étudiant :</p> <p>Développement sur Arduino</p> <ul style="list-style-type: none"> Gérer le mécanisme 6 (capteurs photosensibles, bouton poussoir) Gérer le mécanisme 7 (capteur fin de course) <p>Développement de l'application sur Raspberry</p> <ul style="list-style-type: none"> Mémoriser l'état des mécanismes (valeur des capteurs et état des actionneurs) <p>Configurer le Serveur 13^e Porte</p> <ul style="list-style-type: none"> Installer les serveurs Apache et MySQL Créer la base de données <p>Sciences physiques : Étude des capteurs photosensibles</p>	<p>Installation :</p> <ul style="list-style-type: none"> Raspbian sur Raspberry PI 3B Serveur Apache et MySQL <p>Mise en œuvre :</p> <ul style="list-style-type: none"> Capteurs photosensibles Création de Base de données avec Workbench Développement Arduino Développement Python <p>Configuration :</p> <ul style="list-style-type: none"> Serveur Apache/MySQL Étalonnage des capteurs photosensibles <p>Réalisation :</p> <ul style="list-style-type: none"> Application sur Raspberry Application du mécanisme 6 : l'élément AIR Application du mécanisme 7 : le Katana <p>Documentation :</p> <ul style="list-style-type: none"> Participation à la rédaction du rapport de projet (spécifications, analyse, conception/réalisation) selon les cas d'utilisation confiés à l'étudiant.

image 35 : Bilan des tâches de l'Étudiant

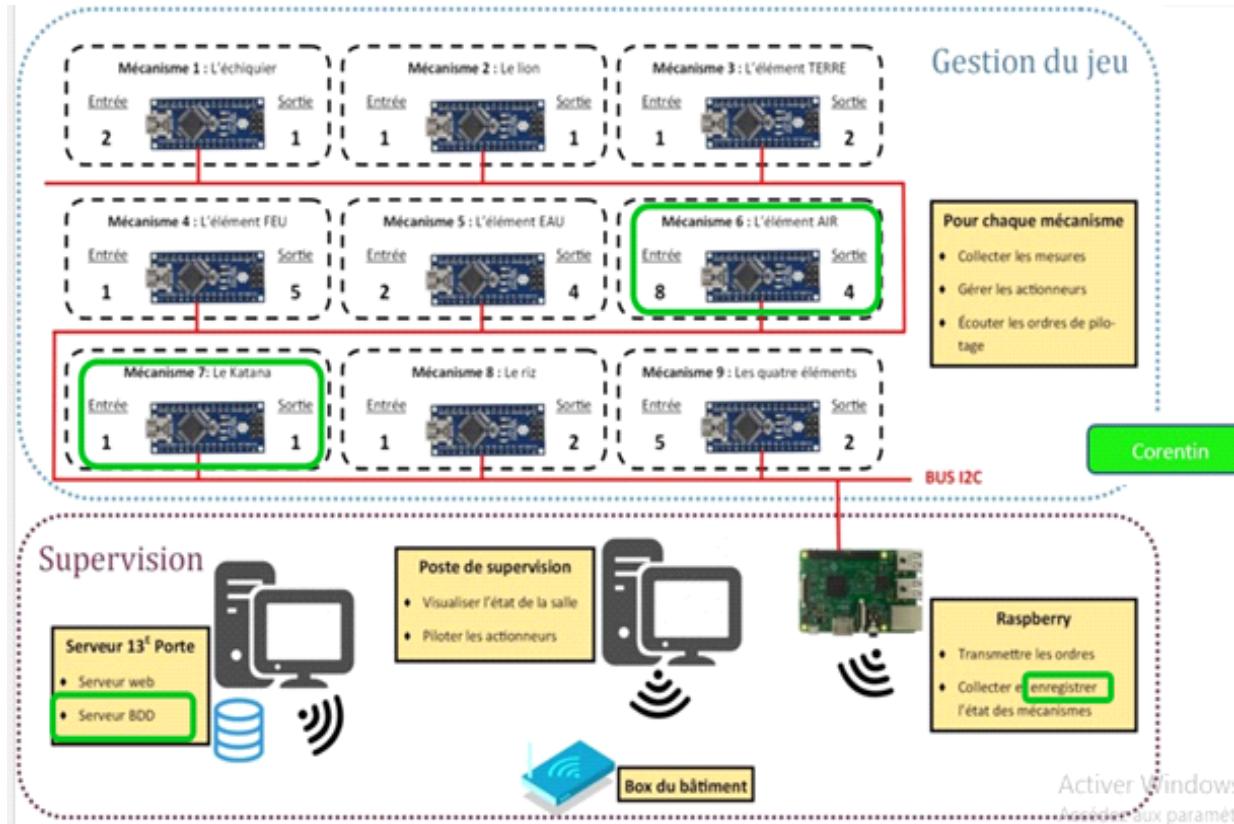
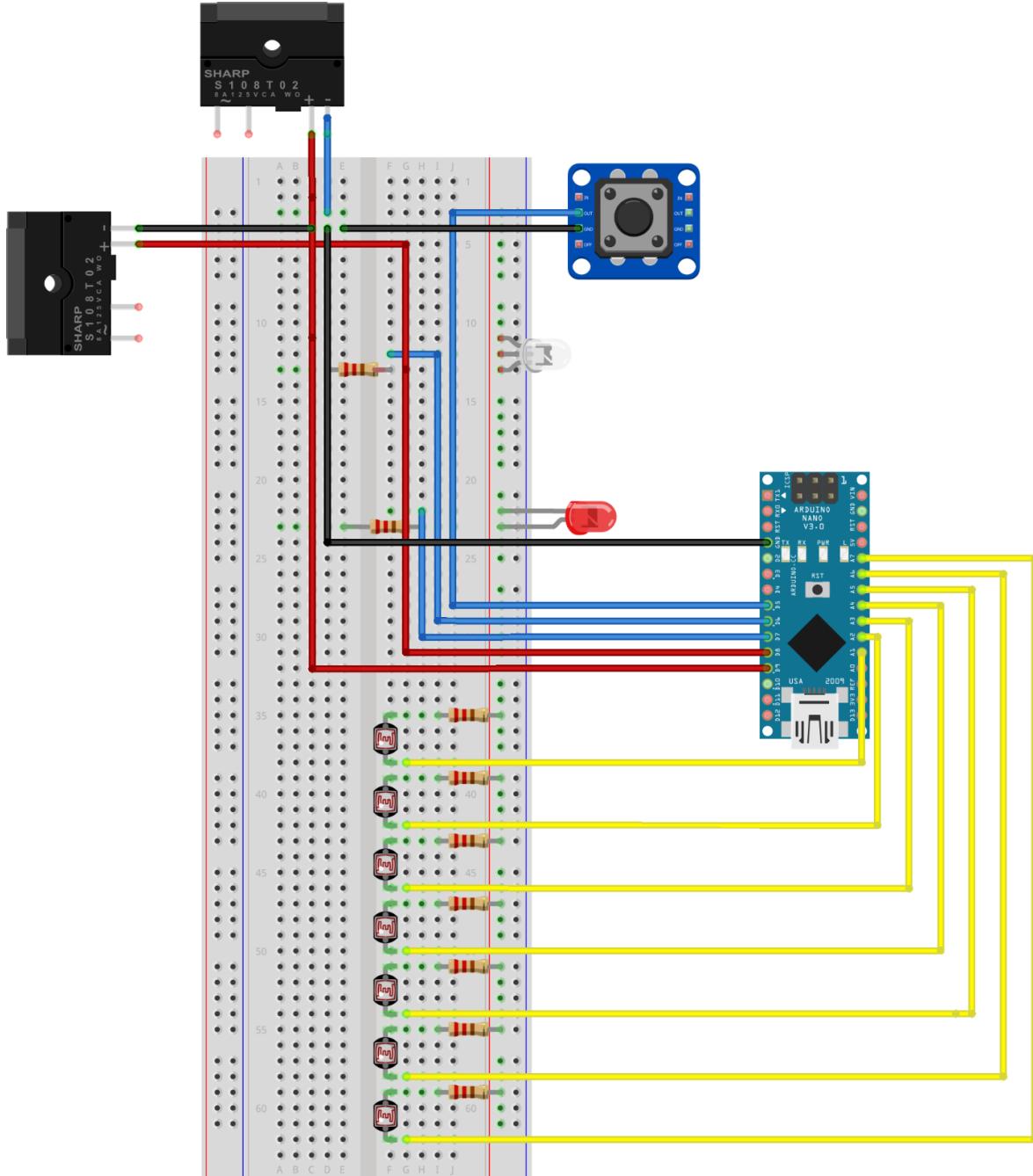


image 36 : Synopsis



B. Réalisation du programme Mécanisme 6 : Air

Schéma câblage



fritzing

image 37 : Câblage mécanisme Air (Corentin)



Programme

Initialisation

On définit les variables pour les Pins utilisées sur l'Arduino :

```
//Fichier Air.H

#include "Wire.h"
#define SLAVE_ADDRESS 0x16 //initialisation de l'Arduino avec l'adresse 0x16

#define CBouton_PIN 5      //Bouton Pousoir Chien
#define SLedVerte_PIN 6    //Led Verte Chien
#define SLedRouge_PIN 7    //Led Rouge Chien
#define SChien_PIN 8       //Relais Souffle Chien
#define SAir_PIN 9         //Relais Sarbacane

long timeChien = 0;      // La dernière fois que la broche de sortie à changé d'état
long debounce = 2000; // Temps de déparasitage (debounce time), a augmenter si la sortie clignote.
long debouncechien = 150;
int first = 0;
```

image 38 : Initialisation mécanisme Air (Corentin)

On initialise le matériel pour le mécanisme :

```
//SETUP
void Air::setupMecanism(){
pinMode(CBouton_PIN, INPUT_PULLUP);

pinMode(SLedVerte_PIN, OUTPUT);

pinMode(SLedRouge_PIN, OUTPUT);
digitalWrite(SLedRouge_PIN, LOW); // LED ROUGE

pinMode(SAir_PIN, OUTPUT);
digitalWrite(SAir_PIN, HIGH); //RELAIS CHIEN
```

image 39 : Setup mécanisme Air (Corentin)



La classe Air

Les principaux attributs :

```
private :  
    bool S_LedVerte; // Led verte pour prévenir la réussite de la combinaison des vannes  
    bool S_Chien; //L'actionneur où le chien souffle  
    bool S_LedRouge; //Led rouge pour prévenir l'échec de la combinaison des vannes  
    bool S_Air; // L'actionneur du relai pour l'air de la sardine  
    bool C_Vanne1; //les actionneurs des 7 vannes  
    bool C_Vanne2;  
    bool C_Vanne3;  
    bool C_Vanne4;  
    bool C_Vanne5;  
    bool C_Vanne6;  
    bool C_Vanne7;  
    bool C_Bouton; // L'actionneur du bouton poussoir  
    bool mecanism_status; // indique si le mécanisme est activé ou non|
```

image 40 : Attributs mécanisme Air (Corentin)

Les principales méthodes :

```
public :  
    Air();  
    void setupMecanism();  
    void execute();  
    void receive_order();  
    void send_status();
```

image 41 : Méthode mécanisme Air (Corentin)

*Le Main*

```
Air Mecanisme = Air();\n\nvoid receive_order(int numBytes) {\n    String data_received;\n\n    while(Wire.available() > 0) {          //Tant que le message n'est pas fini\n        char c = Wire.read();           //On lit le message\n        data_received += String(c);\n    }\n\n    if(data_received != "2") {\n\n        String order = data_received;\n        Serial.print("Order received : ");\n        Serial.println(order);//412221\n\n        if(order[1] == '1'){           //Si le 2eme caractère est 1\n            mechanism.setMechanism_status(true); //On valide le mécanisme\n        }else if(order[1] == '0'){       //Si le 2eme caractère est 0\n            mechanism.setMechanism_status(false); //On invalide le mécanisme\n        }\n\n        for(int i=2; i<sizeof(order); i++) {      //Pour chaque actionneur\n            if(order[i] == '1'){           //Si le caractère est 1\n                mechanism.actuator[i-1] = true; //On valide l'actionneur\n            }else if(order[i] == '0'){       //Si le caractère est 0\n                mechanism.actuator[i-1] = false; //On invalide l'actionneur\n            }\n        }\n    }\n}
```

image 42 : Main mécanisme Air (Corentin)



Exécute

```
void Air::execute() {

    int VanOkSeuil = 10;
    vanne1 = analogRead(S_Vanne1) >= VanOkSeuil ? 1 : 0; //Lis la valeur de la tension à la broche de la vanne 1, si celle-ci est ouverte ou fermée
    vanne2 = analogRead(S_Vanne2) >= VanOkSeuil ? 1 : 0; //Lis la valeur de la tension à la broche de la vanne 2, si celle-ci est ouverte ou fermée
    vanne3 = analogRead(S_Vanne3) >= VanOkSeuil ? 1 : 0; //Lis la valeur de la tension à la broche de la vanne 3, si celle-ci est ouverte ou fermée
    vanne4 = analogRead(S_Vanne4) >= VanOkSeuil ? 1 : 0; //Lis la valeur de la tension à la broche de la vanne 4, si celle-ci est ouverte ou fermée
    vanne5 = analogRead(S_Vanne5) >= VanOkSeuil ? 1 : 0; //Lis la valeur de la tension à la broche de la vanne 5, si celle-ci est ouverte ou fermée
    vanne6 = analogRead(S_Vanne6) >= VanOkSeuil ? 1 : 0; //Lis la valeur de la tension à la broche de la vanne 6, si celle-ci est ouverte ou fermée
    vanne7 = analogRead(S_Vanne7) >= VanOkSeuil ? 1 : 0; //Lis la valeur de la tension à la broche de la vanne 7, si celle-ci est ouverte ou fermée
    long resultCode = vanne1 * 1000000 + vanne2 * 100000 + vanne3 * 10000 + vanne4 * 1000 + vanne5 * 100 + vanne6 * 10 + vanne7; // On calcule le code
    //Afin de valider si les vannes sont bien coordonnées
    // 1 = ouvert, 0 = fermée
    long codeToDo = 11010111; // resultat du code de validité
    // digitalWrite(SLedVerte_PIN, LOW); // LED VERT

    readingchien = digitalRead(CBouton_PIN); // Lis l'état du bouton poussoir
}
```

image 43

```
if (readingchien != previouschien) { // Remettre la minuterie/timer de déparasitage à 0
    timeChien = millis();
}

if ((millis() - timeChien) > debouncechien) {
    if (readingchien == 0) {
        if (codeToDo == resultCode) { // SI le code vanne correspond au code voulu

            if (firstchien == 0) {
                statechien = LOW;
                digitalWrite(SAir_PIN, LOW); // On désactive le Relais sarbacane
                digitalWrite(SLedVerte_PIN, HIGH); // On active la LED vert
                delay(3000); // Attendre 3secondes
                firstchien = 1;
                digitalWrite(SAir_PIN, HIGH); //On reactive le Relais sarbacane
            }
            else {
                digitalWrite(SLedVerte_PIN, LOW); //On désactive la LED verte
                delay(500); // Attendre 0,5 seconde
                digitalWrite(SLedVerte_PIN, HIGH); //On active la LED verte
                delay(500); // Attendre 0,5 seconde
                digitalWrite(SLedVerte_PIN, LOW); //On désactive la LED verte
                delay(500); // Attendre 0,5 seconde
                digitalWrite(SLedVerte_PIN, HIGH); //On active la LED verte
                delay(500); // Attendre 0,5 seconde
            }
        }
    }
}
```

image 44



```

        }
        else {
            if (firstchien == 0 ) {
                digitalWrite(SLedRouge_PIN, HIGH); // On active la LED ROUGE
                digitalWrite(SChien_PIN, LOW); // On désactive le RELAIS SOUFFLE CHIEN
                delay(200); // Attendre 0,2 seconde
                digitalWrite(SChien_PIN, HIGH); //On reactive le RELAIS SOUFFLE CHIEN
                digitalWrite(SLedRouge_PIN, HIGH); //On désactive la LED ROUGE
                delay(1000); // Attendre 1 seconde
                digitalWrite(SLedRouge_PIN, LOW); //On active la LED ROUGE
            }
            else {
                digitalWrite(SLedVerte_PIN , LOW); //On désactive la LED verte
                delay(500); // Attendre 0,5 seconde
                digitalWrite(SLedVerte_PIN , HIGH); //On active la LED verte
                delay(500); // Attendre 0,5 seconde
                digitalWrite(SLedVerte_PIN , LOW); //On désactive la LED verte
                delay(500); // Attendre 0,5 seconde
                digitalWrite(SLedVerte_PIN , HIGH); //On active la LED verte
                delay(500); // Attendre 0,5 seconde
            }
        }
    }
}

digitalWrite(SLedVerte_PIN, not(statechien)); // LED VERT

previouschien = readingchien;

```

image 45

```

Serial.print("vanne1 = "); // Imprime les données sur le port série sous forme de texte
Serial.print(vanne1); // the raw analog reading
Serial.print(" vanne2: ");// Imprime les données sur le port série sous forme de texte
Serial.print(vanne2);
Serial.print(" vanne3: ");// Imprime les données sur le port série sous forme de texte
Serial.print(vanne3);
Serial.print(" vanne4: ");// Imprime les données sur le port série sous forme de texte
Serial.print(vanne4);
Serial.print(" vanne5: ");// Imprime les données sur le port série sous forme de texte
Serial.print(vanne5);
Serial.print(" vanne6: ");// Imprime les données sur le port série sous forme de texte
Serial.print(vanne6);
Serial.print(" vanne7: ");// Imprime les données sur le port série sous forme de texte
Serial.println(vanne7);
Serial.println(readingchien);

```

image 46 : Exécute mécanisme Air (Corentin)



C. Réalisation du programme Mécanisme 7 : Katana

Schéma câblage

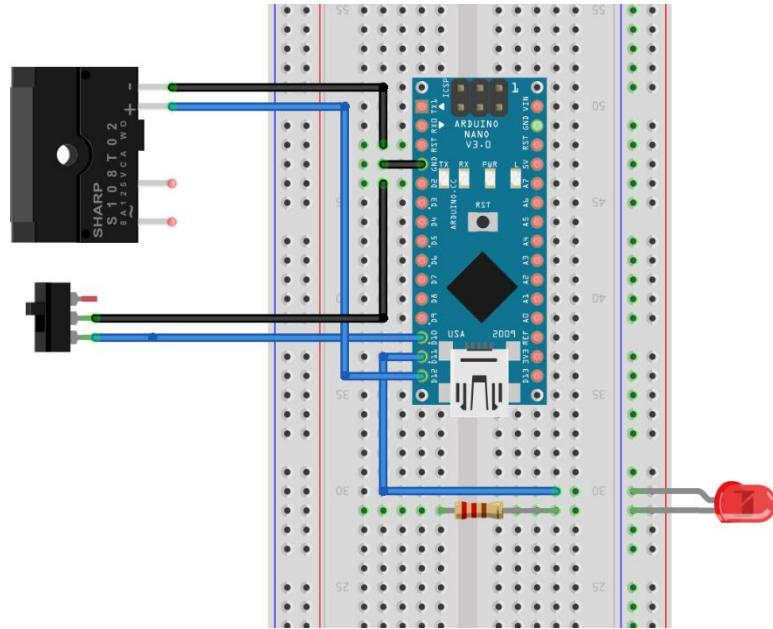


image 47 : Câblage mécanisme Katana (Corentin)

Programme

Initialisation

On définit les variables pour les Pins utilisées sur l'Arduino :

```
#include "Wire.h"
#define SLAVE_ADDRESS 0x16 //initialisation de l'Arduino avec l'adresse 0x16

#define CIInterrupteur_PIN 10 // Interrupteur Katana
#define SLed_PIN 11 // Led de contrôle
#define SKatana_PIN 12 //Relai Katana

int statekatana = HIGH; // Etat actuel de la broche de sortie (la LED)
int readingkatana; // Contient la valeur lue sur la broche d'entrée
int previouskatana = LOW; // Contient la précédente valeur lue sur la broche d'entrée
int firstkatana = 0;
long timeKatana;
```

image 48 : Initialisation mécanisme Katana (Corentin)



On initialise le matériel pour le mécanisme :

```
void Katana::setupMecanism() {  
  
    pinMode(CInterrupteur_PIN 10, INPUT_PULLUP);  
    pinMode(SLed_PIN 11, OUTPUT);  
  
    pinMode(SKatana_PIN 12, OUTPUT);  
    digitalWrite(SKatana_PIN 12, HIGH); // relai doigt
```

image 49 : Setup mécanisme Katana (Corentin)

La classe Katana

Les principales méthodes :

```
public :  
    Katana();  
    void setupMecanism();  
    void execute();  
    void receive_order();  
    void send_status();
```

image 50 : Méthodes mécanisme Katana (Corentin)



Le Main

```

Katana Mecanisme = Katana();

void receive_order(int numBytes) {
    String data_received;

    while(Wire.available() > 0) {           //Tant que le message n'est pas fini
        char c = Wire.read();             //On lit le message
        data_received += String(c);
    }

    if(data_received != "2") {

        String order = data_received;
        Serial.print("Order received : ");
        Serial.println(order);//412221

        if(order[1] == '1'){           //Si le 2eme caractère est 1
            mechanism.setMechanism_status(true); //On valide le mécanisme
        }else if(order[1] == '0'){       //Si le 2eme caractère est 0
            mechanism.setMechanism_status(false); //On invalide le mécanisme
        }

        for(int i=2; i<sizeof(order); i++) {      //Pour chaque actionneur
            if(order[i] == '1'){           //Si le caractère est 1
                mechanism.actuator[i-1] = true; //On valide l'actionneur
            }else if(order[i] == '0'){       //Si le caractère est 0
                mechanism.actuator[i-1] = false; //On invalide l'actionneur
            }
        }
    }
}

```

image 51 : Main mécanisme Katana (Corentin)



Exécute

```
void Katana::execute() {  
  
    readingkatana = digitalRead(CInterruppteur_PIN ); // Lis l'état de l'interrupteur  
  
    //on tient à vérifier si il y a eu un changement de position ou un parasite (bille qui trésaute)...  
    if (readingkatana != previouskatana) { // Remettre la minuterie/timer de déparasitage à 0  
        timeKatana = millis();  
    }  
  
    if (((millis() - timeKatana) > (debounce))) { //On vérifie s'il y a eu un changement de position positif  
        if (readingkatana == 0) {  
            if (firstkatana == 0) {  
                statekatana = LOW;  
                digitalWrite(SKatana_PIN , LOW); // On désactive le relai katana  
                delay(500); // attendre 0,5 seconde  
                digitalWrite(SKatana_PIN , HIGH); // On réactive le relai katana  
                firstkatana = 1; // On change la valeur du katana  
            }  
        }  
        else {  
            firstkatana = 0 ;  
        }  
    }  
    digitalWrite(SLed_PIN 11, not(statekatana));  
    previouskatana = readingkatana;  
}
```

image 52 : Exécute mécanisme Katana (Corentin)



D. Création de la base de données

La base de données a été créée et elle contient trois tables :



Table	Action	Lignes	Type	Interclassement	Taille	Perte
actionneurs	Afficher Structure Rechercher Insérer Vider Supprimer	~15	InnoDB	latin1_swedish_ci	32 Kio	-
capteurs	Afficher Structure Rechercher Insérer Vider Supprimer	~22	InnoDB	latin1_swedish_ci	32 Kio	-
general	Afficher Structure Rechercher Insérer Vider Supprimer	~9	InnoDB	latin1_swedish_ci	16 Kio	-
3 tables	Somme	46	InnoDB	latin1_swedish_ci	80 Kio	0 o

image 53 : Vue générale de la BDD (Corentin)



La table « générale »

			ID_mecanismes	Numero_Salle	Nom_mecanisme	Etat
<input type="checkbox"/>	Modifier	Copier	Effacer	1	1	Echiquier
<input type="checkbox"/>	Modifier	Copier	Effacer	2	1	Le Lion
<input type="checkbox"/>	Modifier	Copier	Effacer	3	1	Terre
<input type="checkbox"/>	Modifier	Copier	Effacer	4	1	Feu
<input type="checkbox"/>	Modifier	Copier	Effacer	5	1	Eau
<input type="checkbox"/>	Modifier	Copier	Effacer	6	1	Air
<input type="checkbox"/>	Modifier	Copier	Effacer	7	1	Katana
<input type="checkbox"/>	Modifier	Copier	Effacer	8	1	Riz
<input type="checkbox"/>	Modifier	Copier	Effacer	9	1	Quatre Elements

0

image 54 : Vue de la table générale de la BDD (Corentin)

La table « actionneurs »

			ID_actionneurs	ID_mecanismes	Numero_Salle	type	Etat	Heure_derniere_mesure
<input type="checkbox"/>	Modifier	Copier	Effacer	1	1	led rouge	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	2	1	led rouge	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	3	1	led rouge	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	4	1	led verte	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	5	1	led rouge	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	6	1	led verte	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	7	1	led verte	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	8	1	led rouge	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	9	1	led verte	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	10	1	led rouge	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	11	1	led rouge	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	12	1	led verte	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	13	1	led verte	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	14	1	led verte	TRUE	2020-03-24 20:51:52
<input type="checkbox"/>	Modifier	Copier	Effacer	15	1	led verte	TRUE	2020-03-24 20:51:52

image 55 : Vue de la table actionneur de la BDD (Corentin)



La table « capteurs »

	<input type="button" value="T"/> <input type="button" value="←"/> <input type="button" value="→"/>		ID_capteurs	ID_mecanismes	Numero_Salle	Type	Valeur	Heure_derniere_mesure
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		1	1	1	Plateau	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		2	1	1	Plateau	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		3	2	1	Hall	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		4	3	1	Hall	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		5	4	1	Cle	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		6	4	1	Trappe	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		7	4	1	Fumee	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		8	5	1	Humidite	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		9	5	1	Fontaine	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		10	5	1	Frigo	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		11	6	1	Vanne	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		12	6	1	Vanne	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		13	6	1	Vanne	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		14	6	1	Vanne	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		15	6	1	Vanne	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		16	6	1	Vanne	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		17	6	1	Vanne	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		18	7	1	Chien	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		19	7	1	Course	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		20	8	1	Poids	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		21	8	1	Tableau	0	2020-03-24 20:51:09
<input type="checkbox"/>	<input type="button" value="Modifier"/> <input type="button" value="Copier"/> <input type="button" value="Effacer"/>		22	9	1	Poussoir	0	2020-03-24 20:51:09

image 56 : Vue de la table capteur de la BDD (Corentin)

Remarques

Dans la création de la base de données, le point le plus complexe à réaliser est la mise en place des clés étrangères afin de permettre aux différentes tables de communiquer entre elles. Pour cela il faut créer une clé primaire qui sera reprise dans la seconde table puis ensuite y indiquer le chemin à prendre dans les contraintes pour qu'elles aient la même valeur.

Cela est explicitement expliqué dans la procédure envoyée à la société si elle devait créer une nouvelle base de données à l'avenir.

Le champs « Numéro_Salle » est présent au cas où l'entreprise voudrait créer une nouvelle salle, alors elle pourrait ajouter les capteurs et actionneurs à cette base de données et n'aurait pas à recréer une nouvelle base de données.

Procédure d'accès à la base de données

Une procédure a été fourni à la société **13ème Porte** afin de permettre à cette entreprise de pouvoir accéder à la base de données depuis un de leur poste informatique.

*Connexion à la base de données*

```
import mysql.connector  
  
conn = mysql.connector.connect(host="localhost", user="root", password="", database="bdd_escape_game")  
cursor=conn.cursor()
```

image 57



Insérer et enregistrer des données dans la base de données

```
#Ecrire les opérations à réaliser sur la bdd

#Insérer des données dans la table
curseur.execute("Insert INTO actionneurs (ID_Actionneur, Etat, Valeur) VALUES (%s, %s, %s)", ("1", "OK", "20"))

#Insérer des données depuis un dictionnaire
capteur = [
    {"ID_Actionneur": "1", "Etat": "OK", "Valeur": "20"},

]
for nbRelevés in actionneur:
    curseur.execute("Insert INTO actionneurs (ID_Actionneur, Etat, Valeur) VALUES (%(ID_Actionneur)s, %(Etat)s, %(Valeur)s)", nbRelevés)

conn.commit() #sert à sauvegarder les données dans la table avant de fermer la connexion.
conn.close()
```

image 58 : Insérer et Enregistrer des données dans la BDD (Corentin)

Adaptation au projet

```
def send_SDtoDataBase(arduino_id, sensor_name, sensor_data):
    """
    send sensor data to DataBase
    """
    #Corentin : Ecrire code
    curseur.execute("Insert INTO capteurs (ID_Capteurs, Nom_Capteurs, Valeur) VALUES (%s, %d, %s)", ("sensor_id", "sensor_name", "sensor_data"))
    conn.commit() #sert à sauvegarder les données dans la table avant de fermer la connexion.

def send_AStoDataBase(arduino_id, actuator_name, actuator_status):
    """
    send actuator status to DataBase
    """
    #Corentin : Ecrire code
    curseur.execute("Insert INTO actionneurs (ID_Actionneur, Nom_Actionneur, Etat) VALUES (%s, %d, %s)", ("actuator_id", "actuator_name", "actuator_status"))
    conn.commit() #sert à sauvegarder les données dans la table avant de fermer la connexion.

def send_MStoDataBase(arduino_id, mechanism_status):
    """
    send mechanism status to DataBase
    """
    #Corentin : Ecrire code
    curseur.execute("Insert INTO general (ID_mecanisme, Etat) VALUES (%s, %s)", ("arduino_id", "mechanism_status"))
    conn.commit() #sert à sauvegarder les données dans la table avant de fermer la connexion.
    conn.close()
```

image 59 : Adaptation pour la BDD (Corentin)



E. La photorésistance en Physique-Appliquée

Un cristal de semi-conducteur à température basse contient peu d'électrons libres. La conductivité du cristal est très faible, proche de celle d'un isolant. Lorsque la température du cristal augmente de plus en plus d'électrons qui étaient immobilisés dans les liaisons covalentes s'échappent et peuvent participer à la conduction. A température constante si le même cristal semi-conducteur est soumis à une radiation lumineuse, l'énergie apportée par les photons peut suffire à libérer certains électrons utilisés dans les liaisons covalentes entre atomes du cristal. Plus le flux lumineux sera intense, plus le nombre d'électrons disponibles pour assurer la conduction sera grand, ainsi la résistance de la photorésistance est inversement proportionnelle à la lumière reçue.

Une photorésistance est généralement utilisée pour mesurer une intensité lumineuse (appareil photo, systèmes de détection...). Elle est constituée d'un matériau semi-conducteur.

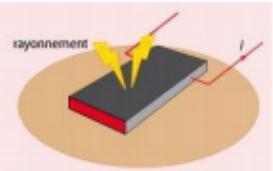


Figure 1. Principe d'une photorésistance.

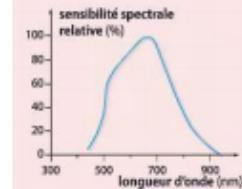


Figure 2. Sensibilité spectrale d'une photorésistance en sulfure de cadmium.



Figure 3. Résistance d'une photorésistance en fonction du flux lumineux.

image 60

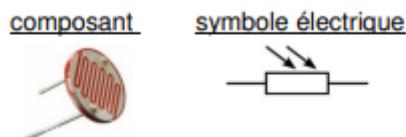


image 61



F. Observer le fonctionnement du mécanisme 6 : Air

Nom :	Observer le fonctionnement du mécanisme 6 : l'Air
Recette :	Technique
Objectif	Lancer le programme afin qu'il s'exécute correctement.
Elément à tester	Air.ino
Pré requis	Ouvrir Air.ino dans l'IDE Arduino et réaliser les câblages

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		Un message indique « téléchargement terminé »	<input type="checkbox"/>
2	Changer la position des vannes		Le mécanisme s'exécute correctement	<input type="checkbox"/>

Rapport de test	Testé par : L'Etudiant 2	Le :
Conformité		
<input type="checkbox"/> Excellente <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible		
Commentaire :		Approbation :



G. Observer le fonctionnement du mécanisme 7 : Katana

Nom :	Observer le fonctionnement du mécanisme 7 : Katana
Recette :	Technique
Objectif	Lancer le programme afin qu'il s'exécute correctement.
Elément à tester	Katana.ino
Pré requis	Ouvrir Katana.ino dans l'IDE Arduino et réaliser les câblages

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		Un message indique « téléchargement terminé »	<input type="checkbox"/>
2	Changer la position du capteur fin de course sur On		Le mécanisme s'exécute correctement	<input type="checkbox"/>

Rapport de test	Testé par : L'Etudiant 2	Le :
Conformité		
<input type="checkbox"/> Excellente <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible		
Commentaire :	Approbation :	



H. Regard critique du projet (Corentin Breny)

J'ai choisi ce projet car le fait d'être en relation avec une entreprise est plus motivant. Cela enrichit également notre réseau professionnel qui peut nous être favorable pour notre avenir. De plus, le travail sur la base de données est un point qui me donnait envie d'en apprendre le plus possible sur cela et de le mettre à l'usage.

I. Connaissances apportées (Corentin Breny)

Ce projet m'a beaucoup apporté notamment sur les points suivants :

- La base de données (création, connexion à distance, clés étrangères, ...)
- Arduino (faire les montages, programmation sur Arduino, ...)

J. Ce qui me reste à faire (Corentin Breny)

- Finalisation des programmes Arduino
- Finalisation de la mémorisation des états des actionneurs
- Effectuer les montages des mécanismes
- Tester le travail

K. Poursuite d'étude (Corentin Breny)

J'aimerais effectuer une licence professionnelle métiers des réseaux informatiques et télécommunication en alternance afin dans l'avenir devenir technicien de maintenance informatique.

Si je ne trouve pas d'alternance, je chercherais donc un emploi dans l'informatique.



IV. Étudiant 3 (Joshua PINNEAU)

A. Rappel des tâches de l'étudiant

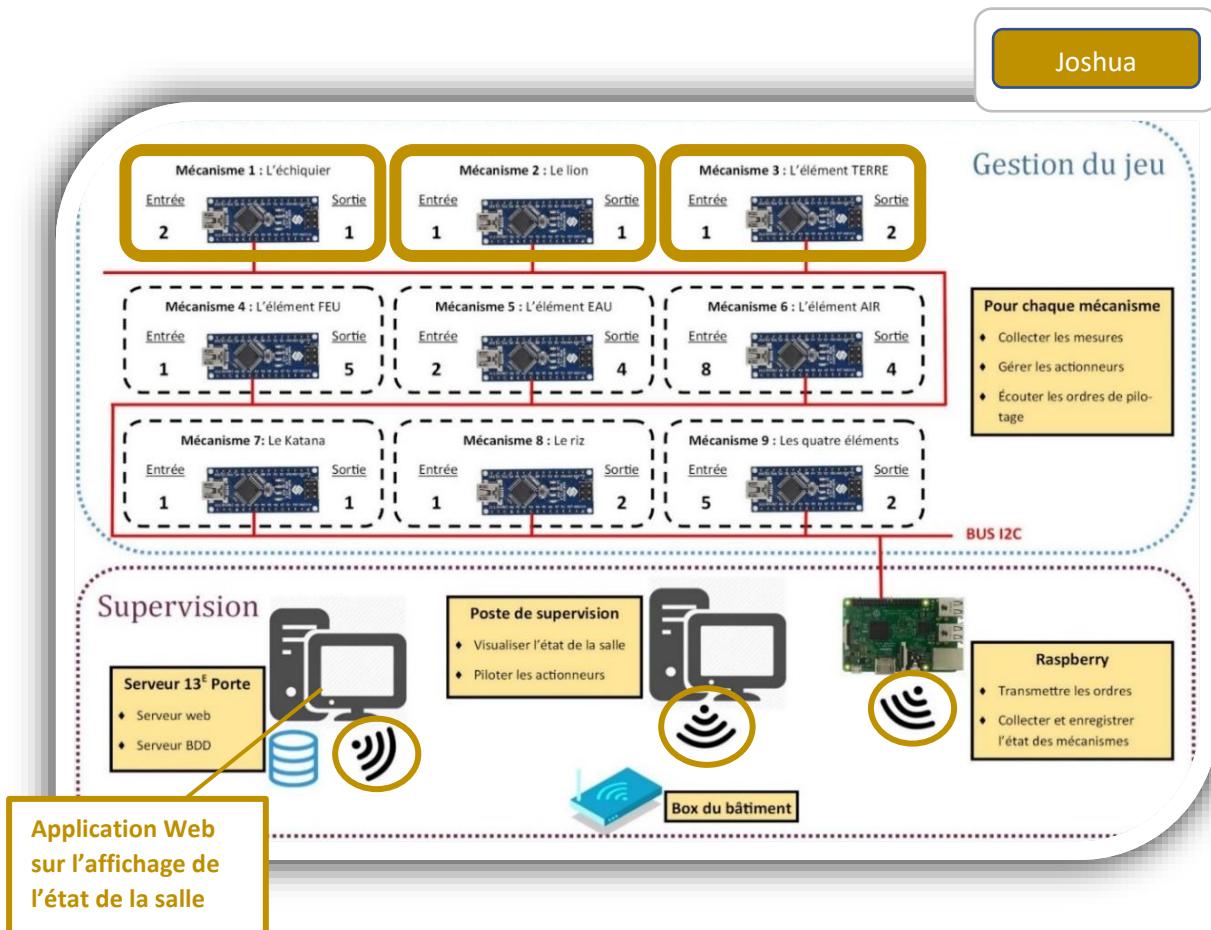


image 62 : Synoptique du projet avec les tâches de l'étudiant

Mécanisme 1 : l'échiquier

Le mécanisme repose sur deux capteurs à effet Hall.

Selon une condition portant sur les valeurs de ces deux capteurs, un électroaimant est activé ou désactivé via un relais et une LED s'allume ou non sur le tableau de contrôle.

Mécanisme 2 : le lion basculant

Le mécanisme repose sur un capteur à effet Hall.

Selon la mesure, un électroaimant est activé ou désactivé via un relais et une LED s'allume ou non sur le tableau de contrôle.

Mécanisme 3 : l'élément TERRE

Le mécanisme repose sur un capteur à effet Hall.



Selon la mesure, (i) un moteur et une LED sont activés ou désactivés via un relais pendant un laps de temps (ii) une sortie est paramétrée en destination du mécanisme des quatre éléments.

Visualiser l'état de la salle

Depuis une application WEB, le superviseur peut visualiser l'état de la salle. Pour chacun des neuf mécanismes, les valeurs des capteurs d'entrée sont affichées ainsi que l'état des différents actionneurs (état allumé ou éteint).

Partie réseau

La partie réseau est constituée d'un schéma réseau de l'installation des différents éléments, un tutoriel pour l'utilisation de chacun des éléments a été transmis à l'entreprise.

Partie physique : l'effet Hall

Une partie physique doit être travaillée concernant l'effet Hall qui est utilisé dans les trois mécanismes de l'étudiant.

B. Partie physique : l'effet Hall

Si une plaque conductrice est :

- Parcourue par un courant
- Plongée dans un champ magnétique

Alors une tension électrique apparaît aux bornes de celle-ci

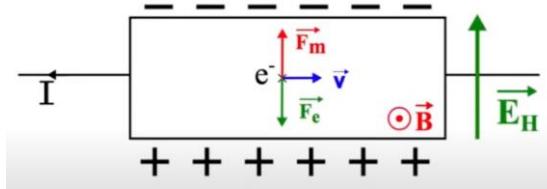


image 63 : Schéma d'une plaque conductrice traversée par un courant « I » plongée dans un champ magnétique « B »

Soit une plaque conductrice traversée par un courant d'intensité « I » plongée dans un champ magnétique « B », les électrons sont déviés vers le haut de la plaque qui se charge négativement. À la suite d'un défaut de charges négatives en bas de la plaque, elle se charge donc positivement. Cette répartition de charge provoque l'apparition d'un champ électrique dirigé de bas en haut de la plaque, de la zone positive vers la zone négative, c'est le champ électrique de Hall, E_H . Une différence de potentiel, ou tension de Hall, restera mesurable entre le bas et le haut, c'est la constante de Hall, « C_H ».

I : intensité du courant

h : hauteur de la plaque

C_H : constante de Hall

$$U_H = \frac{C_H \times I \times B}{h}$$

image 64 : Formule permettant d'obtenir la valeur d'un champ magnétique en fonction de la Constante de Hall



Si on connaît l'intensité du courant qui parcours la plaque « I », sa hauteur « h » ainsi que sa constante de Hall « C_H ». En mesurant la tension de Hall « U_H », on peut remonter à la valeur du champ magnétique « B » par la formule ci-dessus.

Applications possibles de l'effet Hall : Mesure de champ magnétique et capteur de mouvement.

C. Mécanisme N°1 : L'Échiquier

Synopsis du mécanisme

Le mécanisme repose sur deux capteurs à effet Hall.

Selon une condition portant sur les valeurs de ces deux capteurs, un électroaimant est activé ou désactivé via un relais et une LED s'allume ou non sur le tableau de contrôle.

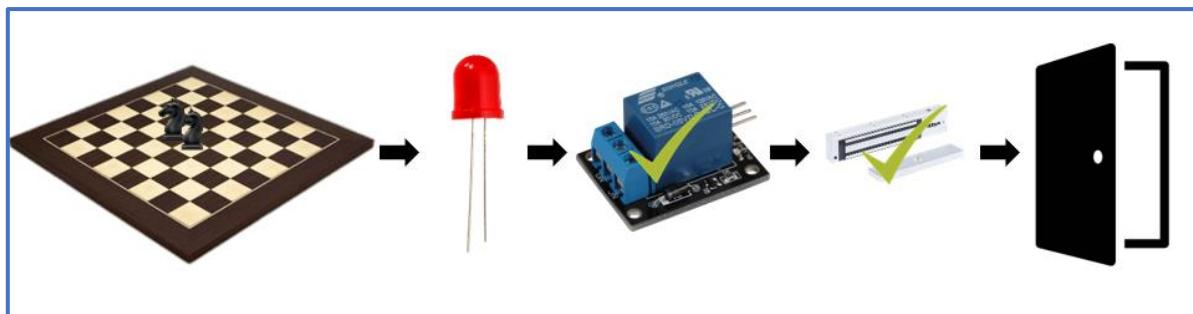


image 65 – Schéma explicatif du mécanisme

Schéma électrique du mécanisme

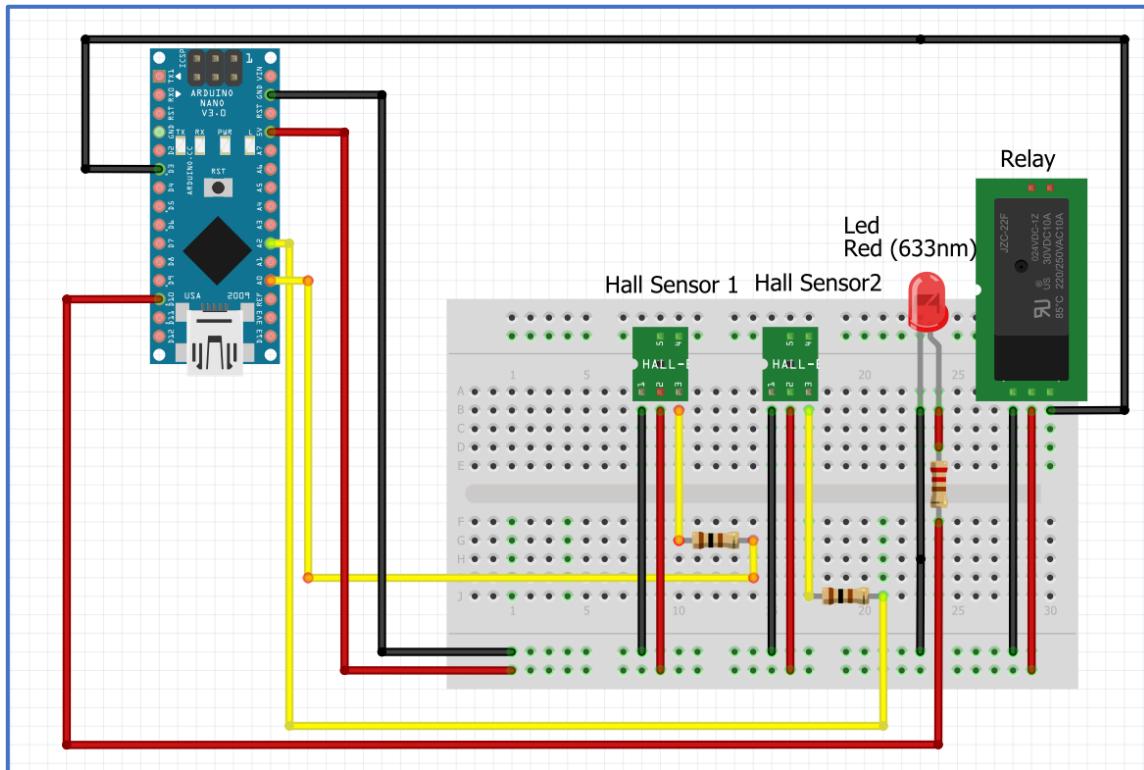


image 66 – Schéma électrique



Programmation du mécanisme

Différents composants du mécanisme

```
#define SLAVE_ADDRESS 0x12
#define CEffetHall1_PIN A0 //Capteur à effet Hall 1
#define CEffetHall2_PIN 2 //Capteur à effet Hall 2
#define SLed_PIN 7 //Led de sortie
#define SEchiquier_PIN 8 //électroaimant
#define DEBOUNCE 2000
```

image 67 – Présentation des différents composants

C'est ici que l'on va déclarer sur quel PIN de la carte Arduino Nano seront les différents éléments présents sur le schéma électrique ci-dessus.

Le premier « `#define` » va servir à l'étudiant n°1 pour le modèle « maître-esclave », il pourra, avec cette ligne, faire interagir les différentes cartes Arduino avec la Raspberry par le biais du bus I2C. Ensuite, nous avons nos deux capteurs à effet Hall, la LED qui sera présente sur le tableau de contrôle du gérant de la société et enfin l'électroaimant qui va libérer la porte.

Fichier Echec.h

La programmation est effectuée sur le logiciel Arduino, on peut y coder nos mécanismes en langage C++. Par conséquent, nous allons coder notre mécanisme Echec avec deux fichiers distincts « `Echec.h` » et « `Echec.cpp` » dans le même projet Arduino, comme suit :

```
class Echec{

private :
    bool C_EffetHall1;
    bool C_EffetHall2;
    bool S_Echiquier;
    bool S_Led;
    bool mechanism_status;

public :
    bool actuator[2] = {S_Echiquier, S_Led};
    int sensor[2] = {C_EffetHall2, C_EffetHall2};
    bool getMechanism_status();
    void setMechanism_status(bool ms);

public :
    Echec();
    void setupMechanism();
    void execute();
};
```

image 68 – Fichier Echec.h



Ceci est la partie du fichier « `Echec.h` », on y définit les deux capteurs à effet Hall, l'actionneur « `S_Echiquier` » et d'autres fonctions qui vont nous permettre de faire fonctionner le mécanisme.

Condition qui valide le mécanisme

```

if (sd_reading == LOW && pin2 == LOW
&& (millis() - ms_time) > DEBOUNCE){      //On vérifie que les pièces sont en place depuis un certain temps (DEBOUNCE)

    C_EffetHall1 = true;                  //On fixe la valeur de l'attribut capteur
    C_EffetHall2 = true;                  //On fixe la valeur de l'attribut capteur

    if (mechanism_status == false) {     //Si il y a eu le premier changement de position

        S_Echiquier = true;              //On active l'électroaimant de la ventouse
        S_Led = true;                   //On allume la led rouge
        mechanism_status = true;        //On valide le mécanisme
    }
} else{                                //Si il n'y a pas eu de changement de position positif

    C_EffetHall1 = false;               //On fixe la valeur de l'attribut capteur
    C_EffetHall2 = false;               //On fixe la valeur de l'attribut capteur
}

```

image 69 – Condition pour vérifier que les pièces sont en place dans le fichier « `Echec.cpp` »

L'image ci-dessus est la condition qui vérifie que les deux cavaliers sont sur les bonnes cases, c'est le test : « `sd_reading == LOW && pin2 == LOW` ». « `sd_reading` » correspond au premier capteur à effet Hall et « `pin2` » au deuxième. La notion de déphasage et de « `DEBOUNCE` » permet d'éviter que le client de l'Escape Game ne choisit pas des cases par hasard, « `DEBOUNCE` » est définie sur 2 secondes, donc le client doit laisser les pièces deux secondes sur les bonnes cases.

Condition lorsque le mécanisme est validé

```

if ( S_Echiquier == true ){           //Pour désactiver l'electroaimant de la ventouse
    delay(100);                     //On attend 0.1 seconde
    digitalWrite(SEchiquier_PIN, LOW); // Écriture du statut sur la ventouse (LOW = on libère l'electroaimant)
    delay(500);
    digitalWrite(SEchiquier_PIN, HIGH);
    delay(4000);                   //On attend 4 secondes
    S_Echiquier = false;            //On change la valeur de l'attribut
}

```

image 70 – Condition selon laquelle le mécanisme est validé

Cette condition nous montre comment l'actionneur fonctionne. Une fois le mécanisme validé, on désactive l'électroaimant, pour libérer la porte, c'est la ligne « `digitalWrite(SEchiquier_PIN, LOW);` » qui nous le permet. Ensuite, on attend un demi second et on réactive l'électroaimant. Finalement, on attend 4 secondes et on change la valeur de l'attribut « `S_Echiquier` » à `false` pour réinitialiser le mécanisme.



D. Mécanisme N°2 : Le lion basculant

Synopsis du mécanisme

Le mécanisme repose sur un **capteur à effet Hall**.

Selon la mesure, un électroaimant est activé ou désactivé via un relais et une LED s'allume ou non sur le tableau de contrôle.

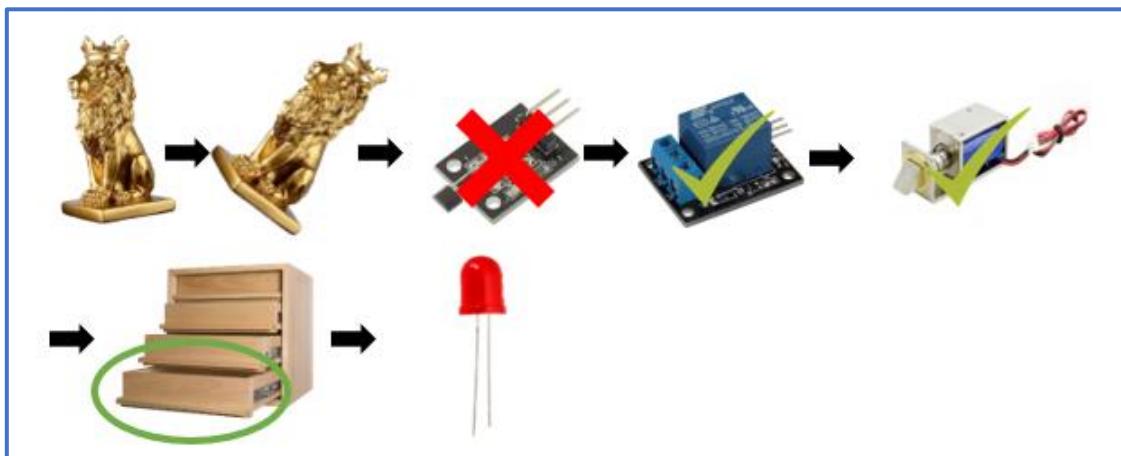


image 71 – schéma explicatif du mécanisme

Schéma électrique du mécanisme

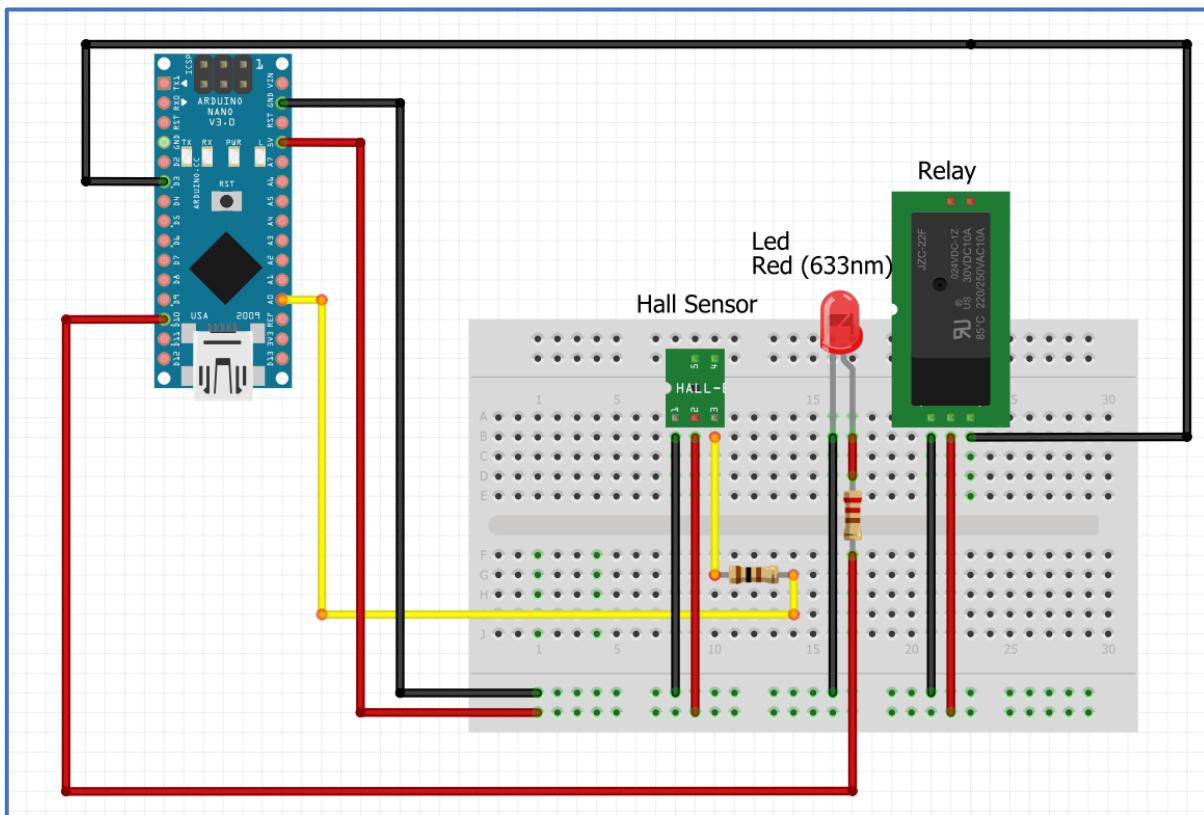


image 72 - Schéma électrique



Programmation du mécanisme

Différents composants du mécanisme

```
#define SLAVE_ADDRESS 0x13
#define CEffetHall1_PIN A0 //Capteur à effet Hall 1
#define SLed_PIN 10 //Led de sortie
#define SLion_PIN 3 //Actionneur
#define DEBOUNCE 2000
```

image 73 – Différents composants

C'est ici que l'on va déclarer sur quel PIN de la carte Arduino Nano seront les différents éléments présents sur le schéma électrique ci-dessus. Contrairement au premier mécanisme, on a ici qu'un seul capteur à effet Hall. Cependant, il y a aussi une Led « `SLed_PIN` » et un actionneur « `SLion_PIN` ».

Fichier LionB.h

Comme pour le mécanisme précédent, on code avec un fichier `LionB.h` et un fichier `LionB.cpp`. Voici le fichier `LionB.h` :

```
class LionB {

private :
    bool C_EffetHall_1;
    bool S_Lion;
    bool S_Led;
    bool mecanism_status;

private :
    bool actuator[2] = {S_Lion, S_Led};
    bool sensor[1] = {C_EffetHall_1};
    bool getMechanism_status();
    void setMechanism_status(bool ms);

public :
    LionB();
    void setupMechanism();
    void execute();
};
```

Image 74 – `LionB.h`

Ceci est le fichier « `LionB.h` » du mécanisme « Le Lion Basculant ». Il est similaire à celui du premier mécanisme. Cependant, il n'y a, ici, qu'un seul capteur à effet hall qui fonctionne dans le sens inverse des deux capteurs du premier mécanisme. Il faut ici désactiver le capteur à effet Hall pour valider le mécanisme.



Condition qui valide le mécanisme

```

sd_reading = digitalRead(CEffetHall1_PIN);           //On récupère la valeur du capteur à effet Hall

if (sd_reading == LOW
&& (millis() - ms_time) > DEBOUNCE){           //On vérifie que la statuette est tournée depuis un certain temps (DEBOUNCE)

    C_EffetHall_1 = true;                         //On fixe la valeur de l'attribut capteur

    if (mechanism_status == false) {             //Si il y a eu le premier changement de position

        S_Lion = true;                          //On active l'électroaimant du tiroir
        S_Led = true;                           //On allume la led rouge
        mechanism_status = true;                //On valide le mécanisme
    }
} else{                                         //Si il n'y a pas eu de changement de position positif

    C_EffetHall_1 = false;                      //On fixe la valeur de l'attribut capteur
}

```

image 75 – Condition qui valide le mécanisme

On commence par récupérer la valeur du capteur à effet Hall et on le met dans la variable « `sd_reading` ». Une fois ceci-fait, on ajoute une condition, il faut que le capteur soit désactivé, comme dit plus haut, pour que le mécanisme soit validé. Il faut aussi que la statuette de Lion soit tournée depuis un certain temps, ce qui correspond, à nouveau, au « `DEBOUNCE` » qui est ici de 2 secondes.

Condition lorsque le mécanisme est validé

```

if (S_Lion == true ){                     //Pour désactiver l'electroaimant du tiroir
    S_Lion = LOW; //Ce qui valide la réussite de l'énigme
    digitalWrite(SLion_PIN, S_Lion);      //reset pour defaire le solenoïde
    ms_time = millis();
    S_Led = HIGH; //La led correspondante sur le panneau du superviseur est allumée
}

```

image 76 – Condition selon laquelle le mécanisme est validé

Lorsque le mécanisme est validé, on valide la réussite de l'énigme en modifiant la valeur de « `S_Lion` » à `LOW`. La ligne « `digitalWrite(SLion_PIN, S_Lion);` » permet de défaire le solénoïde et donc de libérer le tiroir pour que le client puisse découvrir l'indice. Une Led est alors, aussitôt, allumé sur le tableau de contrôle du superviseur, c'est la ligne « `S_Led = HIGH;` » qui le permet.

La fonction Main du mécanisme

```

LionB mechanism = LionB();

void setup() {
    mechanism.setupMechanism();          //On donne une configuration de base au mécanisme
}

void loop() {
    delay(100);                        //On attend 0.1 seconde
    mechanism.execute();                //On exécute le mécanisme
}

```

image 77 – Fonction « `Main()` »



On va devoir créer un objet LionB afin d'exécuter les fonctions que l'on a créé ci-dessus. Cette fonction loop va nous permettre d'exécuter, en boucle, notre fonction « `execute()` »

E. Mécanisme N°3 : L'élément Terre

Synopsis du mécanisme

Le mécanisme repose sur un **capteur à effet Hall**.

Selon la mesure, (i) un moteur et une LED sont activés ou désactivés via un relais pendant un laps de temps (ii) une sortie est paramétrée en destination du mécanisme des quatre éléments.

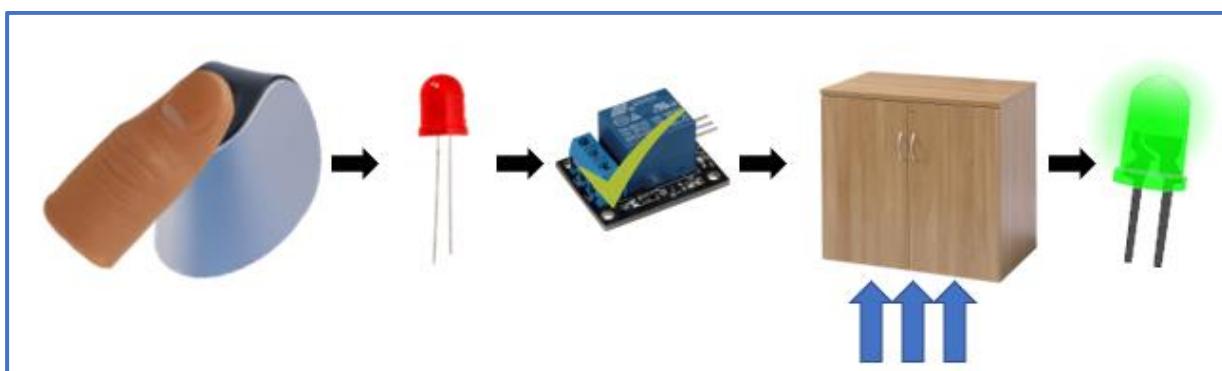


image 78 – Schéma explicatif du mécanisme

Schéma électrique du mécanisme

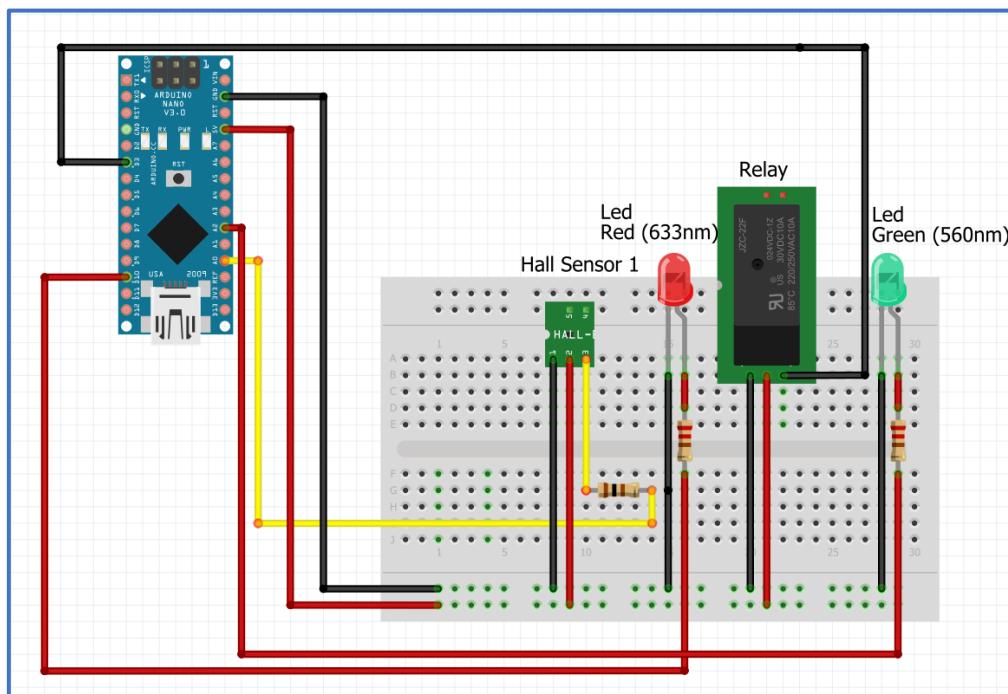


image 79 – Schéma électrique du mécanisme



Programmation du mécanisme

Différents composants du mécanisme

```
#define SLAVE_ADDRESS 0x14
#define C_EffetHall_1_PIN 2 //Capteur à effet Hall 1
#define SLed_PIN A2 //Led de sortie
#define STerre_PIN A0 //Actionneur
#define DEBOUNCE 2000
```

image 80 – Différents composants

C'est ici que l'on va déclarer sur quel PIN de la carte Arduino Nano seront les différents éléments présents sur le schéma électrique ci-dessus. Contrairement au premier mécanisme, on a, encore ici, un seul capteur à effet Hall. Cependant, il y a aussi une Led « `SLed_PIN` » et un actionneur « `STerre_PIN` ».

Fichier Terre.h

Comme pour le mécanisme précédent, on code avec un fichier `Terre.h` et un fichier `Terre.cpp`. Voici le fichier `Terre.h` :

```
class Terre {

private :
    bool C_EffetHall_1;
    bool S_Led;
    bool S_Terre;
    bool mecanism_status;

private :
    bool actuator[2] = {S_Led, S_Terre};
    bool sensor[1] = {C_EffetHall_1};
    const int C_EffetHall_1 = 2;
    bool getMechanism_status();
    void setMechanism_status(bool ms);

public :
    Terre();
    void setupMecanism();
    void execute();
};
```

image 81 – Fichier `Terre.h`

Ceci est le fichier « `Terre.h` » du mécanisme « `Terre` ». On peut y voir les déclarations du capteur à effet Hall « `C_EffetHall_1` », de la Led « `S_Led` » et de l'actionneur « `S_Terre` ». Il y a la fonction « `actuator` » qui prend 2 paramètres « `{S_Led, S_Terre}` » et « `sensor` » qui prend un seul paramètre qui est le capteur. Ces deux fonctions vont être utilisés par l'étudiant n°1 pour les messages du Bus I2C.



Condition qui valide le mécanisme

```

if (sd_reading == LOW
    && (millis() - ms_time) > (DEBOUNCE / 3)) {    //On vérifie s'il y a eu un changement de position positif

    C_EffetHall1 = true;                      //On fixe la valeur de l'attribut capteur

    if (mechanism_status == false) {           //Si il y a eu le premier changement de position

        S_Terre = true;                      //On active l'électroaimant de la ventouse dragon
        S_Led = true;                        //On allume la led rouge
        mechanism_status = true;            //On valide le mécanisme
    }
} else {                                     //Si il n'y a pas eu de changement de position positif

    C_EffetHall1 = false;                    //On fixe la valeur de l'attribut capteur
}

```

image 82 – Condition qui valide le mécanisme

Cette condition est vérifiée lorsque le capteur à effet Hall est activé, il faut que le faux doigt de cinéma, contenant l'aimant, soit positionné sur le lecteur « ADN » qui est en fait un capteur à effet Hall, pendant un certain temps, ici « DEBOUNCE / 3 » qui est donc 2 secondes divisé par 3.

Condition lorsque le mécanisme est validé

```

if (S_Terre == true) {                  //Pour désactiver l'electroaimant de la ventouse
    S_Terre = HIGH;
    digitalWrite(STerre_PIN, HIGH);
    sd_previous = sd_reading;
}

```

image 83 – Condition selon laquelle le mécanisme est validé

Lorsque le mécanisme est validé, on valide la réussite de l'énigme en modifiant la valeur de « `S_Terre` » à `HIGH`. La ligne « `digitalWrite(STerre_PIN, HIGH);` » permet d'activer l'électroaimant qui va faire « monter » un meuble du sol dans lequel on va pouvoir récupérer les derniers éléments de l'Escape Game afin de pouvoir le terminer avec la partie « Quatre Éléments ».



F. Visualiser l'état de la salle

Story Board de l'application Web



image 84 – Story Board de l'application Web

Cette image représente le « Story Board » de l'application Web. C'est-à-dire le premier visuel qui a été imaginé et dessiné avant de commencer la partie programmation de l'application Web.

Aspect final

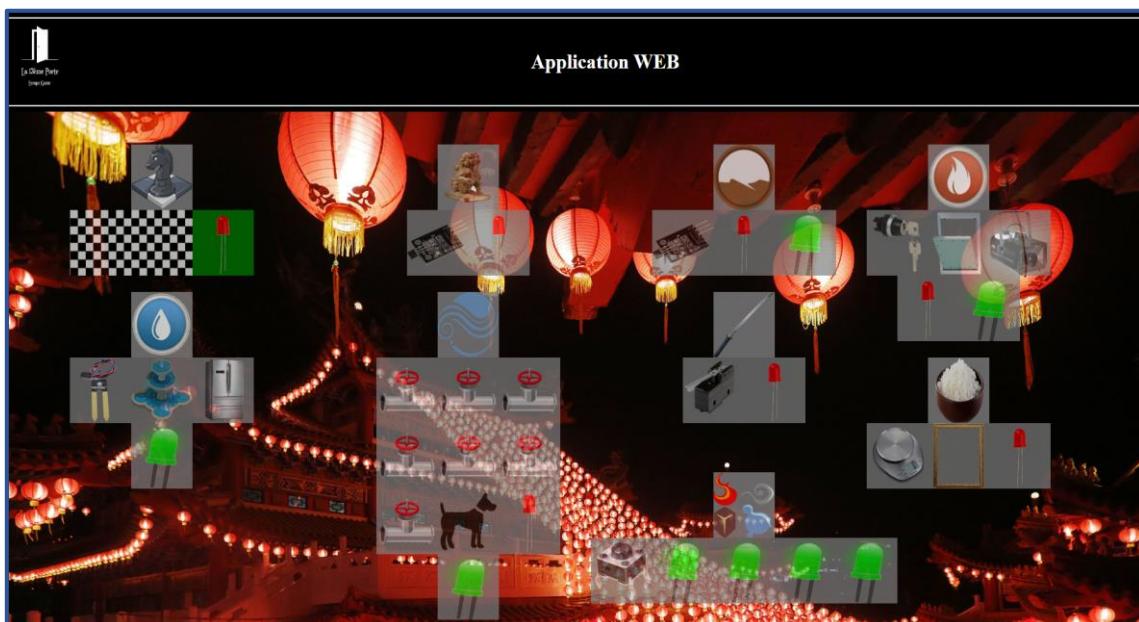


image 85 – Aspect final de l'application Web



Cette image représente l'aspect final de l'application Web. On y voit donc le visuel final de l'application Web après avoir programmé celle-ci. Lorsqu'un capteur est, en booléen, à true, la couleur de la case de ce capteur deviendra verte. Si, au contraire, ce capteur est à false, la couleur de la case deviendra rouge.

Code

Connexion à la base de données

Pour commencer, il nous faut relier notre application WEB à la base de données, pour cela on crée la page connexion.php comme suit :

```
<?php

$dsn = 'mysql:dbname=bdd_escape_game;host=localhost';
$user = 'root';
$password = '';

global $dbh;

try {
    $dbh = new PDO('mysql:host=localhost; dbname=bdd_escape_game; charset=utf8', $user, $password);
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $dbh->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
    echo "Connected successfully";
} catch(PDOException $e)
{
    die('Erreur : ' . $e->getMessage());
}

?>
```

image 86 – page connexion.php de l'application Web

Lorsque la page s'affiche, le mot de passe n'est pas affiché, on ne peut donc pas le voir car ces informations ne sont pas lisibles dans le navigateur. Ceci n'est pas primordial car la seule personne qui verra cette application Web est le superviseur de l'Escape Game.

Les premières lignes nous permettent de renseigner le nom et l'hôte de notre base de données ainsi que l'utilisateur et le mot de passe utilisé.

➔ global \$dbh ;

Cette commande nous permet de pouvoir réutiliser la variable dans laquelle on va mettre les informations récupérées en base de données.

➔ Try
➔ {
➔ } catch
➔ {
➔ }

Grâce au « try/catch », on peut vérifier s'il y a une erreur dans la liaison avec la base de données, ce qui nous afficherait le message d'erreur contenu dans le « die() ».



Code HTML/CSS

Afin de commencer la programmation de l'application WEB de supervision, il faut, dans un premier temps, créer la structure de la page avec la structure traditionnelle HTML retrouvable sur internet :

```
<html lang="fr">
<head>
    <meta charset="utf-8">
    <title>Titre de la page</title>
    <link rel="stylesheet" href="style.css">
    <script src="script.js"></script>
</head>
<body>
    ...
    <!-- Le reste du contenu -->
    ...
</body>
</html>
```

image 87 – Structure HTML de la page Web

Dans un second temps, il faut créer des formes nous permettant d'héberger nos images qui vont ensuite, à l'aide du code PHP, que nous verrons prochainement, nous permettre d'afficher l'état des différents capteurs ou actionneurs. Pour ceci, on procède de la manière suivante :

Cette portion de code va nous permettre d'afficher l'état « 1 » du mécanisme « Échec » par exemple :

```
<div class="carré1">
    
</div>
```

image 88 – Portion de code de la page AppliWeb.php (Partie 1)

Il existe aussi une variante qui va nous permettre d'afficher l'état « 0 » du mécanisme en question :

```
<div class="carré1.1">
    
</div>
<?php
```

image 89 – Portion de code de la page AppliWeb.php (Partie 2)

On utilise la classe « Carré1 » ou « Carré1.1 » juste pour modifier la couleur de la case qui va simuler l'état « 1 » ou « 0 » du mécanisme :

```
.carré1
{
    background-color: green;
    float: left;
    position: absolute;
    width: 100px;
    height: 100px;
    opacity: 0.8;
    margin-top: 50px;
    margin-left: 200px;
}

.carré1.1
{
    background-color: red;
    float: left;
    position: absolute;
    width: 100px;
    height: 100px;
    opacity: 0.8;
    margin-top: 50px;
    margin-left: 200px;
}
```

image 90 – Portion de code du fichier CSS correspondant aux codes ci-dessus

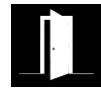


image 91 – Etat d'un mécanisme dans l'application Web pour illustrer l'explication ci-dessus

Code PHP

```
$query1 = ("SELECT * FROM capteurs");
$resultatsCapt = $dbh->query($query1);
$resultatsCapt->setFetchMode (PDO::FETCH_OBJ);
```

image 92 – Partie PHP de la page AppliWeb.php qui montre la récupération de données en BDD pour les actionneurs

Cette portion de code nous permet de récupérer les informations de la table « capteurs » dans notre base de données. La première ligne, à l'aide d'une requête SQL, récupère les informations de la table « capteurs ». Ensuite, il faut mettre toutes ces informations dans une variable, à l'aide de la fonction « query() » qui nous permet de récupérer un jeu de résultat provenant d'une requête SQL et de les récupérer en tant qu'objet PDOStatement.

```
while ($capteurs = $resultatsCapt->fetch())
{
    $capteur = utf8_encode($capteurs->ID_capteurs);
    $etat = utf8_encode($capteurs->Etat);
    $dateMesure = utf8_encode($capteurs->Heure_derniere_mesure);
```

image 93 – Partie PHP de la page AppliWeb.php qui montre la récupération de données en BDD pour les capteurs

Pour utiliser les données récupérées précédemment, il faut faire une boucle et utiliser la fonction « fetch() ». Cette fonction récupère une ligne suivante, d'où l'utilisation d'une boucle pour récupérer toutes les lignes de la base de données.

Une fois toutes les données de la table « capteurs » récupérées, on doit utiliser des variables pour exploiter ces informations. On utilise « \$capteur » pour les données de la colonne « ID_capteurs », « \$etat » pour les données de la colonne « Etat » puis enfin « \$dateMesure » pour la colonne « Heure_derniere_mesure ».



```

if($capteur==1)
{
    if($etat==TRUE)
    {
        ?>
        <div class="carréMecal">
            
        </div>
        <?php
    }
    elseif($etat==FALSE)
    {
        ?>
        <div class="carréMecal.1">
            
        </div>
        <?php
    }
}

```

image 94 – Condition pour récupérer la valeur d'un capteur en fonction de son état dans la BDD

Pour terminer, on doit faire plusieurs boucles comme celle-ci, où on récupère le numéro du capteur, s'il correspond au premier capteur, on entre dans la boucle et on observe l'état du capteur dans la base de données, s'il est à 1, on fait apparaître le carré vert, sinon ce sera le rouge comme suit :



image 95 – Exemple de l'affichage du premier mécanisme dans l'application Web

Si l'état du capteur est à 1, c'est sous cette forme que l'image va apparaître, si l'état est à 0, ce sera alors la même image mais sur un fond rouge. Cela nous permet aisément de différencier l'état des différents capteurs.



G. Partie réseau

Matériel

Ordinateur

Un ordinateur pouvant se connecter à la box internet de la société **13^{ème} Porte**.

Raspberry

Une carte Raspberry qui va servir d'intermédiaire entre les différentes cartes Arduino Nano et le serveur de base de données.

Box internet

Une box internet qui est déjà présente sur le site puisqu'il s'agit du boîtier internet de la Société **13^{ème} Porte**. Celle-ci va nous servir de routeur afin de faire communiquer les différents composants de ce réseau.

Schéma réseau

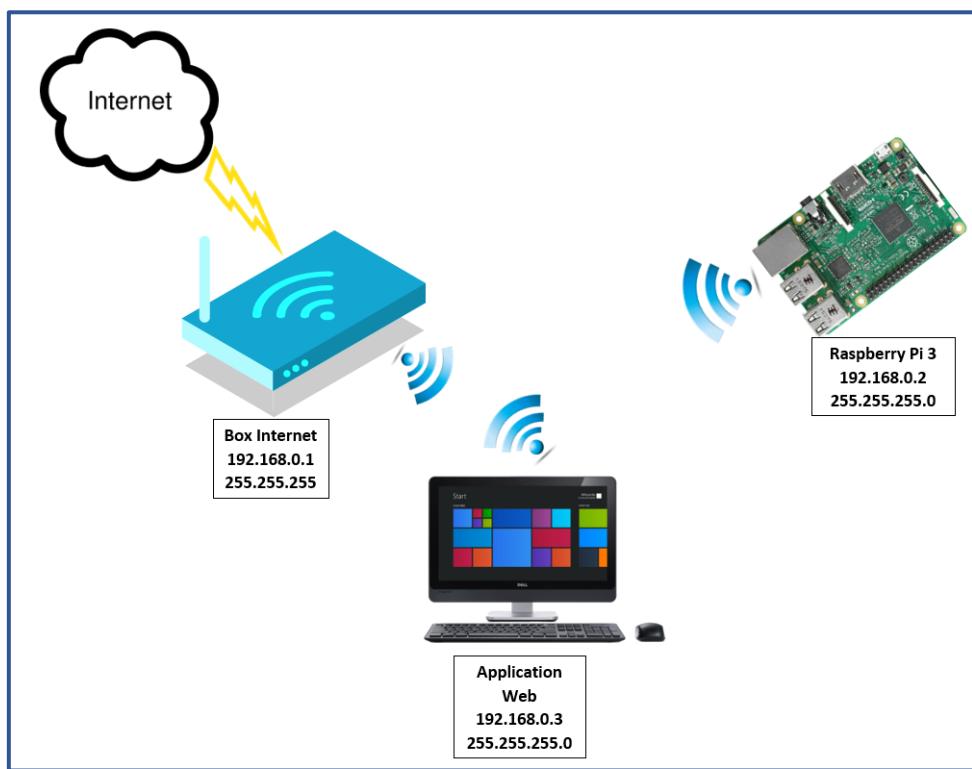


image 96 – Schéma réseau du projet

Tutoriel de connexion

Un tutoriel a été fourni à la société **13^{ème} Porte** afin de permettre à cette entreprise de pouvoir déployer ce réseau en réservant, par exemple, une ou plusieurs adresses IP dans grâce au protocole DHCP.



H. Fiches recettes

Cette partie va permettre de vérifier le bon fonctionnement des différentes tâches de l'étudiant n°3 par le biais des 4 fiches recettes ci-dessous.

Mécanisme n°1 : L'Échiquier

NOM : VÉRIFIER LE FONCTIONNEMENT DU MÉCANISME N°1	
Recette : TECHNIQUE	
Objectif	Vérifier que le mécanisme s'exécute correctement
Elément à tester	Echec.ino
Pré requis	Ouvrir le fichier dans le logiciel Arduino et effectuer le câblage

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		L'affichage d'un message : « téléversement terminé »	<input type="checkbox"/>
2	Approcher un aimant du capteur à effet Hall		La Led s'allume, le mécanisme fonctionne donc correctement	<input type="checkbox"/>

Rapport de test	Testé par l'étudiant n°3	Le 19/05/2020
Conformité		
<input type="checkbox"/> Excellente		
<input type="checkbox"/> Moyenne		
<input type="checkbox"/> Faible		
Commentaire :		Approbation :


Mécanisme n°2 : Le Lion Basculant

NOM : VÉRIFIER LE FONCTIONNEMENT DU MÉCANISME N°2	
Recette : TECHNIQUE	
Objectif	Vérifier que le mécanisme s'exécute correctement
Elément à tester	Lion.ino
Pré requis	Ouvrir le fichier dans le logiciel Arduino et effectuer le câblage

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		L'affichage d'un message : « téléversement terminé »	<input type="checkbox"/>
2	Approcher un aimant du capteur à effet Hall		La Led s'allume, le mécanisme fonctionne donc correctement	<input type="checkbox"/>

Rapport de test	Testé par l'étudiant n°3	Le 19/05/2020
Conformité		
<input type="checkbox"/> Excellente		
<input type="checkbox"/> Moyenne		
<input type="checkbox"/> Faible		
Commentaire :		Approbation :



Mécanisme n°3 : L'Élément Terre

NOM : Vérifier le fonctionnement du mécanisme n°3	
Recette : TECHNIQUE	
Objectif	Vérifier que le mécanisme s'exécute correctement
Elément à tester	Terre.ino
Pré requis	Ouvrir le fichier dans le logiciel Arduino et effectuer le câblage

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		L'affichage d'un message : « téléversement terminé »	<input type="checkbox"/>
2	Approcher un aimant du capteur à effet Hall		La Led s'allume, le mécanisme fonctionne donc correctement	<input type="checkbox"/>

Rapport de test	Testé par l'étudiant n°3	Le 19/05/2020
Conformité		
<input type="checkbox"/> Excellente		
<input type="checkbox"/> Moyenne		
<input type="checkbox"/> Faible		
Commentaire :		Approbation :



Visualiser l'état de la salle

NOM : VÉRIFIER LE FONCTIONNEMENT DE L'APPLICATION WEB	
Recette : TECHNIQUE	
Objectif	Vérifier que l'application Web fonctionne correctement
Elément à tester	AppliWeb.php
Pré requis	Lancer WampServer

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Lancer Localhost		Le navigateur par défaut va se lancer et ouvrir la page Localhost	<input type="checkbox"/>
2	Cliquer sur le lien Application Web		L'application Web se lance et affiche l'état des différents composants du projet	<input type="checkbox"/>

Rapport de test	Testé par l'étudiant n°3	Le 19/05/2020
Conformité <input type="checkbox"/> Excellente <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible		
Commentaire :		Approbation :



I. Conclusion

Communication de groupe

Lors du déroulement du projet et comme chacun le sait, le confinement à modifié la manière de travailler de chacun. Nous avons dû nous adapter et pour cela et nous organiser, nous avons utilisé plusieurs logiciels tels que :

- Discord : Logiciel gratuit basé sur un principe de serveurs qui nous a permis de communiquer les uns avec les autres lorsque nous avions besoin de mettre en commun par exemple.
- Github : Service web d'hébergement et de gestion de développement de logiciel. C'est avec ce logiciel que nous avons pu échanger des fichiers en temps réel sans aucun soucis d'envoi, juste un « Push » nous permet d'envoyer les fichiers souhaités aux autres membres du groupe qui auront à faire un « Pull » pour les récupérer.
- Un journal d'activité sur Excel sur lequel on renseigne nos tâches effectuées chaque jour.

Cette avec cette organisation que nous avons pu venir à bout de notre projet et fournir à la société 13^{ème} Porte, un travail de qualité et un service fonctionnel qui va leur permettre d'améliorer leur Escape Game.

Regard critique du projet

Lorsque j'avais la possibilité de choisir le projet sur lequel je voulais travailler, je me suis orienté vers celui-ci et d'autres qui possédaient une partie application Web car c'est la spécialité que je veux faire en école d'ingénieurs et dans ce sens, je peux avoir plus de facilité à trouver une entreprise pour effectuer une alternance si j'ai déjà des exemples de code que je peux leur fournir.

Connaissances apportées

Ce projet m'a apporté des connaissances qui me seront sûrement très utile dans mon futur métier.

J'ai notamment développé des compétences au niveau Hardware :

- Réaliser un câblage complexe avec plusieurs capteurs à effet Hall et une carte Arduino Nano
- Réfléchir à un schéma réseau permettant la connexion de chacun des composants du projet
- Réfléchir à un tutoriel pour la configuration réseau des différents composants pour éviter des potentielles erreurs

J'ai aussi développé des compétences au niveau Software :

- Réaliser une application Web avec du HTML/CSS, au niveau front end mais aussi au niveau back end avec du PHP pour récupérer des données en BDD
- Programmer une carte Arduino à l'aide de l'IDE Arduino

Poursuite d'étude

Ce projet me sera très utile pour ma poursuite d'étude. Dans la continuité de celui-ci, j'ai été accepté dans l'école d'ingénieurs et du numérique ESIEA sur le campus de Laval pour un cycle d'ingénieurs en trois ans.



La spécialité dans laquelle je souhaite m'orienter, qui porte le nom de majeur en école d'ingénieurs, est l'architecture logicielle axée sur le Web.

Remerciement

Premièrement, je souhaiterais remercier l'entreprise 13^{ème} Porte avec laquelle nous étions en collaboration pour ce projet. Les différents échanges que nous avons pu avoir étaient fluides et professionnels. Nous avons pu découvrir l'entreprise et l'Escape Game sur lequel nous avons travaillé par le biais d'une visite des locaux et d'une rencontre avec le gérant de la société Mr Jean-Charles DOUGUET que je voudrais, par la présente, remercier également.

Deuxièmement, je voudrais remercier l'école Saint Félix La-Salle qui a pu nous proposer ce projet. Je remercie aussi Mr ANGIBAUD avec qui nous avons pu échanger tout au long du projet et encore plus lors du confinement et qui a pu nous conseiller et nous aider sur les différentes tâches que nous avions à exécuter.

Enfin, je souhaiterais remercier les membres de mon groupe qui sont Thomas CADEAU, Constantin MINOS et Corentin BREGNY avec lesquels j'ai pu travailler et échanger tout au long du projet.



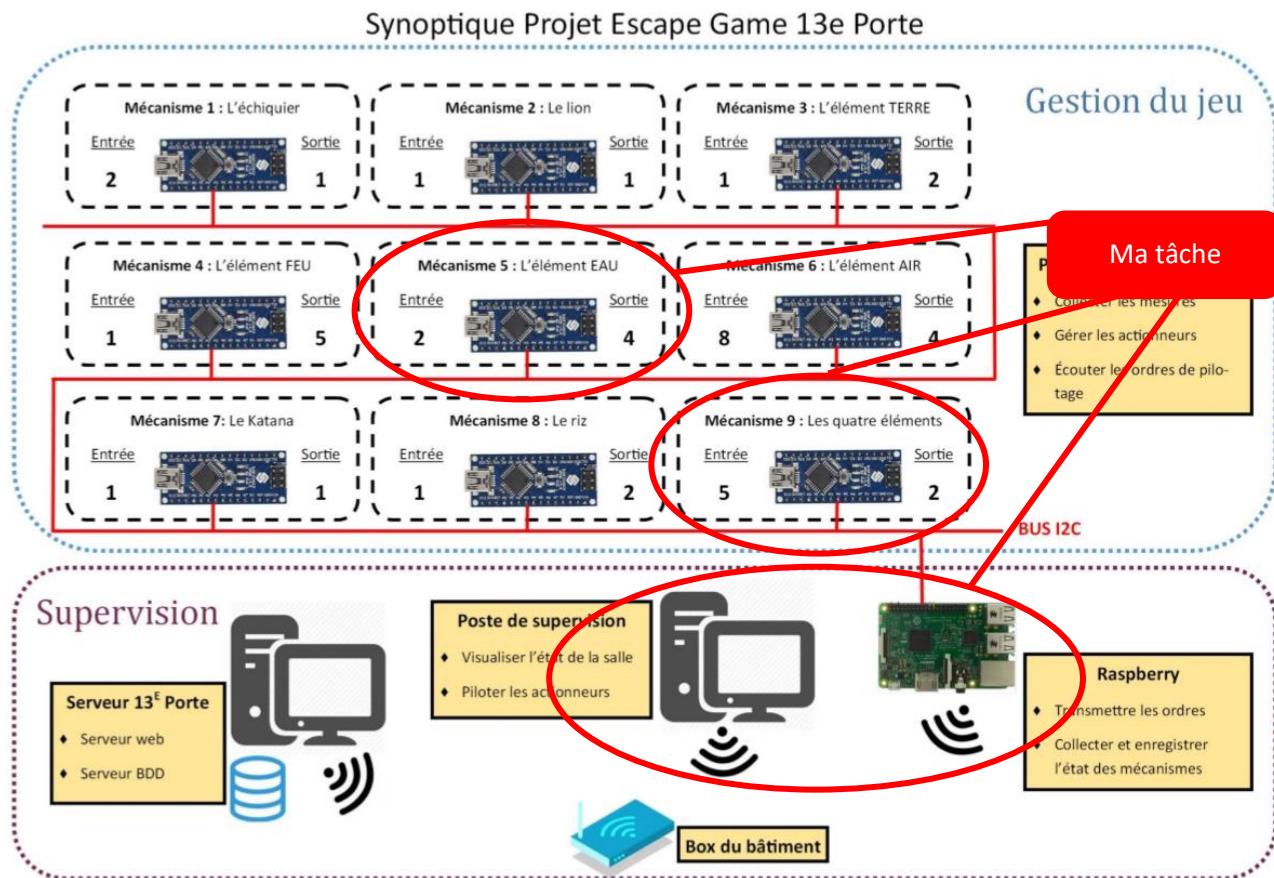
V. Etudiant 4 (Thomas CADEAU)

A. Rappel du cahier des charges

Aperçu des tâches réalisées

Dans le système Escape Game, ma partie de développement consiste à créer plusieurs programmes :

- Deux programmes pour deux mécanismes sur Arduino :
 - Le mécanisme 5 : l'Eau, avec un capteur d'humidité (Langage Arduino ~C++)
 - Le Mécanisme 9 : les 4 Éléments, avec un bouton poussoir (Langage Arduino ~C++)
- Une interface WEB pour la gestion des actionneurs (HTML/CSS/PHP):
 - Envoyé un ordre d'activation ou de désactivation d'un actionneur à la Raspberry.
 - Création d'un serveur sockets pour l'envoie des ces ordres à la Raspberry. (Langage Python)



*Contraintes liées au développement*

A cause du confinement le programme concernant le mécanisme 9 et la programmation par socket n'as pas pu être testé par manque de matériels.



B. Réalisation du programme Mécanisme 5 : l'Eau

Schéma câblage

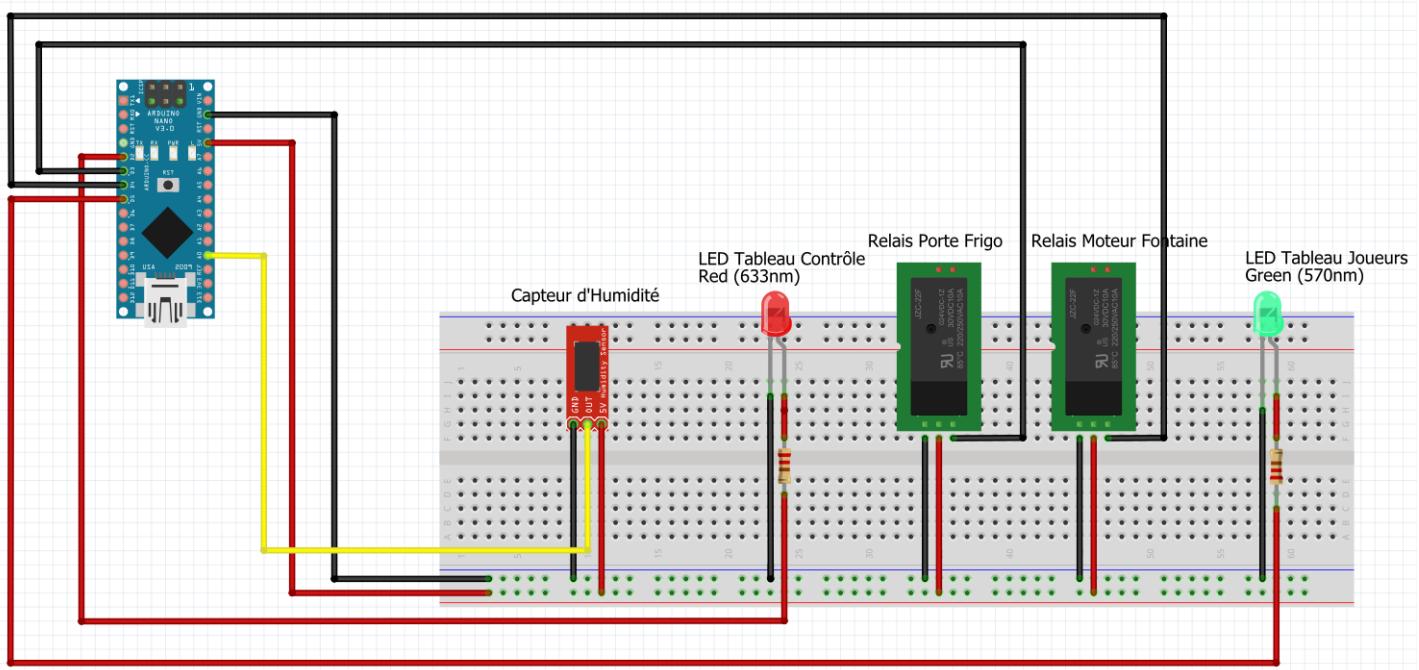
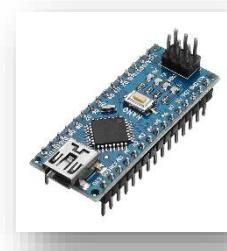


Image 98 : Schéma câblage du mécanisme 5.

C. Présentation du matériel et du mécanisme

Matériels utilisés

- Capteur d'humidité (Water Sensor)
- Un Arduino Nano
- Gâche électrique (Solenoid 12 V)
- 2 Relais (5 V à 220 V)
- Moteur d'une fontaine (220 V)
- 2 LEDS et 2 Résistances (220 Ω)





Description du Mécanisme

Durant l'étape « Elément EAU » de l'escape game, les joueurs doivent verser de l'eau dans une tasse troué. Cette eau s'écoule sur un capteur d'eau (Water Sensor).

Quand le joueur réussit cette énigme, le mécanisme répond ceci :

- Une LED témoin s'allume au tableau de contrôle.
- Une fontaine (220 volts) se met en marche via un relais (5 volts).
- Une gâche électrique (Solenoid 12 volts) se met en marche afin d'ouvrir la porte d'un frigo. (Un indice permettra aux joueurs d'entendre une gâche électrique se mettre en marche et ainsi réaliser que le frigo est désormais ouvert)
- L'élément EAU (LED) est allumé sur la tablette à destination des joueurs.

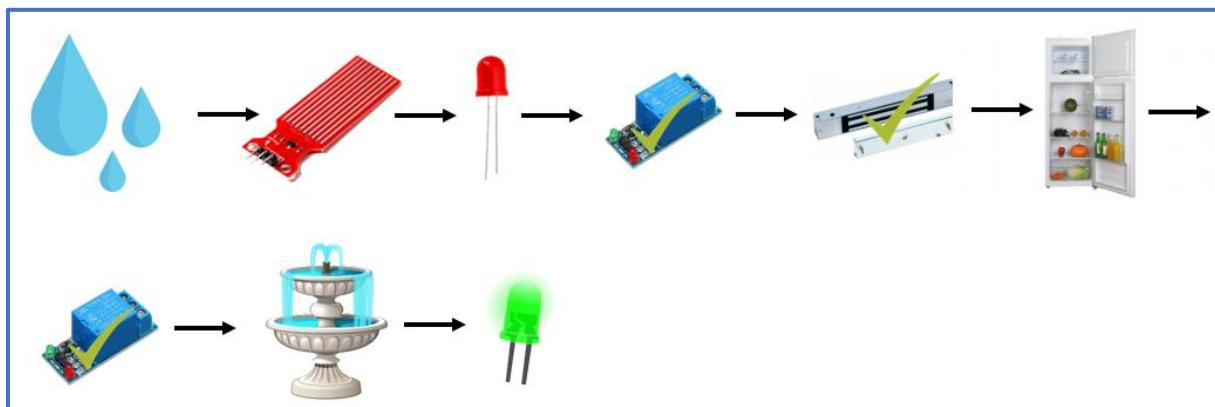


Image 99 : Schéma fonctionnement mécanisme 5.

Capteur d'humidité (Water Sensor)

Le capteur d'eau est un capteur assez sensible qui renvoi une valeur numérique à l'Arduino. Quand le système est mis en route, ce capteur renvoi une première valeur à l'Arduino.

Au niveau du code, cette valeur est stockée dans une variable. Enfin les Sortie S_Frig, S_Fontaine, S_Eau s'activent lorsque le capteur renvoie une valeur supérieure de 180 points de base.

Nous procédons de la sorte car la valeur renvoyée par le capteur dépend du taux d'humidité résiduel encore présent sur le capteur. Celui-ci varie constamment. Ce capteur est le seul élément à l'entrée de l'Arduino (C_Humidite).

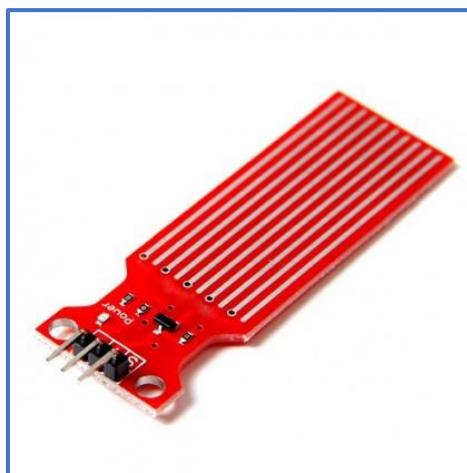


Image 100 : Capteur d'humidité.



D. Programme

Initialisation

On définit des variables pour les pins utilisées de l'Arduino :

```
#define CHumidite_PIN A0 //Capteur d'humidité au pin A0
#define SLed_PIN 2        //Led de contrôle au pin 2
#define SFrigo_PIN 3      //Relais Frigo au pin 3
#define SFontaine_PIN 4    //Relais Fontaine au pin 4
```

Image 101 : Code en tête de Eau.ino

On initialise le matériel utilisé pour le mécanisme :

```
pinMode(CHumidite_PIN, INPUT); //On initialise le pin du capteur d'Humidité en entrée
pinMode(SLed_PIN, OUTPUT); //On initialise la led du tableau de commande en sortie
digitalWrite(SLed_PIN, LOW); //On éteint la led par défaut

pinMode(SFrigo_PIN, OUTPUT); //On initialise le pin du relais frigo en sortie
digitalWrite(SFrigo_PIN, HIGH); //On désactive le relais du frigo par défaut

pinMode(SFontaine_PIN, OUTPUT); //On initialise le pin du relais de la fontaine en sortie
digitalWrite(SFontaine_PIN, HIGH); //On désactive le relais de la fontaine par défaut
```

Image 102 : Code fonction setupMechanism() dans Eau.ino

Le main

Le programme exécute principalement les instructions suivantes :

```
Eau Mecanisme = Eau(); //On instancie un objet de type Eau

void setup() {
    mechanism.setupMechanism(); //On donne une configuration de base au mécanisme
}

void loop() {
    delay(100); //On attend 0.1 seconde
    mechanism.execute(); //On exécute le mécanisme
}
```

Image 103 : Code main dans Eau.ino



La classe Eau

Principaux attributs

Les principaux attributs de la classe sont les suivants :

```
bool S_Frigo;           //Actionneurs qui active/désactive l'électro-aimant de la porte du frigo
bool S_Fontaine;        //Actionneurs qui active/désactive le moteur de la fontaine
bool S_Led;             //Actionneurs qui active/désactive la led de contrôle
bool S_Eau;              //Actionneurs de l'élément Eau sur la tablette des 4 éléments
int C_Humidite;         //Valeur numérique détecter par le capteur d'humidité
bool mecanism_status;   //indique si le mécanisme est activé ou non
```

Image 104 : Code classe Eau dans Eau.ino

Principales méthodes

Les principales méthodes sont les suivantes :

```
public :
Eau();                  //Constructeur par défaut de la classe
void setupMecanism();   //Configuration de la base du mécanisme
void execute();          //Méthode qui fait fonctionner le mécanisme
```

Image 105 : Code classe Eau dans Eau.ino

La méthode execute

Synopsis

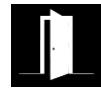
La méthode « **execute** » est la méthode qui fait fonctionner le mécanisme. Elle est appelée dans la fonction loop du programme et donc exécutée en boucle.

Récupérer l'état de l'interrupteur à clef

Pour récupérer l'état de l'interrupteur à clef on utilise l'instruction suivante :

```
ValeurCapteur = analogRead (CHumidite_PIN); //On récupère la valeur du capteur d'humidité
```

Image 106 : Code fonction execute() dans Eau.ino



Valider le mécanisme

Pour valider le mécanisme on exécute les instructions suivantes :

```
if (C_Humidite >=180){          //Si la valeur du capteur est supérieur ou égale à 180 points de base

    S_Fontaine = true;           //On active le moteur de la fontaine
    S_Frigo = true;              //On active l'électro-aimant de la porte du frigo
    S_Led = true;                //On allume la led du panneau de contrôle
    S_Eau = true;                //On allume l'élément Eau des 4 éléments
    mechanism_status = true;     //On valide le mécanisme

}
```

Image 107 : Code fonction execute() dans Eau.ino



E. Réalisation du programme Mécanisme 9 : les 4 Éléments

Schéma câblage

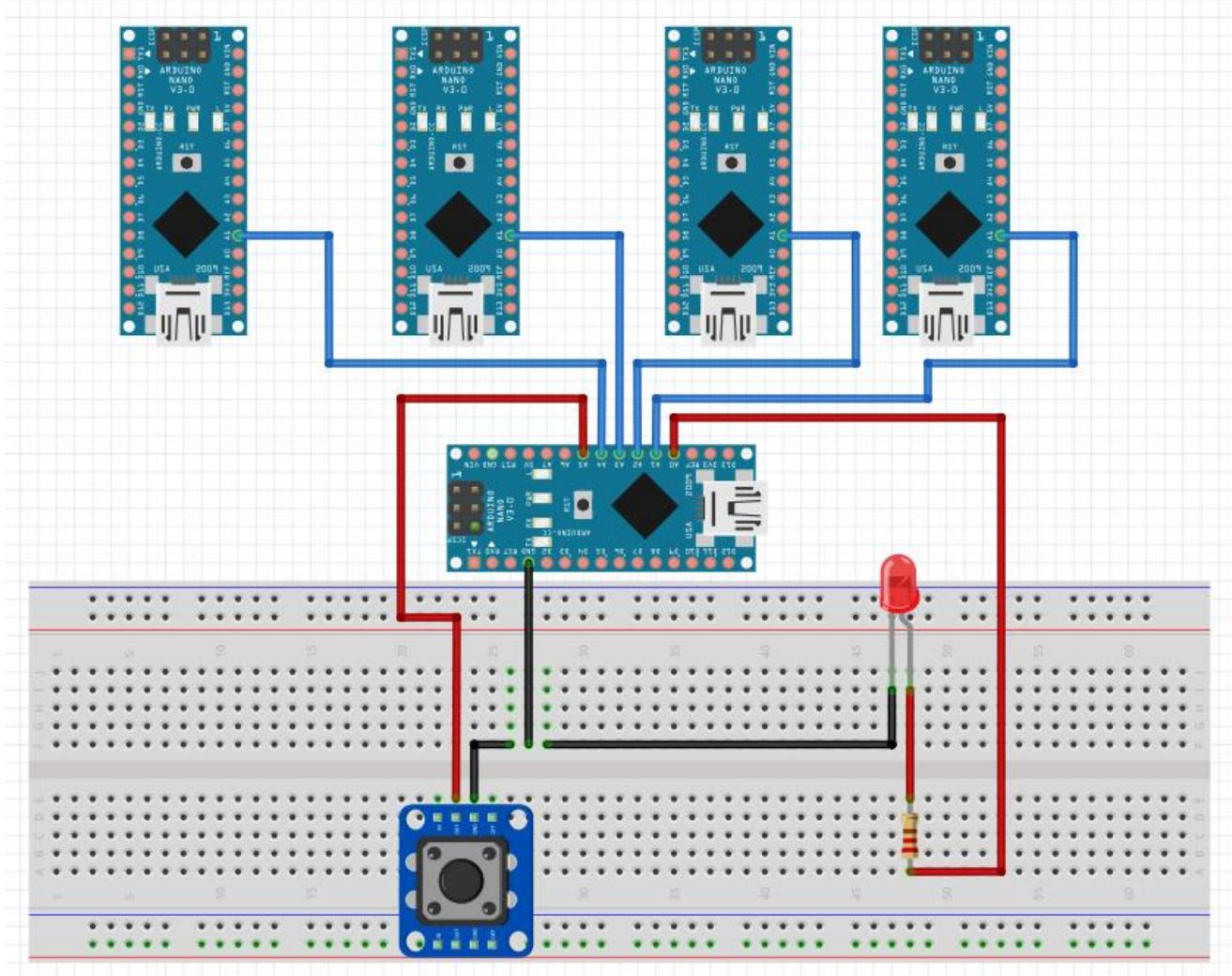


Image 108 : Schéma câblage du mécanisme 9.



F. Présentation du matériels et du mécanisme

Matériels utilisés :

- Bouton poussoir (5 V)
- Le 4 Arduino Nano correspondant aux 4 éléments (Terre, Feu, Eau, Air).
- Electroaimant (24V)
- 5 LEDS et 5 Résistances (220 Ω)
- 1 Relais (5 V à 220 V)



Description du Mécanisme :

Les joueurs appuient sur le bouton poussoir situé sous « La tablette des 4 éléments ». Si les 4 éléments n'ont pas été validés alors une LED rouge à proximité immédiate s'allume quelques secondes. Si les 4 éléments ont préalablement été validé alors l'électroaimant de la porte de sortie est désactivé, ouvrant ainsi la porte Final.

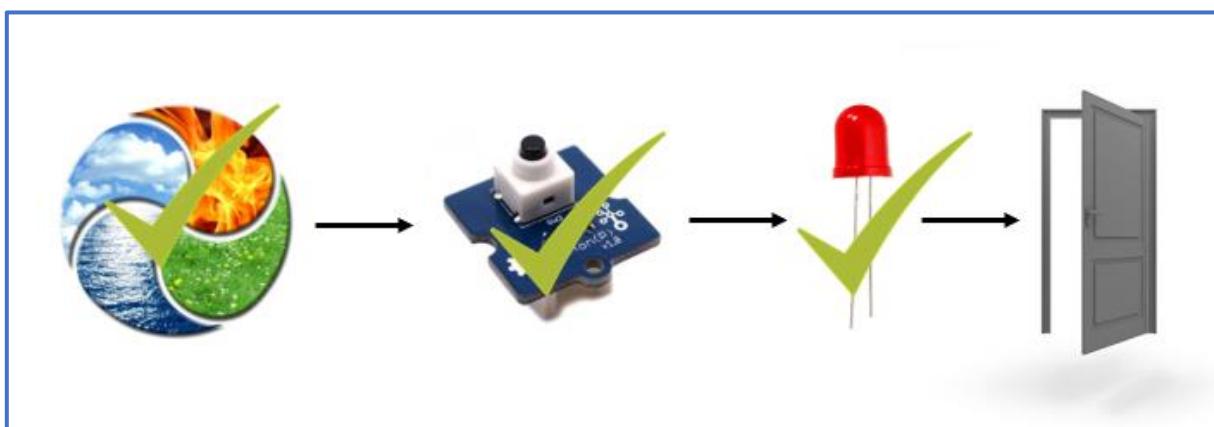


Image 109 : Schéma fonctionnement mécanisme 9.



G. Programme

La réalisation et les tests pour la réalisation de ce programme ont été très compliqué à cause de cette période de confinement. La programmation n'a pas pu se faire dans son intégralité. Je m'en excuse d'avance.

Initialisation

On définit des variables pour les pins utilisés de l'Arduino :

```
#define CPoussoir_PIN 1      //Bouton poussoir au pin 1
#define SLed_PIN 2           //Led rouge des 4 éléments au pin 2
#define SPorteFinal_PIN 3    //Relais de l'électro-aimant de la porte final au pin 3
#define ETerre_PIN 4          //Liaison de l'arduino Terre au pin 4
#define EFeu_PIN 5            //Liaison de l'arduino Feu au pin 5
#define EEau_PIN 6             //Liaison de l'arduino Eau au pin 6
#define EAir_PIN 7             //Liaison de l'arduino Air au pin 7
```

Image 110 : Code en tête de 4Elements.ino.

On initialise le matériel utilisé pour le mécanisme :

```
pinMode(CPoussoir_PIN, INPUT);           //On initialise le pin du bouton poussoir en entrée

pinMode(SLed_PIN, OUTPUT);               //On initialise le pin de la led des 4 éléments en sortie
digitalWrite(SLed_PIN, LOW);             //On éteint la led par défaut

pinMode(SPorteFinal_PIN, OUTPUT);         //On initialise le pin du relais de la porte final en sortie
digitalWrite(SPorteFinal_PIN, HIGH);       //On désactive le relais de la porte final par défaut

pinMode(ETerre_PIN, INPUT);               //On initialise le pin de la liaison de l'arduino Terre en entrée

pinMode(EFeu_PIN, INPUT);                 //On initialise le pin de la liaison de l'arduino Feu en entrée

pinMode(EEau_PIN, INPUT);                 //On initialise le pin de la liaison de l'arduino Eau en entrée

pinMode(EAir_PIN, INPUT);                 //On initialise le pin de la liaison de l'arduino Air en entrée
```

Image 111 : Code fonction setupMechanism() dans 4Elements.ino.



Le main

Le programme exécute principalement les instructions suivantes :

```
4Elements Mecanisme = 4Elements();           //On instancie un objet de type 4Elements

void setup() {
    Mecanisme.setupMecanism();                //On donne un configuration de base au mécanisme
}

void loop() {
    delay(100);                             //On attend 0,1 sec
    Mecanisme.execute();                     //On exécute le mécanisme
}
```

Image 112 : Code main dans 4Elements.ino.

La classe 4Elements

Principaux attributs

Les principaux attributs de la classe sont les suivants :

```
private :
    bool E_Terre;                      //Variable qui informe si ETerre_PIN = TRUE ou FALSE
    bool E_Feu;                        //Variable qui informe si EFeu_PIN = TRUE ou FALSE
    bool E_Eau;                        //Variable qui informe si EEau_PIN = TRUE ou FALSE
    bool E_Air;                        //Variable qui informe si EAir_PIN = TRUE ou FALSE
    bool S_Led;                        //Actionneur qui allume ou éteint la led des 4 éléments
    bool C_Poussoir;                  //Etat de la position détecter par le bouton poussoir
    bool mecanism_status;             //Indique si le mécanisme est activé ou non
```

Image 113 : Code classe 4Elements dans 4Elements.ino.

Principales méthodes

Les principales méthodes sont les suivantes :

```
public :
    4Elements();                      //Constructeur par défaut de la classe
    void setupMecanism();            //Configuration de base du mécanisme
    void execute();                  //Méthode qui fais fonctionner le mécanisme
};
```

Image 114 : Code classe 4Elements dans 4Elements.ino.



H. Développement sur la Raspberry

Pour cette partie, le PC de supervision doit pouvoir gérer et visualiser à distance l'état de chacun des mécanismes via une application WEB. De plus il doit pouvoir récupérer les informations transmis par les mécanismes depuis la Raspberry. J'ai eu beaucoup de contraintes dus au confinement, entre autres le manque de matériel. Cette partie n'est pas terminé.

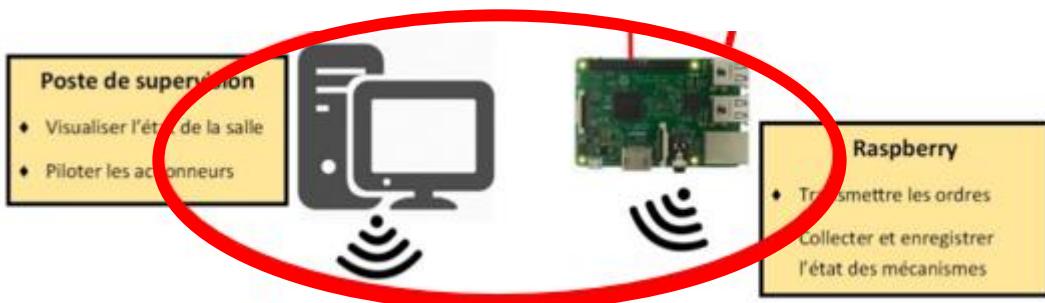


Image 115 : Schéma de la transmission socket, PC Supervision -> Raspberry

La Raspberry relié par liaison I2C avec tous les mécanismes, (les 9 Arduino Nano) reçoit toutes les informations des mécanismes qui ensuite les envoient par sockets au PC de supervision. L'installation et la configuration de la Raspberry sera faite en commun avec l'étudiant 1.



Image 116 : Logos Raspberry et Python

- **Langage de développement :** Programmation par sockets en Python.
- **Logiciel utilisé :** Putty (Emulateur de terminal).



I. Raspberry (Définition)

Le Raspberry pi est un nano ordinateur de la taille d'une carte de crédit que l'on peut brancher à un écran et utilisé comme un ordinateur standard. Sa petite taille, et son prix intéressant fait du Raspberry pi un produit idéal pour tester différentes choses, et notamment la création d'un serveur Web chez soi.



Image 117 : Une Raspberry

J. Programmation par socket

Typiquement, un socket respecte un flux spécifique d'événements pour qu'elle fonctionne. Pour un modèle client-serveur orienté connexion, le socket du processus serveur attend la demande d'un client. Pour ce faire, le serveur doit d'abord établir une adresse que les clients peuvent utiliser pour trouver et se connecter au serveur. Lorsqu'une connexion est établie avec succès, le serveur attend que les clients demandent un service. L'échange de données client-serveur aura lieu si le client se connecte au serveur via le socket. Le serveur répondra alors à la demande du client et lui enverra une réponse.

K. Putty (Définition)

PuTTY est un émulateur de terminal doublé d'un client pour les protocoles SSH, Telnet, rlogin, et TCP brut. Il permet également des connexions directes par liaison série RS-232.

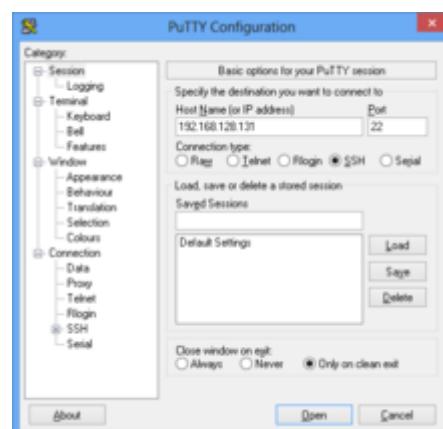
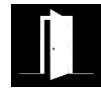


Image 118 : Putty



L. Développement de l'application WEB de supervision

Une application WEB doit être créée pour le poste de supervision de l'administrateur. Une interface pour pouvoir piloter les différents actionneurs doit être réalisé. Le superviseur pourra démarrer ou arrêter chaque actionneur. L'ordre sera tout d'abord transmis par liaison WIFI à la Raspberry, qui transmettra par liaison I2C cet ordre au mécanisme visés (Arduino nano correspondant au mécanisme). Le pilotage à distance des actionneurs devra inhiber la décision décrite dans la section Gérer les neuf mécanismes.



Image 119 : Logo CSS/HTML/PHP

- **Langage de développement :** Programmation PHP/HTML/CSS
- **Logiciel utilisé :** NetBeans + plugin PHP

M. Programmation PHP/HTML/CSS

HTML

L'HTML est un langage informatique utilisé sur l'internet. Ce langage est utilisé pour créer des pages web. L'acronyme signifie « HyperText Markup Language », ce qui signifie en français "Langage de balisage d'hypertexte".

Ce n'est pas à proprement parlé un langage de programmation, mais plutôt un langage qui permet de mettre en forme du contenu. Les balises permettent de mettre en forme le texte et de placer des éléments interactifs, tel des liens, des images ou bien encore des animations. Ces éléments ne sont pas dans le code source d'une page codée en HTML mais "à côté" et la page en HTML ne fait que reprendre ces éléments. Pour visualiser une page en HTML il est nécessaire d'utiliser un navigateur web.



CSS

CSS est l'acronyme de « Cascading Style Sheets » ce qui signifie « feuille de style en cascade ». Le CSS correspond à un langage informatique permettant de mettre en forme des pages web (HTML ou XML).

PHP

PHP est connu comme langage de script utilisé côté serveur. Il est utilisé dans le développement web ainsi que comme langage de programmation général.

Les fichiers PHP peuvent contenir du code utilisé pour exécuter différents processus en ligne. Le moteur PHP sur le serveur web analyse le code PHP contenu dans le fichier et génère dynamiquement le code HTML. C'est ce code HTML et non pas le code PHP sous-jacent qui est visible par l'internaute qui visite une page web.

N. Création de PopUp

Pour la création de ma page WEB, la mise en place de PopUps me paraissait être la meilleure chose. En effet, quand le directeur voudra changer l'état d'un actionneur, il aura juste à cliquer sur le mécanisme en question et une mini page s'ouvrira de la même façon qu'un PopUp. Ensuite, il pourra enfin gérer les actionneurs comme il le voudra.



O. Fiches recettes

Dans cette partie, les quatre fiches recettes sont destinées au client. Elles permettent de valider le fonctionnement de ma tâche dans le système.

Observer le bon fonctionnement du mécanisme 5 : l'Eau

Nom :	Lancer le système de régulation
Recette :	Technique
Objectif	Lancer le mécanisme afin qu'il s'exécute correctement
Elément à tester	Eau.ino
Pré requis	Ouvrir Eau.ino dans l'IDE Arduino et réaliser les câblages

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		Un message indique « téléchargement terminé »	<input type="checkbox"/>
2	Tremper le capteur d'humidité dans l'eau.	Le capteur capte bien une température.	Le mécanisme s'exécute correctement.	<input type="checkbox"/>

Rapport de test	Testé par : L'Étudiant 4	
Conformité		
<input type="checkbox"/> Excellente <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible		
Commentaire :		Approbation :



Observer le bon fonctionnement du mécanisme 9 : Les 4 Éléments

Nom : Lancer le système de régulation	
Recette : Technique	
Objectif	Lancer le mécanisme afin qu'il s'exécute correctement
Elément à tester	4Elements.ino
Pré requis	Ouvrir 4Elements.ino dans l'IDE Arduino et réaliser les câblages

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Téléverser le programme dans une Arduino nano		Un message indique « téléchargement terminé »	<input type="checkbox"/>
2	Recevoir un message True de la part des 4 Éléments.		Le mécanisme s'exécute correctement.	<input type="checkbox"/>

Rapport de test	Testé par : L'Étudiant 4	
Conformité		
<input type="checkbox"/> Excellente	<input type="checkbox"/> Moyenne	<input type="checkbox"/> Faible

Commentaire :

Approbation :



P. Conclusion

Communication du groupe

Durant toute la durée de notre projet, la cohésion de groupe était un atout majeur pour nous tous, car nous étions un groupe de 5, et pourtant nous avons bien réussi à nous ordonner :

- La plateforme GitHub, était d'une grande aide pour le partage de documents et de programmes.
- Le diagramme de Gantt était régulièrement mis à jour, cela était très enrichissant pour l'organisation de la réalisation de nos tâches.
- Beaucoup de réunions entre les membres du groupe par moyen de communication divers (Discord, Teams).
- Une charte graphique commune pour les différents documents.

Toutes ces mesures nous ont aidé à réaliser ce projet sérieusement et en équipe.

D'autant plus qu'ayant dû effectuer notre projet en confinement nous avons pu faire l'expérience du travail de groupe à distance, une expérience enrichissante.

Regard critique du projet

Ce projet m'a beaucoup intéressé au premier regard, car le principe de créer un escape game entièrement informatisé, me paraissait amusant et instructif. La poursuite de recherche sur le serveur socket m'a beaucoup intrigué. La création de page WEB est ce que j'ai apprécié le plus dans toute la réalisation de mon projet. J'ai donc un avis positif sur la réussite de notre projet.

Connaissances apportées

Ce projet m'a apporté bien des connaissances.

Au niveau Hardware j'ai appris à :

- Monter des modules électroniques sur une Arduino.
- Établir une connexion socket entre Poste de supervision et Raspberry.
- Réaliser des schémas électriques des différents mécanismes.
- La liaison entre plusieurs cartes Arduino.

Et au niveau Software dans mes programmes j'ai appris à :

- Réaliser des programmes sockets en Python.
- Programmer une Arduino.
- Faire communiquer un PC de supervision avec une Raspberry
- La communication entre plusieurs cartes Arduino.



Poursuite d'étude

Durant ce confinement, ma candidature à une entreprise a été retenue : AIS Informatique. Je vais effectuer une poursuite d'étude de 2 ans en alternances au sein de cette entreprise, par le biais de l'école ENI informatique situé à Nantes (J'ai été accepté plus tôt dans l'année).

Durant ces 2 années, je serais orienté sur de la création d'application numériques en C# et de la programmation WEB. Ce sont deux spécialités dans lesquelles je voudrais m'orienter dans le futur.

Remerciements

Dans un premier temps, je voudrais remercier la société 13^{ème} porte, pour toutes l'aides qui nous ont apportés, dans la réalisation de ce projet. Les différents échanges que nous avons eu avec le directeur et la visite des locaux m'as aidé à comprendre le fonctionnement d'une entreprise. Je les remercie aussi pour leurs prêts de matériels.

Je remercie l'école, de nous proposer des sujets de projet aussi intéressant, et l'accompagnement pour mener à bien notre projet.

Je voudrais remercier Mr Angibaud, notre professeur tuteur pour notre projet. Pour l'accompagnement et la grande aide que vous avais pu nous donner notamment durant cette période de confinement.

Je remercie les étudiants Joshua PINNEAU, Constantin MINOS et Corentin BRENNY pour leur participation au projet.



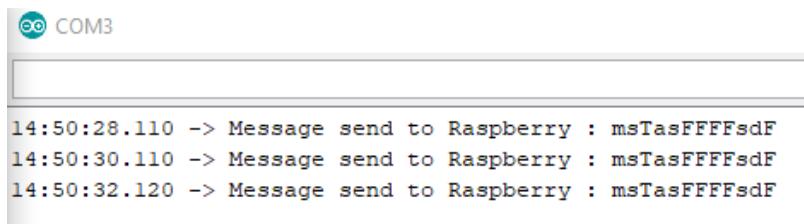
VI. Annexe

A. Annexe 1(Constantin MINOS) :

Test unitaire Etape 1 : 20 cm de câble

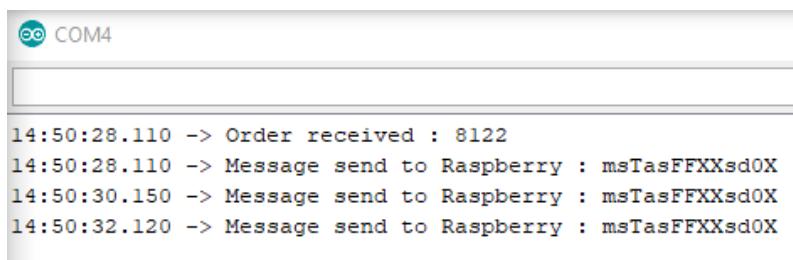
=> L'Arduino 1 a validé l'état de son mécanisme

=> L'Arduino 2 a reçu l'ordre de valider l'état de l'un de ses actionneurs



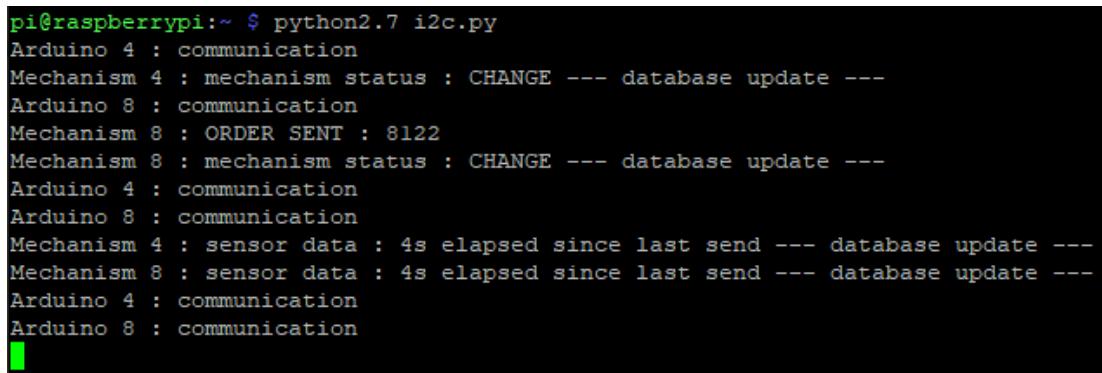
```
14:50:28.110 -> Message send to Raspberry : msTasFFFFsdF
14:50:30.110 -> Message send to Raspberry : msTasFFFFsdF
14:50:32.120 -> Message send to Raspberry : msTasFFFFsdF
```

Image 120 : Capture Arduino 1 du test unitaire



```
14:50:28.110 -> Order received : 8122
14:50:28.110 -> Message send to Raspberry : msTasFFXXsd0X
14:50:30.150 -> Message send to Raspberry : msTasFFXXsd0X
14:50:32.120 -> Message send to Raspberry : msTasFFXXsd0X
```

Image 121 : Capture Arduino 2 du test unitaire



```
pi@raspberrypi:~ $ python2.7 i2c.py
Arduino 4 : communication
Mechanism 4 : mechanism status : CHANGE --- database update ---
Arduino 8 : communication
Mechanism 8 : ORDER SENT : 8122
Mechanism 8 : mechanism status : CHANGE --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
```

Image 122 : Capture Putty du test unitaire

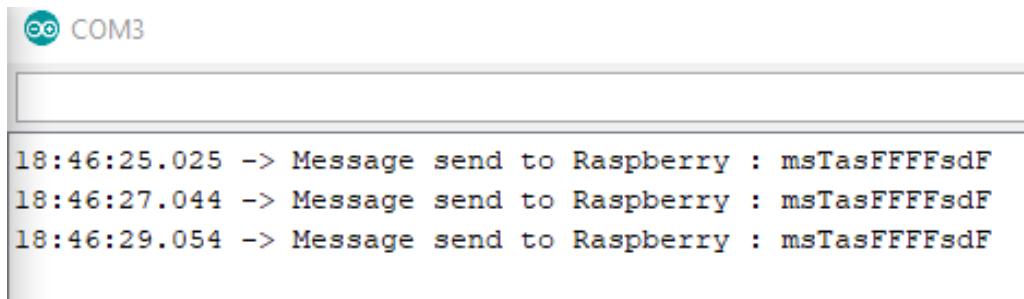


B. Annexe 2 :

Test unitaire Etape 1 : 1m de câble

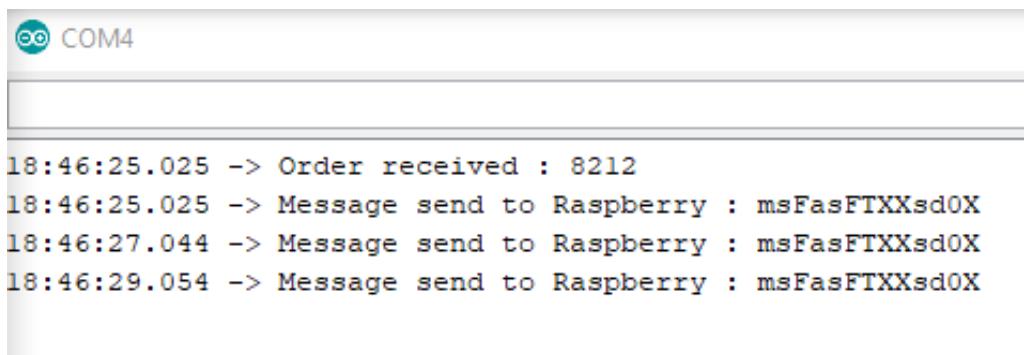
=> L'Arduino 1 a validé l'état de son mécanisme

=> L'Arduino 2 a reçu l'ordre de valider l'état de l'un de ses actionneurs



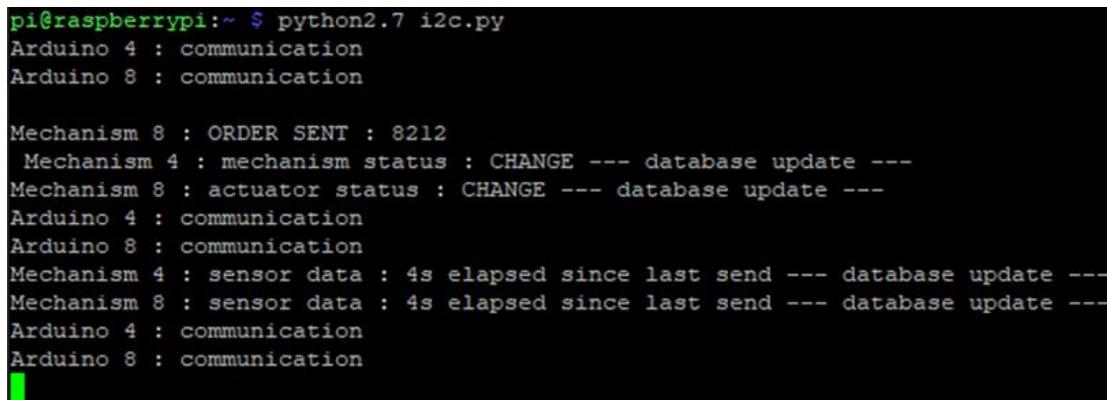
```
18:46:25.025 -> Message send to Raspberry : msTasFFFFsdF
18:46:27.044 -> Message send to Raspberry : msTasFFFFsdF
18:46:29.054 -> Message send to Raspberry : msTasFFFFsdF
```

Image 123 : Capture Arduino 1 du test unitaire



```
18:46:25.025 -> Order received : 8212
18:46:25.025 -> Message send to Raspberry : msFasFTXXsd0X
18:46:27.044 -> Message send to Raspberry : msFasFTXXsd0X
18:46:29.054 -> Message send to Raspberry : msFasFTXXsd0X
```

Image 124 : Capture Arduino 2 du test unitaire



```
pi@raspberrypi:~ $ python2.7 i2c.py
Arduino 4 : communication
Arduino 8 : communication

Mechanism 8 : ORDER SENT : 8212
Mechanism 4 : mechanism status : CHANGE --- database update ---
Mechanism 8 : actuator status : CHANGE --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
```

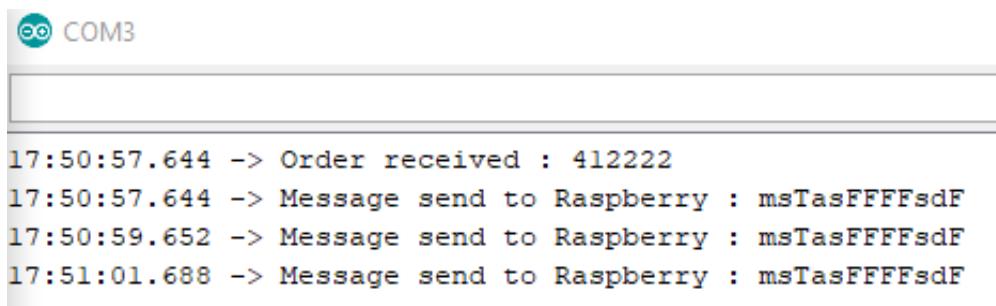
Image 125 : Capture Putty du test unitaire



C. Annexe 3 :

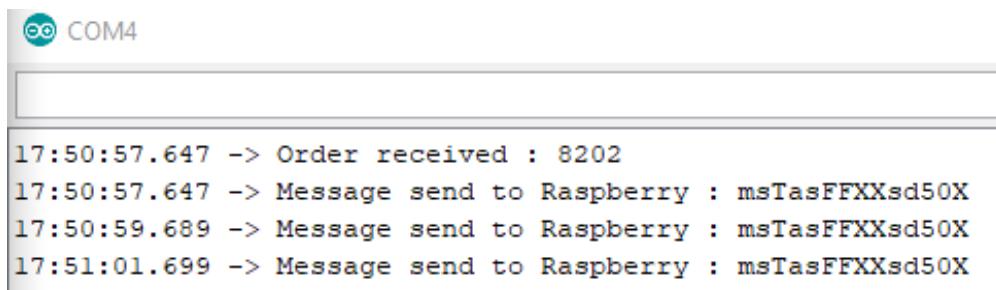
Test unitaire Etape 5 : 20 cm de câble

- => L'Arduino 1 a invalidé l'état de l'un de ses actionneurs
- => L'Arduino 1 a reçu l'ordre de valider l'état de son mécanisme
- => L'Arduino 2 a détecté 50g sur le capteur de poids
- => L'Arduino 2 a reçu l'ordre d'invalider l'état de l'un de ses actionneurs



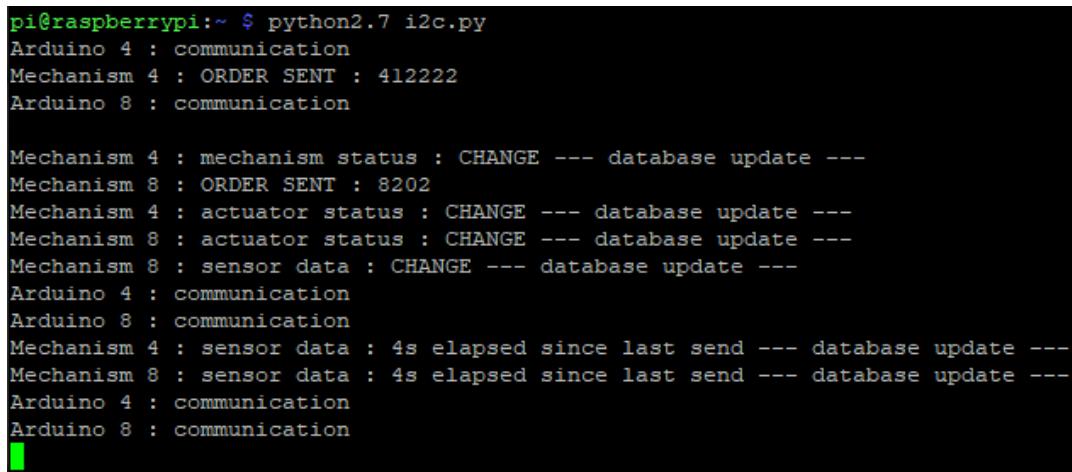
```
17:50:57.644 -> Order received : 412222
17:50:57.644 -> Message send to Raspberry : msTasFFFFsdF
17:50:59.652 -> Message send to Raspberry : msTasFFFFsdF
17:51:01.688 -> Message send to Raspberry : msTasFFFFsdF
```

Image 126 : Capture Arduino 1 du test unitaire



```
17:50:57.647 -> Order received : 8202
17:50:57.647 -> Message send to Raspberry : msTasFFXXsd50X
17:50:59.689 -> Message send to Raspberry : msTasFFXXsd50X
17:51:01.699 -> Message send to Raspberry : msTasFFXXsd50X
```

Image 127 : Capture Arduino 2 du test unitaire



```
pi@raspberrypi:~ $ python2.7 i2c.py
Arduino 4 : communication
Mechanism 4 : ORDER SENT : 412222
Arduino 8 : communication

Mechanism 4 : mechanism status : CHANGE --- database update ---
Mechanism 8 : ORDER SENT : 8202
Mechanism 4 : actuator status : CHANGE --- database update ---
Mechanism 8 : actuator status : CHANGE --- database update ---
Mechanism 8 : sensor data : CHANGE --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
```

Image 128 : Capture Putty du test unitaire



D. Annexe 4 :

Test unitaire Etape 5 : 1m de câble

- => L'Arduino 1 a invalidé l'état de l'un de ses actionneurs
- => L'Arduino 1 a reçu l'ordre de valider l'état de son mécanisme
- => L'Arduino 2 a détecté 50g sur le capteur de poids
- => L'Arduino 2 a reçu l'ordre d'invalider l'état de l'un de ses actionneurs

```
COM3
18:48:38.896 -> Order received : 412222
18:48:38.896 -> Message send to Raspberry : msTasFFFFsdF
18:48:40.904 -> Message send to Raspberry : msTasFFFFsdF
18:48:42.934 -> Message send to Raspberry : msTasFFFFsdF
```

Image 129 : Capture Arduino 1 du test unitaire

```
COM4
.8:48:38.896 -> Order received : 8202
.8:48:38.896 -> Message send to Raspberry : msTasFFXXsd50X
.8:48:40.904 -> Message send to Raspberry : msTasFFXXsd50X
.8:48:42.934 -> Message send to Raspberry : msTasFFXXsd50X
```

Image 130 : Capture Arduino 2 du test unitaire

```
pi@raspberrypi:~ $ python2.7 i2c.py
Arduino 4 : communication
Mechanism 4 : ORDER SENT : 412222
Mechanism 4 : mechanism status : CHANGE --- database update ---
Arduino 8 : communication
Mechanism 8 : ORDER SENT : 8202
Mechanism 4 : actuator status : CHANGE --- database update ---
Mechanism 8 : actuator status : CHANGE --- database update ---
Mechanism 8 : sensor data : CHANGE --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
```

Image 131 : Capture Putty du test unitaire



E. Annexe 5 :

Test unitaire Etape 6-7 : 20 cm de câble

=> Dernier envoi état/valeur capteurs il y a 4s : La Raspberry les envoie en BDD

```
pi@raspberrypi:~ $ python2.7 i2c.py
Arduino 4 : communication
Mechanism 4 : ORDER SENT : 402212
Mechanism 4 : mechanism status : CHANGE --- database update ---
Arduino 8 : communication
Mechanism 8 : ORDER SENT : 8022
Mechanism 4 : actuator status : CHANGE --- database update ---
Mechanism 8 : mechanism status : CHANGE --- database update ---
Mechanism 8 : sensor data : CHANGE --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
[REDACTED]
```

Image 132 : Capture Putty du test unitaire



F. Annexe 6 :

Test unitaire Etape 6-7 : 1m de câble

=> Dernier envoi état/valeur capteurs il y a 4s : La Raspberry les envoie en BDD

```
pi@raspberrypi:~ $ python2.7 i2c.py
Arduino 4 : communication
Mechanism 4 : ORDER SENT : 402212
Mechanism 4 : mechanism status : CHANGE --- database update ---
Mechanism 4 : actuator status : CHANGE --- database update ---
Arduino 8 : communication
Mechanism 8 : ORDER SENT : 8022
Mechanism 8 : mechanism status : CHANGE --- database update ---
Mechanism 8 : sensor data : CHANGE --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
```

Image 133 : Capture Putty du test unitaire



G. Annexe 7 :

Test unitaire Etape 8 : 20 cm de câble

=> Dernier envoi état mécanismes/actionneurs il y a 60s : La Raspberry les envoie en BDD

```
pi@raspberrypi: ~
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 8 : actuator status : 60s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 4 : mechanism status : 60s elapsed since last send --- database update ---
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 4 : actuator status : 60s elapsed since last send --- database update ---
Mechanism 8 : mechanism status : 60s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
```

Image 134 : Capture Putty du test unitaire



H. Annexe 8 :

Test unitaire Etape 8 : 1m de câble

=> Dernier envoi état mécanismes/actionneurs il y a 60s : La Raspberry les envoie en BDD

```
pi@raspberrypi: ~
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : mechanism status : 60s elapsed since last send --- database update ---
Mechanism 4 : actuator status : 60s elapsed since last send --- database update ---
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : actuator status : 60s elapsed since last send --- database update ---
Mechanism 8 : mechanism status : 60s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
Arduino 4 : communication
Arduino 8 : communication
Mechanism 4 : sensor data : 4s elapsed since last send --- database update ---
Mechanism 8 : sensor data : 4s elapsed since last send --- database update ---
Arduino 4 : communication
Arduino 8 : communication
```

Image 135 : Capture Putty du test unitaire



I. Annexe 9 :

Procédure d'installation et de configuration de la Raspberry

Partie Hardware

1) Formater la carte micro-SD (SD Card Formatter)

lien : <https://www.sdcard.org/downloads/formatter/>

2) Télécharger l'image système Raspbian (Version Desktop de préférence)

lien : <https://www.raspberrypi.org/downloads/raspbian/>

3) Ecrire l'image Système sur la carte SD :

- Installer de Win32DiskImager.exe et lancer
lien : <https://win32diskimager.download/>
- Choisir l'image système Raspbian et la carte SD
- Write

4) Insérer la carte Micro-SD dans la Raspberry + réaliser le câblage I2C

5) Alimenter la Raspberry

Partie Software

1) Se connecter :

- Nom d'utilisateur par défaut : **pi**
- Mot de passe par défaut : **raspberry** (Attention clavier QWERTY raspberry = rasperry)

2) Changer le clavier en AZERTY :

- Aller dans l'invité de commande
- Ecrire la commande **sudo dpkg-reconfigure keyboard-configuration**
- Conserver le choix par défaut « Generic 105-Key (Intl) PC » puis « Ok »
- Sélectionner le choix « French » puis « Ok »
- Sélectionner le clavier « The default for the keyboard layout » puis « Ok »
- Sélectionner le choix « No compose key » puis « Ok »
- Terminer la configuration par « Finish »
- Ecrire la commande **sudo shutdown -r now** (Pour redémarrer de la Raspberry)

3) Changer le mot de passe :

- Aller dans l'invité de commande
- Ecrire la commande **sudo raspi-config**
- Sélectionner « Change User Password » puis « Ok »



- Choisir son nouveau mot de passe puis « Ok »

4) Se connecter au Wifi :

- Ecrire la commande **sudo raspi-config**
- Sélectionner « Network Option » puis « Wi-fi »
- Entrer le SSID (nom du réseau) puis le mdp de la Wifi
- Sélectionner « Localisation Options » puis « Change Wifi Country »
- Dans la liste choisir « France » puis « Ok »
- Terminer la configuration par « Finish »

Optionnel : Pour vérifier la connexion Internet et le débit (speedtest) :

- Ecrire la commande **sudo apt-get install python3-pip** (Pour installer pip)
- Ecrire la commande **sudo pip3 install speedtest-cli** (Pour installer speedtest)
 - Ecrire la commande **speedtest-cli --simple** (Pour lancer speedtest)

```
pi@raspberrypi:~ $ speedtest-cli --simple
Ping: 80.409 ms
Download: 2.76 Mbit/s
Upload: 2.65 Mbit/s
```

Image 136

5) Mettre à jour la Raspberry :

- Ecrire la commande **sudo apt-get update**
- Ecrire la commande **sudo apt-get upgrade** puis y pour valider

6) Créer une IP fixe (@192.168.0.10) :

- Ecrire la commande **sudo nano /etc/dhcpcd.conf** (Pour modifier le fichier dhcpcd.conf)
- Supprimer tout son contenu et le remplacer par :

```
interface wlan0
static ip_address=192.168.0.10/24
static routers=192.168.0.1
```

Image 137

CTRL + O pour sauvegarder, ENTRER pour valider puis CTRL + X pour quitter

- Ecrire la commande **sudo shutdown -r now** (Pour redémarrer de la Raspberry)
- Ecrire la commande **ifconfig** (Pour vérifier les changements) :

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.0.10 netmask 255.255.255.0 broadcast 192.168.0.255
              inet6 fe80::ba27:ebff:fe9:63d2 prefixlen 64 scopeid 0x20<link>
                ether b8:27:eb:e9:63:d2 txqueuelen 1000 (Ethernet)
                  RX packets 17013 bytes 13233726 (12.6 MiB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 17173 bytes 11522631 (10.9 MiB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Image 138



7) Activer l'i2c :

- Ecrire la commande **sudo raspi-config**
- Sélectionner « Interfacing Options »
- Sélectionner l'option (P5) I2C puis « Yes » pour activer
- Ecrire la commande **sudo apt-get install i2c-tools** (Pour installer les outils de gestion i2c)
- Ecrire la commande **sudo shutdown -r now** (Pour redémarrer de la Raspberry)
- Ecrire la commande **ls -l /dev/i2c*** (Permet de vérifier l'activation i2c et identifier les bus i2c)

8) Communiquer avec l'Arduino :

- Ecrire la commande **i2cdetect -y 1** (Pour détecter les esclaves sur le bus)
- Ecrire la commande **sudo apt-get install python-smbus** (Pour installer le support i2c pour scripts Python)

9) Lancer le programme « i2c.py » au démarrage de la Raspberry :

- Insérer le programme « i2c.py » dans /home/pi/
- Ecrire la commande **python** (Pour connaître la version de Python) puis CTRL+D (Pour quitter)

```
pi@raspberrypi:~ $ python
Python 2.7.16 (default, Oct 10 2019, 22:02:15)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Image 139

Pour moi Python 2.7

- Ecrire la commande **sudo nano /etc/rc.local** (Pour modifier le fichier rc.local)
- Chercher la ligne « *exit 0* » et la modifier par « **/usr/bin/python2.7 /home/pi/i2c.py &** » (Adapté selon sa version de Python)

Optionnel :

Pour activer le SSH :

- Ecrire la commande **sudo touch /boot/ssh** (Pour créer un fichier ssh)
- Ecrire la commande **sudo shutdown -r now** (Pour redémarrer de la Raspberry)

Pour accéder à l'invité de commande de la Raspberry depuis son ordinateur (connexion SSH) :

- Avoir activé le ssh sur la Raspberry
- Sur son ordinateur :
 - Installer Putty et lancer
lien : <https://www.putty.org/>
 - Dans « Host Name » entrer l'adresse IP de la Raspberry **192.168.0.10** puis Open
 - Une fenêtre s'affiche : Oui



- Entrer ses identifiants Raspberry

Pour insérer ou télécharger des fichiers sur la Raspberry depuis son ordinateur (connexion SFTP) :

- Avoir activé le ssh sur la Raspberry
- Sur son ordinateur :
 - Installer FileZilla et lancer
lien : <https://filezilla-project.org/download.php?type=client#close>
 - Identifier la Raspberry :
 - Dans « Hôte » entrer l'adresse IP de la Raspberry **192.168.0.10**
 - Dans « Identifiant » entrer le nom d'utilisateur de la Raspberry pi
 - Dans « Mot de passe » entrer le mdp de la Raspberry
 - Dans « Ports » entrer **22**
 - Connexion Rapide

J. Annexe 10 (Joshua PINNEAU) :

Configuration IP du PC :

Pour configurer l'adresse IP du PC qui va nous servir à héberger l'Application Web et la base de données, il faut effectuer quelques modifications et notamment quelques commandes dans le « CMD », c'est-à-dire l'invite de commande.

1. Tapez dans la barre de recherche de windows « **CMD** » et appuyez sur entrer.

2. Tapez, dans le cmd, la commande suivante : « **ipconfig/release** ». L'adresse Ip de l'ordinateur sera effacée.

3. Faites un clic droit sur l'icône réseau dans la barre d'état puis cliquez sur « **Ouvrir le Centre Réseau et partage** ».

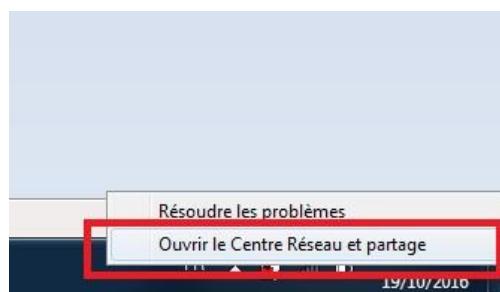


Image 140 : Ouvrir le Centre Réseau et partage

4. La fenêtre suivante s'ouvre, cliquez sur « **Modifier les paramètres de la carte** ».



Image 141 : Modifier les paramètres de la carte

5. Choisissez l'interface de connexion (réseau local, réseau sans fil) dont vous souhaitez changer son adresse IP, puis cliquez avec le bouton droit et sélectionner « propriété »

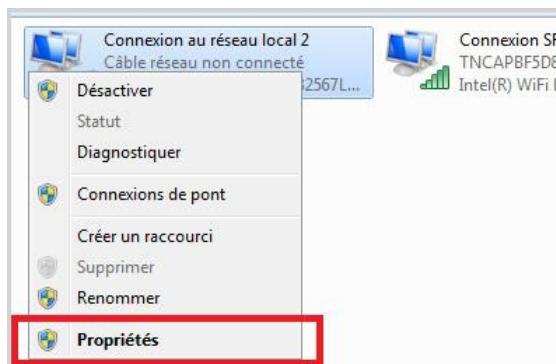


Image 142 : Connexion au réseau local 2

6. Ici, cochez l'option « protocole internet version 4 (TCP/IPv4) » et cliquez sur « propriété »

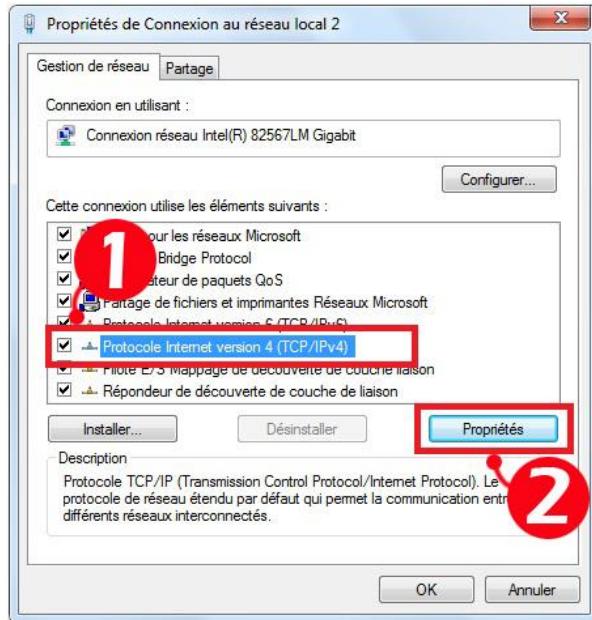


Image 143 : Cocher IPv4

7. Maintenant, cochez l'option « utiliser l'adresse IP suivante » et saisissez votre adresse IP et la passerelle par défaut (l'adresse IP de routeur).

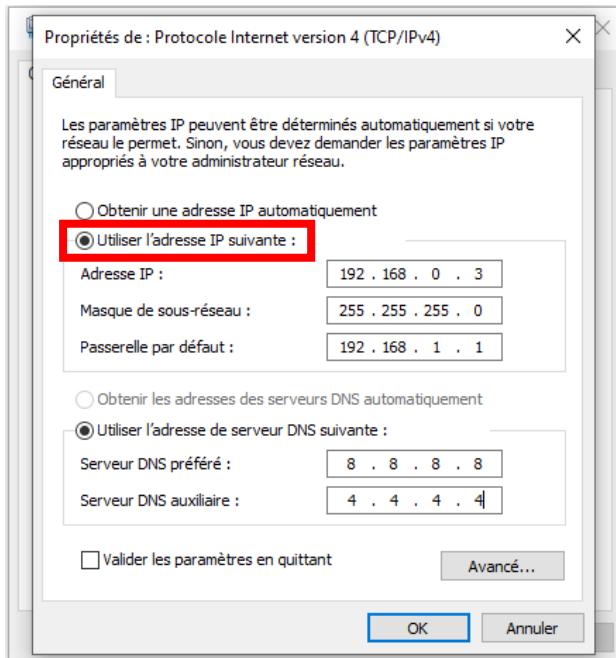


Image 144 : Cocher l'Adresse IP suivante



K. Annexe 11 :

Configuration IP de la Raspberry

1. Avant de commencer à attribuer une adresse IP privée statique pour le Raspberry Pi, **vérifiez si « DHCPCD » est déjà activé** à l'aide de la commande suivante :

« `sudo service dhcpcd status` »

1. (bis) Si ce n'est pas le cas, activez le DHCPCD comme suit :

1. « `sudo service dhcpcd start` »
 2. « `sudo systemctl enable dhcpcd` »

2. Assurez-vous maintenant que la configuration du fichier `/etc/network/interfaces` possède le statut d'origine. Pour cela, la configuration « `iface` » doit être réglée sur « `manual` ».

3. Pour commencer à éditer le DHCPCD activé, ouvrez le **fichier de configuration**

« `/etc/dhcpcd.conf` » et exécutez la commande suivante :
« `sudo nano /etc/dhcpcd.conf` »

4. Pour attribuer une adresse IP au Raspberry Pi, utilisez la commande « `static ip_address=` », suivie de **l'adresse IPv4 souhaitée** puis de « `/24` » (abréviation du masque de sous-réseau `255.255.255.0`). Par exemple, si vous voulez lier l'ordinateur à l'adresse IPv4 `192.168.0.4`, la commande doit être « `static ip_address=192.168.0.4/24` ».

Ensuite, vous devez spécifier l'adresse de votre passerelle et du serveur de nom de domaine (généralement les deux se trouvent sur le routeur). Le Raspberry Pi se tourne vers l'adresse de la passerelle si une adresse IP vers laquelle il veut envoyer quelque chose se trouve en dehors du masque de sous-réseau (dans l'exemple, en dehors de la plage `192.168.0`). Dans la commande suivante, l'adresse IPv4 `192.168.0.1` est utilisée pour la passerelle et le serveur DNS. La commande complète ressemble alors dans notre à ceci :

1. « `interface wifi0` »
 2. « `static ip_address=192.168.0.2/24` »
 3. « `static routers=192.168.0.1` »
 4. « `static domain_name_servers=192.168.0.1` »

5. Sauvegardez les changements avec « `Ctrl + O` » et appuyez ensuite sur Entrée. Utilisez « `Ctrl + X` » pour fermer le fichier de configuration. Un redémarrage reprend les adresses IP statiques nouvellement attribuées dans le réseau :

« `sudo reboot` »



6. Vous pouvez maintenant utiliser une **commande « Ping »** pour vérifier si le Raspberry Pi peut être atteint dans le réseau avec sa nouvelle adresse IP :
« Ping raspberrypi.local »

7. Si l'adresse IP a été liée avec succès, vous pouvez voir que vous pouvez l'atteindre sous la nouvelle adresse IP assignée avec un « **Ping** ».

L. Annexe 12 :

Gérer les adresses IP

En cas de conflit entre les différentes adresses IP des matériaux que nous avons à notre disposition ou juste pour éviter ces conflits, il va falloir ajouter des adresses IP réservés dans le DHCP de notre Box Internet.

Lorsque vous spécifiez une adresse IP réservée pour un PC sur le réseau local, le PC recevra toujours la même adresse IP chaque fois qu'il se connectera au serveur DHCP. Si certains ordinateurs du réseau local requièrent des adresses IP permanentes, veuillez configurer à cet effet la **réservation d'adresses** sur le routeur.

Afin de réserver une adresse IP pour l'un de nos composant, il va falloir suivre ce tutoriel.

1. Ouvrez le navigateur Web et tapez dans la barre d'adresse : « **http://192.168.1.1** » ou « **http://192.168.0.1** » ou « **http://tplinklogin.net** ». Puis appuyez sur **Entrée**. (L'IP LAN change par modèle. Veuillez le trouver sur l'étiquette inférieure du produit.)

Pour accéder aux réglages de votre Livebox
veuillez saisir votre mot de passe

mot de passe

[Mot de passe oublié ou première connexion](#)

connexion

Image 145 : Administration de la Livebox



2. Tapez le nom d'utilisateur et le mot de passe dans la page de connexion.

3. Cherchez l'onglet DHCP et la fonctionnalité permettant d'ajouter une adresse IP réservée.

4. Entrez l'adresse MAC et l'adresse IP ainsi que son statut afin de le mettre en « activé ».

Entrez l'adresse MAC et l'adresse IP, sélectionnez **Status as Enabled**.

Add or Modify an Address Reservation Entry

MAC Address:

Reserved IP Address:

Status:

Save Back

Image 146 : Entrer une adresse MAC

Si vous n'avez pas l'adresse MAC d'un composant, il suffit de faire comme suit :

1. Pour **Windows**, sélectionnez « Démarrer-> Programmes-> Accessoires-> Invite de commandes ».
2. Pour **MAC**, allez sur « Disque dur-> Applications-> Utilitaires-> Terminal ».
3. Pour **Linux**, ouvrez une fenêtre « Telnet / TERMINAL ».

4. Tapez « **ipconfig/all** » dans la fenêtre d'invite, puis appuyez sur « Entrée ». Il montrera l'adresse MAC (adresse physique) et l'adresse IP de cet ordinateur ou Raspberry dans le cas présent.



Image 147 : Trouver son adresse MAC