

# SFL5 : SYSTEME ESCAPE GAME

## 13E PORTE





## Sommaire

I.	Introduction .....	2
A.	Rappel du cahier des charges .....	2
B.	Répartition des tâches .....	4
II.	Analyse .....	5
A.	Les mécanismes .....	5
	Mécanisme 1 : Echiquier .....	5
	Mécanisme 2 : Lion .....	7
	Mécanisme 3 : Terre.....	9
	Mécanisme 4 : Feu .....	11
	Mécanisme 5 : Eau .....	13
	Mécanisme 6 : Air.....	15
	Mécanisme 7 : Katana .....	16
	Mécanisme 8 : Riz.....	17
	Mécanisme 9 : Quatre Eléments .....	19
B.	Application Web de supervision .....	22
	Application Web : Visualiser l'état de la salle.....	22
	Application Web : Piloter les actionneurs .....	25
C.	I2C : Communication entre les Arduinos et la Raspberry .....	28
D.	La Base de Données .....	29
III.	Réalisation du projet.....	30
A.	Etudiant 1 : Constantin .....	30
	Rappel de la tâche de l'étudiant.....	30
	Mécanisme 4 : Feu .....	31
	Mécanisme 8 : Riz.....	32
	Liaison I2C .....	34
B.	Etudiant 2 : Corentin.....	41
	Création de la base de données .....	41
C.	Etudiant 3 : Joshua.....	43
	Rappel de la tâche de l'étudiant.....	43
	Mécanisme 1 : Echiquier .....	43
	Mécanisme 2 : Lion .....	45
	Mécanisme 3 : Terre.....	46
	Application Web : Visualiser l'état de la salle.....	47
	Réseau .....	51
D.	Etudiant 4 : Thomas .....	52
	Mécanisme 5 : Eau .....	52
	Mécanisme 9 : Quatres Eléments.....	54



## I. Introduction

### A. Rappel du cahier des charges

#### Objectifs du projet

Mme. Sterenn LE GOFFIC et M. Jean-Charles Douguet, les deux gérants de la société, souhaitent que le système technique actuellement en place soit recréé entièrement afin de corriger les erreurs de conception et les différents bugs existants. Il est également demandé l'ajout de plusieurs fonctionnalités permettant d'améliorer le travail du superviseur.

#### Système de gestion de la salle

Actuellement, les nombreux mécanismes du jeu dans la salle sont gérés par quatre Arduino UNO. Chaque Arduino a la charge de plusieurs mécanismes distincts. Les gérants souhaitent revoir ce découpage dans le nouveau système en associant à chacune des neuf étapes du jeu un Arduino dédié. Une étape sera appelée mécanisme dans la suite de ce document. Les différents mécanismes du jeu sont nommés ci-dessous.

**Mécanisme 1.** L'échiquier

**Mécanisme 2.** Le lion

**Mécanisme 3.** L'élément TERRE

**Mécanisme 4.** L'élément FEU

**Mécanisme 5.** L'élément EAU

**Mécanisme 6.** L'élément AIR

**Mécanisme 7.** Le Katana

**Mécanisme 8.** Le riz

**Mécanisme 9.** Les quatre éléments

#### Système de supervision

Le commanditaire souhaite disposer de deux fonctionnalités dédiées à la supervision et utiles à la maintenance :

- Le superviseur pourra visualiser depuis un ordinateur l'état du système. L'interface devra afficher à la fois l'état d'avancement dans le jeu, mais aussi la valeur de chaque capteur et actionneur.
- Afin de faciliter la maintenance, une interface devra permettre de piloter depuis un ordinateur chaque actionneur.

#### Situation du projet dans son contexte

Domaine d'activité du système support d'étude :	<input type="checkbox"/> télécommunications, téléphonie et réseaux téléphoniques <input checked="" type="checkbox"/> informatique, réseaux et infrastructures <input type="checkbox"/> multimédia, son et image, radio et télédiffusion <input checked="" type="checkbox"/> mobilité et systèmes embarqués <input type="checkbox"/> électronique et informatique médicale <input checked="" type="checkbox"/> mesure, instrumentation et microsystèmes <input type="checkbox"/> automatique et robotique
---	--



## Cahier des charges – Expression du besoin

Le projet répondra aux besoins suivants :

- ✓ Gérer les neufs mécanismes du jeu
- ✓ Collecter et enregistrer l'état de la salle (valeur de chaque capteur, état de chaque actionneur)
- ✓ Visualiser l'état de la salle (valeur de chaque capteur, état de chaque actionneur)
- ✓ Piloter les actionneurs

## Gérer les neufs mécanismes du jeu

Afin d'assurer la confidentialité du scénario, nous présentons dans ce document les différents mécanismes sans préciser la nature exacte des objets à manipuler dans la salle de jeu. Toutefois, les descriptions permettront d'évaluer le travail à réaliser.

## Matériels utilisés

### *Raspberry*

Une Raspberry Pi 3B est nécessaire pour notre projet et se présente sous cette forme :



### *Arduino*

Neuf Arduinos Nanos sont nécessaires pour les 9 mécanismes du jeu.

Et chaque Arduino nécessite également du matériel.

Voici la liste globale du matériel nécessaire pour toutes les Arduinos :

- ✓ Quatre capteurs à effet hall A3144
- ✓ Module de commutateur de clé 3 pin 1NO1NC
- ✓ Afficheur 7 segments
- ✓ Capteur d'humidité
- ✓ Interrupteur à bascule
- ✓ Sept capteurs photosensibles
- ✓ Boutons poussoir
- ✓ Leds
- ✓ Capteur de fin de course
- ✓ Capteur de poids HX711



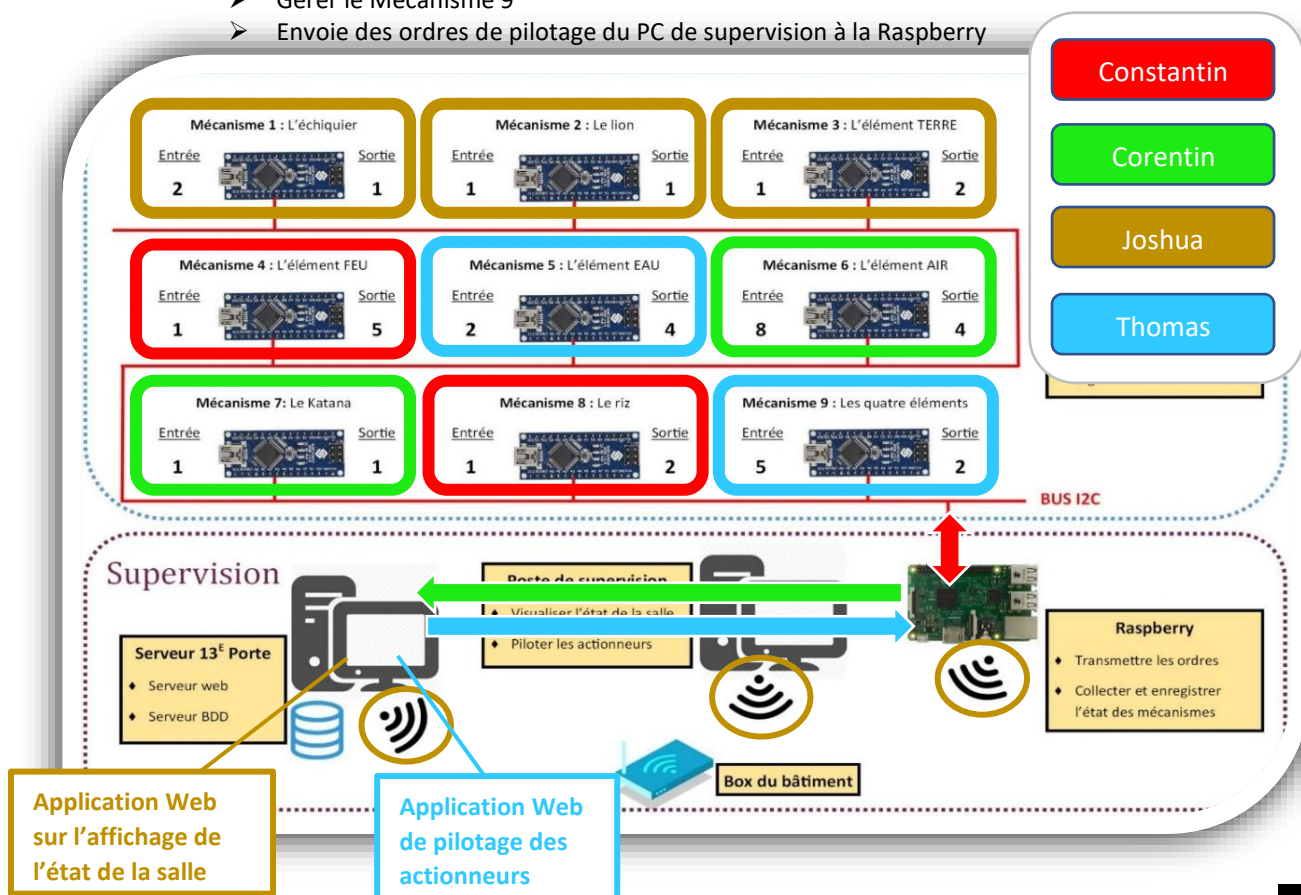
### *Un poste informatique pour le serveur 13e Porte/supervision*

Un poste informatique est nécessaire pour l'application Web et le Serveur BDD.



## B. Répartition des tâches

- **Etudiant 1 : Constantin**
  - Gérer le Mécanisme 4
  - Gérer le Mécanisme 8
  - Gestion du bus I2c entre Arduino et Raspberry :
    - Envoie des ordres de pilotage de la Raspberry à l'Arduino et les traiter
    - Envoi des mesures et des états des actionneurs sur la Raspberry et les traiter
- **Etudiant 2 : Corentin**
  - Créer la base de données
  - Installer les serveurs Apache et MySql
  - Gérer le Mécanisme 6
  - Gérer le Mécanisme 7
  - Récupérer la valeur des capteurs et l'état des actionneurs de la Raspberry au serveur
- **Etudiant 3 : Joshua**
  - Configuration réseau du matériel
  - Création d'une application Web sur l'affichage de l'état de la salle
  - Gérer le Mécanisme 1
  - Gérer le Mécanisme 2
  - Gérer le Mécanisme 3
- **Etudiant 4 : Thomas**
  - Création d'une application Web de pilotage des actionneurs
  - Gérer le Mécanisme 5
  - Gérer le Mécanisme 9
  - Envoi des ordres de pilotage du PC de supervision à la Raspberry





## II. Analyse

### A. Les mécanismes

#### Mécanisme 1 : Echiquier

##### 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.

##### Diagramme de classe

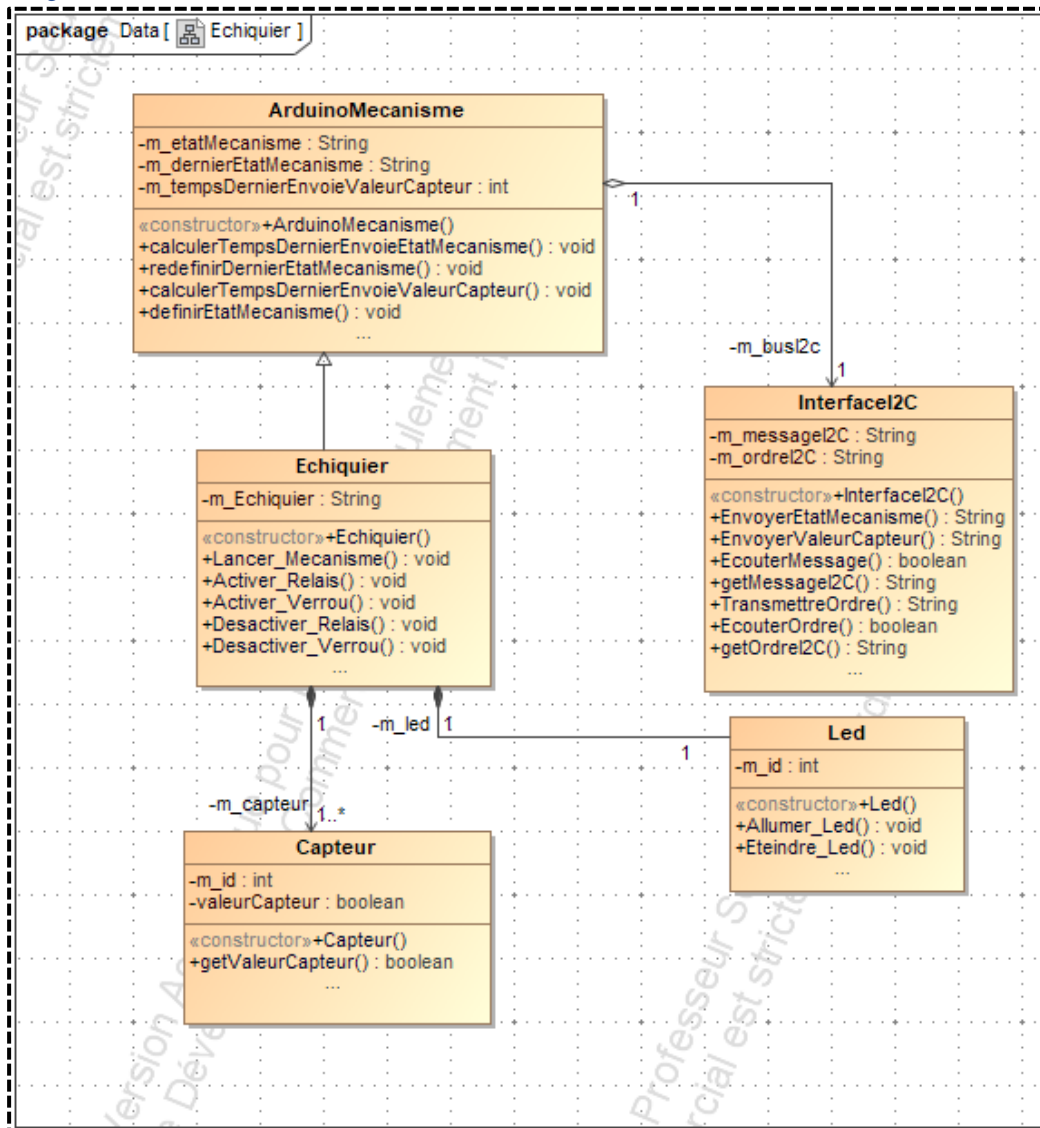
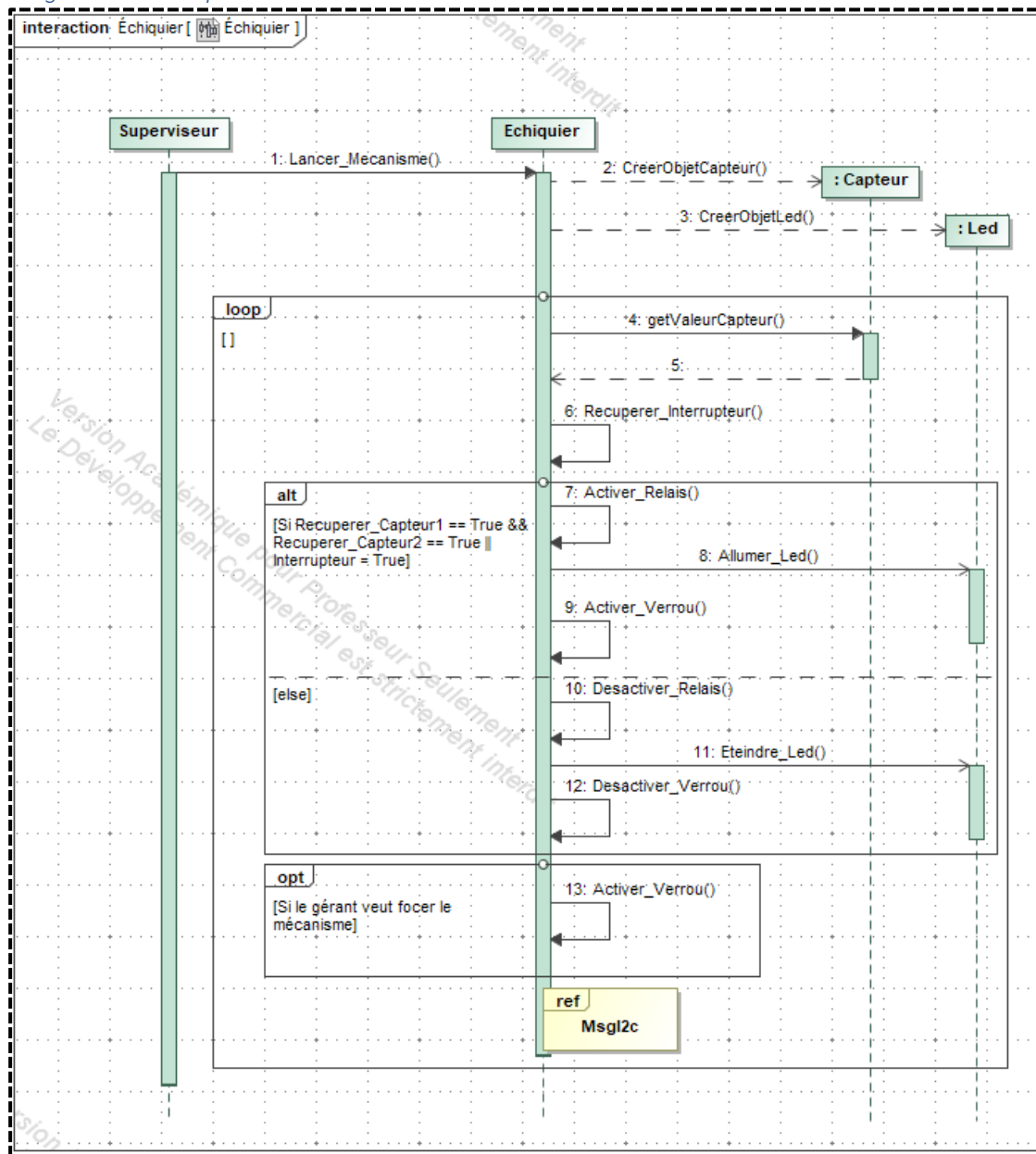




Diagramme de séquence







## Mécanisme 2 : Lion

### 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.

### Diagramme de classe

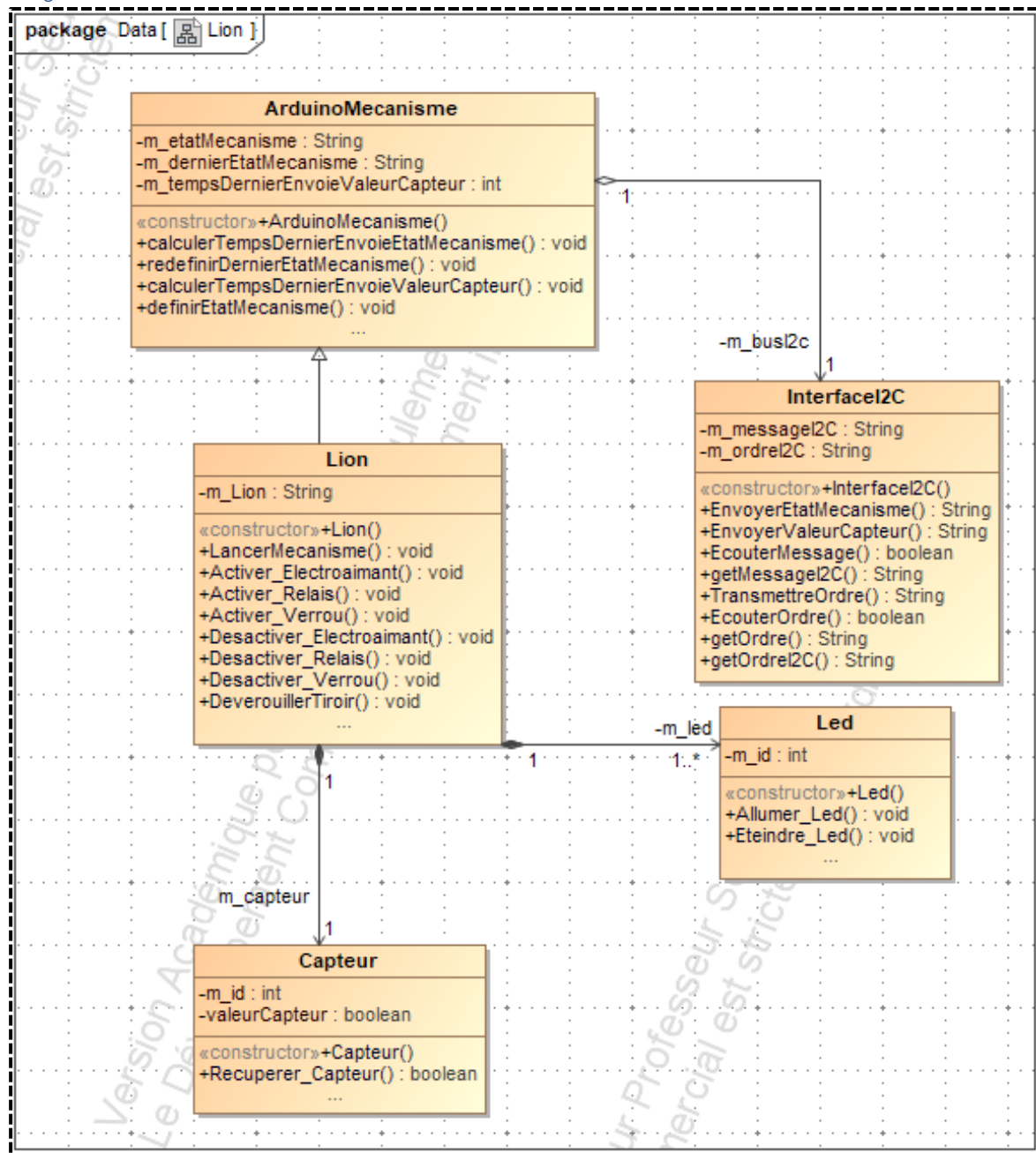
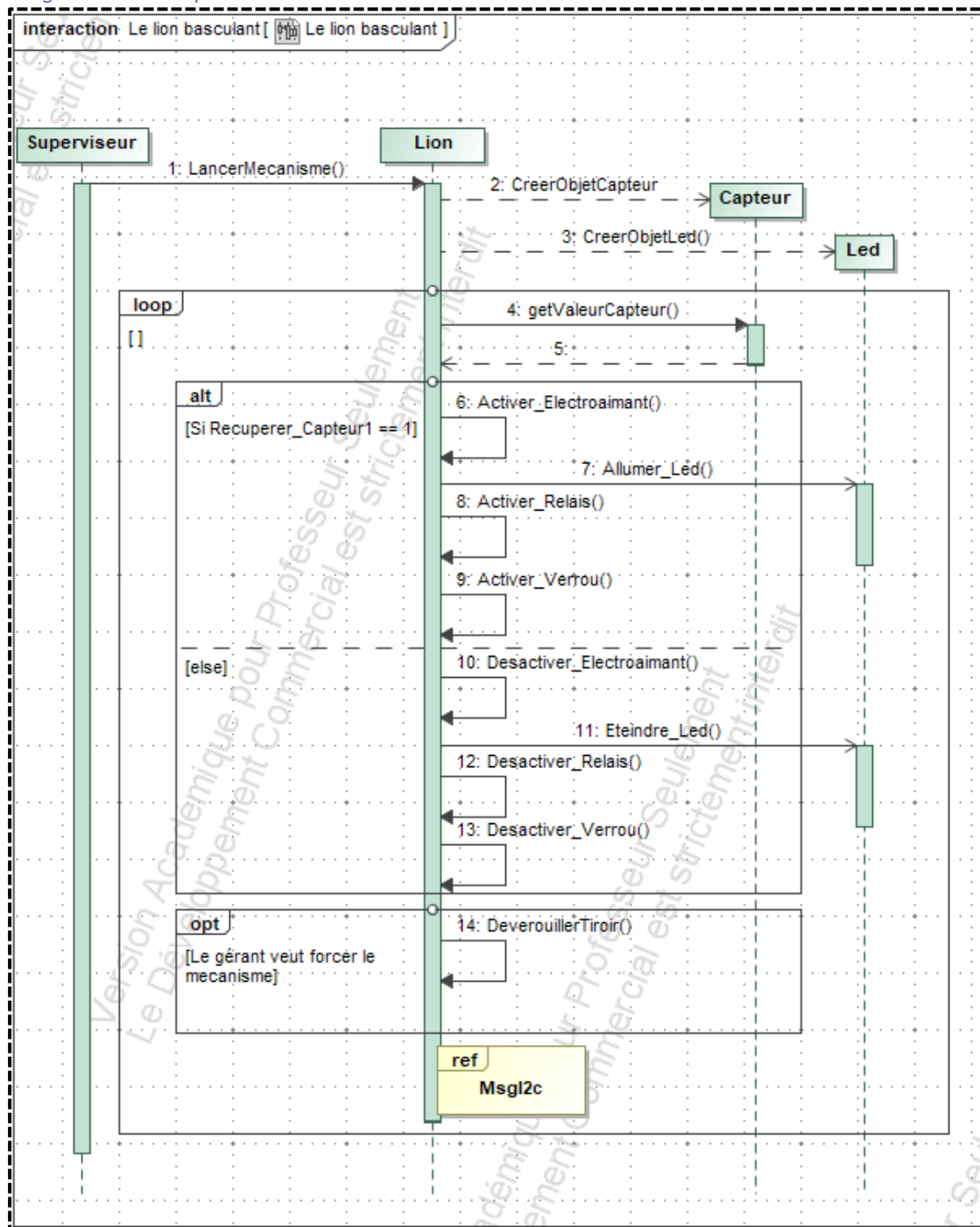






Diagramme de séquence





## Mécanisme 3 : 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.

### Diagramme de classe

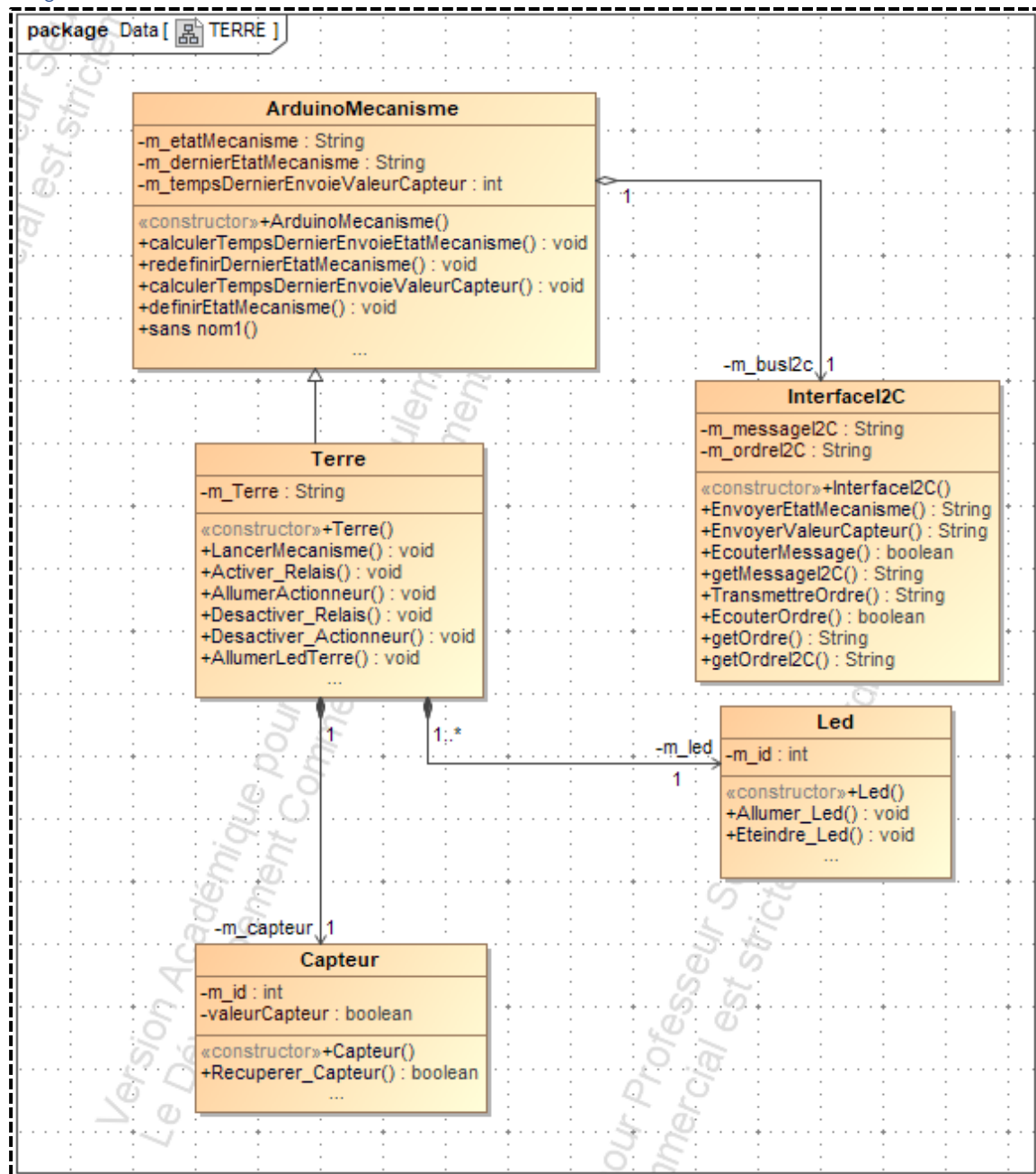
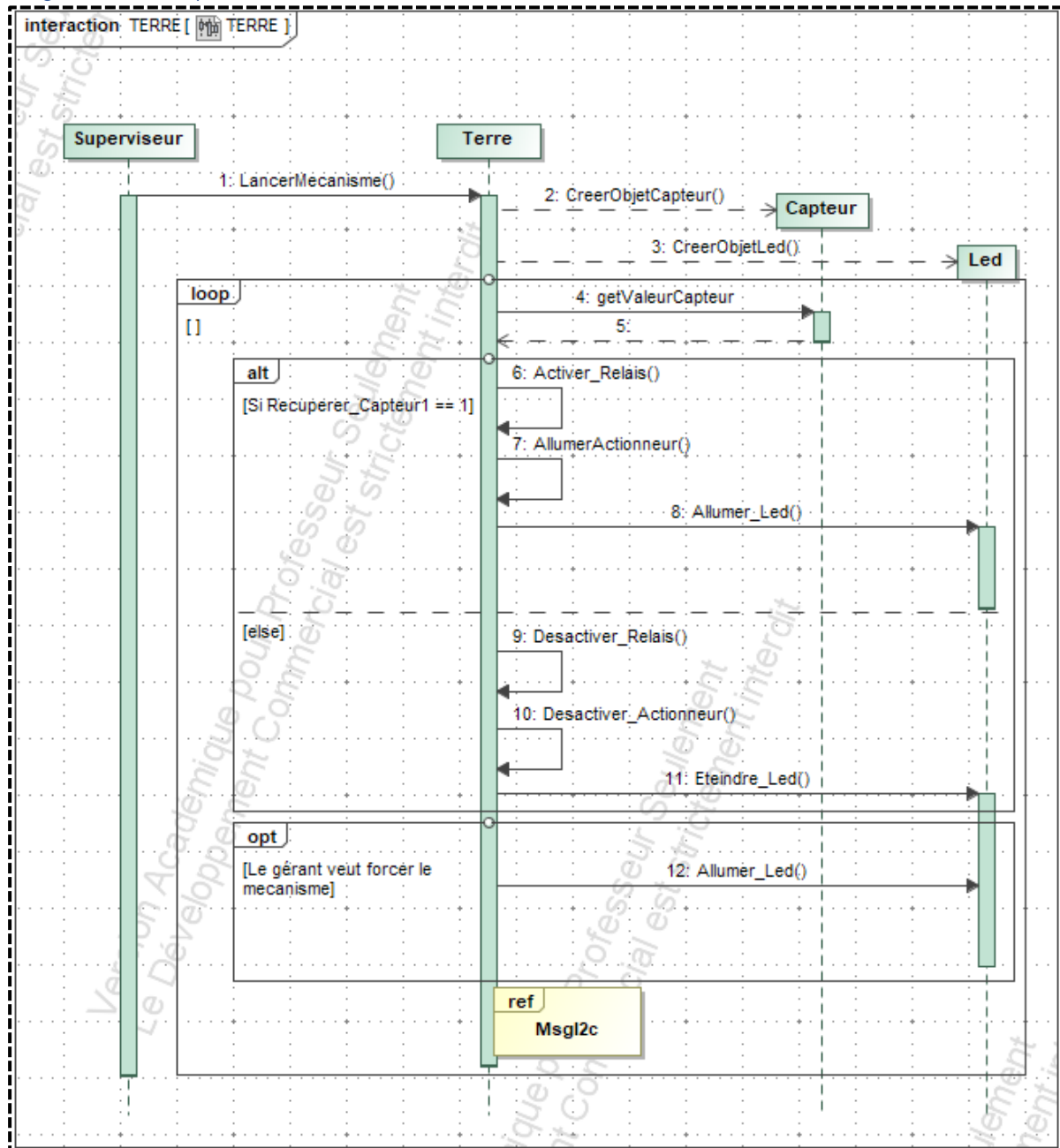




Diagramme de séquence





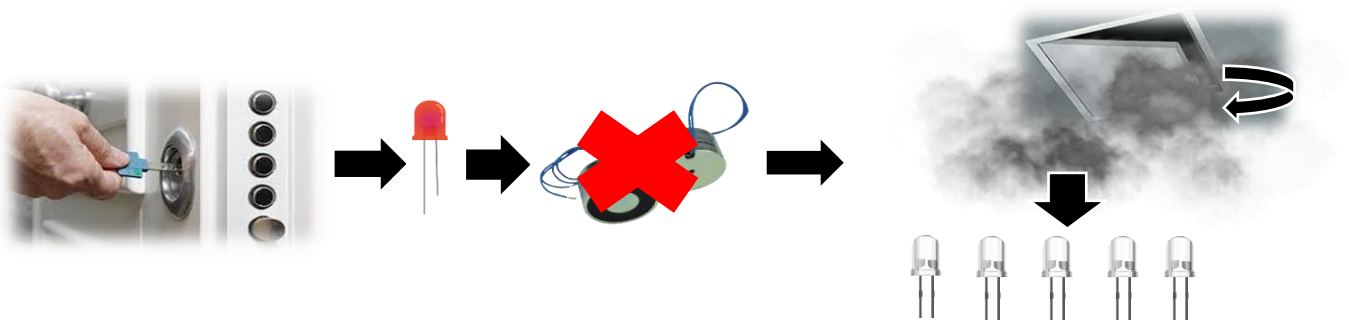
## Mécanisme 4 : Feu

### Synopsis du mécanisme

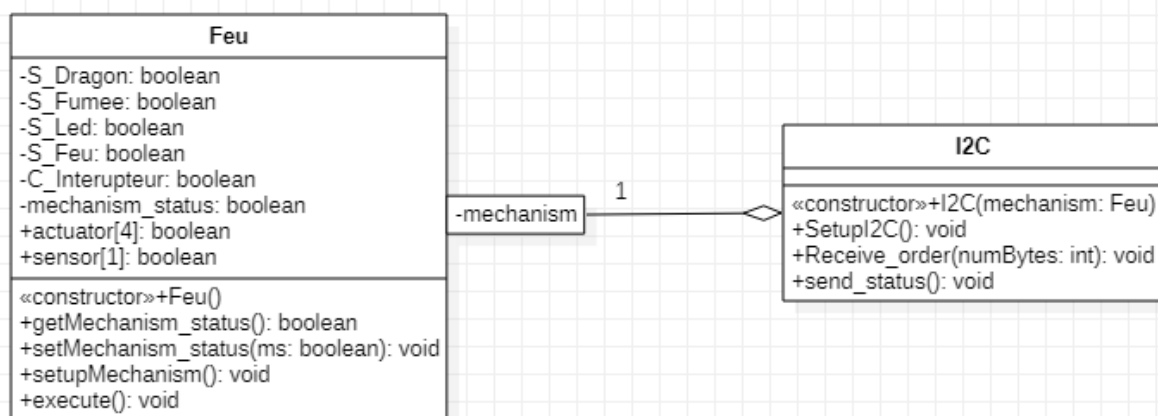
Le mécanisme 4 repose sur **un interrupteur à clef**.

S'il y a un changement de position positif :

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

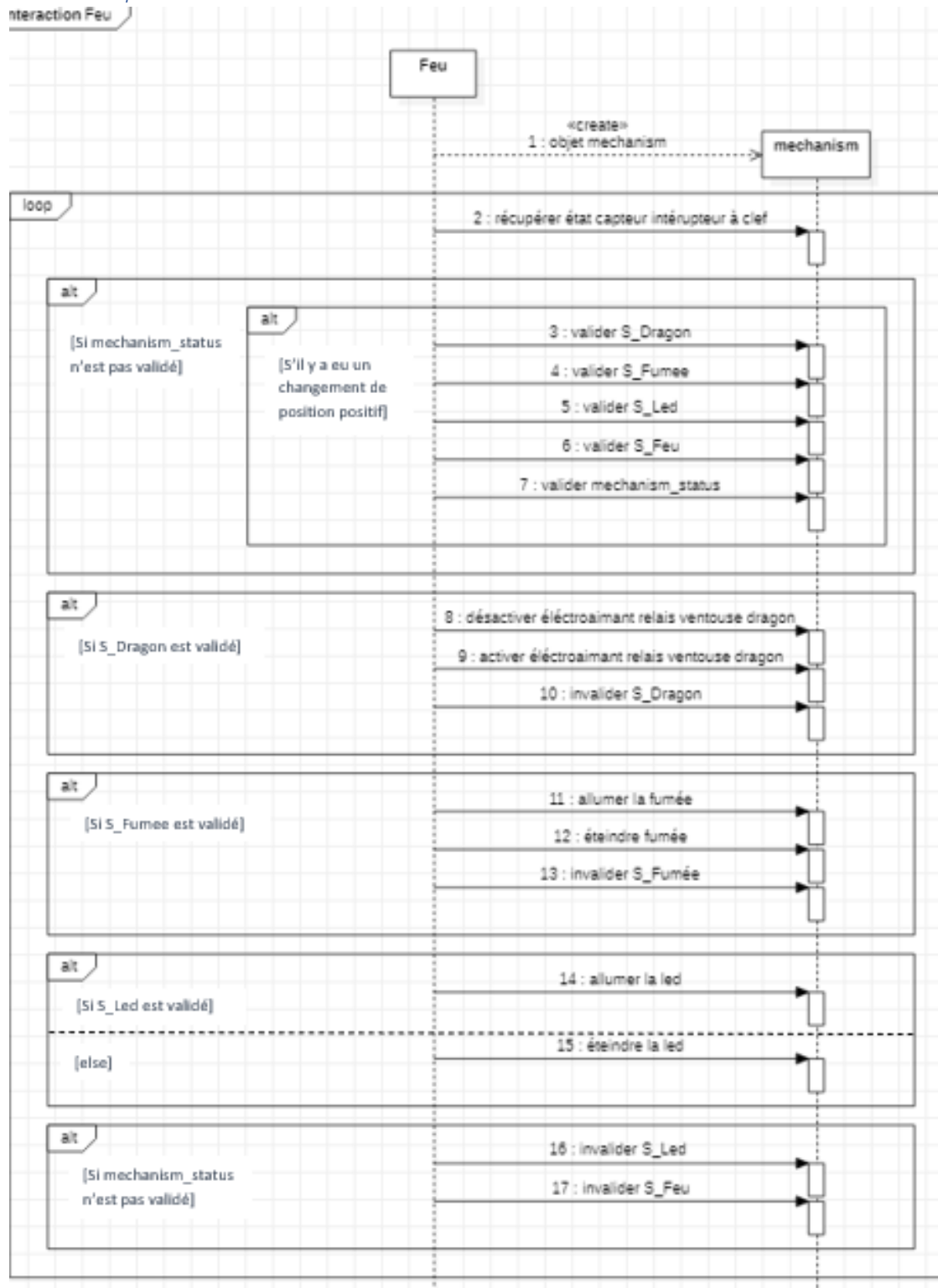


### Diagramme de classe





## Diagramme de séquence



## Mécanisme 5 : Eau

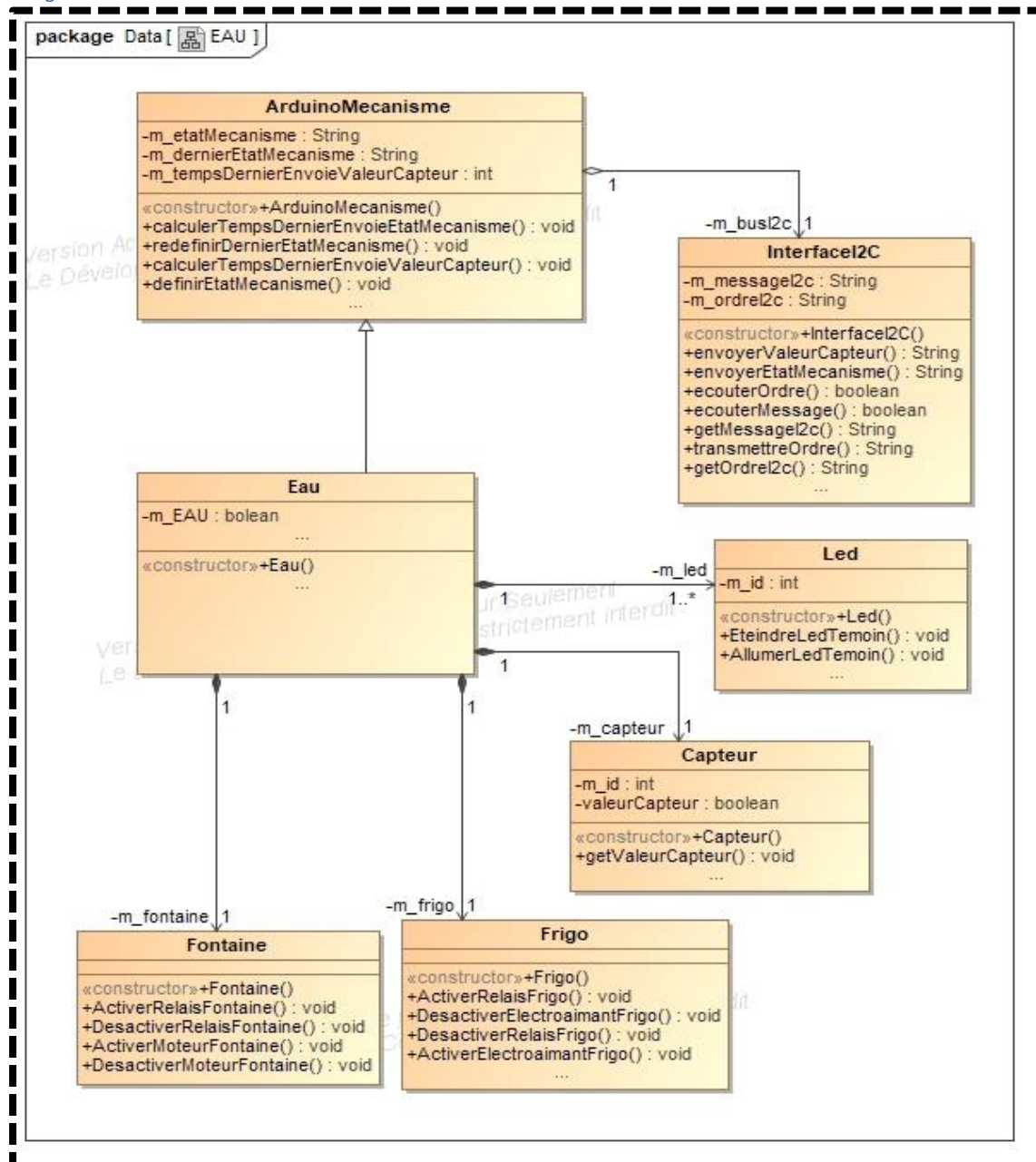
### Synopsis du mécanisme

Le mécanisme repose sur **un capteur d'humidité** et **un interrupteur à bascule**.

Selon la mesure du capteur d'humidité, (i) un électroaimant est activé ou désactivé via un relais (ii) un moteur d'une fontaine est activé ou désactivé via un relais (iii) une sortie est paramétrée en destination du mécanisme des quatre éléments.

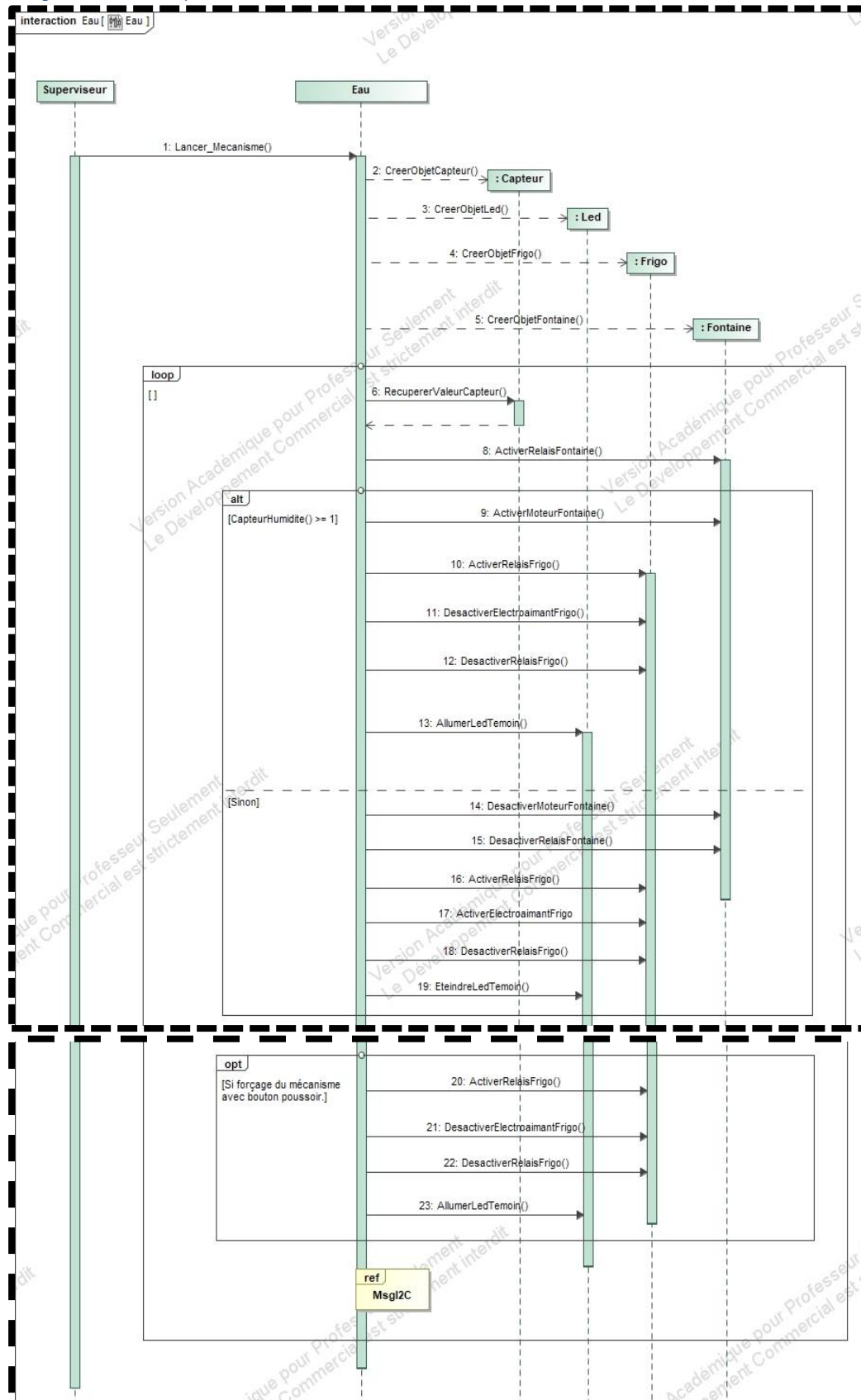
Selon l'état de l'interrupteur à bascule, un électroaimant est activé ou désactivé via un relais

### Diagramme de classe





## Diagramme de séquence







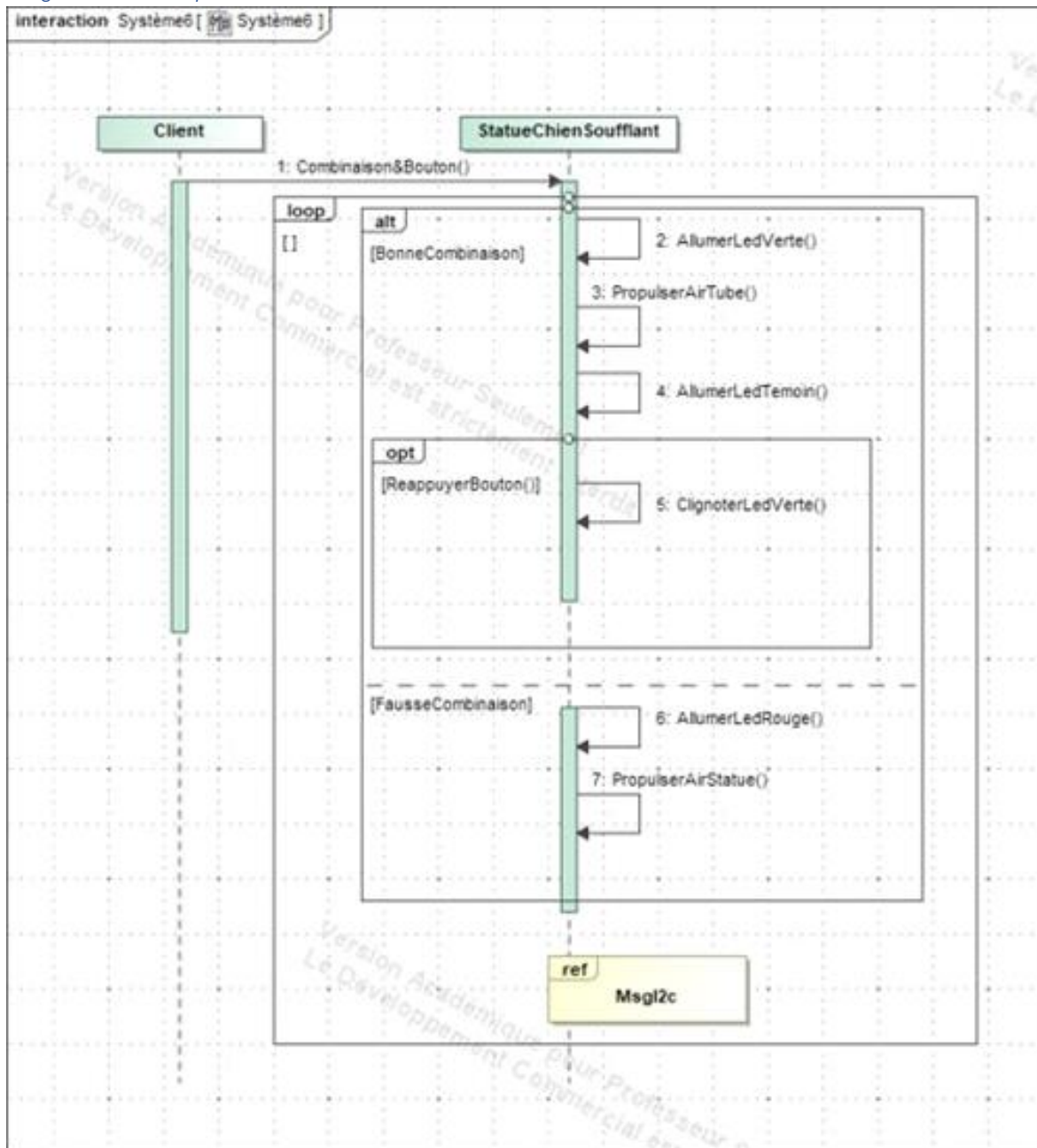
## Mécanisme 6 : Air

### Synopsis du mécanisme

Le mécanisme repose sur **sept capteurs photosensibles** et **un bouton poussoir**.

Lorsque le bouton est pressé, selon des conditions portant sur les valeurs de sept capteurs photosensibles, (i) deux électrovannes sont activées ou désactivées via un relais (ii) une led est allumée ou non (ii) une sortie est paramétrée en destination du mécanisme des quatre éléments.

### Diagramme de séquence





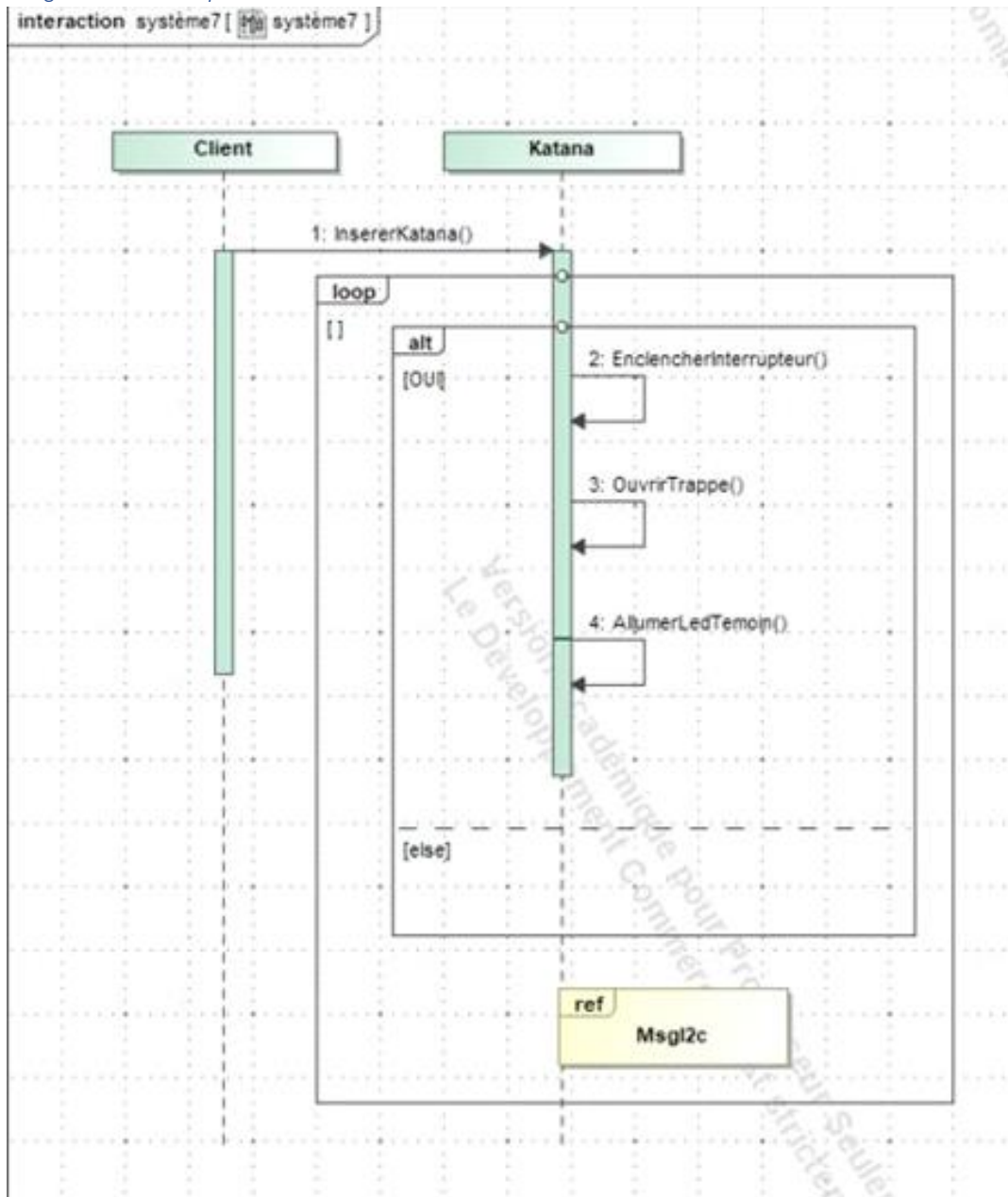
## Mécanisme 7 : Katana

### Synopsis du mécanisme

Le mécanisme repose sur **un interrupteur fin de course**.

Selon l'état de l'interrupteur fin de course, un solénoïde poussant est activé ou désactivé via un relais.

### Diagramme de séquence





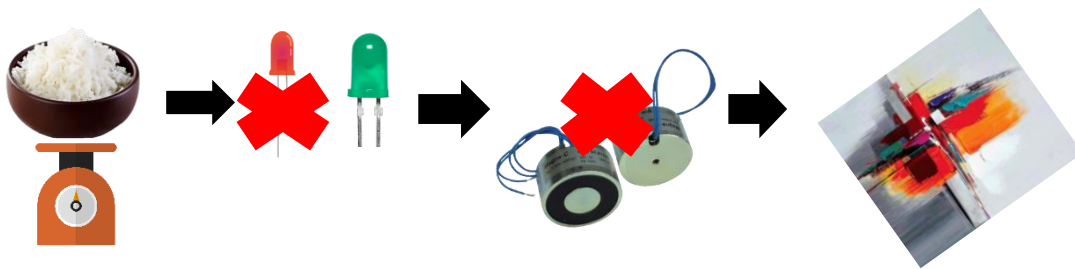
## Mécanisme 8 : Riz

### Synopsis du mécanisme

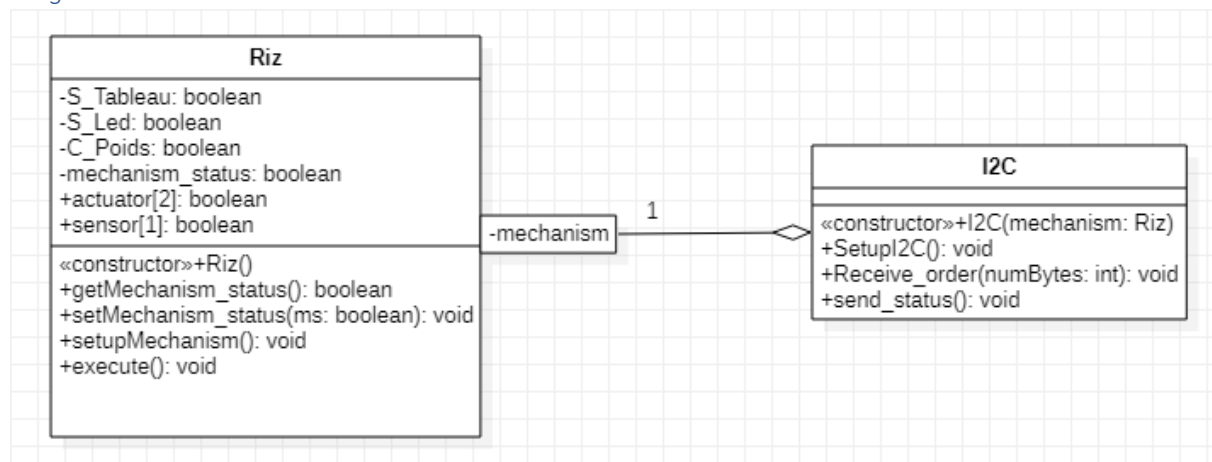
Le mécanisme 4 repose sur **un capteur de poids**.

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

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

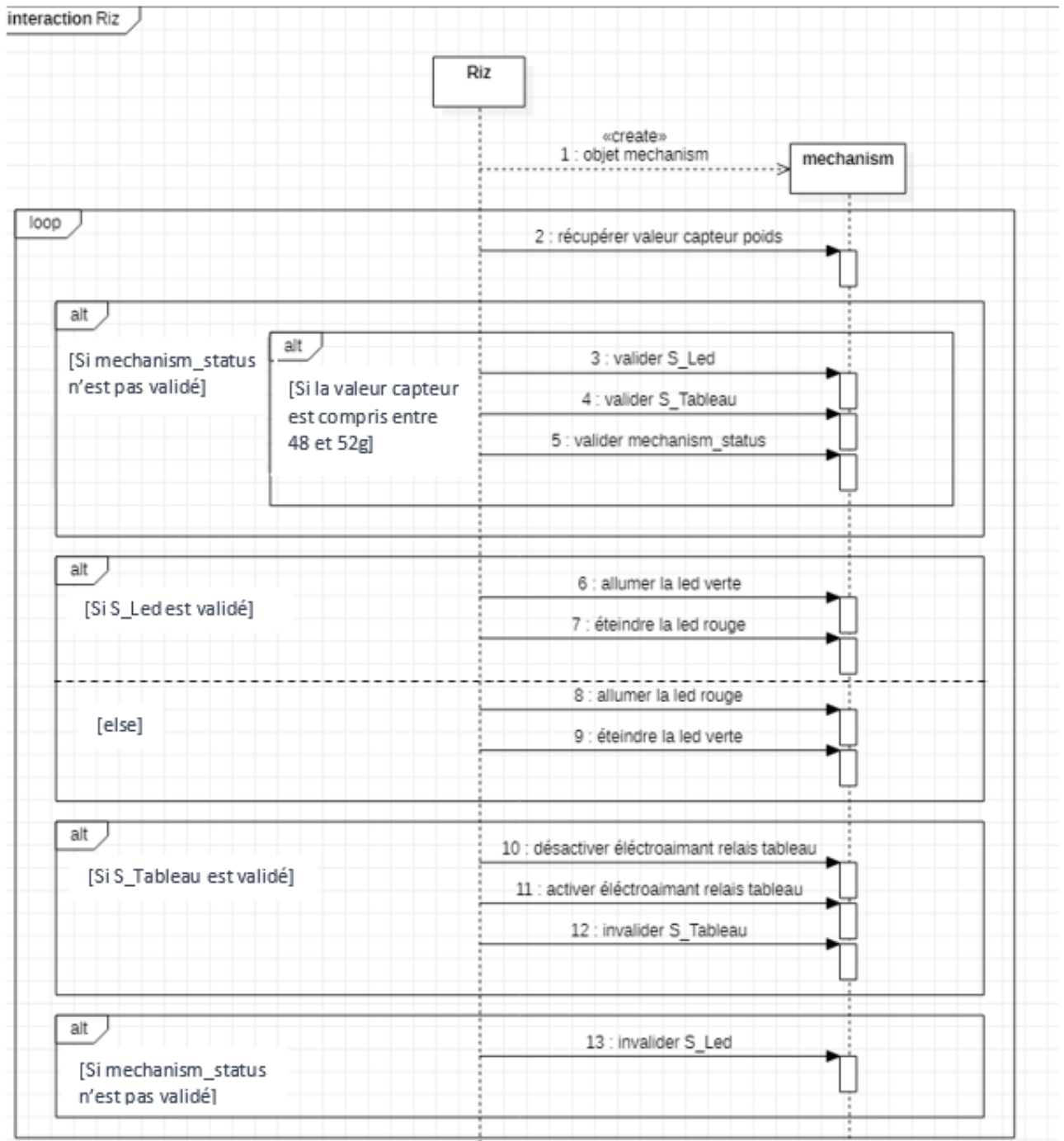


### Diagramme de classe





### Diagramme de séquence





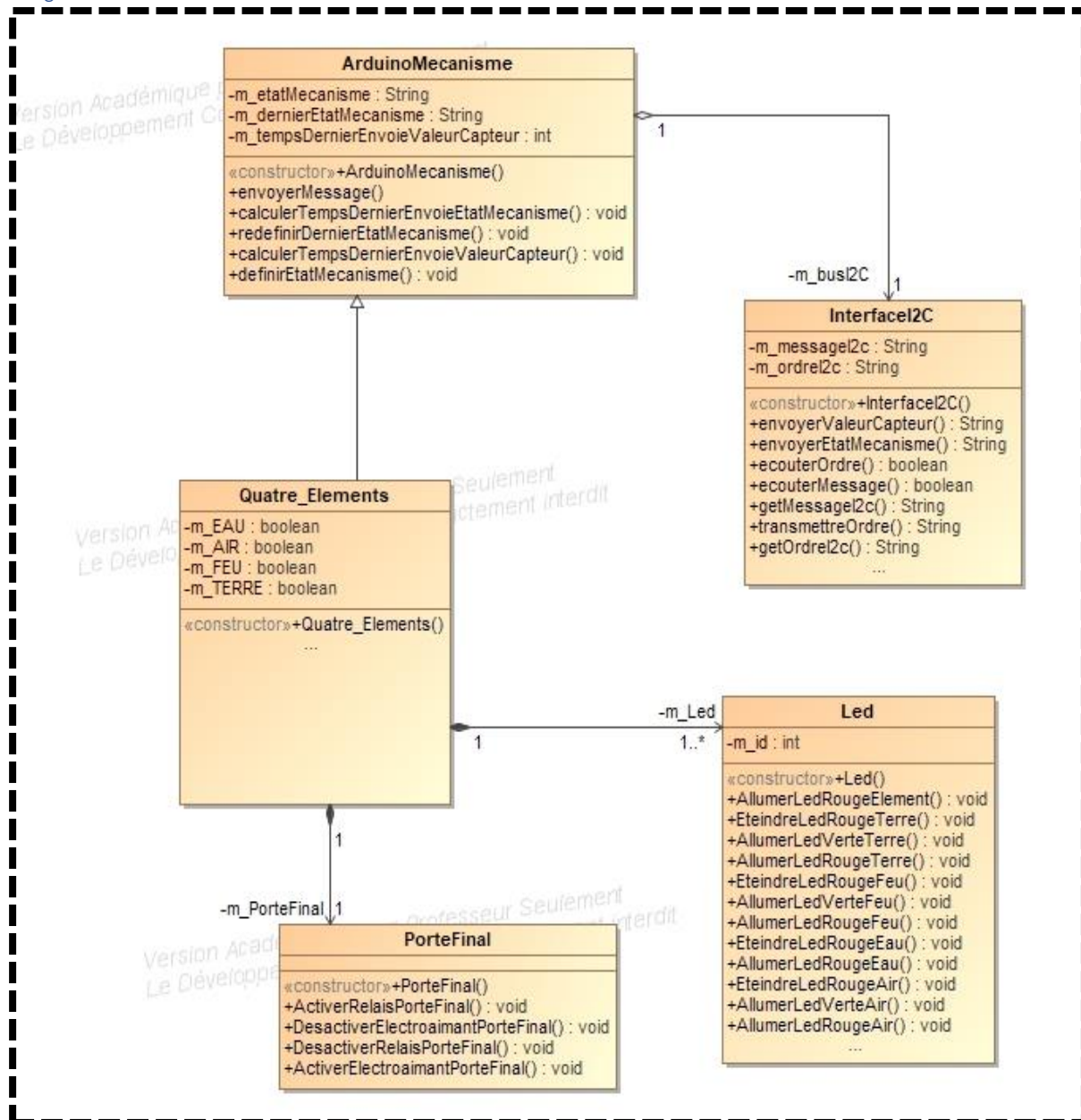
## Mécanisme 9 : Quatre Eléments

### Synopsis du mécanisme

Le mécanisme repose sur un bouton poussoir et les quatre sorties liées aux éléments (mécanismes 3, 4, 5 et 6).

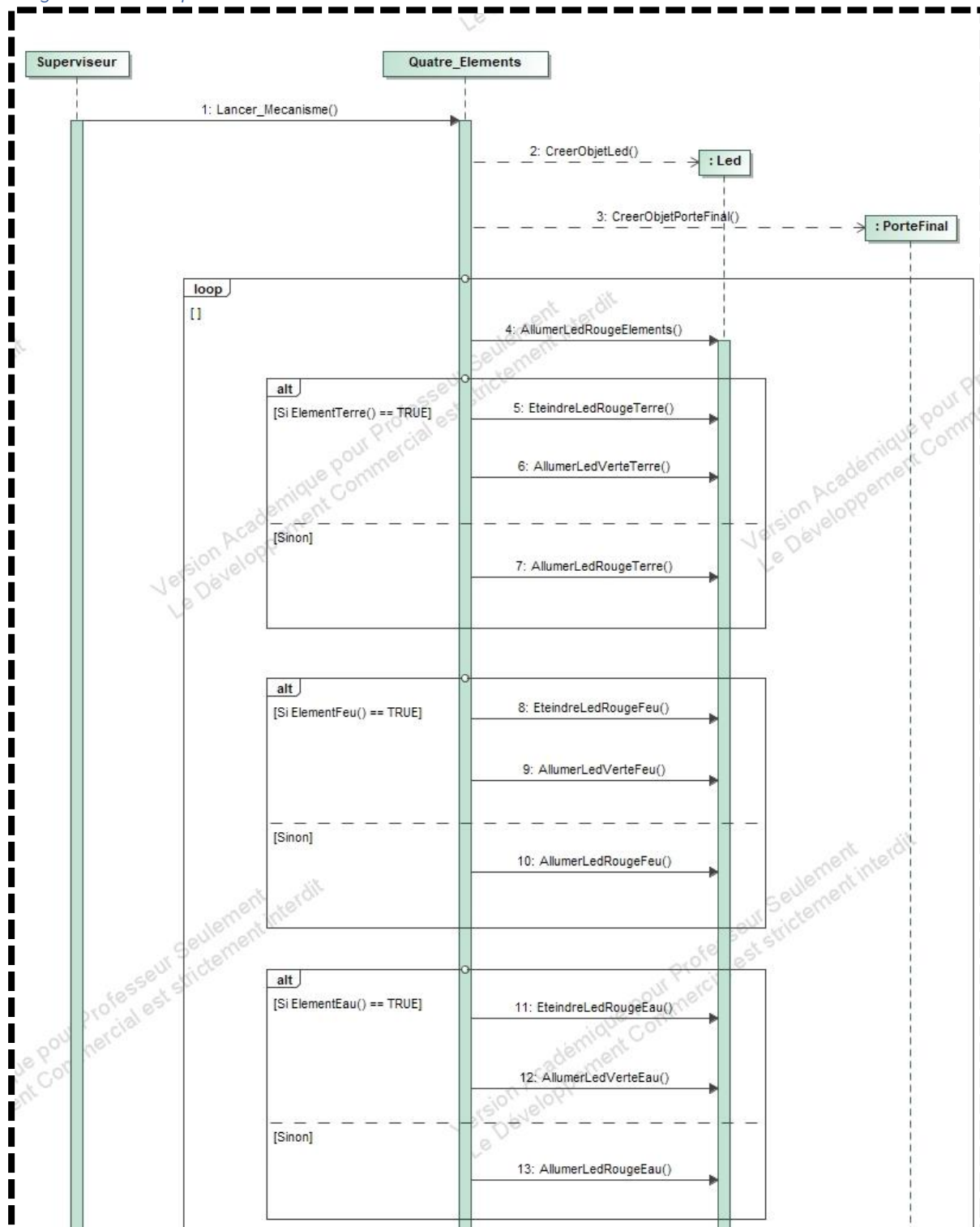
Selon les valeurs de ces quatre sorties, un électroaimant et une led sont activés ou désactivés.

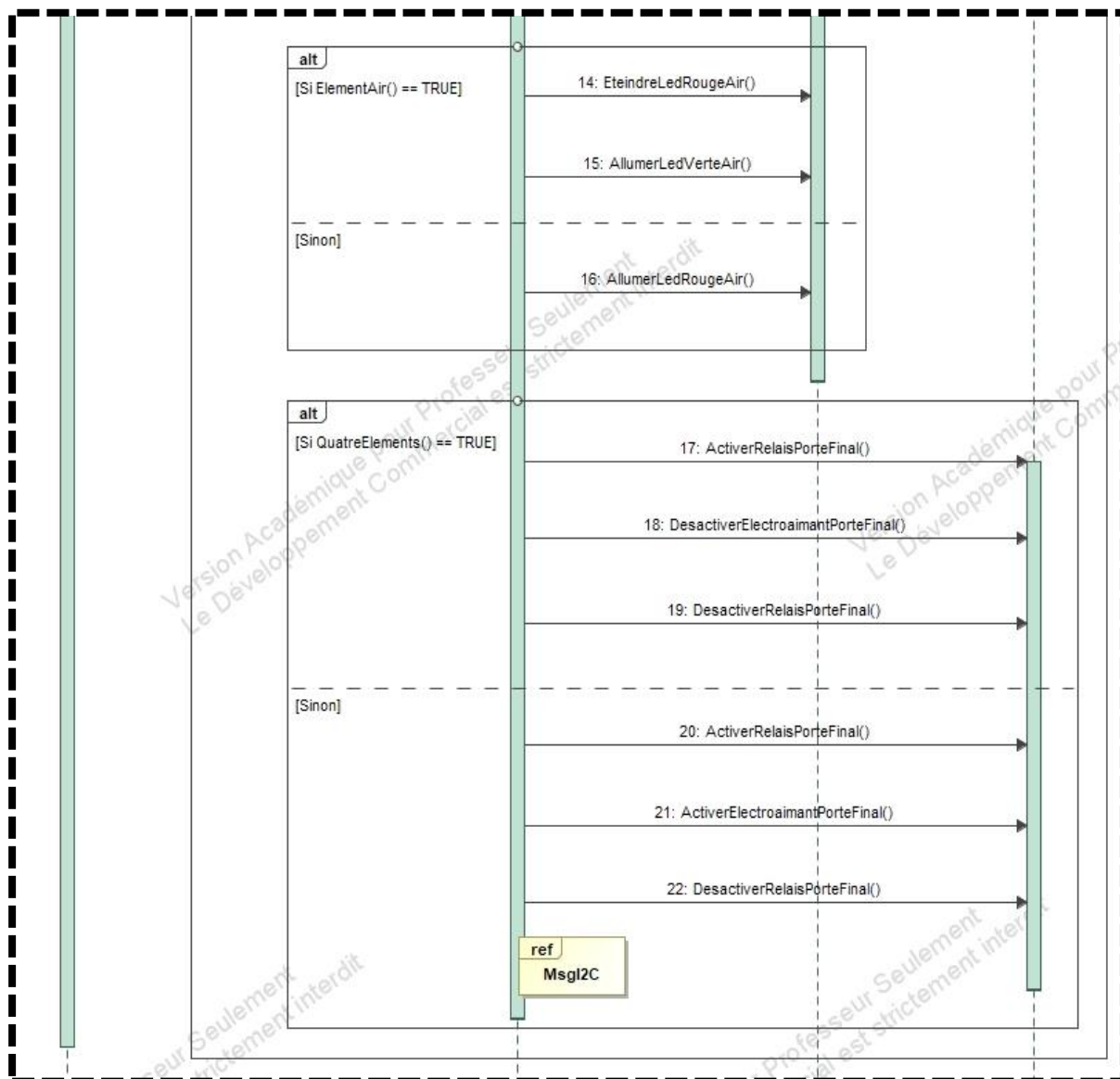
### Diagramme de classe





### Diagramme de séquence







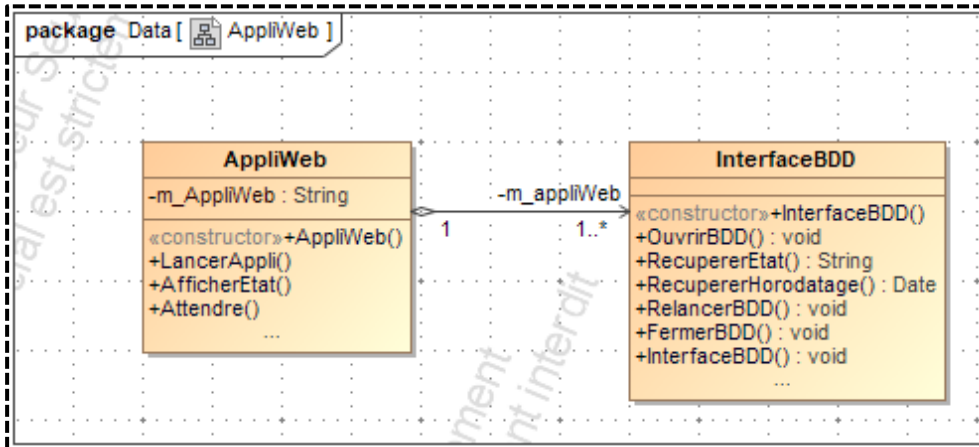


## B. Application Web de supervision

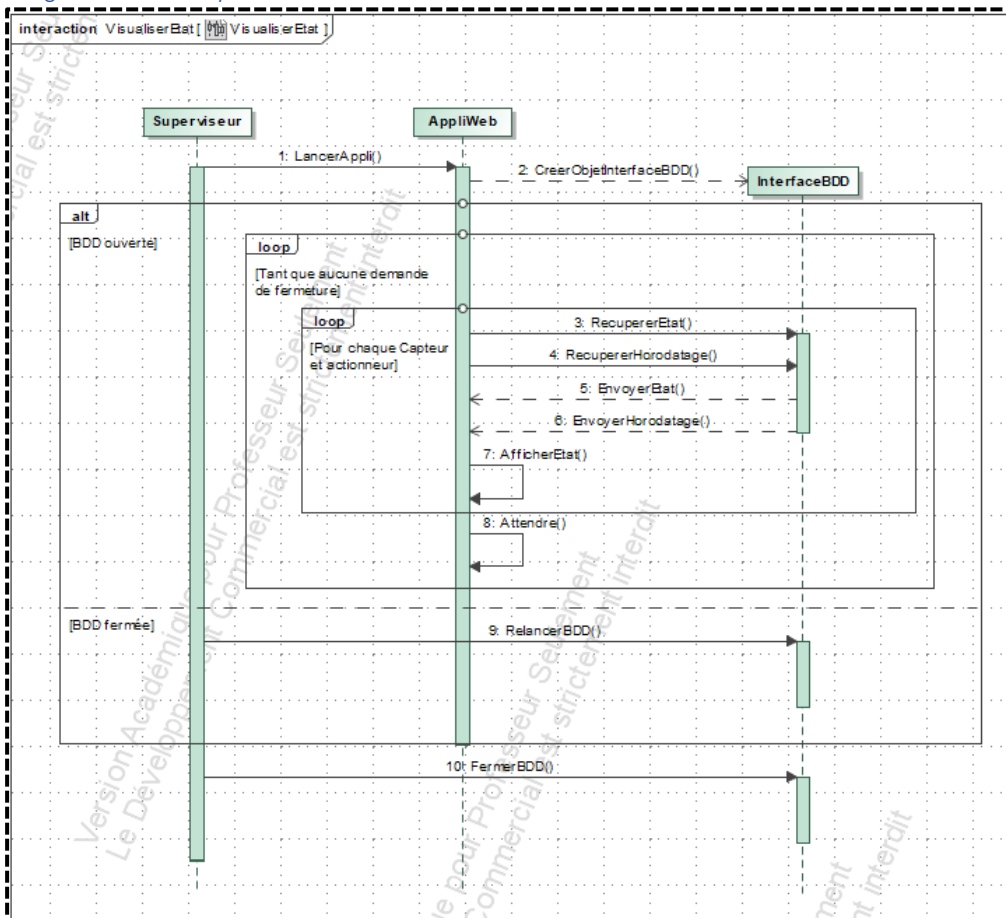
### Application Web : Visualiser l'état de la salle

#### Diagrammes

##### Diagramme de classe



##### Diagramme de séquence





## Développement

### *NetBeans*

**NetBeans** est un environnement de développement intégré (EDI), placé en *open source* par Sun en juin 2000 sous licence CDDL (Common Development and Distribution License) et GPLv2. En plus de Java, NetBeans permet la prise en charge native de divers langages tels le C, le C++, le JavaScript, le XML, le Groovy, le PHP et le HTML, ou d'autres (dont Python et Ruby) par l'ajout de *greffons*. Il offre toutes les facilités d'un IDE moderne (éditeur avec coloration syntaxique, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web).

Compilé en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java). Un environnement Java Development Kit JDK est requis pour les développements en Java.

NetBeans constitue par ailleurs une plateforme qui permet le développement d'applications spécifiques (bibliothèque Swing (Java)). L'IDE NetBeans s'appuie sur cette plateforme.

L'IDE NetBeans s'enrichit à l'aide de greffons.

### *MySQL*

**MySQL** est un système de gestion de bases de données relationnelles (SGBDR).

### *AJAX*

**Ajax** combine JavaScript et DOM, qui permettent de modifier l'information présentée dans le navigateur en respectant sa structure.

Ajax est une architecture informatique qui permet de construire des applications Web et des sites web dynamiques interactifs sur le poste client en se servant de différentes technologies ajoutées aux navigateurs web entre 1995 et 2005.

Ajax est l'acronyme d'**asynchronous JavaScript and XML** : *JavaScript et XML asynchrones*.

### *HTML/CSS*

Le HyperText Markup Language, généralement abrégé HTML ou dans sa dernière version HTML5, est le langage de balisage conçu pour représenter les pages web.

Le terme **CSS** est l'acronyme anglais de **Cascading Style Sheets** qui peut se traduire par "feuilles de style en cascade".

Le **CSS** est un langage **informatique** utilisé sur l'internet pour mettre en forme les fichiers HTML ou XML.

### *Bootstrap*

**Bootstrap** est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. C'est l'un des projets les plus populaires sur la plate-forme de gestion de développement GitHub.



### PHP

**PHP: Hypertext Preprocessor**<sup>7</sup>, plus connu sous son sigle **PHP** (sigle auto-référentiel), est un langage de programmation libre<sup>8</sup>, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP<sup>7</sup>, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet.

PHP a permis de créer un grand nombre de sites web célèbres, comme Facebook, Wikipédia, etc.<sup>9</sup> Il est considéré comme une des bases de la création de sites web dits dynamiques mais également des applications web.

### Apache HTTP Server

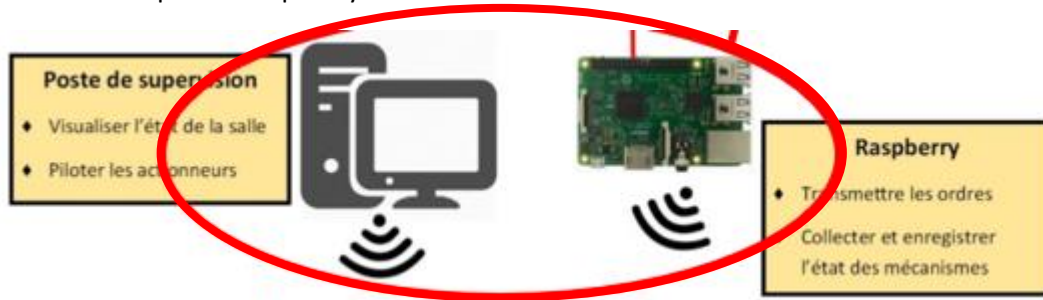
Le logiciel libre **Apache HTTP Server (Apache)** est un serveur HTTP créé et maintenu au sein de la fondation Apache. Jusqu'en avril 2019<sup>3</sup>, ce fut le serveur HTTP le plus populaire du World Wide Web. Il est distribué selon les termes de la licence Apache



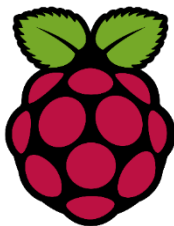
## Application Web : Piloter les actionneurs

### Synoptique

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 transmises par les mécanismes depuis la Raspberry.



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



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



### 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.

### Développement

Une application WEB doit être créée pour le poste de supervision de l'administrateur. Une interface pour pouvoir piloter les différents actionneurs doit être réalisée. 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.



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

### 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).

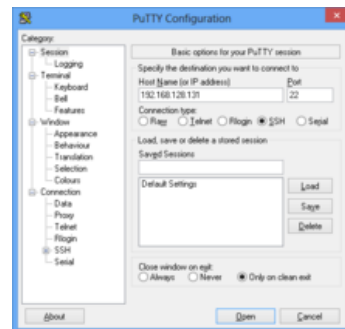
## PHP

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.

## Putty

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.



## Maquette de l'interface

L'échiquier

☐ S1

✓

✗

Le lion basculant

☐ S1

✓

✗

Terre

☐ S1 ☐ S\_TERRE

✓

✗

Feu

☐ S1 ☐ S2 ☐ S3 ☐ S4 ☐ S\_FEU

✓

✗

Eau

☐ S1 ☐ S\_EAU ☐ S2 ☐ S\_PURGE

✓

✗

Air

☐ S1 ☐ S2 ☐ S3 ☐ S\_AIR

✓

✗

Le katana

☐ S1

✓

✗

Le riz

☐ S1 ☐ S2

✓

✗

Les quatre éléments

☐ S1

✓

✗



## C. I2C : Communication entre les Arduinos et la Raspberry

### Synopsis de la liaison i2c

Pour lier les Arduino à la Raspberry nous allons passer par le protocole i2C :

- pour envoyer l'état des actionneurs et les valeurs des capteurs des Arduino à la Raspberry
- pour envoyer les ordres de pilotage du PC de supervision de la Raspberry aux Arduino.

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

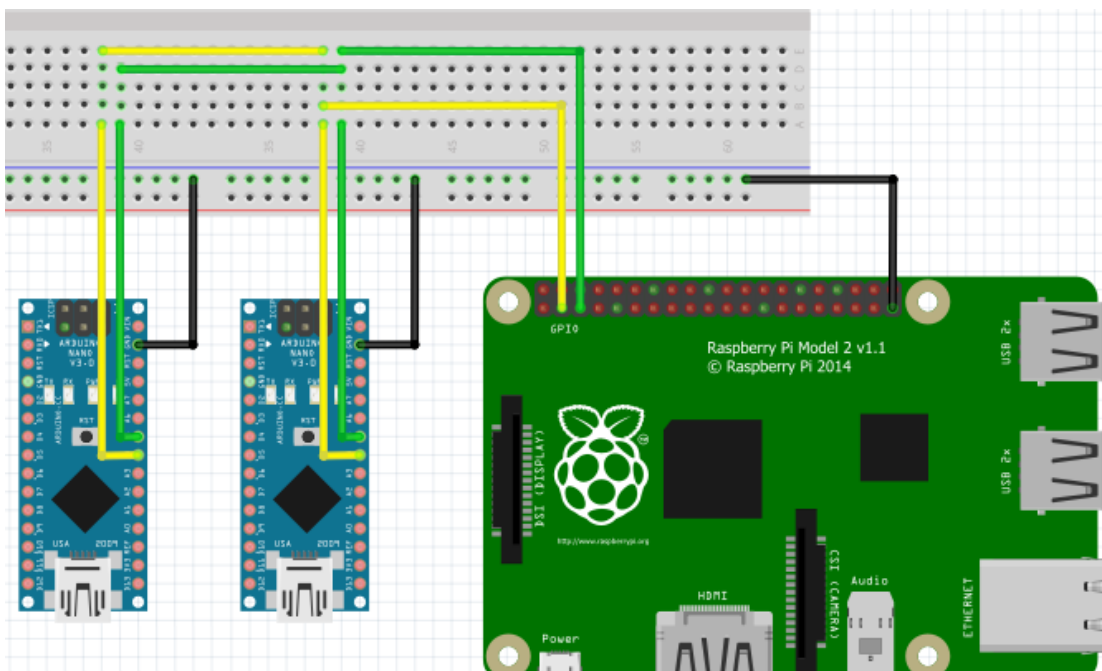
Les données sont transmises en série à 100Kbits/s en mode standard et jusqu'à 400Kbits/s en mode rapide.

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

Il y a un principe de maître et esclave. Ici nous utiliseront 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.

### Schéma Câblage liaison i2c

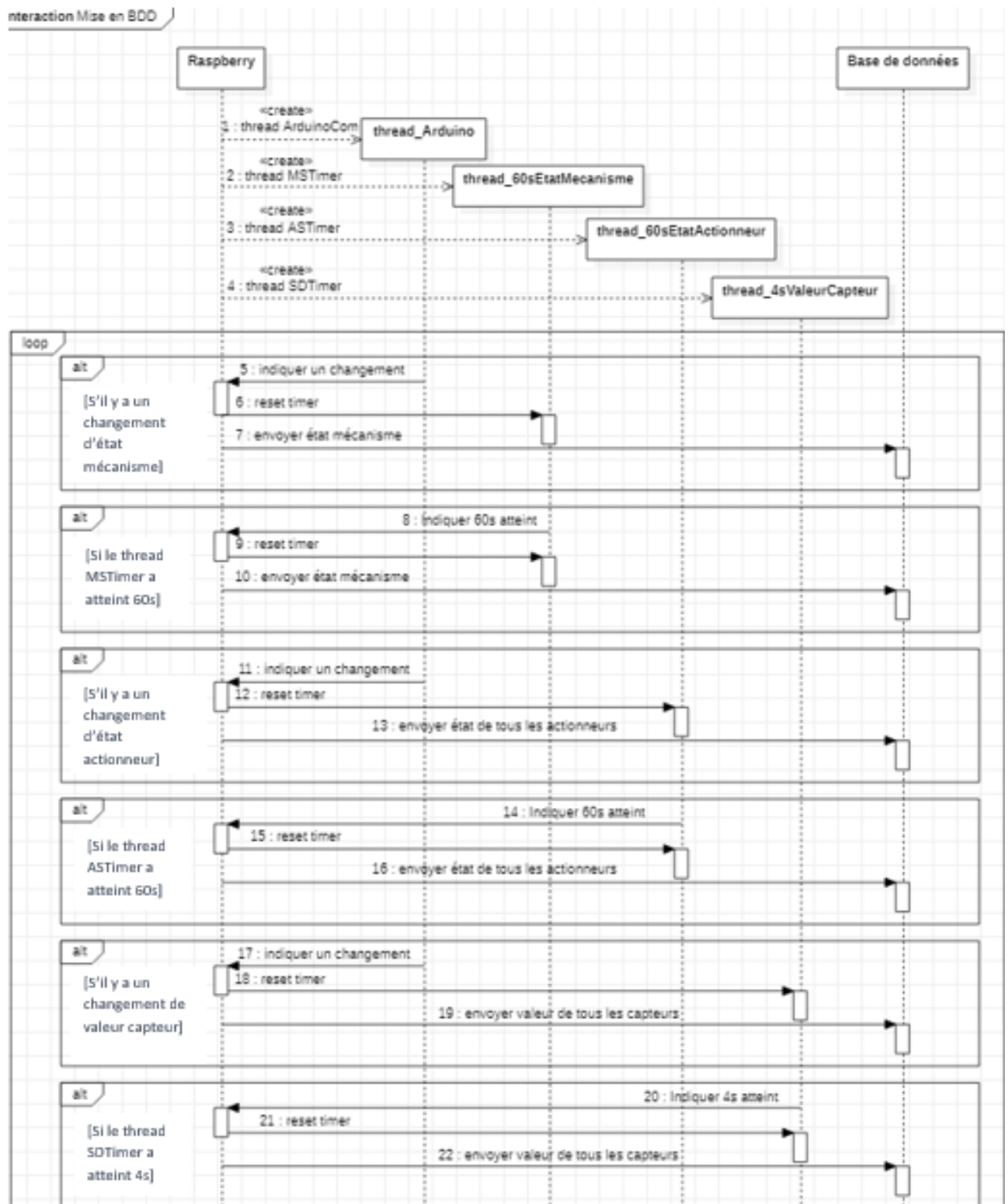






## D. La Base de Données

### Diagramme de séquence





### III. Réalisation du projet

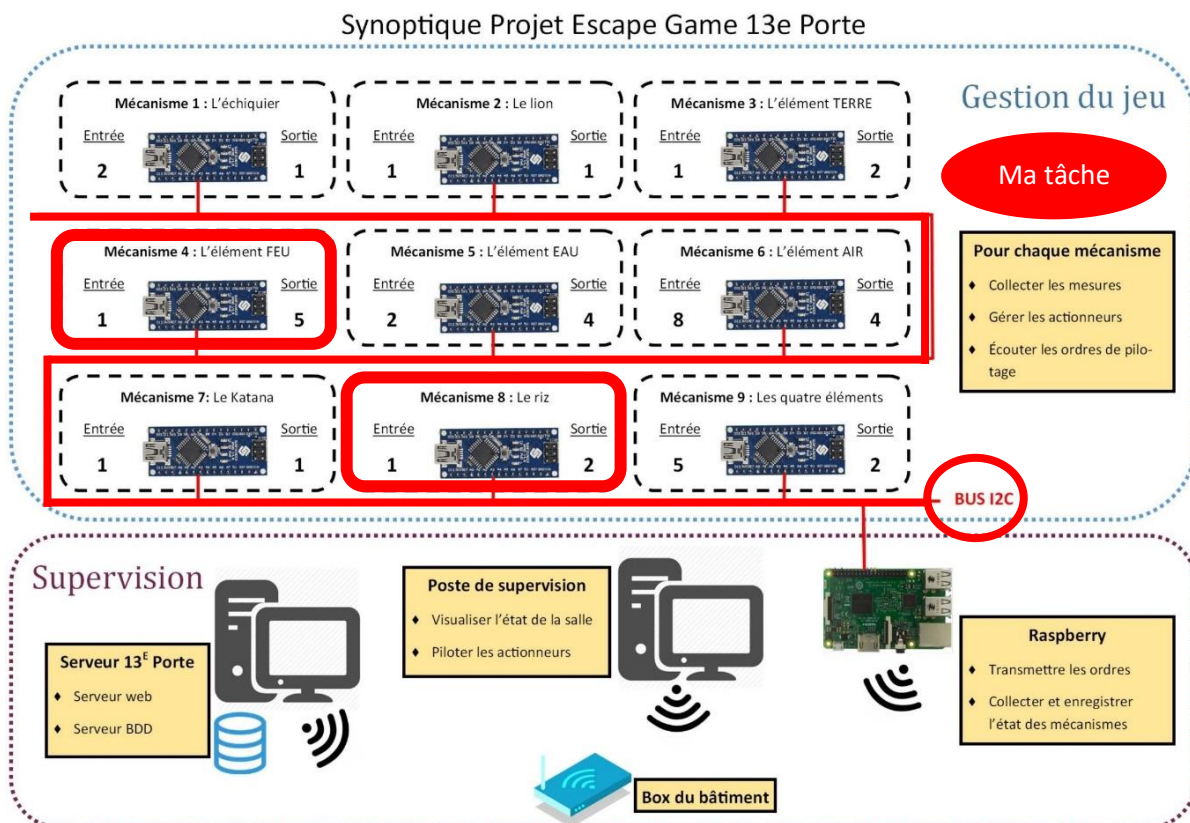
#### A. Etudiant 1 : Constantin

##### Rappel de la tâche de l'étudiant

##### Synoptique

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

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





## Mécanisme 4 : Feu

### Le main

Le programme exécute principalement les instructions suivantes :

```
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 attends 0.1 seconde
    mechanism.execute(); //On exécute le mécanisme
}
```

### 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 controle
bool S_Feu; //actionneur de l'élément FEU sur la tablette des 4 éléments
bool C_Interupteur; //état de la position détecter par l'interrupteur à clef
bool mechanism_status; //indique si le mécanisme est activé ou non
```

*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
```

### La méthode execute

*Synopsis*

La méthode « **exécute** » est la méthode qui fait fonctionner le mécanisme. Elle est appelée dans la fonction loop du programme et donc exécuté 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 interrupteur à clef
```



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
}
```

## Mécanisme 8 : Riz

### 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 attends 0.1 seconde
    mechanism.execute();                                       //On exécute le mécanisme
}
```

### 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 controle
int C_Poids;                                                  //valeur mesuré par le capteur de poids
bool mechanism_status;                                         //indique si le mécanisme est activé ou non
```

*Principales méthodes*

Les principales méthodes sont les suivantes :

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



## La méthode execute

### *Synopsis*

La méthode « **exécute** » est la méthode qui fait fonctionner le mécanisme. Elle est appelée dans la fonction loop du programme et donc exécuté 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
```

### *Valider le mécanisme*

Pour valider le mécanisme on exécute les instructions suivantes :

```
if ( (sd_reading >= 0.48)
    && (sd_reading <= 0.52 )
    && mechanism_status == false ) {                //Si la mesure est comprise entre 0.48 et 0.52 pour la lere fois
    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
}
```



## Liaison 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

#### Les dictionnaires

Il a fallu créer une liste de 9 dictionnaires correspondant aux 9 Arduinos. Les clés de ces dictionnaires sont identiques. Voici un fragment de la liste de dictionnaires :

```
arduinios = [
    {'id': 1, 'address': 0x12,
     'mechanism_status': False, 'ms_noTimer': 0,
     'actuator_status' : {'S_Echiquier': False}, 'as_noTimer' : 0,
     'sensor_data' : {'C_EffetHall1': 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},

    (...) // 7 autres dictionnaires
]
```

Chaque dictionnaire Arduino possède les clés suivantes :

- « **id** » : correspond au numéro du mécanisme assigné à l'Arduino
- « **address** » : correspond à l'adresse esclave assigné à l'Arduino pour l'i2c
- « **mechanism\_status** » : correspond à l'état du mécanisme (valeur True/False)
- « **ms\_noTimer** » : correspond à une valeur entière  
Nécessaire à l'exécution du thread MSTimer pour l'envoi 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.

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



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

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

```
while self.running:

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

        send_order(self.arduino)                #On envoie le message d'ordre a l'Arduino s'il y en a un
        message = get_message(self.arduino)      #On recupere le message i2c venant de l'Arduino
        get_mechanism_status(message, self.arduino) #On verifie l'etat du mecanimse
        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                    #Si il y a une erreur quelconque dans l'execution du thread
        thread = ArduinoCom(self.arduino)       #On arrete le thread
        thread.start()                          #On crée un nouveau thread
        thread.start()                          #On lance le nouveau thread
```

*Envoyer un ordre à l'Arduino*

On appelle la fonction `send_order()`.

Elle exécute principalement les instructions suivantes :

```
if arduino['order'] != 0 :                                #On verifie si l'Arduino a reçu un order

    order = convertStrToListHex(str(arduino['order']))    #On convertit le message socket en Hexadecimal
    bus.write_i2c_block_data(arduino['address'], 0, order) #On envoie le message socket a l'Arduino assignee

    print("Mechanism %s : ORDER SENT : %s" %(arduino['id'],arduino['order']))
    arduino['order'] = 0                                  #On reset l'ordre
```

*Récupérer le message i2c depuis l'Arduino*

On appelle la fonction `get_message()`.

Elle exécute les instructions suivantes :

```
answer = bus.read_i2c_block_data(arduino['address'],0x32) #On recupere 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
```





### 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
    thread = MStimer(arduino)
    thread.start()
    #Pour reset le timer
    #On incremente la valeur "_noTimer" dans le dictionnaire
    #On cree un nouveau thread
    #On lance le nouveau thread

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

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 = self.arduino['ms_noTimer']      #On definit le numero du thread

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

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



### 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'] :

    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)
```

#La partie capteur du message commence au 6eme caractere  
#Pour chaque actionneur du mecanisme

#Si l'actionneur selectionne vient d'etre valide  
#On change la valeur de l'actionneur dans le dictionnaire  
#On reset le timer

#Si l'actionneur selectionne vient d'etre invalide  
#On change la valeur de l'actionneur dans le dictionnaire  
#On reset le timer

#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

De même que pour l'état d'un mécanisme, d'après le diagramme d'exigence s'il n'y a pas de changement d'état pendant 60 secondes il faut envoyer l'état en BDD. Pour ce faire il est judicieux de faire fonctionner un timer de 60s en parallèle. Il a donc fallu utiliser une nouvelle fois des threads.

Ainsi chaque Arduino possède son propre thread qui correspond au timer de l'état des actionneurs. Ce thread est appelé « **ASTimer** »

Le thread `ASTimer` exécute principalement les instructions suivantes :

```
no_thread = self.arduino['as_noTimer']

while self.running:
    time.sleep(60)

    if self.arduino['as_noTimer'] != no_thread :
        self.running = False
    else :
        send_AStoDataBase(self.arduino, console_message)
```

#On definit le numero du thread

#On attend 60 secondes

#Si le thread n'est plus le timer actuel  
#On arrete le thread  
#Sinon  
#On envoie l'etat des actionneurs en BDD



### 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'] :

    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)
```

#La partie capteur du message commence au 12eme caractere  
#Pour chaque capteur du mecanisme

#Si le capteur selectionne vient d'etre valide  
#On change la valeur du capteur dans le dictionnaire  
#On reset le timer

#Si le capteur selectionne vient d'etre invalide  
#On change la valeur du capteur dans le dictionnaire  
#On reset le timer

#Si le capteur possede une valeur numerique  
#On recupere la valeur numerique  
#Si la valeur numerique vient de changer  
#On change la valeur du capteur dans le dictionnaire  
#On reset le timer

#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

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 = self.arduino['sd_noTimer']

while self.running:
    time.sleep(4)

    if self.arduino['sd_noTimer'] != no_thread :
        self.running = False
    else :
        send_SDtoDataBase(self.arduino, console_message)
```

#On definit le numero du thread

#On attend 4 secondes

#Si le thread n'est plus le timer actuel  
#On arrete le thread  
#Sinon  
#On envoie la valeur des capteurs en BDD



## Partie I2C pour Arduino

### Synoptique

La partie i2c du programme de chaque Arduino consiste à :

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

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

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

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

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

```
#include "Wire.h"
```

### Envoyer les informations du mécanisme à la Raspberry

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

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

3eme caractère                      6eme – 9eme caractère                      12eme caractère et +

**Le 3<sup>ème</sup> 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<sup>ème</sup> - 9<sup>ème</sup> 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<sup>ème</sup> caractère et +** : correspond à la valeur / état des capteurs du mécanisme

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



Pour envoyer le message i2c on exécute les instructions suivantes :

```
void send_status() {
    Wire.write(getMessagei2c()); //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 utilisé par l'Arduino
    Wire.onRequest(send_status); //On envoie le messagei2c sur le bus i2c
}
```

### Exécuter les ordres reçus depuis la Raspberry

Pour exécuter les ordres envoyés depuis la Raspberry et à postériori 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 reçues
    }

    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 utilisé par l'Arduino
    Wire.onReceive(receive_order); //On récupère le message s'ordre reçu sur le bus i2c via la fonction receive_order
}
```

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

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

1er caractère    2eme caractère    3eme caractère et +

**Le 1<sup>er</sup> caractère** : correspond au numéro du mécanisme

**Le 2<sup>ème</sup> 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<sup>ème</sup> 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**



## B. Etudiant 2 : Corentin

### Création de la base de données

Table mécanisme

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

Table actionneurs

	ID_actionneurs	ID_mecanismes	type	Etat	Valeur_numerique	Heure_derniere_mesure
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	1	1	led rouge	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	2	2	led rouge	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	3	3	led rouge	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	4	3	led verte	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	5	4	led rouge	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	6	4	led verte	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	7	5	led verte	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	8	6	led rouge	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	9	6	led verte	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	10	7	led rouge	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	11	8	led rouge	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	12	9	led verte	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	13	9	led verte	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	14	9	led verte	TRUE	0	2020-03-24 20:51:52
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	15	9	led verte	TRUE	0	2020-03-24 20:51:52





Table capteurs

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

### 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

### 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.



## C. Etudiant 3 : Joshua

### Rappel de la tâche 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.

### Mécanisme 1 : Echiquier

#### Programmation du mécanisme

Un PIN est une entrée ou une sortie de la carte Arduino.

On commence par renseigner les PIN sur lesquelles sont installés nos composants.

```
#define SLAVE_ADDRESS 0x12
#define C_EffetHall1_PIN A0 //Capteur à effet Hall 1
#define C_EffetHall2_PIN 2 //Capteur à effet Hall 2
#define S_LedR_PIN 3 //Led de sortie
```

Définition des PIN





Le premier « #define » va être utile à l'étudiant n°1 pour expliquer le modèle « maître-esclave ». Le second et le troisième seront les PIN de nos deux capteurs à effet hall. Le dernier sera le PIN de la LED rouge qui sera représentée sur le tableau de contrôle.

La programmation est effectuée sur le logiciel Arduino. Celui-ci peut contenir du langage C ou C++. On va donc coder avec un fichier Echec.h et un fichier Echec.cpp.

```
class Echec{
private :
    const int C_EffetHall1;
    const int C_EffetHall2;
    bool S_Echiquier;
    bool mecanism_status;

public :
    bool actuator[1] = {S_Echiquier};
    int sensor[2] = {C_EffetHall2, C_EffetHall2};
    bool getMechanism_status();
    void setMechanism_status(bool ms);

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

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.

```
if ((pin1 == LOW) && (pin2 == LOW))    // Si les deux pieces sont à la bonne place
{
    ECHECstate = LOW; // On valide l'enigme
} else LEDstate = HIGH;
```

La condition qui va vérifier que les pièces sont en place

Cette condition va vérifier, à l'aide des deux capteurs à Effet Hall qui sont représentés par « pin1 » et « pin2 », que les deux pièces sont bien en place sur les bonnes cases afin de valider le mécanisme.



## Mécanisme 2 : Lion

### Programmation du mécanisme

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 :
    const int C_EffetHall1;
    bool S_Lion;
    bool mecanism_status;

private :
    bool actuator[1] = {S_Lion};
    bool sensor[1] = {C_EffetHall1};
    bool getMechanism_status();
    void setMechanism_status(bool ms);

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

Le fichier LionB.h

Ceci est donc le fichier LionB.h, 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.

```
if (readingLIONB == HIGH)
{
    LIONBstate = LOW; //Ce qui valide la réussite de l'énigme
    digitalWrite(3, LIONBstate); //reset pour defaire le solenoide
    timeLIONB = millis();
    LedLIONB = HIGH; //La led correspondante sur le panneau du superviseur est allumée
}
```

La condition qui va valider l'énigme

Ceci est une partie de la fonction « `execute()` » du fichier Terre.cpp. On y voit la condition qui définit l'action de l'électroaimant du doigt de cinéma. Lorsque l'électroaimant est sur le capteur à effet hall pendant un temps donné par la première condition, le mécanisme se valide.



## Mécanisme 3 : Terre

### Programmation du mécanisme

Comme pour les deux premiers mécanismes, on code avec un fichier Terre.h et un fichier Terre.cpp. Voici le fichier Terre.h :

```
class Terre {

    private :
        const int C_EffetHall_1;
        bool S_LedR;
        bool S_LedV;
        bool mecanism_status;

    private :
        bool actuator[2] = {S_LedR, S_LedV};
        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();

};
```

Terre.h

Ceci est donc le fichier Terre.h, on retrouve un seul capteur à effet hall. Cette fois-ci, il y a une LED verte, représentée par « S\_LedV » qui va représenter la LED qui se trouve sur la tablette des quatre éléments. On y retrouve les différentes fonctions qui sont présentes dans le code des autres mécanismes.

```
if ((millis() - timeDOIGT) > (debounce / 3)) // attendre que l'on ai dépassé le temps de déparasitage
{
    if ((pindoigt == LOW))
    {
        digitalWrite(4, HIGH); //LED CONTROLE doigt
        DOIGTstate = LOW;
    }
}
if (pindoigt == HIGH)
{
    DOIGTstate = HIGH;
}
```

Partie de la fonction « execute() »

Ceci est une partie de la fonction « execute() » du fichier Terre.cpp. On y voit la condition qui définit l'action de l'électroaimant du doigt de cinéma. Lorsque l'électroaimant est sur le capteur à effet hall pendant un temps donné par la première condition, le mécanisme se valide.





## 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());
}

?>
```

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>
```

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>
```

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

```
<div class="carré1.1">

</div>
<?php
```

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;
}
```

## Code PHP



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

Cette portion de code nous permet de récupérer les informations de la table « **capteurs** » dans notre base de données. Le 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);
```

Pour utiliser les données récupérées précédemment, il faut faire une boucle et utiliser la fonction « **fetch()** ». Cette fonction récupère une la 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
    }
}
```

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 :

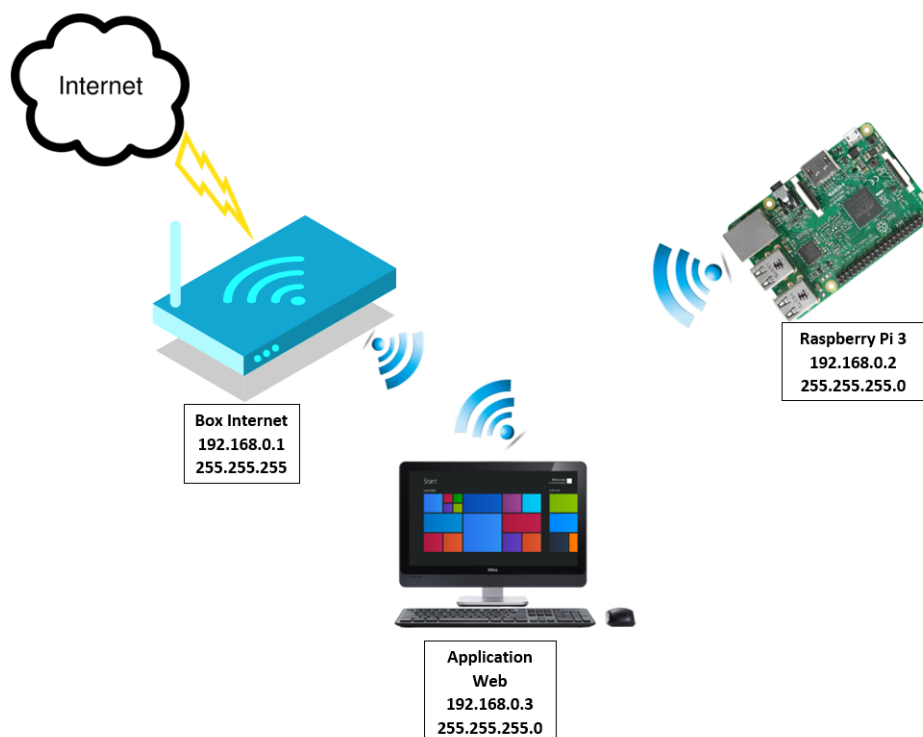




Si l'état du capteur est à 1, c'est sous cette forme que l'image va apparaître, si l'état est à 0, ce sera alors la même image mais sur un fond rouge. Cela nous permet aisément de différencier l'état des différents capteurs.

## Réseau

### Schéma réseau



### Tutoriel de connexion

Un tutoriel a été fourni à la société **13<sup>ème</sup> 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.





## D. Etudiant 4 : Thomas

### Mécanisme 5 : Eau

#### Description du Sous-système

Durant l'étape « Elément EAU » de l'escape game, les joueurs doivent verser de l'eau dans une tasse trouée. 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 (Solénoïde 12 volts) se met en marche afin d'ouvrir la porte d'un frigo suivant la séquence suivante :

```
digitalWrite(3, LOW); //RELAIS H2O FRIGO
delay(1000);
digitalWrite(3, HIGH);
delay(1000);
digitalWrite(3, LOW); //RELAIS H2O FRIGO
delay(1000);
digitalWrite(3, HIGH);
delay(1000);
digitalWrite(3, LOW);
delay(1000);
digitalWrite(3, HIGH);
delay(5000);
digitalWrite(3, LOW);
delay(20000);
```

L'activation de la gâche électrique via cette séquence permet aux joueurs d'entendre la 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.

#### Capteur d'humidité (Water Sensor)

Le capteur d'eau est un capteur assez sensible qui renvoie une valeur numérique à l'Arduino. Quand le système est mis en route, ce capteur renvoie une première valeur à l'Arduino. Au niveau du code, cette valeur est stockée dans une variable. Enfin les Sortie S1, S2, 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).



### Programmation du mécanisme

Pour programmés nos différents mécanismes, la société 13 ème Porte nous avais fournis leurs codes de base, pour améliorer un maximum leurs codes. Leurs programmes étaient très brouillon et désorganisée.

Nous nous sommes tout de suite renseigné sur le logiciel Arduino et son langage de programmation. Ainsi, nous avons appris que le langage Arduino était un langage qui se rapprochait beaucoup du C et du C++ : nous en avons conclut qu'il fallait réorganisé tous ces programmes en classes et fonctions.

Tout d'abord on renseigne sur quelle PIN se trouvent les différents relais et capteurs :

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

Ensuite, nous déclarons la classe Eau.

*Déclaration de la classe Eau :*

```
class Eau{
    private :
        bool S_Frigo;
        bool S_Fontaine;
        bool S_Led;
        bool S_Eau;
        bool C_Humidite;
        bool mecanism_status;

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

Comme vous pouvez le voir ci-dessus, les attributs privés de la classe Eau sont tous des Boolean, afin de simplifier l'envoi des états des différents capteurs et actionneurs (true : **Activer** ; false : **Désactiver**). En seconde partie, les fonctions de la classe Eau en public :

- **Eau()** : Constructeur par défaut.
- **void setupMecanism()** : La fonction setup, qui informe au logiciel Arduino quelles capteurs ou actionneurs est en sortie ou en entrée de l'arduino. Pour ce mécanisme, seul le capteur d'humidité est en entrée de l'arduino.
- **void execute()** : La fonction où le mécanisme s'exécute.



*Fonction void execute() :*

Tout d'abord, il faut activer le capteur et lire sa première valeur (ValeurCapteur est initialisé à 0) :

```
ValeurCapteur = analogRead (CHumidite_PIN); //Lecture de la valeur du capteur
```

On stocke ensuite cette valeur dans une variable « ValeurCapteurInitiale ». Ensuite, on utilise une deuxième valeur du capteur pour pouvoir faire la différence entre la première valeur : ce qui nous donnera la valeur exacte du capteur que l'on stockera dans **C\_Humidite**.

Pour finir, si **C\_Humidite** est supérieur ou égale à 180, alors le mécanisme se lance

```
switch(EtatEau){  
  case 0 :  
    int ValeurCapteurInitial = ValeurCapteur;  
    EtatEau = 1;  
    break;  
  case 1 :  
    C_Humidite = ValeurCapteur - ValeurCapteurInitiale;  
  
    if (C_Humidite >=180){  
  
      S_Fontaine = true;  
      S_Frigo = true;  
      S_Led = true;  
      S_Eau = true;  
      mechanism_status = true;  
    }  
}
```

## Mécanisme 9 : Quatres Eléments

### Description du Sous-système

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 de sortie.