



## Rapport

# Développement et mise en place d'un Escape Game

Semestre 3 - 2° Année



## Sommaire

<b>Sommaire .....</b>	<b>2</b>
<b>Introduction.....</b>	<b>3</b>
<b>Le serveur .....</b>	<b>7</b>
<b>Le site des joueurs .....</b>	<b>10</b>
<b>Le site de contrôle .....</b>	<b>16</b>
<b>Le système de logs .....</b>	<b>20</b>
<b>Le répondeur Asterisk .....</b>	<b>22</b>
<b>Le détecteur Sismique .....</b>	<b>23</b>
<b>Le coffre infrarouge.....</b>	<b>33</b>
<b>Le jeu du Simon.....</b>	<b>42</b>
<b>Conclusion .....</b>	<b>58</b>

## Introduction

Cette SAE avait pour objectif la réalisation d'un Escape Game complet faisant intervenir des moyens de transmission comme le Wi-Fi.

Afin de tirer le mieux partie de cette SAE et de plus vite se rendre compte des énigmes à mettre en place, nous avons d'abord commencé par trouver une petite histoire à notre jeu et ainsi le rendre plus intéressant.

L'idée d'empêcher une catastrophe naturelle de ravager le monde est venue assez naturellement car nous avions déjà en notre possession un détecteur de tremblements de terre, fait avec deux Arduinos connectés en Wi-Fi.

Nous nous sommes donc vite mis d'accord sur une histoire et un contexte : un énorme tremblement de terre risque de se produire dans les 30 prochaines minutes et seule la technologie de l'Institut de Recherche Scientifique sur les Catastrophes Naturelles (IRSCN) peut sauver l'humanité. La tâche consiste donc à passer parmi plusieurs niveaux de sécurité (les énigmes), afin d'activer cette fameuse technologie et sauver le monde.

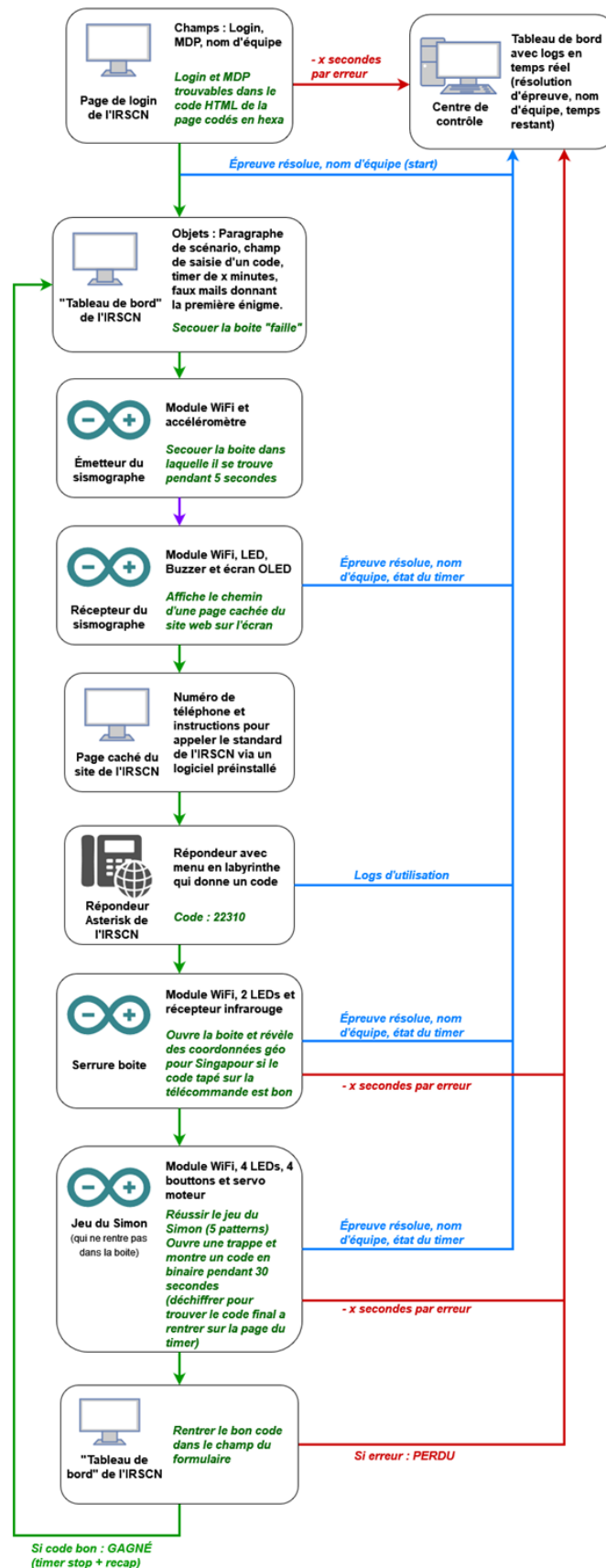
Une fois cela mis en place, il ne nous restait plus qu'à trouver les énigmes à mettre en place. Il était essentiel et logique pour nous de développer un site web et d'avoir plusieurs Arduinos dans notre jeu, avec un serveur central qui gèrerait le jeu de manière complètement automatique.

Les cours de la première année sur la téléphonie, et notamment sur les serveurs Asterisk, nous ont donné l'idée d'implémenter une énigme faisant appel à un appel justement, vers un serveur de VoIP.

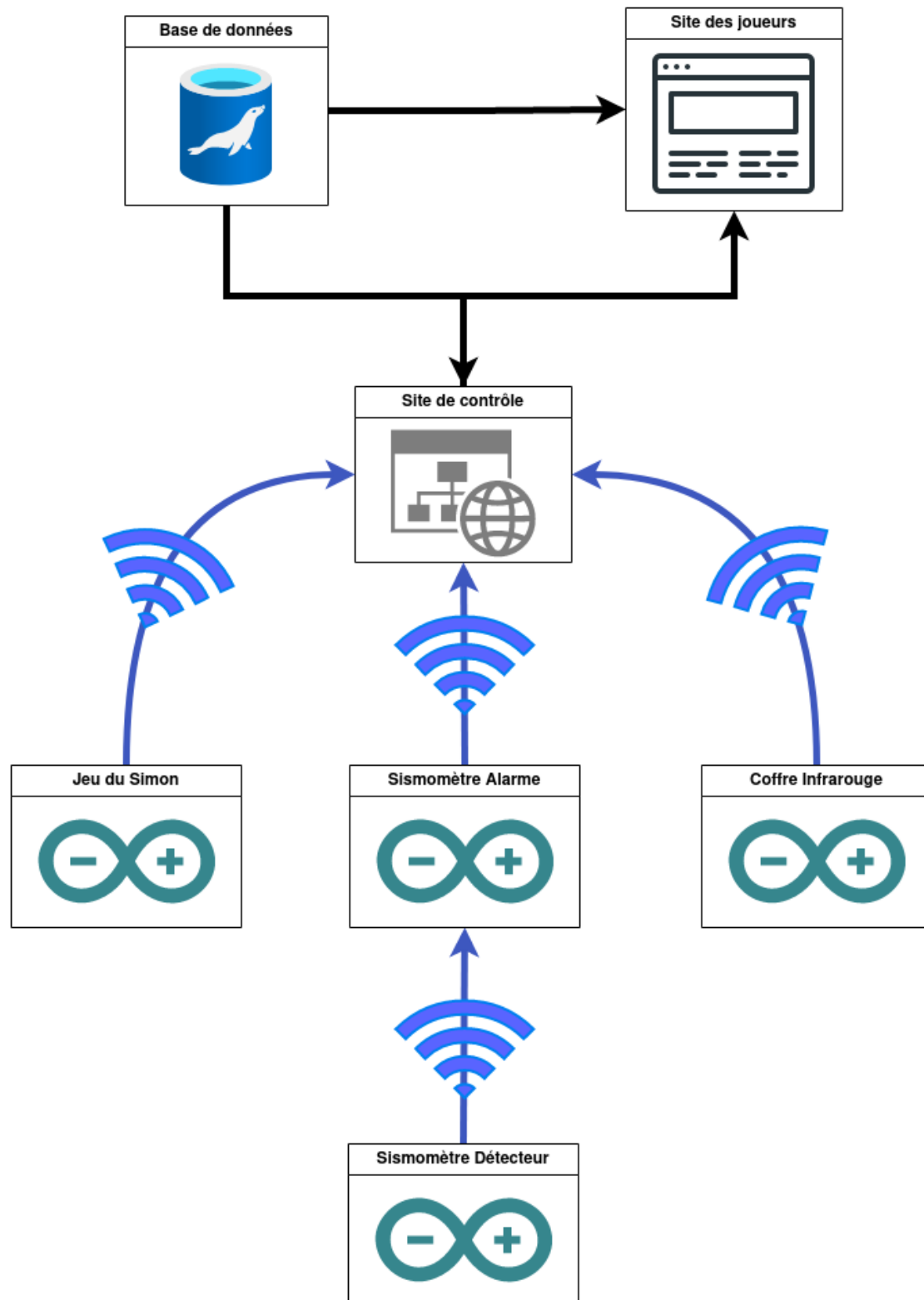
Une fois ces idées en place et la liste des énigmes terminée, nous avons dû dresser un diagramme du déroulé complet de l'Escape Game avec ses épreuves et les liens entre celles-ci.

De là, nous avons dressé la liste des composants nécessaires à la réalisation du projet, nous nous sommes réparti les tâches et nous avons dressé un Trello afin de suivre l'avancement du projet.

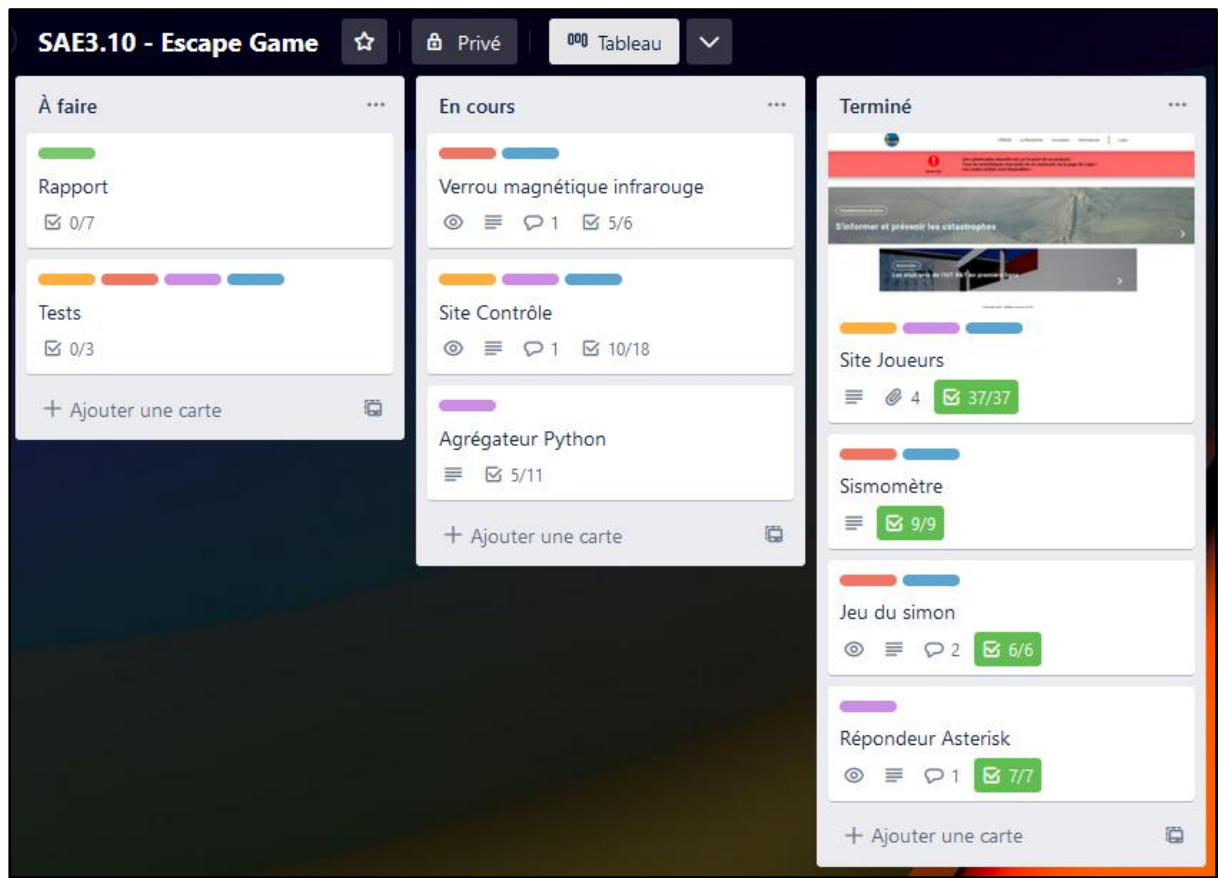
Le diagramme de l'Escape Game avec ses différentes épreuves :



Voici le schéma des transmissions complètes de l'Escape Game :



Voici le Trello du projet avec les différentes parties et les tâches qui y sont assignées :



Nous avons travaillé avec GitHub afin de centraliser nos codes, le répertoire est disponible en suivant ce lien : [SAE301-Escape\\_Game](#)

## Le serveur

Pierre angulaire du projet, le serveur hébergé sur Thorin, gère l'entièreté de l'Escape Game. Il permet d'héberger le site web des joueurs, celui des maîtres du jeu, le programme Python de gestion des messages des Arduinos, le serveur Asterisk et le système automatique de logs.

Nous avons commencé par mettre en place l'environnement Apache qui allait servir à héberger les deux sites web du jeu.

Afin d'héberger deux sites différents sur le même serveur, il nous a fallu utiliser des ports différents dans les fichiers de configuration des sites.

Pour le site des joueurs, **PlayerWeb**, nous avons gardé le port **80** parce qu'il s'agit du port par défaut et que l'accès au site est donc plus facile puisqu'il nous suffit de rentrer l'adresse IP du serveur afin d'y accéder.

Voici à quoi ressemble le fichier **/etc/apache2/playerweb.conf** sur notre serveur de jeu :

```
<VirtualHost *:80>
    ServerName 127.0.0.1
    ServerAdmin root@localhost
    DocumentRoot /var/www/playerweb
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Concernant le site des maîtres du jeu, **ControlWeb**, nous avons choisi de l'héberger sur le port **8888**. Ce n'est pas un port utilisé ni un port par défaut, nous devons donc le spécifier manuellement dans le champ d'URL du navigateur.

Voici maintenant le contenu du fichier **/etc/apache2/controlweb.conf** sur notre serveur de jeu :

```
<VirtualHost *:8888>
    ServerName 127.0.0.1
    ServerAdmin root@localhost
    DocumentRoot /var/www/controlweb
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Une fois cela fait et après avoir activé les sites au moyen de la commande **a2ensite**, nous pouvons commencer à utiliser les répertoires que l'on a spécifié dans les fichiers de configuration, afin d'y déposer les pages web et les documents qui seront disponibles sur les sites.

Note : Nos sites utilisent PHP, il fallait donc l'installer lui, ainsi que ses dépendances (notamment **libapache2**) grâce à **apt**.

Il a fallu ensuite se pencher sur la base de données qui contient les informations nécessaires au bon fonctionnement automatique de l'Escape Game.

Nous avons rapidement listé les données dont nous allions avoir besoin grâce à notre diagramme mis en place dès le début de notre réflexion.

Voici la définition des tables présentes dans la base de données **sae301** gérée par **mariadb** qui sont accessibles via l'utilisateur **control**. Les données en gras représentent les valeurs par défaut (les valeurs présentes lorsqu'aucune partie n'est lancée).

users
username
password

gamecontrol	
team_name	none
penalties	0
hints	6
finishdate	none
enddate	none
result	none
score	0
result_enigmas	0

Nous avons tout d'abord créé une table **users** qui se charge de contenir les informations nécessaires afin de se connecter à la page de login sur le site des joueurs que nous verrons plus tard.

Ensuite, la table **gamecontrol** contient les informations essentielles au bon fonctionnement de l'Escape Game, qui sont récupérées et éditées automatiquement par les différentes parties du jeu.

Notes :

- **penalties** correspond aux pénalités de l'équipe joueuse en secondes ;
- **hints** correspond aux indices restants ;
- **finishdate** indique la date à laquelle le jeu se fini (30 minutes après le lancement de celui-ci par les joueurs) ;
- **enddate** correspond à la date à laquelle les joueurs ont fini l'Escape Game, afin de calculer le temps restant et les points ;
- **result** définit le résultat du jeu (Gagné ou Perdu) ;
- **score** contient le score final de l'équipe joueuse ;
- **result\_enigmas** contient le total des points gagnés au fur et à mesure de la résolution des épreuves.



Le grand avantage pour nous d'avoir cette base de données est que nous pouvons avoir accès aux informations essentielles du fonctionnement de la partie en cours, et ainsi pouvoir visualiser certaines informations en temps réel, aussi bien sur le site des joueurs (le temps et les indices restants et les pénalités) que sur le site de contrôle.

Le serveur nous sert également de point d'accès Wi-Fi, afin d'y connecter nos Arduinos et qu'ils puissent envoyer des messages qui servent d'identifiant d'énigme pour les logs.

## Le site des joueurs

Nous avons commencé le gros du développement par le site qu'utilisent les joueurs pour suivre leur avancement dans l'Escape Game et qui héberge aussi quelques énigmes et indices.

Les pages restent simples dans leur mise en page mais se complexifient avec le code PHP qui leur permet de rester synchronisées entre elles et avec le serveur.

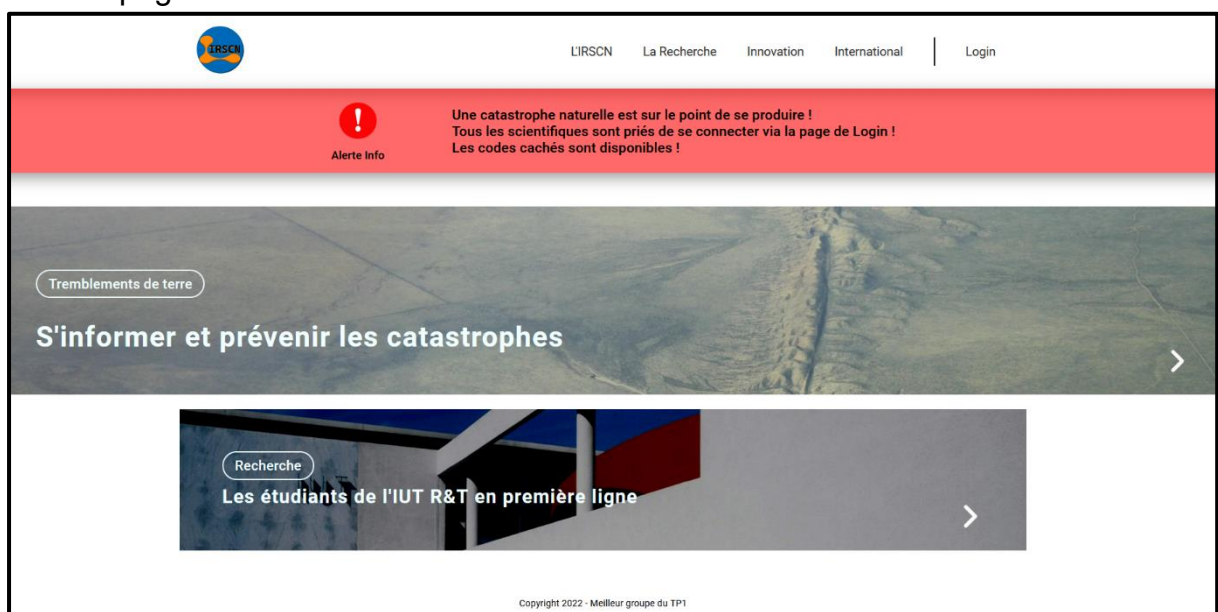
Tout d'abord, voici la liste des pages présentes sur le site **PlayerWeb** :

- Page d'accueil : **index.php**
- Page d'erreur : **error.php**
- Page de connexion : **login.php**
- Page du tableau de bord : **board.php**
- Page cachée : **hidden.php**
- Page des résultats : **results.php**

Au tout début du jeu, les joueurs arrivent sur la page d'accueil et doivent chercher un code en hexadécimal dans le code source de la page qui, une fois décodé, leur donne l'identifiant et le mot de passe à rentrer dans le formulaire présent sur la page **login.php** afin d'accéder au tableau de bord.

L'identifiant à trouver est : **iutchercheurs**, et le mot de passe : **no\_earthquake**.

Voici la page d'accueil :



Dès lors que les joueurs appuient sur le bouton redirigeant vers la page de connexion, l'Escape Game est lancé et le temps défile ! Il faut donc prendre son temps et bien lire les indices présents sur la page d'accueil avant de se précipiter en cliquant sur tous les liens.

Au moment de la redirection vers la page de connexion, le site web fait appel au script **init.php** qui initialise les variables de session propres à PHP et qui permettent aux différentes pages de partager les mêmes variables, ce qui n'est pas possible avec JavaScript.

Par exemple, pour les pénalités, le script va chercher la valeur des pénalités présente dans la base de données et initialise la variable de session :

```
if (!isset($_SESSION['penalties'])) {  
    $penalties = mysqli_fetch_array(mysqli_query($con, "SELECT penalties  
        FROM gamecontrol LIMIT 1"));  
    $_SESSION['penalties'] = $penalties['penalties'];  
}
```

Note : La configuration de l'accès à la base de données est définie dans un autre script PHP : **config.php** que l'on inclut dans tous les scripts où une action envers la base de données est nécessaire.

Une fois cela fait, et lors de l'initialisation de la variable **finishdate** vue précédemment, c'est le script **init.php** qui se charge d'initialiser les logs au format JSON :

```
$game_logs = '/home/sae301/logs/game-logs.json';  
$logsFile = fopen($game_logs, 'w');  
  
$logs = array(  
    "team_name" => "not_set",  
    "logs" => [ array(  
        "id" => 'Init'.strval($timeRemaining),  
        "enigma" => "Init",  
        "time" => date('H:i:s'),  
        "status" => "Résolue",  
        "time_left" => $timeRemaining,  
        "penalties" => $penalties['penalties'],  
        "hints" => $hints['hints']  
    )  
]  
);  
  
fwrite($logsFile, json_encode($logs));  
fclose($logsFile);
```

Ici, le script crée un fichier sur le serveur dans le répertoire **/home/sae301/logs/** qui contiendra les logs en temps réel de l'Escape Game jusqu'à la fin de la partie.

Les dates sont formatées selon la fonction **date**.

Voici l'initialisation de la variable **finishdate** :

```
$_SESSION['finishdate'] = date("r", strtotime("+1800 sec"));
```

On peut voir que l'on prend bien la date présente à laquelle on rajoute 1800 secondes soit 30 minutes.

Voici enfin le calcul du temps restant, calculé à partir de l'heure au moment de l'exécution, de la valeur de la variable **finishdate** et des pénalités :

```
$timeRemaining = ((strtotime($_SESSION['finishdate']) -  
$penalties['penalties']) - strtotime(date("r")));
```

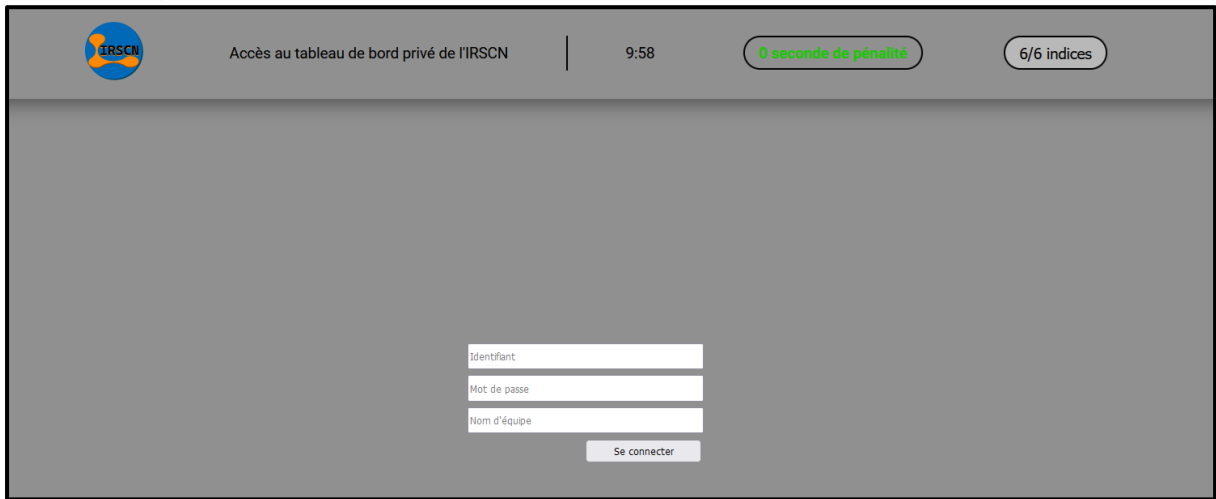
Une fois cela fait, les joueurs arrivent sur la page de connexion qui, comme toutes les pages du site, vérifie si l'utilisateur y a accès en vérifiant la valeur de la variable de session **user**.

```
session_start();  
if(isset($_SESSION['user'])) {  
    header("location: ./board.php");  
    die();  
}
```

Dans ce code, la page **login.php** vérifie si les joueurs se sont déjà connectés en testant l'existence de la variable de session **user**. Si les joueurs se sont déjà connectés via la page de connexion, ils seront automatiquement redirigés vers la page du tableau de bord s'ils tentent de revenir sur la page de connexion.

Le même principe est utilisé pour autoriser ou non l'accès à des pages nécessitant d'abord d'être connecté, comme, par exemple, les pages **board.php** et **hidden.php**. Evidemment, on peut décliner cette fonctionnalité pour tester si le jeu est fini et ainsi automatiquement rediriger vers la page des résultats ou au contraire en interdire l'accès tant que le jeu n'est pas fini.

Voici la page de connexion :



Note : Le nom d'équipe est obligatoire et sert à organiser les logs ainsi qu'à déterminer les grands sauveurs de l'humanité.

Sur toutes les pages qui diffusent le temps restant, les indices et les pénalités (toutes les pages sauf la page d'accueil et d'erreur), un code JavaScript s'occupe de récupérer les valeurs des variables de session importantes depuis PHP toutes les secondes et les affiche de manière formatée sur la page.

Note : PHP tourne côté serveur tandis que JavaScript tourne côté client. Afin de pouvoir récupérer des variables PHP en JavaScript, il faut faire appel à Ajax.

```
var penalties = 0;
$.ajax({
  url: "./elements/penalties.php",
  async: false,
  success: function(data){
    penalties = data;
  }
});
```

Ce code JavaScript présent sur toutes les pages du site des joueurs, permet de faire appel au script **penalties.php** qui va chercher la valeur des pénalités présente dans la base de données et retourne cette valeur avec un simple **echo \$\_SESSION['penalties'];**.

Le code Ajax récupère ensuite le résultat de l'écho de **penalties.php** et le place dans la variable locale **penalties**.

Le même système est mis en place pour les indices et pour le timer.

Une fois connectés, les joueurs arrivent sur leur tableau de bord avec, toujours : le temps restant en temps réel, les indices restants et les pénalités en secondes. Ce tableau de bord comporte : un champ de formulaire qui servira à rentrer le code final pour gagner (ou perdre) l'Escape Game, des indications sur le début de l'Escape Game et des faux mails qui donnent des indices sur la première épreuve.

Voici la page du tableau de bord :

The screenshot shows a web interface for an escape game. At the top, there's a header with a logo, the text 'Tableau de bord', a timer '9:27', a green button '0 seconde de pénalité', and a button '6/6 indices'. Below the header, a red alert message states: 'Un gigantesque tremblement de terre a été détecté et se produira dans les prochaines minutes ! L'Institut de Recherche Scientifique sur les Catastrophes Naturelles a mis au point un dispositif permettant de contrer cette menace ! Seul le bon code rentré dans le champ ci-dessous pourra activer ce système et sauver l'humanité ! Plusieurs dispositifs de sécurité ont été mis en place afin de garantir que le système ne puisse pas être abusé. Les directives parviennent souvent par mail...'. Below this is a form with 'Code secret' and 'Valider' buttons. A 'Mails' section follows, containing two email snippets. The first email is from 'Service de Surveillance de l'IRSCN' with the subject 'ATTENTION !' and mentions a seismic event at Bourges. The second email is from 'Service de Surveillance de l'IRSCN' with the subject 'Attaques en phishing' and warns about phishing attempts. The interface is clean and professional, using a grey and white color scheme with red for alerts.

La page cachée, atteignable en changeant l'URL après avoir pris connaissance de son existence au terme de la première épreuve, affiche elle aussi le temps restant, les indices restants et les pénalités. En plus de ces informations, la page cachée indique que les joueurs peuvent appeler un serveur distant en VoIP et joint une marche à suivre en PDF.

Cette page cachée possède directement dans son code la capacité de générer son propre log. En effet, cette page est une étape à elle seule et il faut que les maîtres du jeu soient au courant du moment où les joueurs l'atteignent.

La page cachée va donc elle-même chercher des informations dans la base de données, avant de les formater en JSON et de les envoyer dans le fichier des logs.

Enfin, la page des résultats se contente de récupérer les informations de la base de données au moyen des scripts **getScore.php**, **getHints.php** et **penalties.php**, selon la même méthode que lors de la récupération des pénalités sur les autres pages grâce à Ajax.

Elle les formate ensuite, afin que celles-ci soient affichées de manière claire et concise aux joueurs.

Voici la page des résultats :



**Résultats**

**Bravo équipe TP1 !**

**Vous avez gagné !**

**Il vous restait : 8 minutes et 35 secondes**

**Vous aviez : 20 secondes de pénalité !**

**Vous avez consommé : 2 indices !**

**Votre score final est de : 85 points !**

Copyright 2022 - Meilleur groupe du TP1

Passons rapidement sur les différents scripts utilisés qui n'ont pas encore été mentionnés, en résumant leurs fonctionnalités :

- **access.php** : vérifie lors de l'envoi du formulaire de connexion, que les identifiants sont bien conformes et ajoute un log en fonction de la réussite ou de l'échec de la connexion ;
- **WinOrLost.php** : vérifie si le code final entré depuis le tableau de bord est valide et calcule le score final après avoir déclaré l'équipe victorieuse ou non ;
- **reset.php** : est appelé manuellement par les maîtres du jeu à la fin de la partie et permet de remettre à zéro la base de données, les variables de session, ainsi que de ranger les logs dans un fichier au nom de l'équipe.

## Le site de contrôle

Nous savions que nous ne voulions pas avoir à tenir de registre papier ou à constamment être derrière les joueurs, tant pour notre confort que pour le leur. Il nous fallait donc trouver une solution afin d'avoir les yeux partout tout en restant immobiles derrière un seul écran.

Nous avons donc choisi de développer un site accessible uniquement par les maîtres du jeu, qui diffuse en temps réel le temps restant, les indices restants, les pénalités, les logs (quelle épreuve, quand, qui), ainsi qu'un résumé final à la fin de la partie.

Le site de contrôle se compose de différents blocs permettant chacun de visualiser une des informations citées plus haut.

Là où ce site se différencie du site des joueurs au niveau de la réalisation, c'est qu'il ne comporte que deux scripts PHP (sans compter le script de connexion à la base de données) qui lui permettent de récupérer TOUTES les informations de la partie, tandis que le site des joueurs en contient plus de cinq.

Voici la magie qui rend tout ça possible :

```
$result = mysqli_query($con, "SELECT team_name, penalties, hints,
                                finishdate, enddate, result, score FROM gamecontrol");

$data = array();

while ($row = mysqli_fetch_assoc($result)) {
    $data[] = $row;
}

echo json_encode($data);
```

Ce seul code du script **getData.php** permet de renvoyer toutes les données sélectionnées dans la base de données au script Ajax qui a fait appel à **getData.php**, sous la forme d'un dictionnaire.



De ce fait, il n'y a qu'un seul script Ajax sur la page **index.php** du site de contrôle qui récupère toutes les informations :

```
$.ajax({
  url: "../elements/getData.php",
  async: false,
  success: function(PHPdata){
    let data = JSON.parse(PHPdata)
    team_name = data[0].team_name;
    penalties = Number(data[0].penalties);
    hints = Number(data[0].hints);
    finishdate = data[0].finishdate;
    enddate = data[0].enddate;
    result = data[0].result;
    score = Number(data[0].score);
  }
});
```

La partie JavaScript doit pouvoir lire le fichier de logs et ainsi afficher chaque log en temps réel. En effet, le script lit le fichier de logs toutes les secondes et ajoute les logs qu'il n'a pas déjà ajouté à la page.

Pour ce faire, le script fait appel à **getLogs.php** qui via une requête Ajax. Celui-ci lit le fichier, et le renvoie sous la forme d'un dictionnaire au programme JavaScript.

Après l'obtention du dictionnaire contenant les logs, le script vérifie que le log n'est pas déjà présent sur la page. Ensuite, il ajoute récursivement les éléments nécessaires à l'affichage des logs : un élément div général, trois divs enfants pour les informations générales, les informations propres à l'énigme et les informations générales de la partie. Puis, chaque information est ajoutée au moyen d'un élément paragraphe dont le contenu est formaté en fonction de l'information qu'il contient (texte en rouge si l'énigme est échouée, en vert si elle est réussie).

```
if (logs != null) {
  // Pour chaque entrée dans la liste "logs", on vérifie si le log n'est pas
  // déjà présent
  logs.forEach(log => {
    if (!logs_ids.includes(log.id)) {
```

Afin de différencier chaque log sur la page web, ceux-ci sont ajoutés avec l'id correspondant dans le fichier de logs.

Dès que le site détecte un résultat valide dans la base de données, il effectue le même programme que sur la page **results.php** du site **PlayerWeb** et affiche un compte rendu de la partie en récupérant les informations de la base de données une dernière fois à la fin de la partie.

## Le programme Python

Nos énigmes reposant massivement sur des Arduinos, il est essentiel de pouvoir récupérer des messages envoyés par ceux-ci via la connexion Wi-Fi. C'est pourquoi, nous avons développé un programme Python tournant en arrière-plan sur le serveur, qui écoute sur le port **9999**, un port non utilisé, et attend la connexion d'un client et est prêt à recevoir un message.

Pour cela, nos TP de Python de deuxième année nous ont bien servi, puisqu'il a simplement fallu créer un socket :

```
self.serveur = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.serveur.bind(('', 9999))
```

Et écouter dessus :

```
self.serveur.listen(5)
connexion, adresse = self.serveur.accept()
```

Il a cependant fallu que le programme Python soit capable de recevoir les messages de plusieurs clients en même temps et donc d'autoriser de multiples connexions sur le même port.

Nous avons pour cela utilisé un système de threads, ce qui signifie que différents processus seront alloués pour chaque client qui se connectera au programme, permettant ainsi à plusieurs clients de se connecter simultanément.

```
thread = threading.Thread(target=self.traitement_connexion, args=(connexion,
                                                                    adresse))
thread.start()
```

Le programme teste les messages reçus et agit en conséquence. Par exemple, si le message provient du Sismomètre, le programme le renvoie d'abord à l'alarme afin que celle-ci s'active ou se désactive, puis le programme crée un log en fonction du message.

Tout comme les sites web, le programme Python récupère les informations essentielles de la partie depuis la base de données, puis les formate selon le format des logs donné par la suite.

Si besoin, comme pour les sites web, le programme Python met à jour les informations de la base de données. Par exemple, si une énigme est échouée et renvoie le code **E**, il faut que 10 secondes soient ajoutées aux pénalités.

## Le système de logs

Le système de logs se doit d'être simple et facilement évolutif. C'est pourquoi nous avons développé le système suivant :

- Un fichier de logs est créé dans le répertoire **/home/sae301/logs** et est lu par les deux sites web et le programme Python ;
- A la fin de la partie, lors de la remise à zéro manuelle, le fichier est renommé avec le nom de l'équipe et est déplacé dans le répertoire **/home/sae301/archived-logs** ;
- La structure du fichier de logs est la suivante :

```
{
  "team_name": "TP1",
  "logs": [
    {
      "id": "LR50000",
      "enigma": "Login",
      "time": "16:58:44",
      "status": "Résolue",
      "time_left": 50000,
      "penalties": 20,
      "hints": 2
    },
    {
      "id": "SE48888",
      "enigma": "Simon",
      "time": "16:59:44",
      "status": "Échouée",
      "time_left": 48888,
      "penalties": 30,
      "hints": 1
    }
  ]
}
```

La valeur **team\_name** est modifiée après une connexion réussie via la page **login.php**. Autrement, cette valeur reste en **undefined**.

La liste logs contient tous les événements de l'Escape Game comprenant à chaque fois :

- L'identifiant généré à partir de l'initiale de l'épreuve, suivie de **R** ou **E** pour Réussie ou Échouée et du temps restant au moment de l'événement en secondes ;
- Le nom complet de l'énigme ;
- L'heure réelle à laquelle l'événement s'est réalisé ;
- Le statut complet de l'énigme ;
- Le temps restant en secondes au moment où l'événement s'est réalisé ;
- Les pénalités en secondes au moment où l'événement s'est réalisé ;
- Les indices restants au moment où l'événement s'est réalisé.

Toutes ces informations sont générées sur l'un des deux sites web ou depuis le programme Python. Le système ayant généré le log, vérifie son existence ou non dans le fichier de logs. Si le log n'existe pas ou n'intervient pas après un log d'une énigme réussie, il est ajouté à la suite des éléments de la liste **logs**.

## Le répondeur Asterisk

Après avoir trouvé la page cachée, les joueurs doivent appeler le serveur Asterisk via l'application **twinkle**. Le numéro pour appeler le serveur est le **00310**. Les joueurs sont alors redirigés vers un menu interactif dont voici les menus et leurs options :

Menu principal	Menu informations	Menu Nouvelle découverte	Menu Plan de Surveillance et d'Action
Informations sur l'institut	Histoire de l'institut	Scientifique ?	Accepter et activer
Nouvelle découverte	Projets de l'institut	Responsable organisation ?	Réécouter
Info sur les catastrophes	Réécouter	Réécouter	Menu principal
Activer le plan de surveillance et d'action	Menu principal	Menu principal	
Réécouter			
Quitter l'appel			

Menu P.S.A demande code
Écouter le code IR
Réécouter
Menu principal

A partir des indices précédents, les joueurs se rendent dans le menu du Plan de Surveillance et d'Action et acceptent deux fois les conditions en appuyant sur 1. La voix du répondeur indique qu'il faut trouver un coffre étiqueté P.S.A et l'ouvrir au moyen d'une télécommande infrarouge avec le code qu'elle dicte juste après (22310).

Voici un exemple du contenu du fichier **/etc/asterisk/extensions.conf** qui sert à créer un tel menu :

```
[menuInfos]
exten => s,1,Background(/home/sae301/audios-Asterisk/menu_infos,m)
same => n,Goto(menu,s,10)

exten => 1,1,Goto(menuInfos-History,s,1)
exten => 2,1,Goto(menuInfos-Projects,s,1)
exten => 3,1,Goto(menuInfos,s,1)
exten => 4,1,Goto(menu,s,10)
```

L'instruction **Background** avec le paramètre **m**, joue un fichier audio en attendant que l'utilisateur appuie sur une touche du pavé numérique.

Si l'utilisateur n'appuie sur aucune touche, le serveur Asterisk renvoie l'utilisateur au menu principal à la fin du fichier audio.

## Le détecteur Sismique

Le montage qui nous a lancés sur cet Escape Game est un détecteur sismique développé à partir de deux Arduinos, d'un accéléromètre, d'un écran OLED, d'une LED, d'un Buzzer et de deux modules ESP8266 servants à connecter les Arduinos en Wi-Fi.

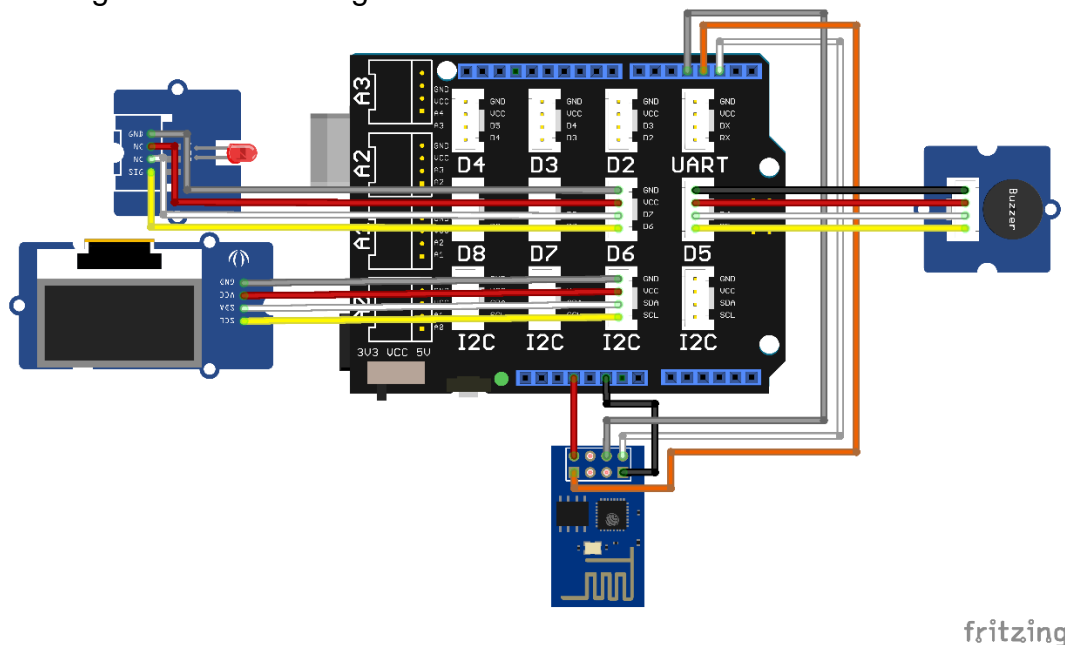
Ce détecteur sismique est divisé en deux parties : un détecteur composé uniquement d'un Arduino connecté en Wi-Fi via un ESP8266 et d'un accéléromètre ; et une alarme composée du deuxième Arduino avec son ESP8266, de l'écran, de la LED et du Buzzer.

L'alarme est développée et lancée en première, car c'est elle qui sert de serveur. En effet, lorsque le détecteur sismique ressent une secousse importante, il envoie un message à l'alarme lui indiquant qu'il faut sonner et ainsi afficher sur son écran l'URL **/hidden.php**, afin que les joueurs prennent connaissance de son existence et poursuivent le jeu.

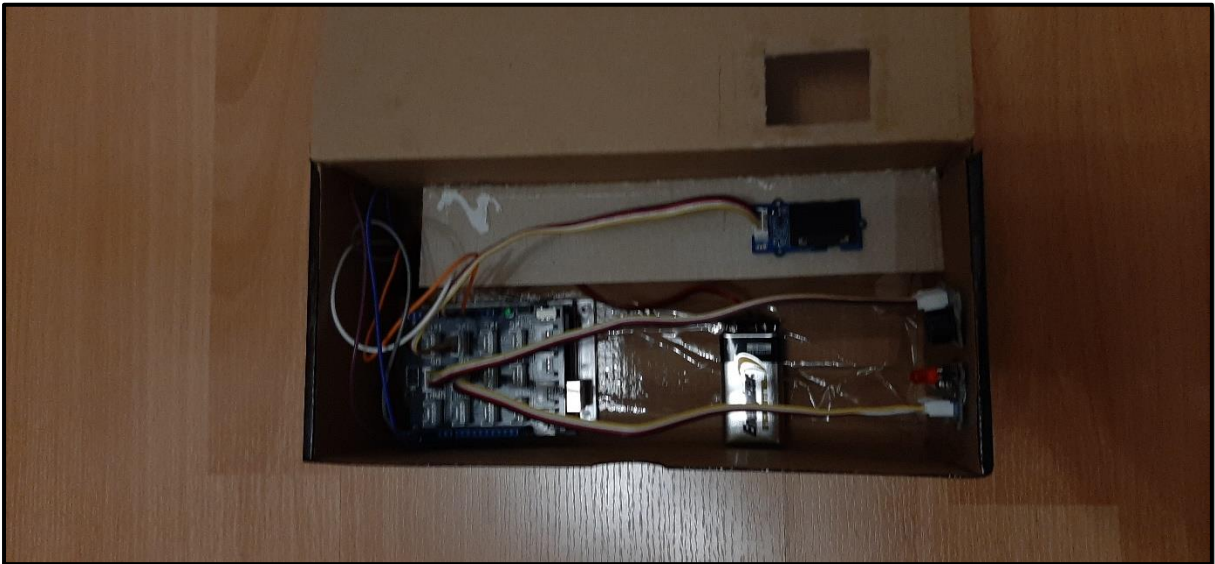
Voici le pinout de l'alarme :

Pins Arduino	D5	D6	I2C	GND	3,3V	2	3	4
Pin/Module correspondant	Buzzer	LED	Ecran OLED	GND ESP8266	3,3V ESP8266	TX ESP8266	RX ESP8266	CH_PD ESP8266

Le diagramme du montage :



Voici l'alarme dans sa boîte :





Voici le code commenté :

```
#include <SoftwareSerial.h> // Bibliothèque SoftwareSerial pour utiliser les fonctionnalités de la communication série logicielle

#include "Arduino_SensorKit.h" // Bibliothèque Arduino_SensorKit pour utiliser les fonctionnalités du shield et des composants comme l'écran OLED et le Buzzer

#include <SerialESP8266wifi.h> // Bibliothèque SerialESP8266wifi pour utiliser les fonctionnalités de la communication wifi

// Serial config
#define sw_serial_rx_pin 2 // Pin vers TX
#define sw_serial_tx_pin 3 // Pin vers RX
#define esp8266_reset_pin 4 // Pin vers CH_PD, pas reset

// Crée une instance de la communication série logicielle en utilisant les broches définies précédemment
SoftwareSerial swSerial(sw_serial_rx_pin, sw_serial_tx_pin);
SoftwareSerial AT(sw_serial_rx_pin, sw_serial_tx_pin);

// Crée une instance de la communication wifi en utilisant l'instance de la communication série logicielle et la broche de réinitialisation définies précédemment
SerialESP8266wifi wifi(swSerial, swSerial, esp8266_reset_pin, Serial);

// User config
#define ssid "ArduinoEarth" // Wifi SSID
#define password "no_earthquake" // Wifi Password

String inputString;

#define LED 6 // Slot D6 utilisée pour la LED
#define BUZZER 5 // Slot D5 utilisée pour le Buzzer

// Définit l'état par défaut de l'alarme
bool alarm_state = false;
```

```
void setup() {
    // Démarre les composants et la communication série
    Serial.begin(9600);
    swSerial.begin(9600);
    pinMode(LED, OUTPUT);
    pinMode(BUZZER, OUTPUT);
    Oled.begin();
    Oled.setFlipMode(false);

    // Tant que la communication série n'est pas prête
    while (!Serial);
    Serial.println("Starting wifi");
    // Configure la communication wifi en mode TCP
    wifi.setTransportToTCP();
    // Configure la fin de la transmission wifi avec un saut de ligne
    wifi.endSendWithNewline(true);
    // Démarre la communication wifi
    wifi.begin();
    // Se connecte au réseau wifi défini avec le ssid et le password
    wifi.connectToAP(ssid, password);
    // Affiche son adresse IP
    wifi.getIP();
    // Démarre un serveur local sur le port 9990
    wifi.startLocalServer("9990");

    AT.begin(9600);
}

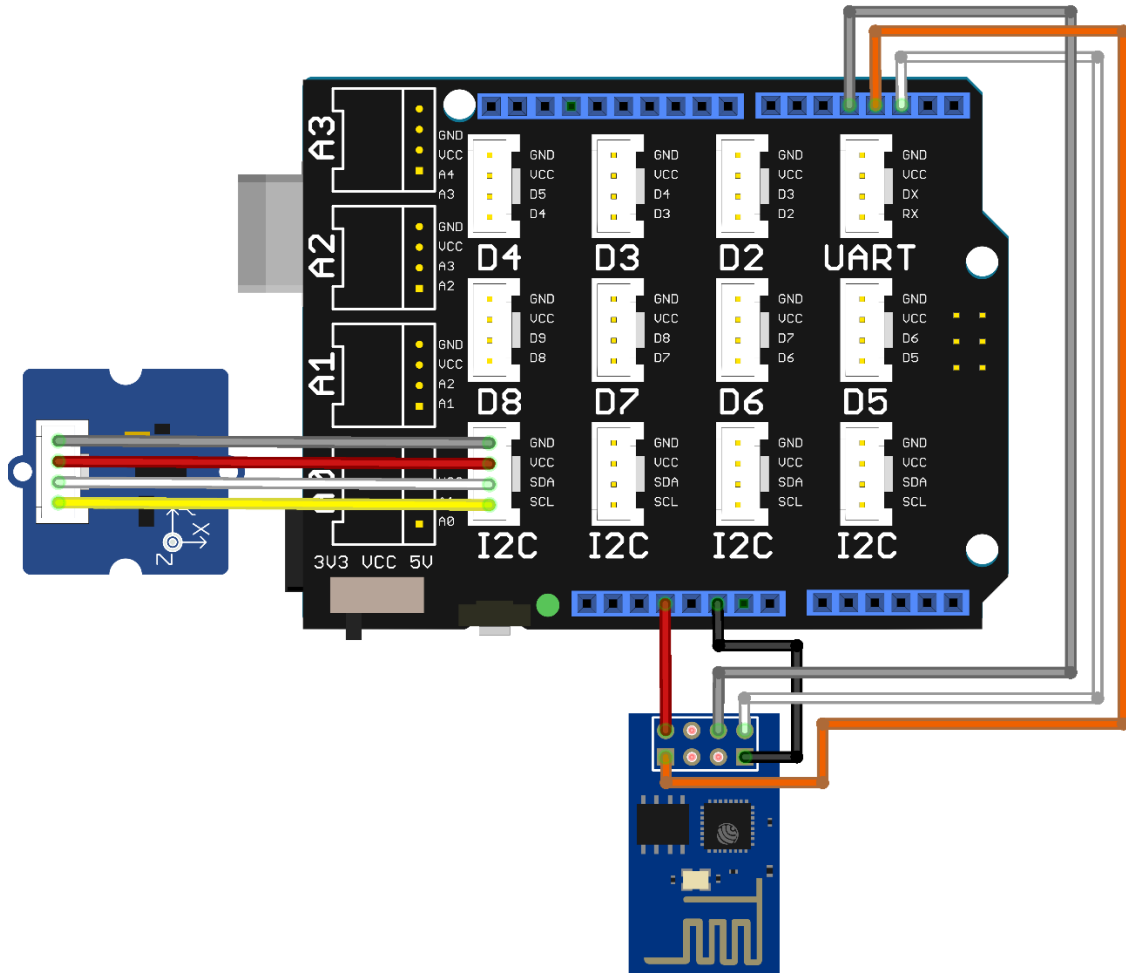
void loop() {
    if (!wifi.isStarted()) {
        wifi.begin();
        wifi.getIP();
    }
    // Lit les messages reçus sur le port ouvert
    while (AT.available() > 0) {
        String message = AT.readString();
        // Si le message est "SyR" : active l'alarme
        if (message.indexOf("SyR") >= 1) {
            alarm_state = true;
        }
        // Si le message est "SyE" : désactive l'alarme
        if (message.indexOf("SyE") >= 1) {
            alarm_state = false;
        }
        Serial.println("State: " + String(alarm_state));
    }
    Alarm();
}
```

```
void Alarm(){
    // Si l'alarme est active : fait sonner le Buzzer, fait clignoter
    la
        LED et affiche un message clignotant sur l'écran
    if (alarm_state == true) {
        digitalWrite(LED, HIGH);
        tone(BUZZER, 90);
        Oled.setFont(u8x8_font_chroma48medium8_r);
        Oled.setCursor(0, 43);
        Oled.print("                ");
        Oled.setCursor(35, 33);
        Oled.print("ALERTE");
        Oled.setCursor(50, 53);
        Oled.print("/hidden.php");
        Oled.refreshDisplay();
        delay(1000);
        digitalWrite(LED, LOW);
        noTone(BUZZER);
        Oled.setCursor(35, 33);
        Oled.print("ALERTE");
        Oled.setCursor(50, 53);
        Oled.print("                ");
        Oled.refreshDisplay();
        delay(1000);
    }
    // Si l'alarme est inactive : éteint le Buzzer et la LED et af-
    fiche
        le texte "Aucune alerte" sur l'écran
    else {
        noTone(BUZZER);
        digitalWrite(LED, LOW);
        Oled.setFont(u8x8_font_chroma48medium8_r);
        Oled.setCursor(35, 33);
        Oled.print("                ");
        Oled.setCursor(50, 53);
        Oled.print("                ");
        Oled.setCursor(0, 43);
        Oled.print("Aucune alerte!");
        Oled.refreshDisplay();
    }
}
```

Pour le détecteur qui va venir se connecter au serveur de l'Arduino de l'alarme pour y envoyer ses messages, voici son pinout :

Pins Arduino	I2C	GND	3,3V	2	3	4
Pin/Module correspondant	Accéléromètre	GND ESP8266	3,3V ESP8266	TX ESP8266	RX ESP8266	CH_PD ESP8266

Le diagramme du montage :



fritzing

Voici le détecteur dans sa boîte :



Voici enfin le code commenté :

```
#include <SoftwareSerial.h> // Bibliothèque SoftwareSerial pour utiliser les fonctionnalités de la communication série logicielle

#include "Arduino_SensorKit.h" // Bibliothèque Arduino_SensorKit pour utiliser les fonctionnalités du shield et des composants comme l'écran OLED et le Buzzer

#include <SerialESP8266wifi.h> // Bibliothèque SerialESP8266wifi pour utiliser les fonctionnalités de la communication wifi

// Serial config
#define sw_serial_rx_pin 2 // Pin vers TX
#define sw_serial_tx_pin 3 // Pin vers RX
#define esp8266_reset_pin 4 // Pin vers CH_PD, pas reset

// Crée une instance de la communication série logicielle en utilisant les broches définies précédemment
SoftwareSerial swSerial(sw_serial_rx_pin, sw_serial_tx_pin);

// Crée une instance de la communication wifi en utilisant l'instance de la communication série logicielle et la broche de réinitialisation définies précédemment
SerialESP8266wifi wifi(swSerial, swSerial, esp8266_reset_pin, Serial);

// User config
#define ssid "ArduinoEarth" // Wifi SSID
#define password "no_earthquake" // Wifi Password
```

```
String inputString;

// Définit l'état par défaut de l'alarme
bool alerte = false;

// Définit les valeurs par défaut des axes
float x1 = 0.0;
float y1 = 0.0;
float z1 = 0.0;

float x = 0.0;
float y = 0.0;
float z = 0.0;

void setup() {
  // Démarre les composants et la communication série
  Serial.begin(9600);
  swSerial.begin(9600);
  inputString.reserve(20);

  Accelerometer.begin();

  // Tant que la communication série n'est pas prête
  while (!Serial);
  Serial.println("Starting wifi");
  // Configure la communication wifi en mode TCP
  wifi.setTransportToTCP();
  // Configure la fin de la transmission wifi avec un saut de ligne
  wifi.endSendWithNewline(true);
  // Démarre la communication wifi
  wifi.begin();
  // Se connecte au réseau wifi défini avec le ssid et le password
  wifi.connectToAP(ssid, password);
  // Se connecte au socket du programme Python sur le port 9999
  wifi.connectToServer("192.168.122.103", "9999");
}

void loop() {
  if (!wifi.isStarted()) {
    wifi.begin();
  }

  // Lit les valeurs des axes
  x1 = Accelerometer.readX();
  y1 = Accelerometer.readY();
  z1 = Accelerometer.readZ();
}
```

```
delay(100);

// Calcul la différence entre les valeurs des axes au moment
// présent et celles prises il y a 100 millisecondes
x = Accelerometer.readX()-x1;
y = Accelerometer.readY()-y1;
z = Accelerometer.readZ()-z1;
int alerte_confirm = 0;

// Vérifie si la différence dans au moins un des axes est plus
// important que 0.1%
while ((x > 0.1 || x < -0.1) || (y > 0.1 || y < -0.1) || (z > 0.1 || z < -
0.1)){
    if (alerte == false) {
        alerte_confirm++;
        if (alerte_confirm == 5) {
            // Envoie le message "SyR" au serveur de l'alarme si une
            // différence importante qui perdure pendant 5 secondes
            Serial.println("TREMBLEMENT DE TERRE !");
            alerte = true;
            wifi.send(SERVER, "SyR");
        }
        Serial.println(alerte_confirm);
    }
    if (alerte == true) {
        Serial.println("ALERTE ENVOYEE !");
    }

    // Continue de vérifier si la différence dans au moins un des
    // axes est plus important que 0.1%
    delay(500);

    x1 = Accelerometer.readX();
    y1 = Accelerometer.readY();
    z1 = Accelerometer.readZ();

    delay(100);

    x = Accelerometer.readX()-x1;
    y = Accelerometer.readY()-y1;
    z = Accelerometer.readZ()-z1;

    delay(500);
}
```

```
if (alerte == true) {  
    // Si une alarme est lancée et qu'il n'y a plus de différence  
    importante dans au moins un des axes, envoie le message "SyE"  
    au serveur de l'alarme et fait s'arrêter l'alarme  
    alerte = false;  
    wifi.send(SERVER, "SyE");  
}  
Serial.println("RAS");  
delay(500);  
}
```

Avec ce système, il suffit donc de secouer la boîte pendant quelques secondes, afin de voir apparaître l'URL **/hidden.php** sur l'écran de l'alarme.



## Le coffre infrarouge

L'objectif de ce jeu est de, grâce à la solution du jeu précédent, taper un code sur une télécommande en direction d'un capteur infrarouge, afin d'ouvrir un verrou électromagnétique.

En fonction du code final, une LED verte s'allume pour prévenir les joueurs de l'ouverture du verrou électromagnétique, sinon, une LED rouge informe les joueurs de leur erreur. Pour être certain qu'un appui sur une touche à bien fonctionné, une LED bleue s'allume à chaque fois qu'une touche sera prise en compte. De plus, un Buzzer sonne lorsque le bon code est tapé, au moment de l'ouverture du verrou, qui ouvrira la boîte du jeu, révélant l'indice pour le jeu suivant.

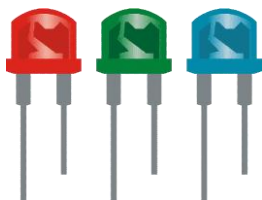
La transmission de données se fait par un capteur infrarouge, qui capte la télécommande, et reçoit les informations, que l'on peut retrouver sur le moniteur Arduino.

Si on se penche un peu plus sur ce que nous propose la [documentation](#), nous pouvons voir que pour utiliser le capteur infrarouge, il nous faut 2 parties. Une partie émettrice, et une partie réceptrice.

Pour câbler le capteur infrarouge (RC5), il faut vérifier le sens des pins. Surtout, ne pas se tromper dans le câblage, sinon le capteur peut brûler et devenir obsolète. Ensuite, du côté de l'émission, nous utilisons une télécommande. Pour la partie réceptrice, nous avons donc le capteur infrarouge et nous n'utilisons pas le programme présent dans la [documentation](#).

Pour mettre en place la partie de ce jeu, nous devons utiliser plusieurs éléments :

3 LEDs



Buzzer



Arduino



Capteur Infrarouge



Verrou



Télécommande



Voici le pinout de ce coffre infrarouge :

<b>Pins Arduino</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
Pin/Module correspondant	LED Bleue	LED Rouge	LED Verte	Buzzer	Verrou

Après avoir lu la documentation, nous avons pu voir que le capteur fournissait un code type, que l'on a testé. Lorsque nous avons mis en place le capteur seul, nous avons pu voir sur le moniteur, qu'un code en hexadécimal représentant la touche qui a été actionnée, est affiché. Nous avons pu remarquer par la suite, que lorsqu'un appui prolongé sur une des touches était effectué, un buffer de type « FFFFFFFF » est renvoyé en plus du code de la touche. Nous avons par la suite pris en compte cette fonctionnalité.

Nous avons décidé de reprendre le programme de base, puis de lui ajouter notre touche personnelle. Cependant, nous avons rencontré beaucoup de difficultés. Nous avons voulu mettre sous forme de liste le code à obtenir, afin de pouvoir le comparer avec une autre liste, à laquelle on ajouterait successivement chaque touche pressée de la télécommande.

À partir de ce moment-là, nous avons rencontré un problème. Le fait d'ajouter à une liste un code reçu par le moniteur n'est pas chose simple. Au début, nous avons décidé de résoudre le problème en concaténant chaque code hexadécimal entre eux. Mais, une fois de plus, problème, car les codes hexadécimaux n'étaient pas toujours les mêmes.

Pour y remédier, nous avons remarqué que dans le code, une ligne convertissait la valeur reçue par la télécommande. Cela veut dire que le code était d'abord reçu en base 10, puis converti en base 16 (hexadécimal). Nous avons alors décidé de simplifier le code en ne convertissant rien, et en gardant le code brut.

N'oublions pas que le fait d'avoir une liste, que ce soit la liste avec le code de référence, ou la liste qui recevra le code implémenté, nous posait un souci. Pour régler ce problème, nous avons retiré l'idée des listes, qui fonctionnent avec du « char » ou du « char\* » dans le langage Arduino, par des chaînes de caractères, qui, elles, fonctionnent avec du « str ». Le fait d'avoir comme type du « str », nous permet de concaténer plus facilement des chaînes de caractères.

À partir de ce moment-là, nous avons dû reprendre tout le code à partir du début. C'est à dire que la liste avec le code de référence a dû être transformée en une liste de caractère, avec le code, cette fois-ci en base 10, et non en hexadécimal.

Comme dit au début, lorsque l'on appuie longtemps sur la même touche, le moniteur nous renvoi un buffer avec « FFFFFFFF », sauf que si on reçoit ce message, il va s'ajouter dans notre chaîne de caractère à comparer avec le code final. Alors pour remédier à ce problème, nous avons dû créer dans la boucle, une condition « if », qui nous permet à l'aide d'une fonction, trouvée dans la documentation, de comparer une certaine partie d'une chaîne de caractère. Nous avons décidé de comparer les trois premiers caractères de chaque code d'une touche. Nous avons pu comparer simplement ce début de code, parce que nous avons remarqué que chaque code commençait de la même manière (par les chiffres 167).

Pour la partie du verrou, il n'y a rien de plus simple. Il suffit d'initialiser le fait que le verrou doit être un **OUTPUT**, puis affirmer, dès que le jeu commence, que l'état du verrou est « **HIGH** », ce qui signifie qu'un courant électrique lui sera envoyé et lui permettra de devenir aimanté et ainsi de maintenir le couvercle de la boîte fermé.

Si on s'intéresse aux états que peut avoir le verrou, on retrouve deux positions :

- HIGH : Signifie que le verrou est fermé (aimanté)
- LOW : Signifie que le verrou est ouvert (non aimanté)

Dans chaque condition, il faut préciser l'état que le verrou devra avoir. Dans notre cas, il doit tout le temps être sur la position « HIGH », sauf quand le jeu est réussi, alors son état passera en « LOW », pour permettre l'ouverture de la boîte.

Pour finir, il nous reste la partie Wi-Fi.

En premier lieu, nous devons importer deux bibliothèques : **<SoftwareSerial.h>** et **<SerialESP8266wifi.h>**.

Voici le pinout spécifique à l'Arduino de cette énigme et à son ESP8266 :

Pins Arduino	A0	A1	A2	GND	3,3V
Pin ESP8266	TX	RX	CH_PD	GND	3,3V

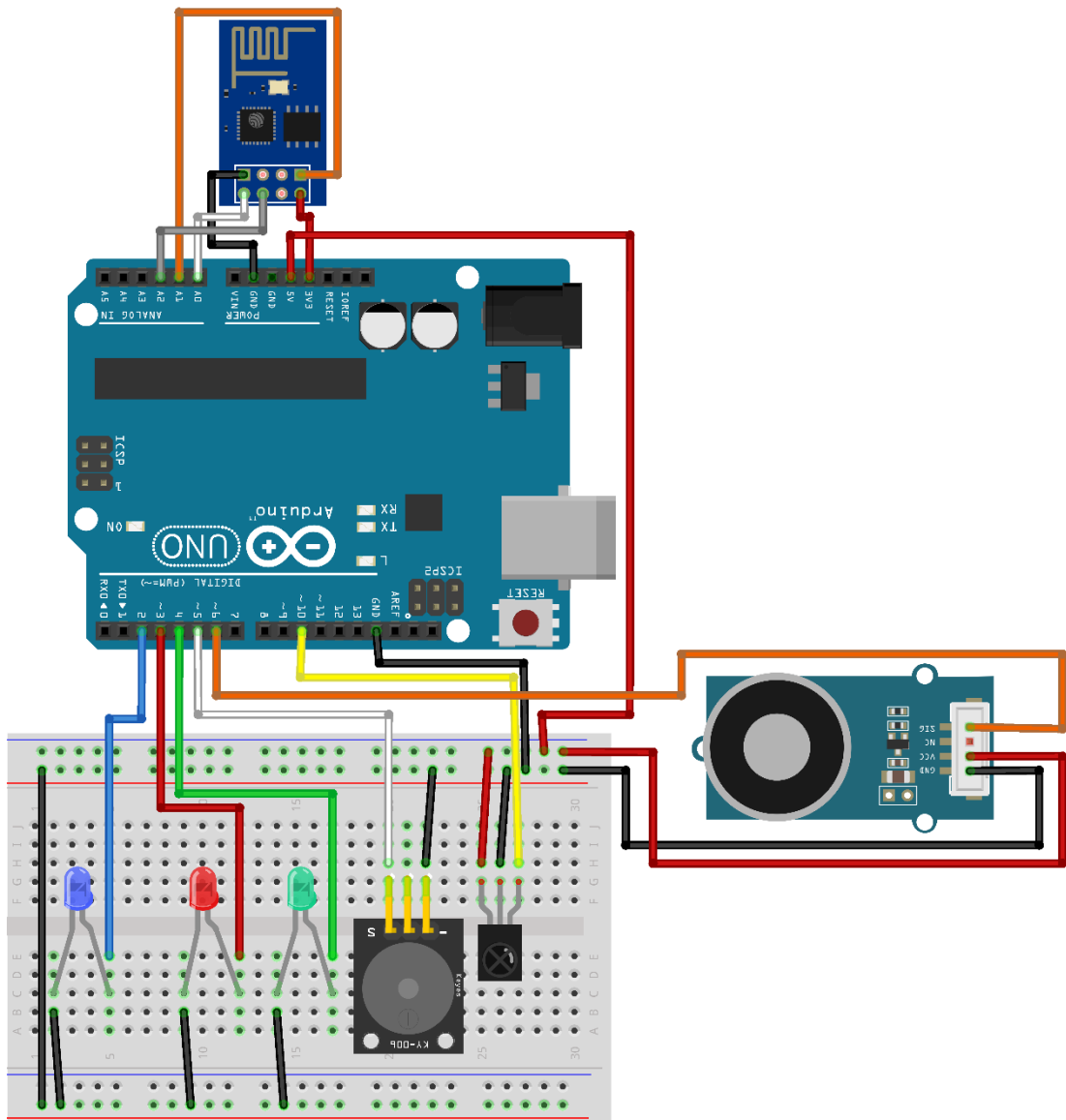
Ensuite, nous devons configurer l'utilisateur qui se connectera au Wi-Fi. Il faut alors définir le SSID et le mot de passe de notre point d'accès Wi-Fi.

Il nous faut ensuite connecter l'ESP8266 au programme Python en cours d'exécution sur le serveur et ainsi utiliser un thread et pouvoir envoyer des messages.

Une fois que nous avons toutes ces informations, nous pouvons alors implémenter cette partie dans le code.

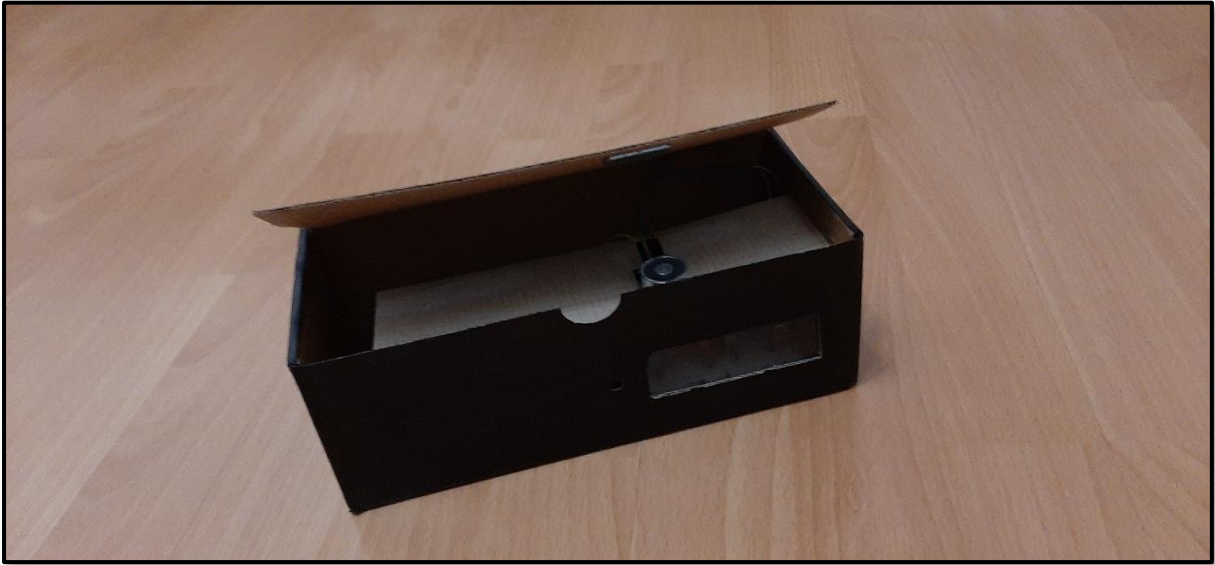
Si la finalité du jeu est bonne, alors on envoie en Wi-Fi le message « **LoR** » au programme Python via la connexion établie pour dire que le jeu est réussi, sinon, on renvoie « **LoE** ».

Voici le diagramme du montage :



fritzing

Le verrou infrarouge dans sa boîte :



Voici enfin le code commenté :

```
// Importe la bibliothèque IRremote pour utiliser les
fonctionnalités de la télécommande infrarouge
#include <IRremote.h>

// Importe la bibliothèque SoftwareSerial pour utiliser les
fonctionnalités de la communication série logicielle
#include <SoftwareSerial.h>

// Importe la bibliothèque SerialESP8266wifi pour utiliser les
fonctionnalités de la communication wifi
#include <SerialESP8266wifi.h>

// Serial config
#define sw_serial_rx_pin A0 // Pin vers TX
#define sw_serial_tx_pin A1 // Pin vers RX
#define esp8266_reset_pin A2 // Pin vers CH_PD, pas reset

// Crée une instance de la communication série logicielle en
utilisant les broches définies précédemment
SoftwareSerial swSerial(sw_serial_rx_pin, sw_serial_tx_pin);

// Crée une instance de la communication wifi en utilisant
l'instance de la communication série logicielle et la broche de
réinitialisation définies précédemment
SerialESP8266wifi wifi(swSerial, swSerial, esp8266_reset_pin,
Serial);

// User config
#define ssid "ArduinoEarth" // Wifi SSID
#define password "no_earthquake" // Wifi Password

String inputString;

int broche_reception = 11; // Broche 10 utilisée pour la réception
IRrecv Receiver(broche_reception); // Crée une instance de réception
decode_results decode_ir; // Décodage et stockage des données reçues

// Initialisation du bon code à avoir
String code = "1671805516718055167430451672417516738455";
```

```
// Déclare une variable pour stocker le code final constitué de la
concaténation des différents messages envoyés par la télécommande
String test_code = "";

// Déclare une variable pour stocker le message infrarouge reçu
String msg = "";

const int BLUE = 2;
const int RED = 3;
const int GREEN = 4;
const int BUZZER = 5;
const int LOCK = 6;

void setup() {
    // Démarre les composants et la communication série
    Serial.begin(9600);
    swSerial.begin(9600);
    inputString.reserve(20);

    Receiver.enableIRIn(); // Démarre la réception
    pinMode(BLUE, OUTPUT);
    pinMode(RED, OUTPUT);
    pinMode(GREEN, OUTPUT);
    pinMode(BUZZER, OUTPUT);
    pinMode(LOCK, OUTPUT);
    digitalWrite(LOCK, HIGH); // Verrou sur la position fermée

    // Tant que la communication série n'est pas prête
    while(!Serial);
    Serial.println("Starting wifi");
    // Configure la communication wifi en mode TCP
    wifi.setTransportToTCP();
    // Configure la fin de la transmission wifi avec un saut de ligne
    wifi.endSendWithNewline(true);
    // Démarre la communication wifi
    wifi.begin();
    // Se connecte au réseau wifi défini avec le ssid et le password
    wifi.connectToAP(ssid, password);
    // Se connecte au socket du programme Python
    wifi.connectToServer("192.168.122.103", "9999");
}
```

```
void loop() {
  if (!wifi.isStarted()) {
    wifi.begin();
  }

  // Vérifie si la longueur du code reçu n'est pas égale à la
  // longueur du code attendu
  if (test_code.length() != code.length()) {
    if (Receiver.decode(&decodedir)) {
      msg = decode_ir.value;
      Serial.println(msg);

      // Si les 3 premiers caractères sont égaux à "167"
      if (msg.substring(0, 3) == "167") {
        test_code.concat(msg); // On ajoute chaque morceaux du
                               // message à la variable "msg"

        Serial.println(test_code);
        digitalWrite(BLUE, HIGH); //Allume la LED bleue
        tone(BUZZER, 300, 50); // Le Buzzer sonne
        delay(100); // Pause de 0.1s
        noTone(BUZZER); // Le Buzzer ne sonne plus
        delay(100); // Pause de 0.1s
      }
      Receiver.resume(); // Reçoit le prochain code
    }
  }

  else {
    if (test_code == code) {
      digitalWrite(GREEN, HIGH); // La LED verte s'allume pour
                                  // indiquer que le code est bon
      analogWrite(LOCK, LOW); // Le verrou s'ouvre (non aimanté)
      tone(BUZZER, 400, 100); // Le Buzzer sonne
      delay(500); // Pause de 0.5s
      noTone(BUZZER); // Le Buzzer ne sonne plus
      test_code = ""; // test_code redevient vide
      wifi.send(SERVER, "LoR");
      delay(10000); // Pause de 10s
    }
  }
}
```



```
else {
    digitalWrite(RED, HIGH); // La LED rouge s'allume pour
                             // indiquer que le code est incorrect
    tone(BUZZER, 200, 50); // Le Buzzer sonne
    delay(100); // Pause de 0.1s
    noTone(BUZZER); // Le Buzzer ne sonne plus
    test_code = ""; // test_code redevient vide
    wifi.send(SERVER, "LoE"); // Envoie le message "LoE"
    delay(1000); // Pause de 1s
}
}

delay(100); // Pause de 0.1s
digitalWrite(LOCK, HIGH); // Verrou fermé (aimanté)
digitalWrite(BLUE, LOW); // LED bleue éteinte
delay(100); // Pause de 0.1s
digitalWrite(RED, LOW); // LED rouge éteinte
digitalWrite(GREEN, LOW); // LED verte éteinte
}
```

## Le jeu du Simon

Le jeu du Simon est un jeu de mémoire et de réflexion. Il consiste en un dispositif comprenant quatre boutons de couleur (généralement rouge, jaune, bleu et vert) qui s'allument dans un certain ordre. Le joueur doit reproduire la séquence d'allumage en appuyant sur les boutons dans le bon ordre. A chaque fois que le joueur réussit à reproduire la séquence, celle-ci s'allonge d'un cran et ainsi de suite jusqu'à la dernière étape. Si le joueur réussit chaque étape sans se tromper, il gagne la partie.

**Voici un exemple du jeu du Simon :**



## Notre reprise du jeu :

Notre but est de reproduire le jeu du Simon à l'aide d'un code et d'une carte Arduino, de LEDs, de boutons et d'un buzzer pour faire du son.

Pour démarrer le jeu, le joueur doit appuyer sur le bouton tout à droite (celui qui n'est pas devant une LED et qui est à l'opposé du buzzer).

Une fois le bouton appuyé, une animation sonore et lumineuse se joue. C'est le signe que la partie va commencer.

Une fois l'animation terminée, le premier code couleur se lance. Il faut appuyer sur les boutons situés devant les LED correspondantes pour reproduire le code.

Si le code reproduit est le bon, alors une nouvelle animation s'effectue avant de lancer le code suivant. Dans le cas contraire, le LED rouge s'allume, il faut donc recommencer en appuyant sur le bouton de démarrage. Il y a en tout 5 évolutions du code couleur. Il faut réussir les 5 de suite pour gagner le jeu.

Un buzzer est installé pour émettre un son différent pour chaque LED. A chaque fois qu'une LED s'allume, le buzzer joue le son correspondant.

Lorsque le joueur appuie sur un bouton, la LED correspondante s'allume.

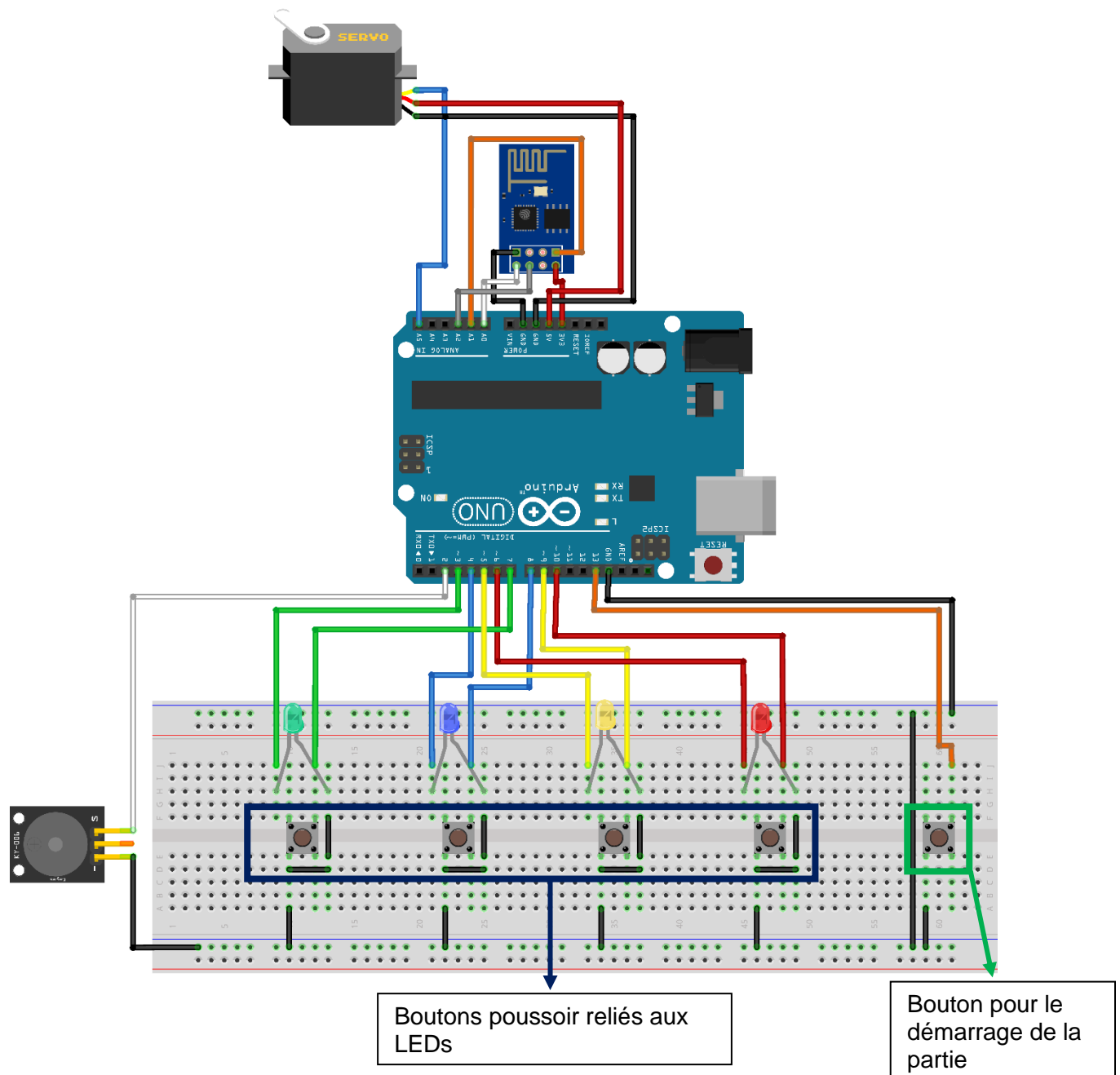
Si le joueur gagne, cela déclenchera un servomoteur qui dévoilera un code.

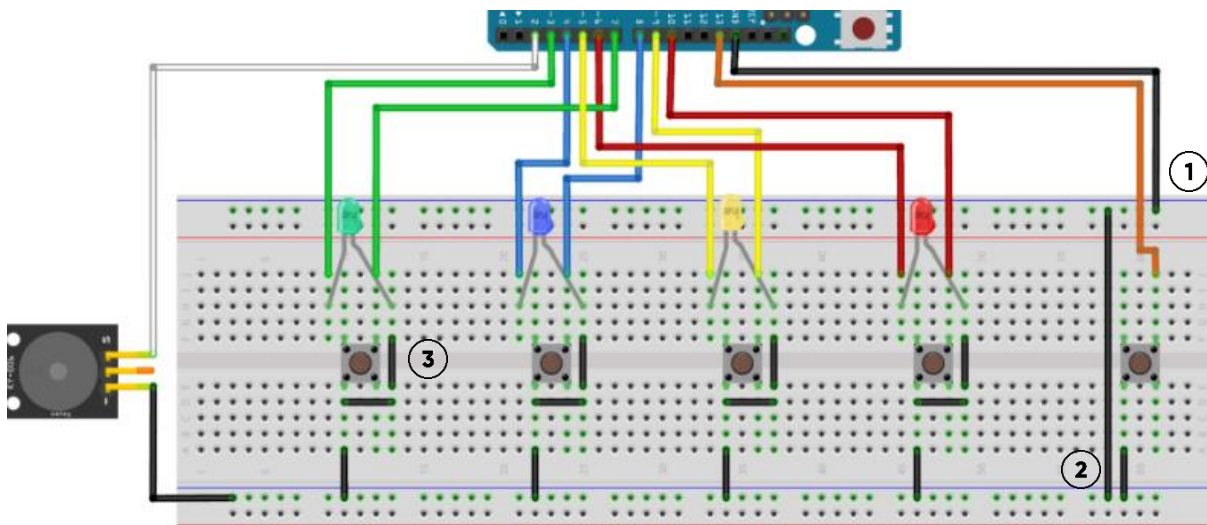
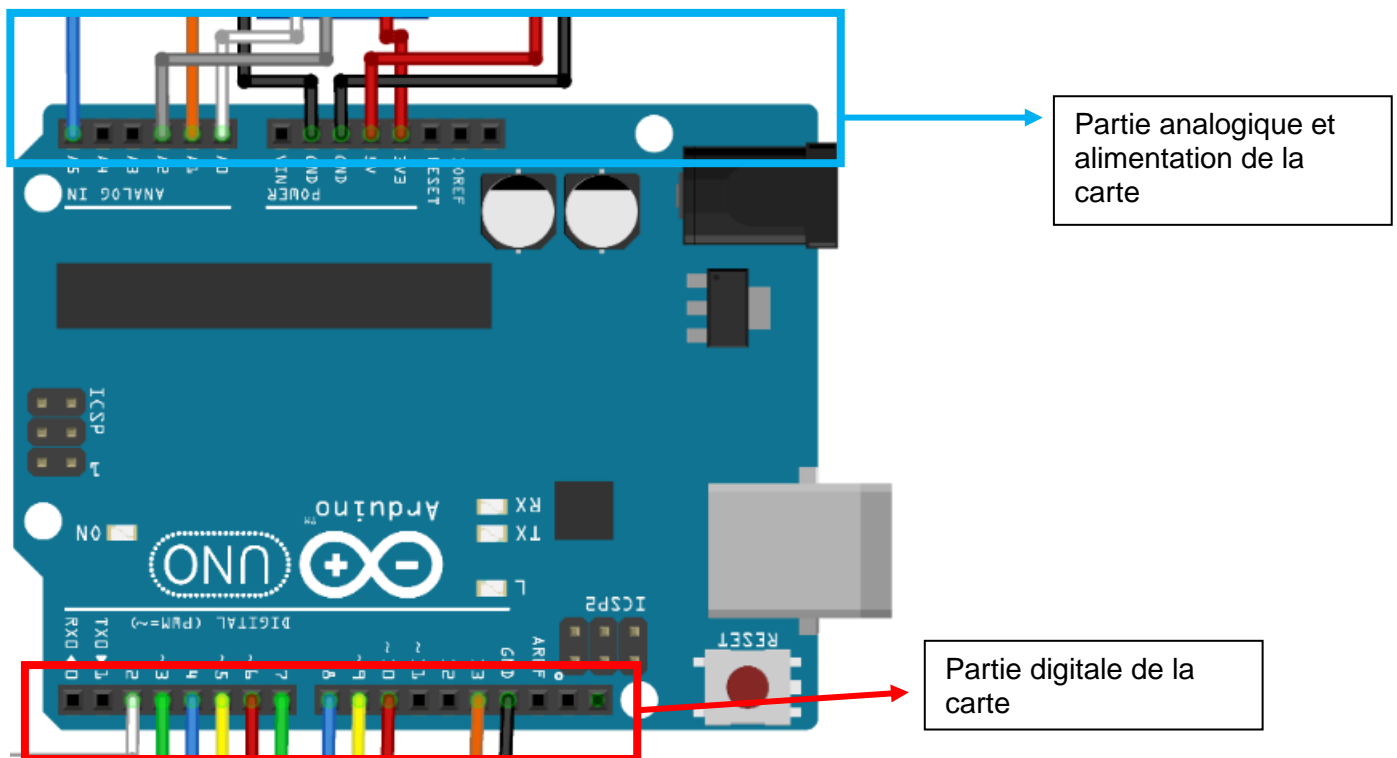
Comment avons-nous réalisé ce jeu ?

Matériel nécessaire :

- Carte Arduino Uno
- Tablette électronique
- 4 LEDs (rouge, bleue, jaune, et verte)
- 5 boutons
- 1 Buzzer
- Des fils électroniques
- 1 ESP8266

Montage détaillé et explications :





On peut tout d'abord voir que l'alimentation de tous les composants se fait via la carte Arduino. Pour la **masse** (pin « GND ») :

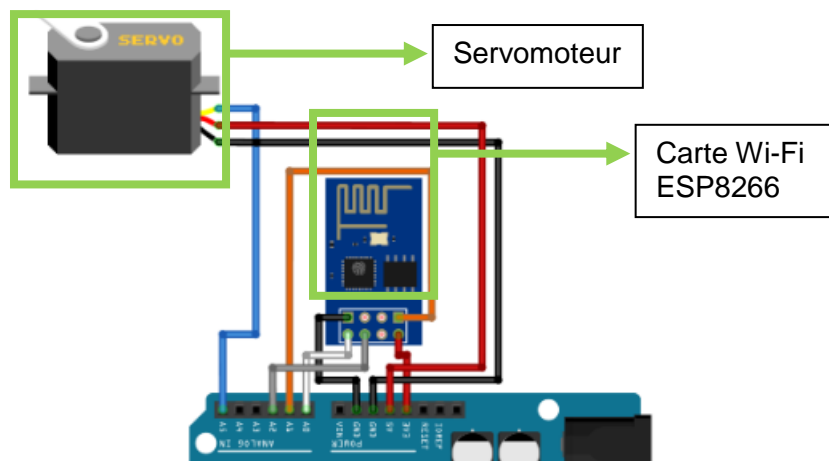
1. Elle est premièrement branchée en haut de la « breadbord »
2. Elle est redistribuée vers le bas de la breadbord pour pouvoir lier les boutons et le buzzer à la masse.
3. Les câbles relient la masse au bouton et la redirigent vers les LEDs.

Ce procédé de « redistribuer » la masse permet de faciliter le branchement et d'éviter l'utilisation excessive de câbles.

Les broches analogiques sont ensuite utilisées en fonction des besoins du code. Vous trouverez ci-dessous un tableau récapitulatif des pins utilisés, ainsi que le composant auquel ils sont reliés :

Tableau récapitulatif des broches DIGITALES pour le jeu du Simon	
2	Buzzer
3	LED Verte
4	LED Bleue
5	LED Jaune
6	LED Rouge
7	Bouton pour LED Verte
8	Bouton pour LED Bleue
9	Bouton pour LED Jaune
10	Bouton pour LED Rouge
13	Bouton pour lancer le jeu

Partie analogique de la carte :



On peut remarquer que cette partie est très différente de la précédente. Seulement 2 composants utilisent un bon nombre de broches mais cela est dû au fait que ces derniers ont besoin de plusieurs broches pour pouvoir fonctionner convenablement. Sous la forme d'un tableau nous allons récapituler quelle broche est branchée à quelle partie de quel composant :

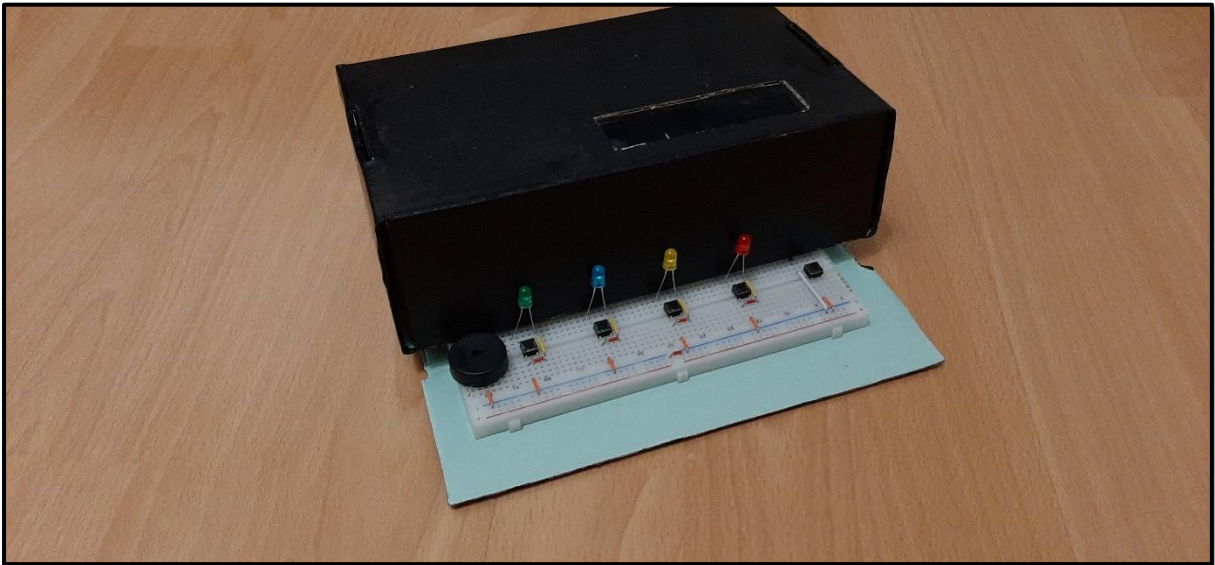
Tableau récapitulatif des broches ANALOGIQUES pour le jeu du Simon	
A0	Broche Tx <sup>1</sup> de la Carte Wi-Fi
A1	Broche Rx <sup>2</sup> de la Carte Wi-Fi
A2	CH_PD <sup>3</sup> de la Carte Wi-Fi
A5	Servomoteur
GND	Masse du servomoteur
GND	Masse de la Carte Wi-Fi
5V	Alimentation du servomoteur
3.3V	Alimentation de la Carte Wi-Fi

<sup>1</sup> Broche de transfert des données

<sup>2</sup> Broche de réception des données

<sup>3</sup> Broche de reset

Voici le Simon dans sa boîte :



Voici le code commenté :

```
// Importe la bibliothèque IRremote pour utiliser les
fonctionnalités de la télécommande infrarouge
#include <IRremote.h>

// Importe la bibliothèque SoftwareSerial pour utiliser les
fonctionnalités de la communication série logicielle
#include <SoftwareSerial.h>

// Importe la bibliothèque SerialESP8266wifi pour utiliser les
fonctionnalités de la communication wifi
#include <SerialESP8266wifi.h>

// Serial config
#define sw_serial_rx_pin A0 // Pin vers TX
#define sw_serial_tx_pin A1 // Pin vers RX
#define esp8266_reset_pin A2 // Pin vers CH_PD, pas reset

// Crée une instance de la communication série logicielle en
utilisant les broches définies précédemment
SoftwareSerial swSerial(sw_serial_rx_pin, sw_serial_tx_pin);

// Crée une instance de la communication wifi en utilisant
l'instance de la communication série logicielle et la broche de
réinitialisation définies précédemment
SerialESP8266wifi wifi(swSerial, swSerial, esp8266_reset_pin,
Serial);

// User config
#define ssid "ArduinoEarth" // Wifi SSID
#define password "no_earthquake" // Wifi Password

String inputString;

Servo servo;
int pos = 90;

int y = 0;
String reponse = ""; // Initialisation de la variable de réponse qui
se remplira avec les valeurs des boutons
appuyés par les joueurs
```



```
const int GREEN = 3;
const int BLUE = 4;
const int YELLOW = 5;
const int RED = 6;
const int GREEN_BTN = 7;
const int BLUE_BTN = 8;
const int YELLOW_BTN = 9;
const int RED_BTN = 10;
const int BUZZER = 2;
const int RESET = 13;

const int SERVO = A5;

// Déclare un tableau de chaînes de caractères (un tableau de pointeurs de char) nommé "codes". Les chaînes de caractères qui sont incluses dans le tableau sont des séquences de chiffres qui représentent les combinaisons de couleurs à afficher dans le jeu du Simon. Chaque chaîne de caractères est une séquence de chiffres séparés par des virgules, où chaque chiffre correspond à une couleur (3 -> vert, 4 -> bleu, 5 -> jaune, 6 -> rouge)
const char* codes[] = {"4,5,3", "4,5,3,6,5,4,5", "4,5,3,6,5,4,5,6,3,4,6", "4,5,3,6,5,4,5,6,3,4,6,4,6,3,6,4", "4,5,3,6,5,4,5,6,3,4,6,4,6,3,6,4,5,3,6,5"};

// Le tableau de structure "Note" nommé "notes" est déclaré et initialisé avec quatre éléments, chacun associant une fréquence de son à un code de couleur
struct Note {
    int frequency;
    char code;
};
Note notes[] = {
    {300, '3'},
    {350, '4'},
    {400, '5'},
    {450, '6'}
};

unsigned long startTime;
```



```
void setup() {
    // Démarre les composants et la communication série
    Serial.begin(9600);
    inputString.reserve(20);
    swSerial.begin(9600);

    // Place le servomoteur à 90°
    servo.attach(SERVO);
    servo.write(pos);

    pinMode(GREEN, OUTPUT);
    pinMode(BLUE, OUTPUT);
    pinMode(YELLOW, OUTPUT);
    pinMode(RED, OUTPUT);
    pinMode(GREEN_BTN, INPUT_PULLUP);
    pinMode(BLUE_BTN, INPUT_PULLUP);
    pinMode(YELLOW_BTN, INPUT_PULLUP);
    pinMode(RED_BTN, INPUT_PULLUP);
    pinMode(BUZZER, OUTPUT);
    pinMode(RESET, INPUT_PULLUP);

    // Tant que la communication série n'est pas prête
    while(!Serial);
    Serial.println("Starting wifi");
    // Configure la communication wifi en mode TCP
    wifi.setTransportToTCP();
    // Configure la fin de la transmission wifi avec un saut de ligne
    wifi.endSendWithNewline(true);
    // Démarre la communication wifi
    wifi.begin();
    // Se connecte au réseau wifi défini avec le ssid et le password
    wifi.connectToAP(ssid, password);
    // Se connecte au socket du programme Python
    wifi.connectToServer("192.168.122.103", "9999");
}

void play_note(char code, int del) {
    for (int i = 0; i<4; i++) {
        if (notes[i].code == code) {
            tone(BUZZER, notes[i].frequency, del);
            delay(del*2);
            noTone(BUZZER);
            break;
        }
    }
}
```

```
void boutons() {
  if (digitalRead(GREEN_BTN) == LOW) {
    reponse += 3;
    digitalWrite(GREEN, HIGH);
    play_note('3', 50);
    digitalWrite(GREEN, LOW);
    Serial.println(reponse);
    startTime = millis();
    delay(500);
  }

  else if (digitalRead(BLUE_BTN) == LOW) {
    reponse += 4;
    digitalWrite(BLUE, HIGH);
    play_note('4', 50);
    digitalWrite(BLUE, LOW);
    Serial.println(reponse);
    startTime = millis();
    delay(500);
  }

  else if (digitalRead(YELLOW_BTN) == LOW) {
    reponse += 5;
    digitalWrite(YELLOW, HIGH);
    play_note('5', 50);
    digitalWrite(YELLOW, LOW);
    Serial.println(reponse);
    startTime = millis();
    delay(500);
  }

  else if (digitalRead(RED_BTN) == LOW) {
    reponse += 6;
    digitalWrite(RED, HIGH);
    play_note('6', 50);
    digitalWrite(RED, LOW);
    Serial.println(reponse);
    startTime = millis();
    delay(500);
  }
}
```

```
void animationSuccess() {
    digitalWrite(GREEN, HIGH);
    play_note('3', 40);
    digitalWrite(GREEN, LOW);
    digitalWrite(BLUE, HIGH);
    play_note('4', 40);
    digitalWrite(BLUE, LOW);
    digitalWrite(YELLOW, HIGH);
    play_note('5', 40);
    digitalWrite(YELLOW, LOW);
    digitalWrite(RED, HIGH);
    play_note('6', 40);
    digitalWrite(RED, LOW);
    delay(100);

    digitalWrite(GREEN, HIGH);
    digitalWrite(BLUE, HIGH);
    digitalWrite(YELLOW, HIGH);
    digitalWrite(RED, HIGH);
    tone(BUZZER, 500, 100);
    delay(200);
    noTone(BUZZER);
    digitalWrite(GREEN, LOW);
    digitalWrite(BLUE, LOW);
    digitalWrite(YELLOW, LOW);
    digitalWrite(RED, LOW);
}

void doCode(char* code_source) {
    char* code = strdup(code_source);

    String codeTest(code);
    codeTest.replace(", ", "");

    char* codeArr = strtok(code, ",");

    while (codeArr != NULL) {
        for (int i = 0; i < 4; i++) {
            if (notes[i].code == *codeArr) {
                tone(BUZZER, notes[i].frequency, 50);
                digitalWrite(atoi(codeArr), HIGH);
                delay(500);
                digitalWrite(atoi(codeArr), LOW);
                break;
            }
        }
    }
}
```

```

    codeArr = strtok(NULL, ",");
}

free(code);
reponse = "";
startTime = millis();

while (reponse.length() != codeTest.length()) {
    boutons();
    if(millis() - startTime >= 10000) {
        break;
    }
}

if (reponse == codeTest) {
    animationSuccess();
}

else if (reponse != codeTest){
    digitalWrite(RED, HIGH);
    tone(BUZZER, 200, 250);
    delay(500);
    noTone(BUZZER);
    digitalWrite(RED, LOW);
    wifi.send(SERVER, "SiF");
    reponse = "";
    y = 0;
}
}

void win() {
    servoOpen();
    for (int i = 0; i<4; i++) {
        digitalWrite(GREEN, HIGH);
        digitalWrite(BLUE, HIGH);
        digitalWrite(YELLOW, HIGH);
        digitalWrite(RED, HIGH);
        tone(BUZZER, 500, 250);
        delay(500);
        noTone(BUZZER);
        digitalWrite(GREEN, LOW);
        digitalWrite(BLUE, LOW);
        digitalWrite(YELLOW, LOW);
        digitalWrite(RED, LOW);
        delay(500);
    }
}

```

```
wifi.send(SERVER, "SiR");
delay(50000);
digitalWrite(RED, HIGH);
tone(BUZZER, 500, 250);
servoClose();
noTone(BUZZER);
digitalWrite(RED, LOW);
}

void servoOpen() {
  for (pos = 90; pos <= 180; pos += 1) {
    servo.write(pos);
    delay(15);
  }
}

void servoClose() {
  for (pos = 180; pos >= 90; pos -= 1) {
    servo.write(pos);
    delay(15);
  }
}

void loop() {
  if (!wifi.isStarted()) {
    wifi.begin();
  }

  if ((digitalRead(RESET) == LOW) && (y == 0)) {
    Serial.println("OKAY !");
    jeu();
  }
}
```

```
void jeu() {  
  
    // Initialisation du jeu  
    if (y == 0) {  
        for (int i=0; i<6; i++) {  
            digitalWrite(GREEN, HIGH);  
            delay(75);  
            digitalWrite(GREEN, LOW);  
            play_note('3', 40);  
            digitalWrite(BLUE, HIGH);  
            delay(75);  
            digitalWrite(BLUE, LOW);  
            play_note('4', 40);  
            digitalWrite(YELLOW, HIGH);  
            delay(75);  
            digitalWrite(YELLOW, LOW);  
            play_note('5', 40);  
            digitalWrite(RED, HIGH);  
            delay(75);  
            digitalWrite(RED, LOW);  
            play_note('6', 40);  
        }  
        y = 1;  
  
        for (int cds = 0; cds < 5; cds++) {  
            delay(1000);  
            Serial.println(codes[cds]);  
            doCode(codes[cds]);  
            if (y == 0) {  
                break;  
            }  
        }  
        if (y != 0) {  
            y = 0;  
            win();  
        }  
    }  
}
```

### Explication des fonctions :

#### **"Play\_note()" :**

La fonction "play\_note()" est utilisée pour jouer la note correspondant à la couleur du bouton pressé. Elle prend en paramètre un code de couleur (un char) et un délai (un entier) et effectue les actions suivantes :

- Elle parcourt le tableau "notes" en utilisant une boucle "for" et vérifie si l'élément courant du tableau a un code de couleur correspondant au code de couleur passé en entrée ;
- Si le code de couleur correspond, la fonction "tone()" est utilisée pour jouer la fréquence de son associée à cet élément du tableau, sur la broche du buzzer. Elle utilise également la fonction "delay()" pour attendre le délai spécifié. Ensuite, la fonction "noTone()" pour arrêter le son ;
- Si le code de couleur ne correspond pas, la fonction passe à l'élément suivant du tableau et recommence la vérification. Si aucun élément du tableau n'a le bon code de couleur, la fonction ne fait rien.

La fonction "play\_note()" est appelée dans la fonction "boutons()" chaque fois qu'un bouton de couleur est pressé, avec le code de couleur du bouton et un délai de 50 millisecondes en entrée. Cela permet de jouer la note de couleur correspondante lorsque le bouton est pressé.

#### **Fonction "Boutons()" :**

La fonction "boutons()" est utilisée pour gérer les interactions du joueur avec les boutons de couleur. Lorsqu'un bouton est pressé, la fonction effectue les actions suivantes :

- Elle ajoute le code de couleur correspondant à la variable "reponse" ;
- Elle allume la LED de couleur correspondante en utilisant la fonction "digitalWrite()" ;
- Elle joue la note de couleur correspondante en appelant la fonction "play\_note()" ;
- Elle éteint la LED de couleur correspondante en utilisant la fonction "digitalWrite()" ;
- Elle affiche la chaîne de caractères "reponse" dans la console série en utilisant la fonction "Serial.println()" ;
- Elle met à jour la variable "startTime" avec le temps actuel en millisecondes en appelant la fonction "millis()" ;
- Elle attend 500 millisecondes en appelant la fonction "delay()".

La fonction "boutons()" vérifie l'état des quatre boutons de couleur en appelant la fonction "digitalRead()" sur chaque broche de bouton. Si l'état d'un bouton est "LOW" (c'est-à-dire que le bouton est pressé), la fonction effectue les actions décrites ci-dessus pour ce bouton. Si aucun bouton n'est pressé, la fonction ne fait rien.

### **Fonction "animationSuccess()" :**

La fonction "animationSuccess()" est utilisée pour afficher une animation de succès lorsque le joueur a réussi à reproduire correctement une séquence de couleurs. Elle effectue les actions suivantes :

- Elle allume successivement chaque LED de couleur en utilisant la fonction "digitalWrite()" et joue la note de couleur correspondante en appelant la fonction "play\_note()" ;
- Elle attend 100 millisecondes en appelant la fonction "delay()" ;
- Elle allume toutes les LEDs de couleur en utilisant la fonction "digitalWrite()" et joue une note de haute fréquence sur la broche du buzzer en utilisant la fonction "tone()". Elle attend 200 millisecondes en appelant la fonction "delay()" ;
- Elle éteint toutes les LEDs de couleur en utilisant la fonction "digitalWrite()" et arrête de jouer la note en utilisant la fonction "noTone()".

### **Fonction "doCode()" :**

La fonction "doCode()" fait les actions suivantes :

- Elle copie la séquence de couleurs dans une nouvelle chaîne de caractères "code" en utilisant la fonction "strdup()" ;
- Elle crée une nouvelle chaîne de caractères "codeTest" à partir de "code" et supprime les virgules de cette chaîne en utilisant la fonction "replace()" ;
- Elle décompose la chaîne de caractères "code" en une série de codes de couleur en utilisant la fonction "strtok()" ;
- Elle parcourt chaque code de couleur en utilisant une boucle "while" et joue la note de couleur correspondante en appelant la fonction "play\_note()". Elle allume également la LED de couleur correspondante en utilisant la fonction "digitalWrite()" ;
- Elle libère la mémoire allouée à "code" en appelant la fonction "free()" ;
- Elle réinitialise la chaîne de caractères "reponse" et enregistre le temps actuel en utilisant la fonction "millis()" ;
- Elle utilise une boucle "while" pour attendre que le joueur ait entré tous les codes de couleur de la séquence ou que 10 secondes se soient écoulées. Pendant cette boucle, elle appelle la fonction "boutons()" pour détecter les entrées du joueur et met à jour la chaîne "reponse" ;



Si la chaîne "reponse" du joueur correspond à la chaîne "codeTest", cela signifie que le joueur a correctement reproduit la séquence de couleurs et la fonction "animationSuccess()" est appelée pour jouer une animation de réussite.

Sinon, si la chaîne "reponse" du joueur ne correspond pas à la chaîne "codeTest", cela signifie que le joueur n'a pas correctement reproduit la séquence de couleurs. La fonction allume la LED rouge et fait sonner le buzzer, puis envoie un message au serveur en appelant la fonction "wifi.send()". Elle réinitialise également la chaîne "reponse" et remet la variable "y" à 0.

### **Fonction "win()" :**

La fonction "win()" est appelée lorsque le joueur a réussi à reproduire toutes les séquences de couleurs du jeu du Simon. Elle envoie un message au serveur en appelant la fonction "wifi.send()", puis fait clignoter toutes les LEDs, commande au servomoteur de s'ouvrir et fait sonner le buzzer en boucle pendant quelques secondes pour célébrer la victoire du joueur.

### **Fonction "loop()" :**

Si le bouton de réinitialisation (RESET) est enfoncé et que la variable "y" est égale à 0, cela signifie que le jeu vient de démarrer ou qu'il vient d'être réinitialisé. Dans ce cas, la fonction appelle la fonction "jeu()" pour démarrer ou redémarrer le jeu.

### **Fonction "jeu()" :**

La fonction "jeu()" initialise le jeu et gère les différentes étapes de celui-ci. Elle s'exécute lorsque la variable "y" est égale à 0.

La fonction commence par une boucle qui allume et éteint les différentes LEDs et joue la note correspondante pour montrer que le jeu se lance.

Ensuite, la variable "y" est mise à 1.

Puis, une boucle itère 5 fois. A chaque itération, la fonction "doCode()" est appelée avec en paramètre un élément du tableau "codes". La boucle s'arrête si la variable "y" est mise à 0 dans la fonction "doCode()".

Si la boucle se termine normalement, c'est-à-dire si "y" n'a pas été mise à 0, alors la fonction "win()" est appelée et la variable "y" est mise à 0.

## **Conclusion**

Au cours de cette SAÉ, nous avons pu mettre en pratique nos connaissances acquises en électronique, en Python, mais aussi en bases de données et en développement web, en faisant en plus référence à des compétences passées lors de la première année, en mettant en place un serveur Asterisk.

Cette SAE nous a encore une fois permis de développer notre travail d'équipe et notre cohésion étant donné que chacun s'occupait d'une partie qui ne pouvait pas exister sans les autres.

Nous avons dû travailler différemment en centralisant toutes nos productions et en établissant un cadre précis pour que notre travail soit fluide, efficace et de qualité.

Malgré les nombreux problèmes rencontrés, nous avons su retomber trouver des solutions rapidement et efficacement, afin que cela n'impacte pas le déroulé de la production et que le rendu final se rapproche au plus de ce que nous avons imaginé au départ.

Enfin, voici le lien vers une vidéo montrant l'Escape Game en fonctionnement et expliquant les aspects principaux de celui-ci : [Vidéo Présentation de l'Escape Game](#)