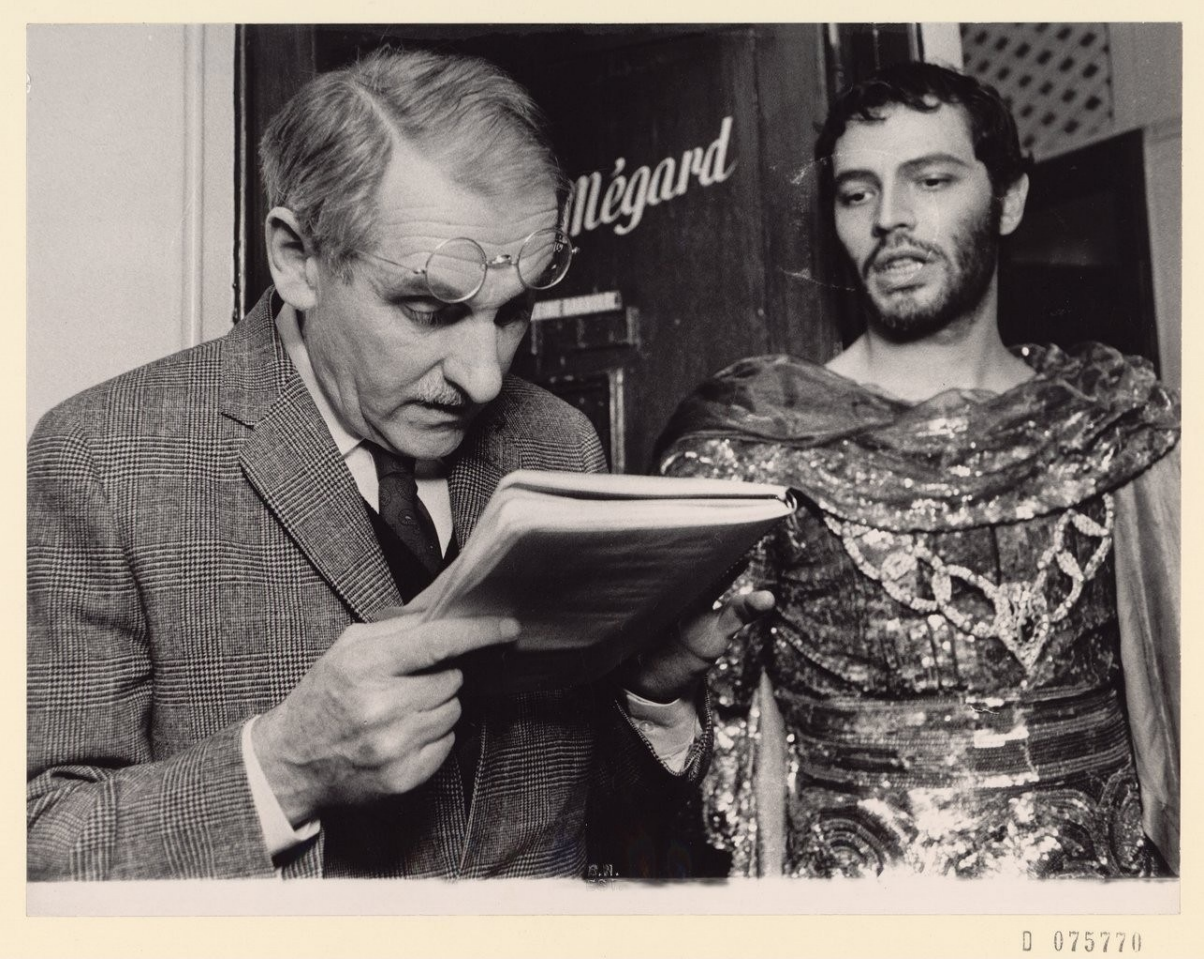
Table des matières

Conclusion

Préambule



Source gallica.bnf.fr / Bibliothèque nationale de France

Objectifs

Dans le cadre d'un projet domotique lié à une pièce de théâtre réalisé par une classe de première ES de notre lycée, nous avons été consulté afin de concevoir un "pupitre" accompagné de plusieurs lumières qui s'allument et s'éteignent de façon aléatoire et d'une musique de fond.

Ce projet est en vérité un sous-projet, en effet ce pupitre sera joint à d'autres projets tel qu'un ascenseur ou une application mobile capable d'orchestrer le pupitre et l'ascenseur.

l'Équipe

L'équipe en charge du pupitre est composé de quatre personne : Rached MEJRI, Hugo BESSARD, Lucas THOMAS et Methéo PIRAT. Chaque membre est en charge d'un périmètre spécifique en lien avec le pupitre.

Date limite

Nous avons cinq séances de quatre heures afin de réaliser le projet.

NOTE: Le projet à été travaillé en dehors des heures scolaires, ainsi, la date limite fixé en fonction des séances n'est pas représentatif du travail fourni.

Avant propos

Nous avions réalisé en début de projet un programme spécifique à la carte Arduino afin de gérer le son et la lumière du pupitre.

Suite à plusieurs contraintes tel que la gestion de la concurrence et le manque de documentation viable concernant un module lié au son, nous nous sommes plutôt tourné vers le nano-ordinateur Raspberry PI afin de réaliser notre projet.

Vous pouvez consulter la page des problèmes rencontrés afin d'en savoir plus.

Installer l'ensemble du projet sur un Raspberry PI

NOTE : Si les instructions ci-dessous vous posent problème, vous pouvez passer cette page et mettre en place pas à pas le projet.

Ce gitbook explique pas à pas via les présentations des différentes technologies et la présentation des étapes de l'élaboration du projet comment mettre en place le projet.

En contre partie, cette page propose pour les utilisateurs avancées l'ensemble des commandes et configuration à faire afin d'implémenter ce projet.

Dépendances

Depuis pypi.org

- pyserial
- RPi.GPIO
- omxplayer-wrapper

```
sudo pip3 install pyserial RPi.GPIO omxplayer-wrapper
```

Sur le Raspberry PI cible

- Python >= 3.5
- omxplayer

```
sudo apt update; sudo apt upgrade
sudo apt install omxplayer python3
```

Installation

```
git clone https://github.com/Reachip/projet-theatre
sudo python3 projet-theatre/pypitre
```

Utilisez le fichier `config.json` afin d'indiquer au programme les trois PIN GPIO qui serviront à allumer les lumières et ou trouver le fichier qui jouera la musique.

Mise en place d'un deamon

La mise en place de deamon est disponible ici.

Chaîne d'information et d’énergie

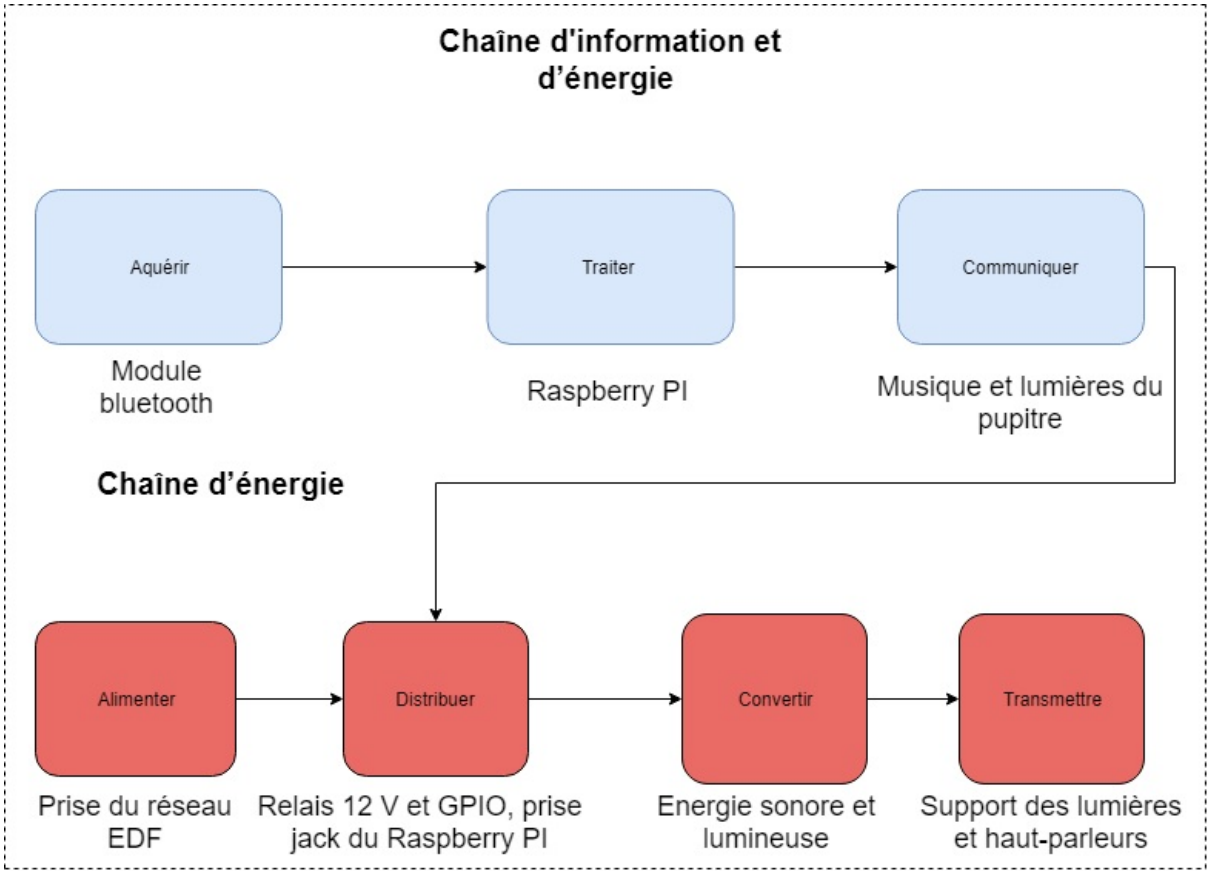


Diagramme des exigences

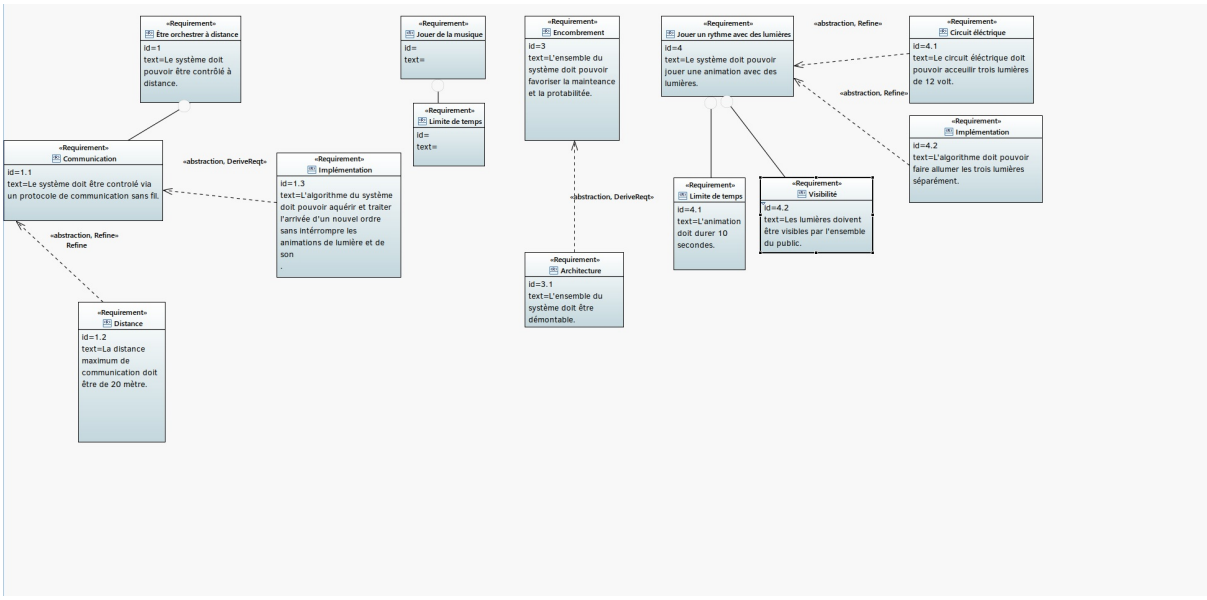


Diagramme de contexte

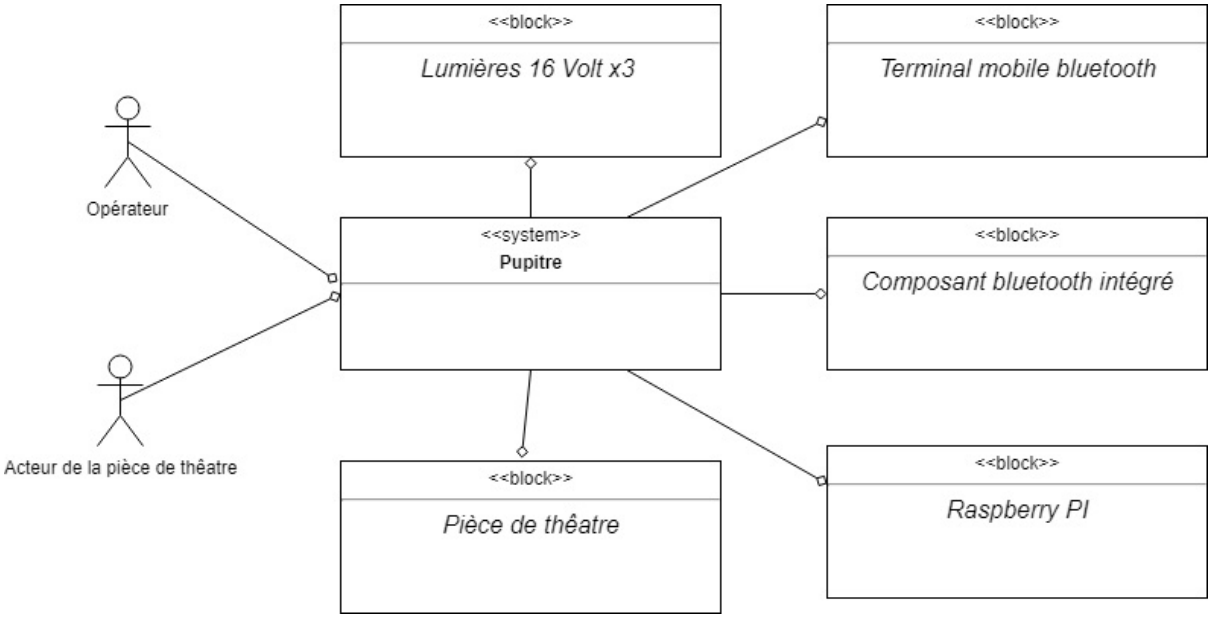
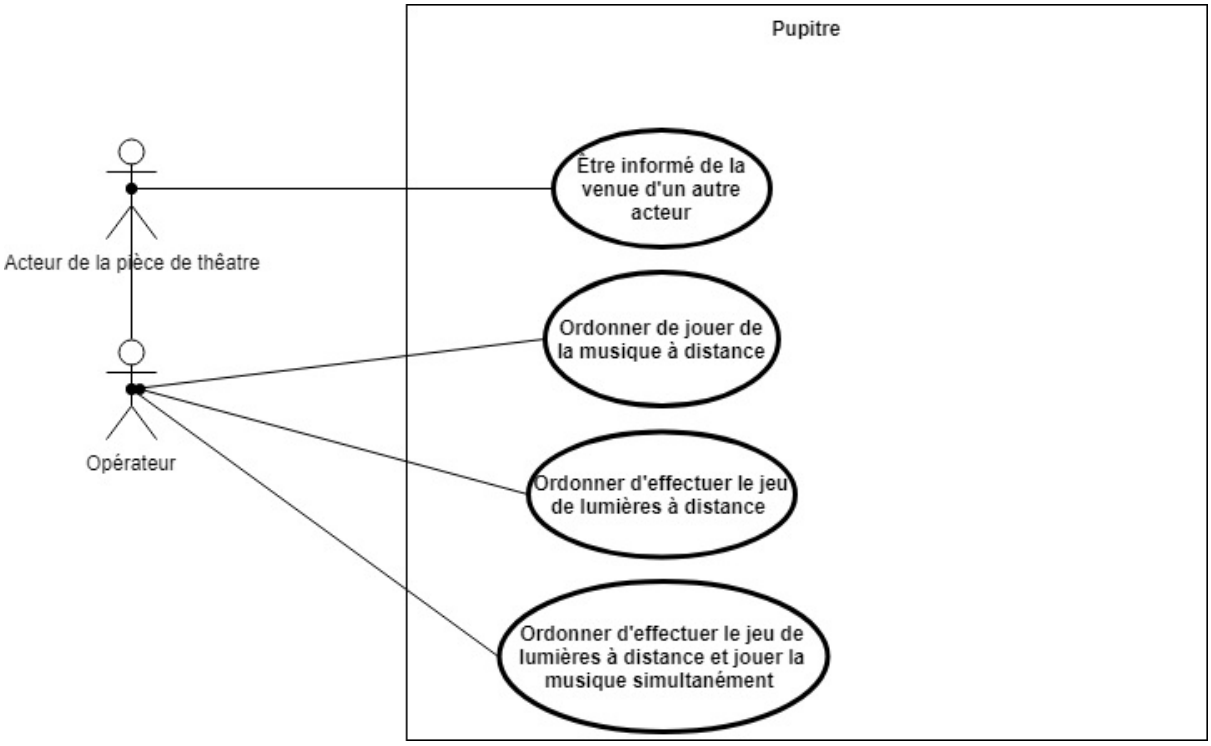


Diagramme de cas d'utilisation



Etapes de mise en oeuvre

- Remplir la fiche de validation.
- Réaliser les différents digrammes.
- Organiser l'équipe vers les différents sujets transverses.
- Construire le pupitre.
- Disposer les lumières sur le pupitre.
- Écrire le programme charger d'allumer les lumières aléatoirement et de jouer la musique de fond en parallèle graçe à l'envoi d'informations par liaison bluetooth.
- Souder le montage.

Code source

Suite à plusieurs problématiques rencontrées lors du projet, nous avons abandonné le développement du projet à l'aide de l'environnement Arduino au profit du Raspberry PI. **Ainsi, le code-source Arduino est malheureusement manquant.**

Avec le Raspberry PI

Le code source du projet pupitre implémenté en langage Python est disponible à cette adresse : https://github.com/projet-theatre/pupitre

A ne pas oublier : L'arboresence suivante est proposé afin de regrouper tous les fichiers nécessaires au projet

```
pupitre
├── LICENSE
├── Makefile
├── README.md
├── __init__.py
├── __main__.py
├── config.json
└── core
    ├── __init__.py
    ├── event.py
    └── light.py
```

Le fichier Makefile est expliqué plus loin sur ce gitbook. Il n'est pas obligatoire mais il permet de gagner en productivité.

```
// Fichier config.json
// Il permet de livrer un programme générique,
// donc facilement personnalisable sans devoir éditer directement
// l'implémentation en Python.
{
    // Les PIN en liaison avec les différents relais de lumières
    // Ici, il en existe trois mais il peux en avoir plus ou moins.
    "leds_handlers": [17, 20, 21],

    // Le chemin absolu vers le fichier audio.
    "music_path": "/home/pi/buzzer.mp3",

    // Le nombre de fois ou les animations de lumières
    // seront effectuées.
    "animation_iterations": 5
}
```

```


# Fichier __main__.py
```

```
import json
import time
import logging
from logging.handlers import RotatingFileHandler
from pathlib import Path

from omxplayer.player import OMXPlayer
import RPi.GPIO as GPIO

from core import event
from core import light

# Mise en place d'un fichier de log
# afin d'obtenir un rapport d'anomalies
# probables.
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

formatter = logging.Formatter("%(asctime)s :: %(levelname)s :: %(message)s")
file_handler = RotatingFileHandler("/home/pi/pupitre.log", "a", 1000000, 1)
file_handler.setLevel(logging.DEBUG)
file_handler.setFormatter(formatter)
logger.addHandler(file_handler)

stream_handler = logging.StreamHandler()
stream_handler.setLevel(logging.DEBUG)
logger.addHandler(stream_handler)

logger.info("le script est initialisé")

event = event.Event()
event.start()


# Ouverture de fichier de configuration
with open("/home/pi/pupitre/config.json") as json_file:
    global json_datas
    # On récupère dans un dictionnaire
    # les pins chargés d'allumer les lumères
    # puis le chemin absolu vers le fichier audio.
    json_datas = json.load(json_file)

GPIO.setmode(GPIO.BCM)
[
    GPIO.setup(handler, GPIO.OUT, initial=GPIO.HIGH)
    for handler in json_datas["leds_handlers"]
]

logger.info("GPIO initialisés")


@event.on_with_just_sound
def with_just_sound():
  """ Quand l'utilisateur ne demande qu'à avoir du son ... """
  logger.info("demande de jouer le son")
  path = Path(json_datas["music_path"])
  OMXPlayer(path)
```

```
  logger.info("jouer le son fini")


@event.on_with_sound_and_leds
def with_sound_and_led():
  """ Quand l'utilisateur demande à avoir du son  et l'animation des lumières ... """
  logger.info("jouer le son et animer les lumières demandé")
  path = Path(json_datas["music_path"])
  OMXPlayer(path)
  light.animation(5, json_datas["leds_handlers"])
  logger.info("jouer le son et animer les lumières fini")


@event.on_with_leds
def with_leds():
  """ Quand l'utilisateur ne demande qu'à avoir l'animation des lumières ... """
  logger.info("animer les lumières")
  light.animation(json_datas["animation_iterations"], json_datas["leds_handlers"])
  logger.info("animer les lumières fini")


event.join()
```

```
# Fichier core/event.py
import logging
import threading
from logging.handlers import RotatingFileHandler

import serial
import RPi.GPIO as GPIO


class Event(threading.Thread):
    """
    Cette classe se sert du pattern dispatcher
    afin de répondre à des événements.

    Dans le cas présent, l'envoie de données par le terminal
    mobile destinée au terminal bluetooth du Raspberry.
    """
    def __init__(self):
        threading.Thread.__init__(self)

        # Stockage des fonctions qui seront appellées si tel ou tel nombre est envoyé
        self.callbacks = {}

        # Mise en place d'un système de log, qui s'avère important
        self.logger = logging.getLogger()
        self.logger.setLevel(logging.DEBUG)

        formatter = logging.Formatter("%(asctime)s :: %(levelname)s :: %(message)s")
        file_handler = RotatingFileHandler("/home/pi/pupitre.log", "a", 1000000, 1)
        file_handler.setLevel(logging.DEBUG)
        file_handler.setFormatter(formatter)
        self.logger.addHandler(file_handler)
```

```
        stream_handler = logging.StreamHandler()
        stream_handler.setLevel(logging.DEBUG)
        self.logger.addHandler(stream_handler)

        try: # Tentative
          # bluetooth_module fait référence au module bluetooth du Rasoberry
          self.bluetooth_module = serial.Serial(
              port="/dev/ttyAMA0", # Définition du PIN on l'on va lire.
              baudrate=9600,
              parity=serial.PARITY_NONE,
              stopbits=serial.STOPBITS_ONE,
              bytesize=serial.EIGHTBITS,
              timeout=1,
          )

        except Exception as error: # Si une erreur se produit
          # On écrit sur le fichier de log
          self.logger.warning("Erreur concernant le serial du Raspberry : " + error)

    def on_with_sound_and_leds(self, callback):
        """
        Décorateur qui stockera la fonction passé en
        appel de retour dans le dictionnaire. Cette
        dernière pourra être appellé graçe à la clé b"2"

        Quand la donnée "2" sera envoyée, on appellera
        la fonction passer au décorateur.
        """

        # Quand la donnée "2" sera envoyée, on appellera
        # la fonction passer au décorateur.
        self.callbacks[b"2"] = callback
        return callback

    def on_with_just_sound(self, callback):
        """
        Décorateur qui stockera la fonction passé en
        appel de retour dans le dictionnaire. Cette
        dernière pourra être appellé graçe à la clé b"1"

        Quand la donnée "1" sera envoyée, on appellera
        la fonction passer au décorateur.
        """

        self.callbacks[b"1"] = callback
        return callback

    def on_with_leds(self, callback):
       """
        Décorateur qui stockera la fonction passé en
        appel de retour dans le dictionnaire. Cette
        dernière pourra être appellé graçe à la clé b"3"

        Quand la donnée "3" sera envoyée, on appellera
        la fonction passer au décorateur.
        """
```

```
        self.callbacks[b"3"] = callback
        return callback

    def run(self):
        """
        Méthode qui vérifiera indéfiniment la valeur envoyée
        par bluetooth en passant cette dernière au dictionnaire
        qui vérifiera si la valeur envoyée correspond à
        une fonction. Si c'est le cas, on appelle la
        fonction prévu à cet effet.
        """
        while True:
            try:
                req = self.bluetooth_module.readline()
                self.callbacks[req]()

            except KeyboardInterrupt:
                # On réinitialise les PIN du Raspberry
                GPIO.cleanup()

            except KeyError:
                pass
```

```
# Fichier core/light.py
import time
import RPi.GPIO as GPIO


def animation(period, handlers):
    """
    Méthode générique réalisant l'animation
    des lumières (effet mors avec les
    lumières qui s'allument et s'éteignent très vite).

    La periode correspond au nombre de fois que l'animation
    sera réalisé puis handlers correspond à une liste de PIN
    ou sont branchées les lumières.
    """
    for loop in range(period):
        for handler in handlers:
            GPIO.output(handler, GPIO.LOW)
            time.sleep(0.1)

        for handler in reversed(handlers):
            GPIO.output(handler, GPIO.HIGH)
            time.sleep(0.1)
```

Avec l'environnement Arduino

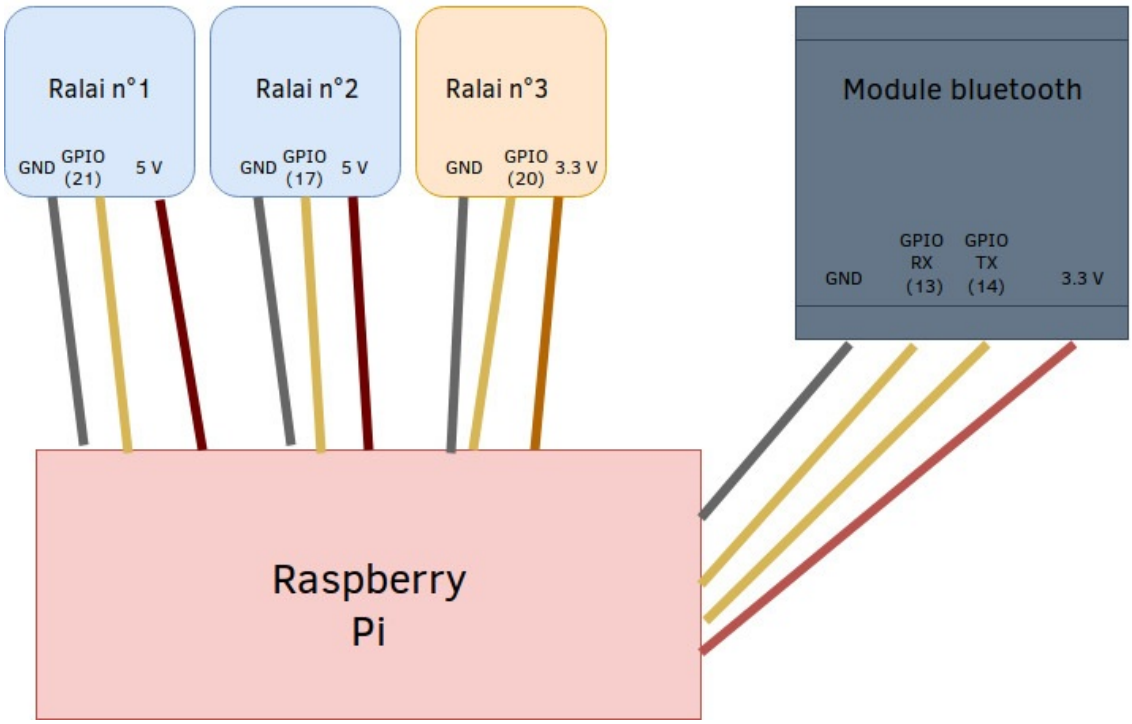
Le programme en langage C

Voici le code source du programme téléverser dans la carte Arduino destiné à gérer les leds du pupitre et ainsi les allumer et les éteindre aléatoirement :

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  randomSeed(analogRead(0));
}

void loop() {
  const byte LED_PIN = random(2, 4); // On séléctionne un nombre qui réprésente un PIN de
LED
  delay(1000); // On attend 1 seconde

  /*
   * On initialise une variable "state" qui aléatoirement
   * possédera la valeur HIGH (led allumé) ou LOW (led étinte).
   */
  byte state = 0;

  /*
  * On définit aléatoirement la valeur de la variable state
  * graçe à cette condition
  */
  if (random(0, 10) > 5)
    state = HIGH;

  else
    state = LOW;
 /*
 * Si la valeur généré alétoirement est supérieur à 5,
 * On fait une pause de x temps ou x est aussi aléatoirement
 * compris entre 1 et 3 secondes
 */

  if (random(0, 10) > 5)
    delay(random(1000, 3000));

  /* La led séléctionné aléatoirement s'étindera ou s'allumera
  aléatoirement, comme convenu */
  digitalWrite(LED_PIN, state);
}
```

Circuit éléctrique

NOTE: Vous pouvez retrouver l'emplacement des GPIO, du GND et des tensions 3.3 Volt et 5 V dans la partie "Raspberry PI" de la documentation.



Problèmes rencontrés

La fonction random() de la librairie standard d'Arduino

Aussi étrange que cela puisse paraître, Arduino ne générait pas de nombre aléatoire d'une échelle de zéro à trois. N'ayant pas trouvé d'explications rationnel à ce problème, nous avons décidé de générer un nombre aléatoire grâce à cette fonction mais cette fois de zéro à dix en adaptant ce comportement l'algorithme de gestion des leds, ce qui a fonctionné.

Le module Grove MP3 v2.0 pour Arduino

Dû à un manque critique de documentation officiel, l'implémentation de se module s'est avéré être un échec. En effet, les sources de documentation improvisées nous ont guidés vers la composition du schéma électrique mais la mise en place de la partie programmation manquait de source et semblait archaïque ou point de manipuler des données en hexadécimal.

Il a fallu abandonner nos travaux sur l'Arduino et s’intéresser au Raspberry PI afin de continuer le projet qui requiert une prise en charge de l'audio.

La prise en charge du Raspberry PI par un écran VGA standard

La solution à ce problème s'est avéré plus facile qu'elle en avait l'air. Après plus d'une heure d'investigation, il fallait finalement changer une ligne du fichier de configuration qui se trouvait à la racine de la carte SD ou se trouvait Raspbian.

L’exécution concurrente entre deux tâches

Le pupitre doit pouvoir gérer l'algorithme de lumières, dit l'éclairage, et la bande de son de la pièce de théâtre simultanément.

A partir de se postulat nous avons donc remplacé l'Arduino au profit du Raspberry PI car en effet, Arduino ne supporte aucun design-pattern lié à la concurrence. En passant par les threads voir même l’asynchronisme, aucune solution viable n'a été trouvé. Il a donc fallu se tourner vers un dispositif plus puissant, avec un environnement qui offre de plus amples libertés du point de vue des performances.

Le choix d'une technologie afin de jouer un fichier audio sur le Raspberry PI

Voici la problématique qui a sans doute l'élément le plus retardant. En effet, il a fallu choisir la bonne librairie Python afin de gérer l'audio car certaines :

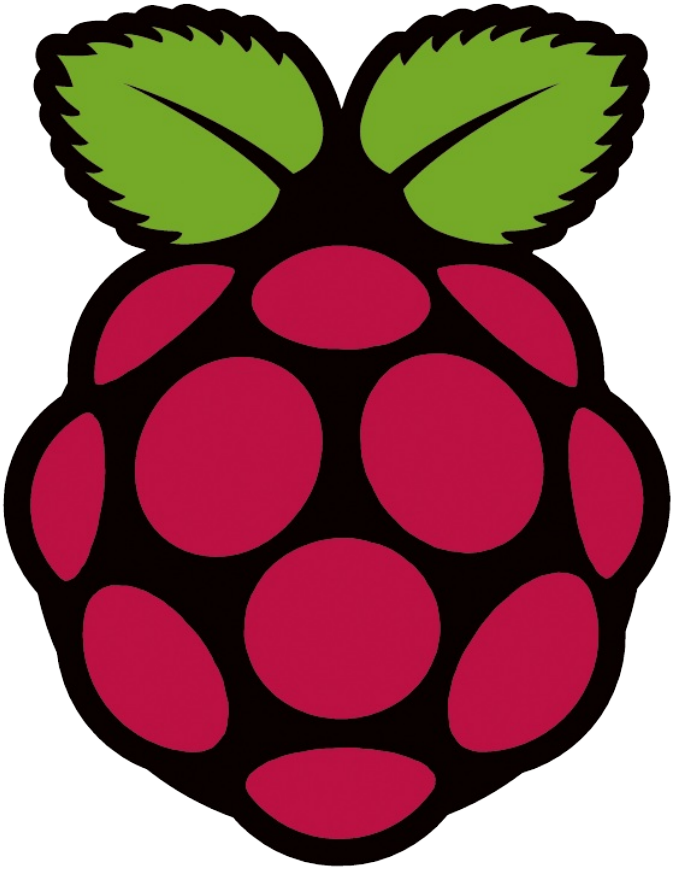
- Ne supportaient pas le support jack du Raspberry PI (module playsound)
- Ne délivrait pas un son "correcte", il s'agissait de grésillement ou d'un son trop fort pour les notes graves. (package pygame)
- Ne supportaient pas la mise en service dite "deamon" (package pygame)
- Ne pouvaient pas "mettre en pause" l'échantillon audio sans passer par des pipes UNIX (processus mpg123)
- Requéraient une dépendance à installer, ceci étant fastidieux car il fallait choisir une version exacte de la dépendance en question (pyglet)

Il a donc fallu investiguer pendant un long moment pour pouvoir trouver une technologie adéquate, qu'elle soit d'un point de vue processus ou qu'il s'agisse d'un module/package Python.

Par conséquent, nous avons choisi un wrapper du programme omxplayer disponible dans les dépôts de PyPi (support officiel pour les librairies et packages Python). Ce dernier répond à toutes les problématiques énoncées ci-dessus tout en restant élégant du point de vue de son implémentation.

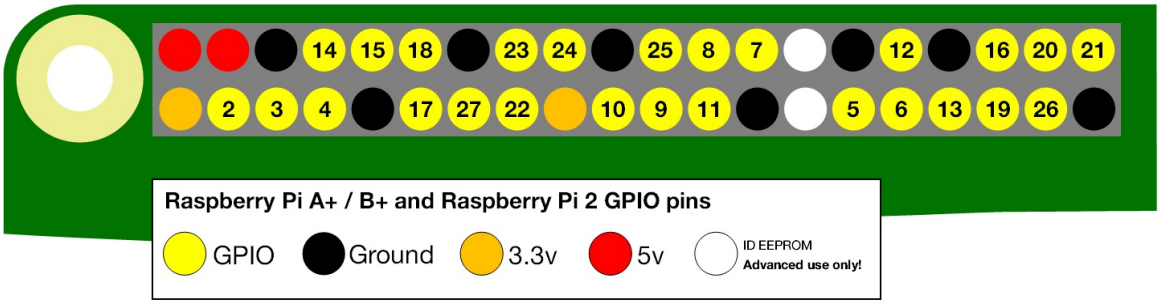
L'ensemble de ses problèmes nous a fait prendre conscience que l'Arduino est selon nous un choix idéal pour réaliser ce genre de projet dit "embarqué". En effet le manque d'ergonomie se fait sentir avec un Raspberry PI : Il n'y a aucun moyen de téléverser de pair à pair en liaison physique les différents programmes et le lancement d'un service au démarrage s'avère lent.

Raspberry PI



Le Raspberry PI est un nano-ordinateur. Il est donc programmable et peut ainsi envoyer des "données" (envoyer du courant afin d'allumer une led par exemple) ou réceptionner des "données" (réceptionner les données d'un capteur thermique par exemple) tout comme un Arduino. Il est cependant plus "puissant" concernant le processeur et la mémoire vive, pouvant supporter un système d'exploitation ou exécuter des algorithme implémenter dans des langages dit haut-niveau comme le langage Python.

Alimentation	Documentation	Datasheet
5 Volt	https://www.raspberrypi.org/documentation/	https://static.raspberrypi.org/files/product briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf



Installation du système d'exploitation Raspbian pour le Raspberry PI

Raspbian est une distribution Linux conçu pour le Raspberry PI. Pour l'installer il suffit de se rendre sur le site officiel du Raspberry PI et d'aller vers l'onglet "download", l'image ISO est disponible en cliquant sur l'onglet "Raspian".

Note : L'image ISO doit être transféré vers une carte SD qui à l'issu devra être inséré dans le Raspberry PI, nous avons utilisé **le logiciel Etcher** afin d'effectuer cette manœuvre.

Premier démarrage

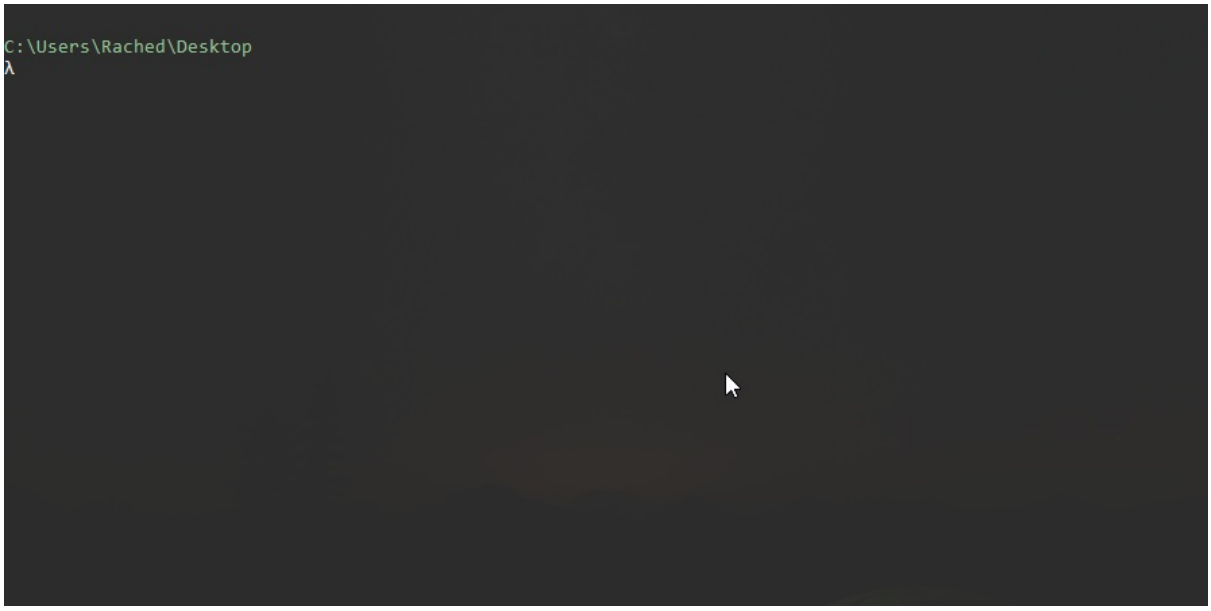
Si les premiers pas ont été effectués et le Raspberry PI allumé, il suffit de se connecter avec le login ***pi*** et le mot de passe : ***raspberry***.

Linux

Linux est un système d'exploitation, c'est-à-dire, un ensemble de programmes qui permet d'agir sur la machine et de lancer d'autres programmes.

Ce système d'exploitation est par défaut utilisé par le Raspberry PI de par sa faible consommation de ressources, sa gratuité, sa robustesse.

L’interpréteur de commande (shell)



NOTE: Les animations gifs ne fonctionnent malheureusement pas sur la version PDF et docx.

L'interpréteur de commandes est l'interface entre l'utilisateur et le système d'exploitation, d'où son nom anglais "shell", qui signifie "coquille".

Il nous permet donc de réaliser des actions sur le Raspberry PI sans interface graphique, ce qui dans le cas contraire aurait été fatale car le Raspberry PI possède peu de mémoire vive et de puissance de calcul.

Le gestionnaires de paquet Debian

Un gestionnaire de paquets est un (ou plusieurs) outil(s) automatisant le processus d'installation, désinstallation, mise à jour de logiciels installés sur un système informatique.

Nous avons utilisée le gestionnaires de paquets Debian afin d'installer Python 3.5, le gestionnaire de paquets Python, le package RPi.GPIO et omxplayer.

Installation des dépendances sur le Raspberry PI

Il faut donc utiliser interpréteur de commande ainsi que le gestionnaire de paquets Debian nommé apt comme ci-dessous :

```
sudo apt update # On met à jour les dêpots
sudo apt upgrade # On met à jour les packages du système Raspbian

# On installe le package qui servira à lancer
# la musique du pupitre et le gestionnaire de packages de Python
sudo apt install python3-pip

# On installe finalement le package pyserial afin de
# lire les informations données par le
# module bluetooth, le package Rpi.GPIO
# afin de d'agir sur les différents GPIO et omxplayer-wrapper
# afin de gérer le fichier audio.

pip3 install --user pyserial RPi.GPIO omxplayer-wrapper
```

Systemd

A ne pas oublier : L'arboresence suivante est proposé afin de regrouper tous les fichiers nécessaires au projet

```
pupitre
├── LICENSE
├── Makefile
├── README.md
├── __init__.py
├── __main__.py
├── config.json
└── core
    ├── __init__.py
    ├── event.py
    └── light.py
```

Systemd est un ensemble de programme permettant de créer des services et les lancer au démarrage du système cible sous Linux.

Par conséquent, nous avons créer un fichier représentatif de notre besoin pour systemd afin de lancer notre projet au démarrage du Raspberry PI nano-ordinateur utilisé pour le projet.

Nous utiliserons l'éditeur vi afin d'éditer le fichier pupitre.service

```
sudo vi /etc/systemd/system/pupitre.service
```

Nous y ajoutons donc ces lignes :

```
[Unit]
Description=Projet pypitre issu du projet thêatre de Monsieur Rocher (Lycée Jean Monnet)
After=multi-user.target

[Service]
Type=idle
ExecStart=/usr/bin/python3 /home/pi/pupitre/__main__.py

[Install]
WantedBy=multi-user.target
```

Puis nous activons le service :

```
sudo systemctl enable pupitre.service
```

OpenSSH et Makefile

OpenSSH

OpenSSH est un ensemble d'outils libres pour établir des communications chiffrés, donc sécurisées, sur un réseau informatique grâce au protocole SSH.

Nous avons par conséquent utilisé le programme `scp` afin de transférer des fichiers locaux du poste de travail vers le Raspberry PI ainsi que le protocole ssh afin d'accéder au terminal de se dernier à distance car le Raspberry PI ne possédait pas d'interface graphique.

Makefile

A ne pas oublier : L'arboresence suivante est proposé afin de regrouper tous les fichiers nécessaires au projet

```
pupitre
├── LICENSE
├── Makefile
├── README.md
├── __init__.py
├── __main__.py
├── config.json
└── core
    ├── __init__.py
    ├── event.py
    └── light.py
```

Les Makefiles sont des fichiers, généralement appelés makefile ou Makefile, utilisés par le programme make pour exécuter un ensemble d'actions, comme la compilation d'un projet, l'archivage de document, la mise à jour de site et dans le cas présent le téléverser le projet stocké en local vers le Raspberry PI et l'exécuter depuis ce dernier.

Nous avons jugé cette méthode comme étant la plus optimal car elle a la spécificité d'être native sur les systèmes UNIX-like tout en ayant une syntaxe relativement minimaliste.

Voici ci-dessous le Makefile utilisé :

```
push:
    scp -r * pi@192.168.0.6:~/pupitre # l'adresse ip du raspberry peut changer d'un réseau
 local à un autre.
    ssh pi@192.168.0.6 python3 pupitre # On exécute le package Python du pupitre depuis le
 Raspberry PI
```

Nous pouvons donc exécuter les commandes écrites sur le Makefile grâce à cette commande depuis un shell Linux :

```
make push
```

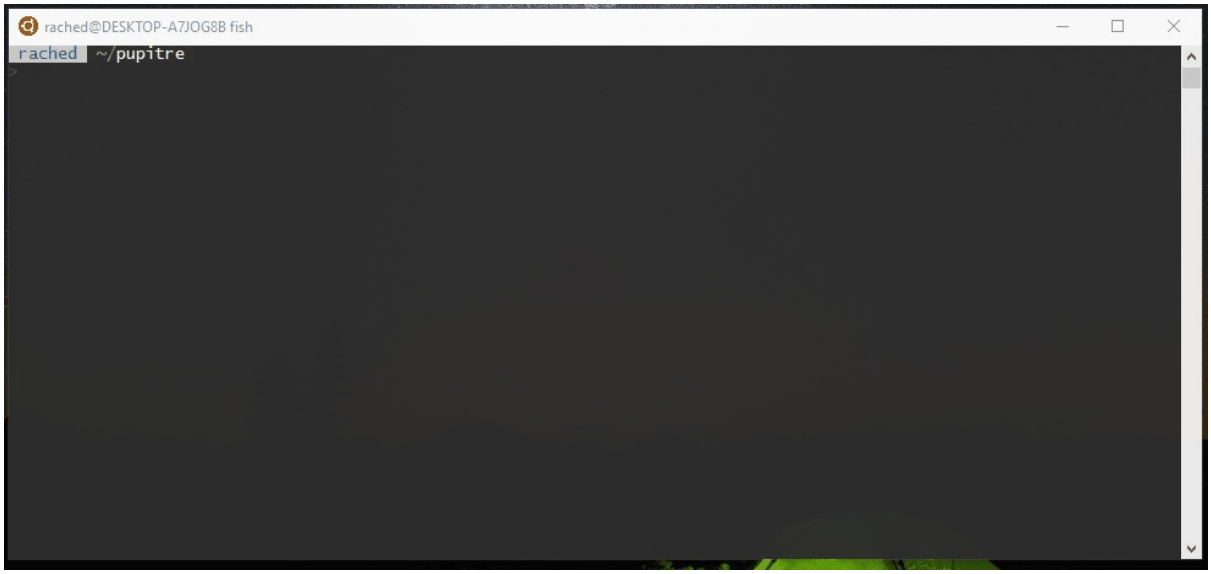
Le cas de l'interface graphique sur le Raspberry PI

En effet, travailler directement sur le Raspberry PI aurait été une façon de gagner en efficacité. En revanche, il faut garder en tête que ce nano-ordinateur n'a pas la puissance d'un véritable ordinateur traditionnel. La mise en place d'une interface graphique aurait été donc dégradante pour les performances du système.

De plus, l'accès au terminal n'est pas un point faible. Il permet une certaine souplesse dans le contrôle du système que l'on ne retrouve dans une interface graphique.

OpenSSH et Makefile par l'exemple

Voici ci-dessous une vidéo au format gif qui montre l’exécution d'un Makfile afin d'automatiser nos opérations :



NOTE: Les animations gifs ne fonctionnent malheureusement pas sur la version PDF et docx.

Python 3

Python

Le langage Python est l'un des langages les plus polyvalent et facile à apprendre. Sa syntaxe et ses mécanismes élégants font de Python l'un des langages les plus populaires en 2019 d'après l'index TIOBE.

Python est un langage interprété, une fois l'algorithme écrit en Python, il faut le faire passer par un logiciel, l'interpréteur, qui se chargera de transformer le code source en code machine compréhensible par un langage plus performant : le langage C.

PIP et PyPi

PIP est le gestionnaire de paquets officiel de Python. Ce dernier récupère les paquets depuis PyPi, l'index officiel des paquets pour Python. Il a été utilisé afin de réaliser ce projet, les instructions afin de l'installer sont disponibles ici.

Composants externes

Le module bluetooth DF-BluetoothV3

Description

Ce module Bluetooth permet d'ajouter une connexion série sans fil Bluetooth sur le microcontrôleur cible. Il communique avec ce dernier via un port série TTL.

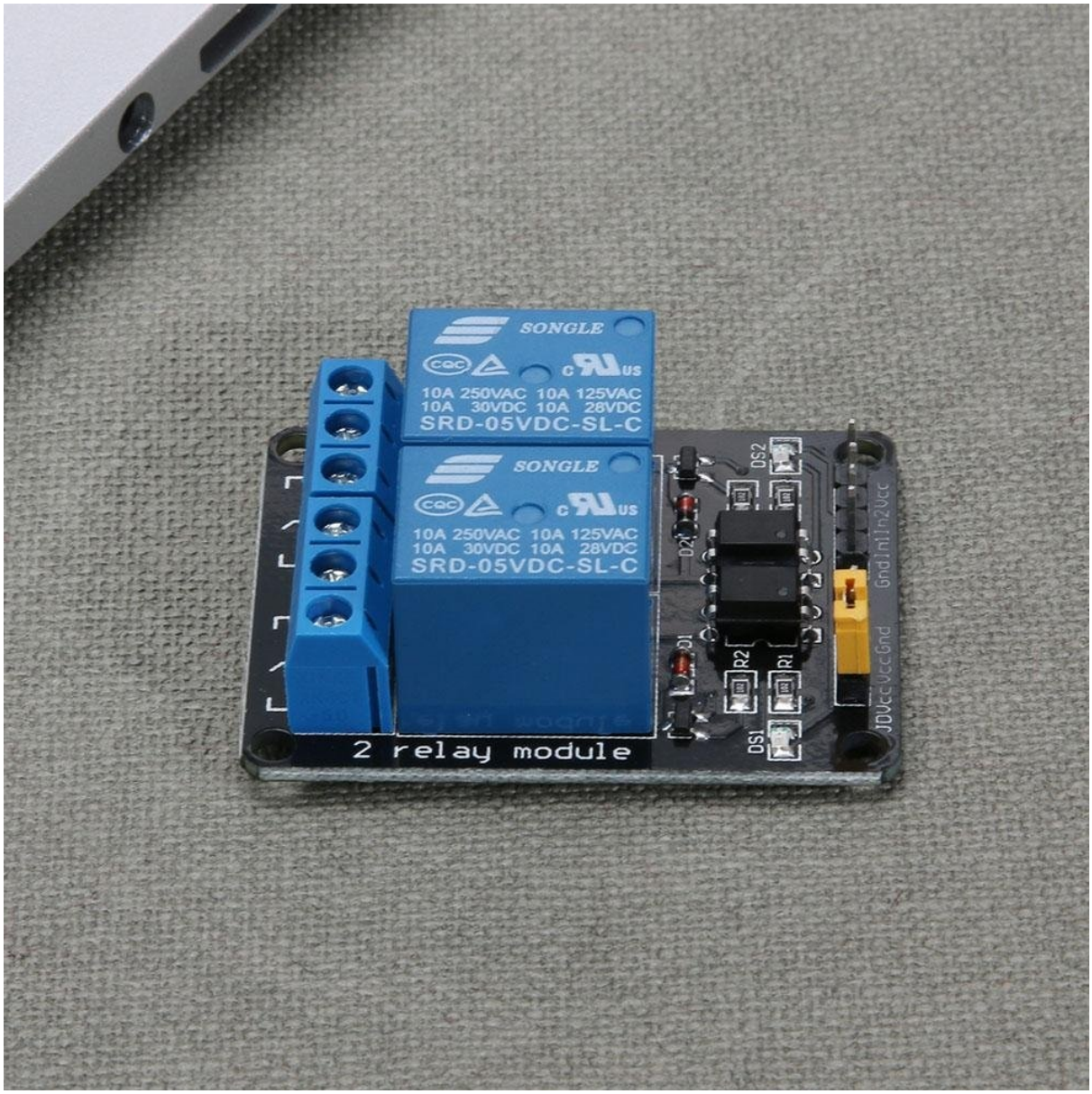
Tableau des caractéristiques

Alimentation	Bluetooth	Dimension	Datasheet
3,5 à 8 Volt	Version 2.0	40 x 20 x 13 mm	http://image.dfrobot.com/image/data/TEL0026/TEL0026_I

Emettre des informations au Raspberry PI

IMPORTANT : La broche TX du module doit être branché à la broche RX du Raspberry PI (PIN numéro 15 du Raspberry PI d'après la norme BCM) et la broche RX du module doit être branché à la broche TX du Raspberry PI (PIN numéro 14 du Raspberry PI d'après la norme BCM).

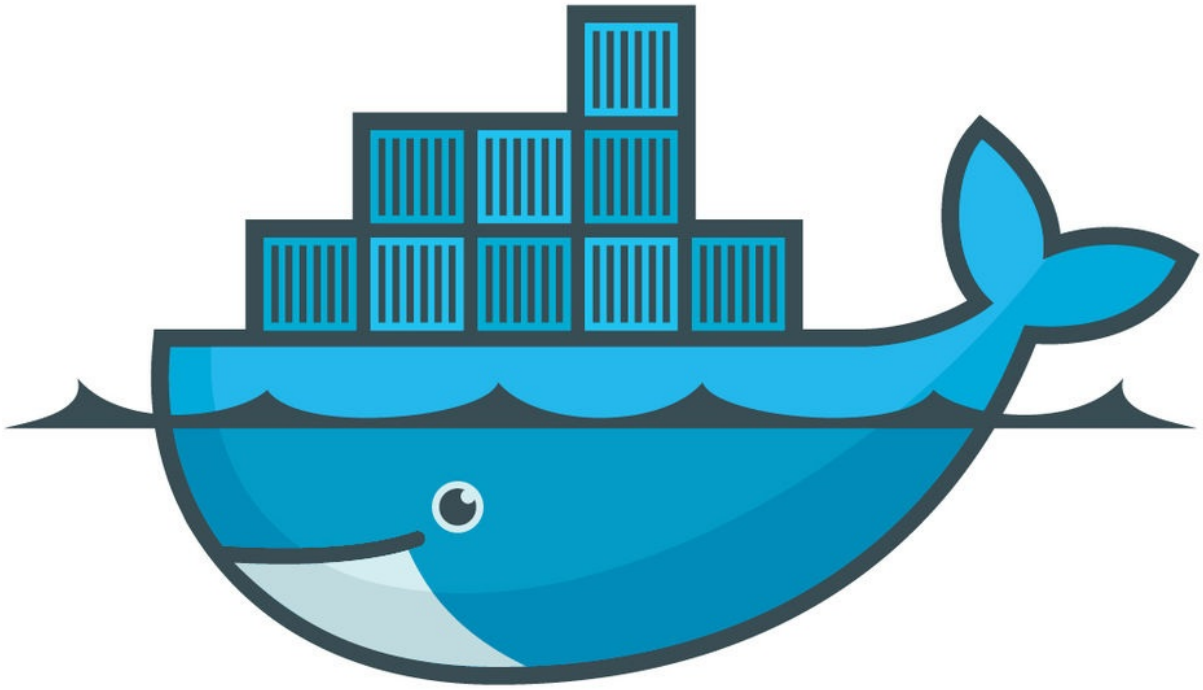
Le relais Hillrong 12 V



Ce relais possède en entrée une broche vers un PIN délivrant 3.3 Volt ou 0 Volt, une broche d'alimentation 5 Volt ou 3.3 Volt puis une broche de mise à la terre.

De l'autre côté, nous avons relié un câble partant du trou du milieu du relai vers la broche de haut-potentiel de la lumière puis nous avons relié de nouveau un câble partant du trou de droite du relai (normalement ouvert) vers l'alimentation délivrant du 16 Volt afin d'alimenter des lampes nécessitant une telle tension.

Docker



Docker est une technologie de virtualisation. Elle permet de créer un système isolé à travers le concept de container Docker et d'installer n'importe quel système d'exploitation sur ce dernier.

L'avantage étant donc de pouvoir installé n'importe quel version de n'importe quel programme sans effet de bord. Il était donc important de s’intéresser à cette technologie afin de pouvoir installer les dépendances du projet et garder une certaine flexibilité dans la façon de maintenir les différents scripts du pupitre.

Docker sur un Raspberry PI

Malheureusement, il s'est avéré que Docker n'était pas supporté par le modèle de Raspberry PI utilisé pour ce projet (Raspberry PI 2 model B+ édition 2014).

En effet, Docker subissait une erreur de segmentation, aucune issue n'a été trouvé afin de résoudre ce problème à ce jour.

L'alternatif

Aucune alternatif n'a été trouvé afin de pallier à la problématique de Docker. Il a fallu s'adapter à cette contrainte tout gardant à l'esprit que notre système doit rester "propre" afin d'éviter tout effet de bord.

Git

Git est un logiciel de gestion de versions décentralisé qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus accompagné de messages décrivant toutes les modifications apportées. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes.

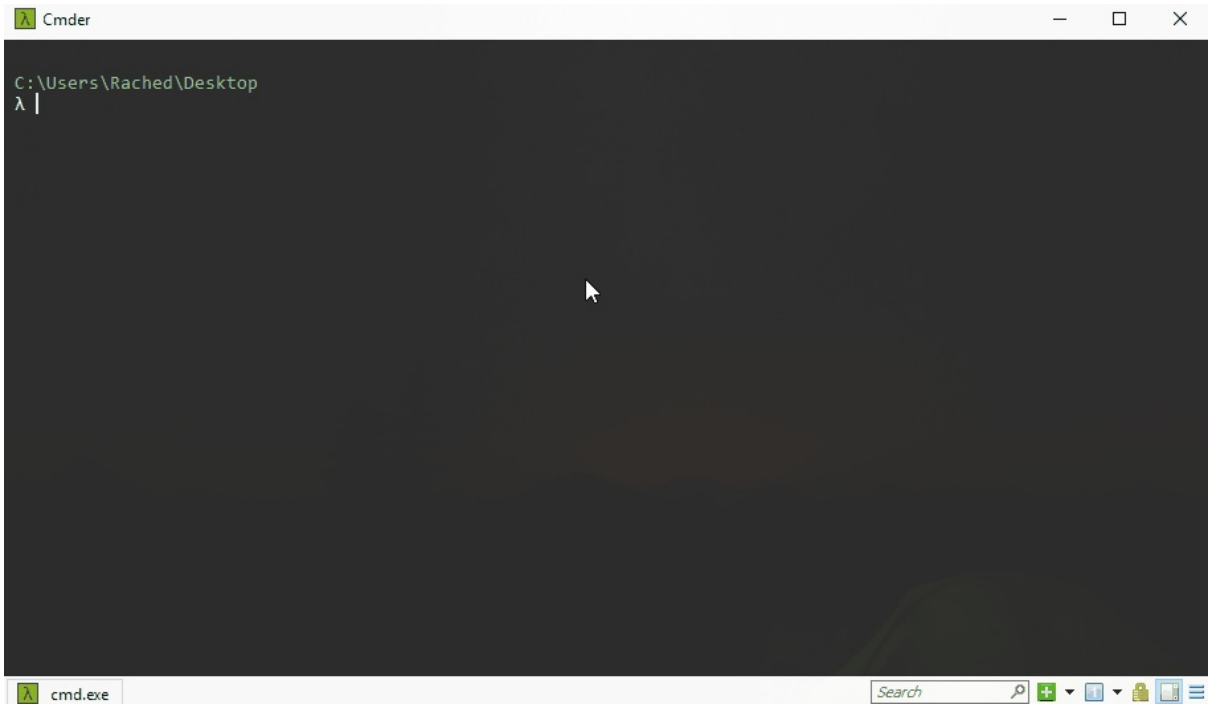
Ce dernier a été utilisé dans le cadre du projet, les commandes de bases peuvent être trouvées à cette adresse, une introduction à ces commandes demanderait tout un chapitre : https://git-scm.com/book/fr/v2

Il est vivement recommandé voir impératif d'utiliser un logiciel de gestion de versions au sein d'un projet de quelquonque taille. La moindre modification d'une ligne de code peut entraîner de nombreux dysfonctionnement, il est donc impératif de garder des versions saines du projet en utilisant ce genre de logiciel.

Utilisation de git afin de récupérer le projet

La commande `git clone` permet de récupérer l'entièreté du projet depuis la plateforme Github à cette adresse : https://github.com/projet-theatre/pupitre. Malgré cette possibilité de récupération, tout ce qui concerne le code source du projet est disponible sur cette page.

"git clone" par l'exemple

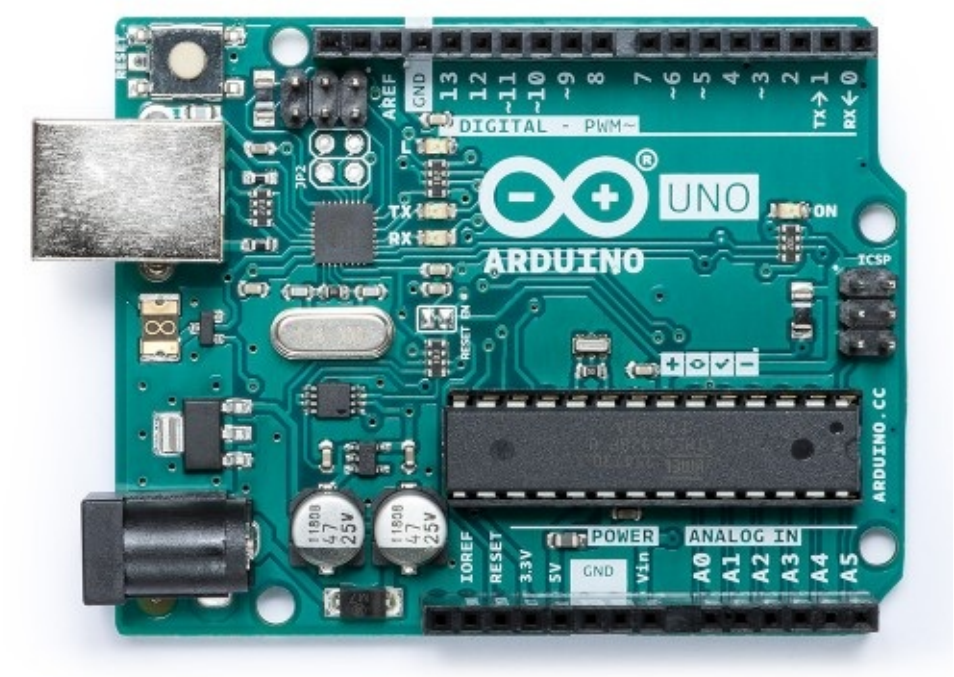


NOTE: Les animations gifs ne fonctionnent malheureusement pas sur la version PDF et docx.

L'environnement Arduino

Arduino est un microcontrôleur. Il est donc programmable et peut ainsi envoyer des "données" (envoyer du courant afin d'allumer une led par exemple) ou réceptionner des "données" (réceptionner les données d'un capteur thermique par exemple).

Alimentation	Documentation officiel	Le Language Reference	
5 Volt	https://www.arduino.cc/en/main/docs	https://www.arduino.cc/reference/en/	



Sa simplicité en fait un choix idéal pour réaliser des projets concrets avec une limite de temps courte sans devoir se soucier des détails complexes.

Fiche de validation

Télécharger la fiche de validation

NOTE: Le téléchargement de la fiche de valisation ne fonctionne malheureusement pas sur la version PDF et docx.

Conclusion

Ce projet pupitre est un moyen pour nous de se confronter à la réalité d'un véritable projet en classe d'STI2D.

Il a fallu prendre conscience des attentes, organiser son temps et son équipe afin de réaliser un travail optimale pour les prochaines séances à venir.

Ceci étant la partie théorique, d'un point de vue pratique cette fois, il a fallu réfléchir aux différentes solutions à mettre au point lors des phases de programmation notamment dans le cadre de la concurrence des tâches et construire "manuellement" le pupitre en prenant en compte les différentes mesures et l’accueil des différents composants électriques.