PROJET DE FIN D'ANNÉE

« BRIGHTMIRROR »

Réalisé par Sean Marotta – 4ème Année

Promotion PASCAL
Système Embarqué

Année 2019 – 2020
Tuteur : Monsieur Charles Meunier

TABLE DES MATIÈRES

LISTE DES FIGURES

REMERCIEMENTS

Tout d'abord, j'aimerais remercier l'ESIREM pour m'avoir donné le temps ainsi que l'opportunité de réaliser un projet qui me tenait à cœur de réaliser depuis quelques temps maintenant, un projet au départ personnel qui est donc réalisé dans un cadre scolaire. J'ai pu apprendre énormément d'un point de vue l'électronique et quelques rappels en informatique, mais cela m'a surtout permis de créer quelque chose de concret qui me sera utile dans la vie de tous les jours.

Et enfin, je souhaite remercier mes parents qui ont non seulement financé l'intégralité du projet mais qui m'ont également conseillé dans la réalisation du cadre. Ils furent d'une aide précieuse dans les moments les plus délicats.

INTRODUCTION

L'idée du projet m'ait venu quelques temps avant même d'étudier dans les Systèmes Embarqués lorsque je naviguais sur les réseaux sociaux où je suis tombé sur plusieurs projets Arduino/Raspberry Pi. Ainsi, un des projets m'a particulièrement plu, d'apparence, c'était un simple miroir mais de plus près, on pouvait y voir tout une interface qui s'affichait sur celui-ci. Et depuis ce jour, j'ai voulu en réaliser un pour ma propre utilisation et c'est dans ce cadre scolaire que j'ai pu le concevoir.

En clair, j'ai élaboré un miroir intelligent que j'ai nommé BrightMirror, « Bright » voulant dire Brillant mais aussi et surtout Intelligent en anglais et « Mirror » pour miroir. Mon objectif principal pour ce BrightMirror, est qu'il puisse afficher des données (que l'on appellera module) comme l'heure, un calendrier, la météo etc… Mais j'aimerais y apporter des modules créés par moi-même, en l'occurrence un module qui puisse afficher les horaires de passages des trams de Dijon. J'y ajouterai également quelques capteurs comme un capteur de présence (PIR) ou ultrasonique, des composants électroniques comme des LEDs, des boutons, et des potentiomètres. Et finalement j'aimerais y intégrer la fonction Alexa en ajoutant donc un micro, et des mini haut-parleurs, le BrightMirror serait donc un objet connecté dans la maison.

Je vais vous expliquer tous ce que j'ai pu réaliser lors de ces deux mois en détaillant les différentes étapes de la réalisation du BrightMirror avec les multiples difficultés que j'ai pu rencontrer.

Je vous souhaite une bonne lecture.

a. *CADRE EN BOIS ET COMPOSANTS NECESSAIRES*

Ma première étape a été de concevoir l'objet sur papier sur lequel figure toutes les dimensions des différents composants comme : le cadre en bois qui sert de structure générale, le tissu occultant permettant d'empêcher la lumière provenant de l'intérieur du miroir de sortir, le film miroir sans tain qui permet de refléter tel un miroir tout en laissant passer une petite quantité de lumière à travers et la plaque de Plexiglass qui donne au film sa rigidité. Cependant, l'élément essentiel au démarrage de la conception est l'écran (PC ou TV) qui sera donc l'interface sur lequel figure toutes nos données. La largeur de l'écran donnera hauteur du miroir mais la longueur de l'écran n'a pas d'importance car il sera orienté ainsi :

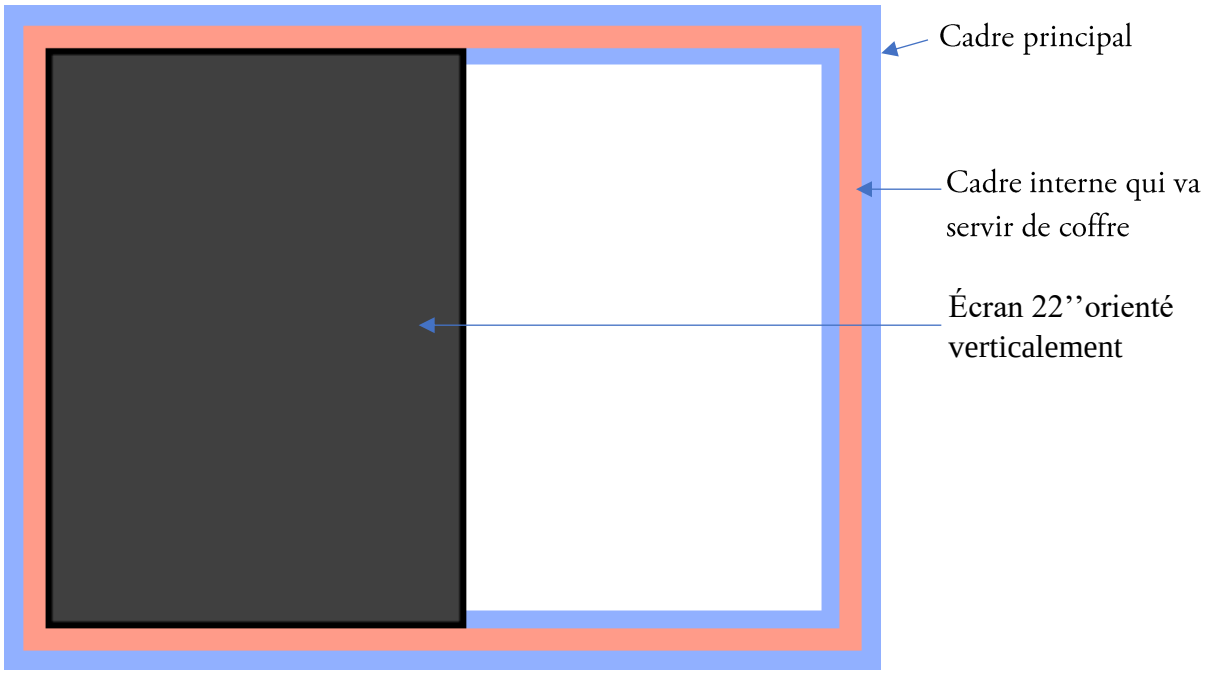


Figure 1. Schéma du cadre avec l'écran d'ordinateur 22" en vue de derrière

Après l'obtention de l'écran, j'ai pu identifier les dimensions pour le cadre en bois, tissu occultant, film miroir sans tain, et Plexiglass. Les détails des dimensions et des schémas sont présentés en annexe 1, 2 et 3.

La plupart des matériaux du projet ont été acheté à Leroy Merlin, cela concerne le plateau de table en pin qui m'a servi à créer les tasseaux de bois pour la réalisation du cadre. C'est là-bas que j'ai également pu trouver le film miroir sans tain, la colle à bois, les clous et les outils nécessaire. Concernant le tissu, étant en période de déconfinement récente, il fût de longue attente devant Mondial Tissu pour finalement ne rien trouver. J'ai donc trouvé le tissu convenable à Toto Tissu. La plaque de Plexiglass a été le plus difficile à trouver dû à la demande soudaine engendrée par les précautions dans les magasins, mais j'en ai finalement trouvé à BricotDépôt. À partir de maintenant, tous les composants électroniques qui vont être cités ont été acheté sur le site d'Amazon.

Après la conception de l'armature du miroir, vient la partie qui nous intéresse particulièrement, les composants de l'intérieur de celui-ci. Par-là, j'entends tous les capteurs et alimentations :

- Potentiomètre
- Breadboard
- LED strip
- Haut-parleurs
- Microphone
- Capteur de mouvement
- Mini amplificateur audio
- Multiprise
- Alimentation 12v, 5V, 3.3V
- Boutons poussoirs

b. *LED STRIP : WS2812B*

Parlons-en plus en détails à commencer par le système d'éclairage LED. En effet, sur le coté du cadre sera présent deux bandes LED (à gauche et à droite) qui permettra de donner une ambiance au miroir. Le modèle de la bande LED est nommé WS2812b, c'est une bande composée de 60 LEDs RVB (taille variable selon les besoins, pour nous 60 LEDs suffisent) adressable individuellement, c'est-à-dire que pour chaque LEDs, nous pouvons lui donner une couleur sous format RVB (par exemple, pour obtenir du rouge, il faut lui donner la valeur de 255, 0, 0) ainsi qu'une intensité lumineuse voulu (de 0 à 255). Il possède trois pins d'entrées respectivement **VCC | Data | GND.** La tension d'entrée est donc de +5V.

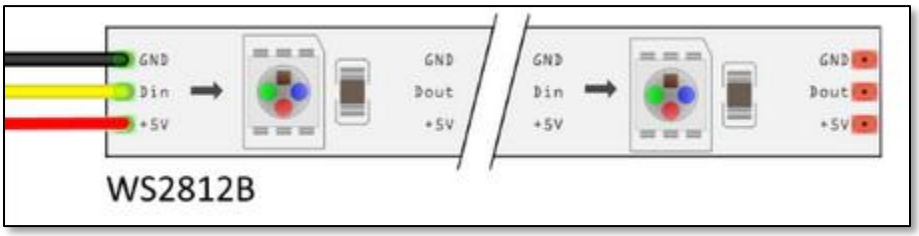


Figure 2. Schéma d'une partie du WS2812b avec les pins

c. *ENCODEUR ROTATIF : KY-040*

Un potentiomètre, ou plus exactement un encodeur rotatif ou encore un module KY-040 sera présent sur le côté du miroir qui permettra à l'utilisateur de choisir la couleur des LEDs. Ce module possède 5 pins respectivement **GND | VCC | SW (Bouton) | DT (Output B) | CLK (Output A)**. La tension d'entrée du module est de 3.3V. Le fonctionnement est très simple, à l'intérieur de celui-ci, il y a des surfaces de contact (Common C) et deux sorties (A et B). À chaque rotation, les deux sorties entrent en contact avec le Common C ce qui génère un front montant (un 1 en niveau logique) et un front descendant lorsqu'il n'est plus en contact (un 0). Pour déterminer le sens de rotation, il suffit de comparer les deux valeurs à chaque mouvement.

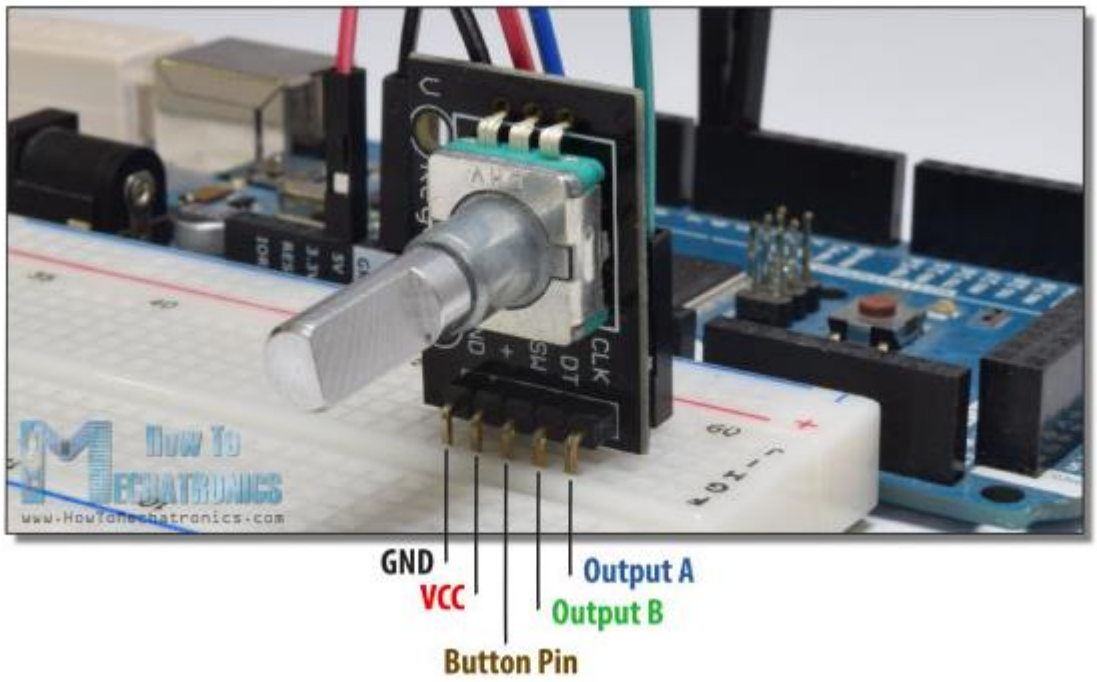


Figure 3. Module KY-040

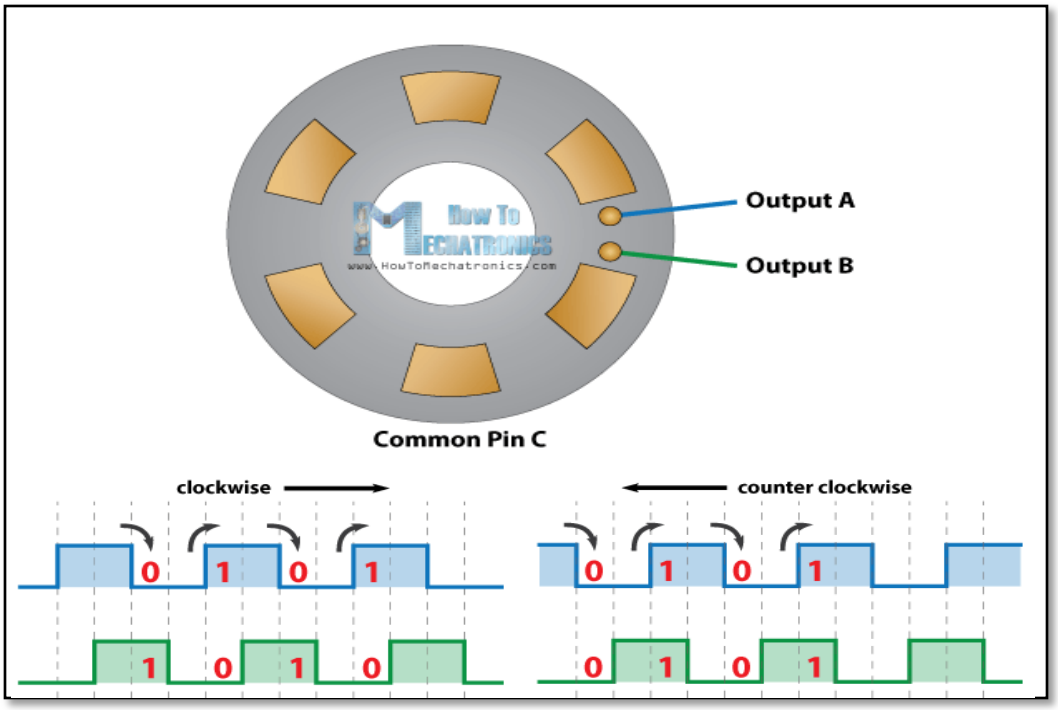


Figure 4. Schéma de l'intérieur d'un KY-040 et son fonctionnement

d. CAPTEUR PIR : HC-SR501

L'allumage des LEDs s'effectuent en fonction de la présence d'une personne en face du miroir afin d'économiser de l'énergie. Pour cela, j'ai utilisé un capteur de présence, le modèle HC-SR501 (*PIR sensor* en anglais pour *Passive Infra-Red sensor*).

C'est un capteur qui possède 3 pins d'entrée, **VCC | OUT | GND** respectivement. La tension d'entrée est de 5V. Le capteur est donc basé sur la détection de radiation infrarouge que les corps chauds émettent, il renvoie 1 s'il y a quelqu'un et 0 s'il ne détecte rien. De plus, il possède deux potentiomètres permettant de régler à la fois la distance de détection ou sensitivité (allant de 3 à 7 mètres) et le temps d'allumage (allant de 3 secondes à 5 minutes). Le temps d'allumage est utile si le fonctionnement du capteur en mode L, c'est-à-dire qu'il reste allumé en fonction de ce temps d'allumage. Et il y a le mode H, où le capteur renvoie 1 tant qu'il y a une présence dans le rayon de détection. Dans notre cas, il est plus adapté d'utiliser le mode H avec la distance minimum de 3 mètres.

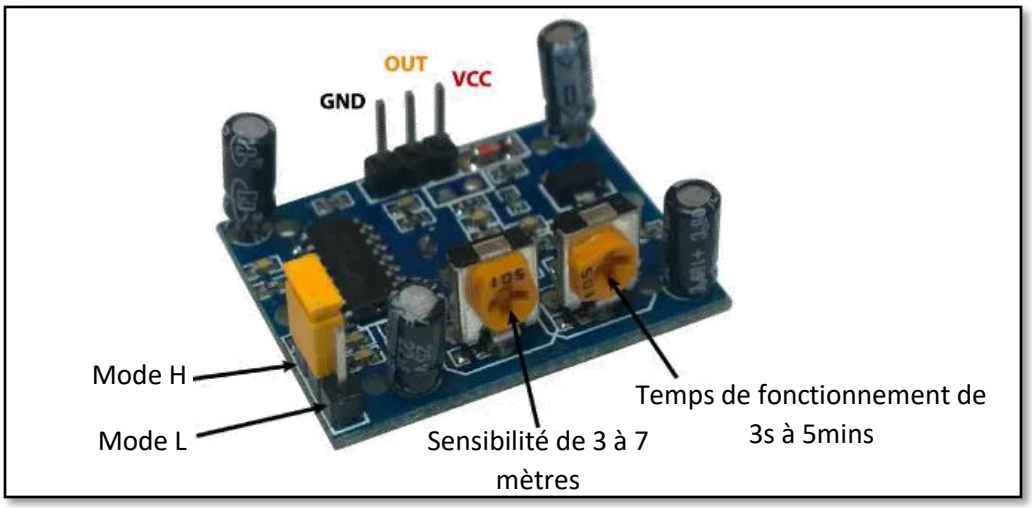


Figure 5. Photo d'un capteur PIR et son fonctionnement

e. MICROPHONE, HAUT-PARLEURS ET MINI-AMPLI PAM8610

Et enfin, nous utiliserons un microphone USB et deux haut-parleurs 5 watts 8 Ohms afin d'assurer le fonctionnement de la reconnaissance et de l'assistance vocale Alexa.

Figure 6. Photo du microphone USB et des deux haut-parleurs

Pour faire fonctionner les haut-parleurs, il faut les brancher sur un mini-amplificateur audio, le module PAM8610 qui un mini-ampli numérique classe D à deux canaux. On peut facilement caractériser les haut-parleurs par deux paramètres. Le premier c'est le gain. Un amplificateur qui a un gain de 10 multiplie par 10 l'amplitude du signal en entrée. On lui donne plus ou moins 2V en entrée, il sort plus ou moins 20V. C'est vrai tant que l'amplitude de sortie ne dépasse une valeur maximale. Si

vous donnez un signal de 3V à cet ampli et qu'il est alimenté en 20V il ne pourra pas vous sortir 30V. Le signal de sortie sera coupé en haut et en bas, cela s'appelle la saturation. Le deuxième paramètre qui caractérise un ampli c'est sa résistance interne qu'on appelle aussi impédance de sortie. Plus elle est faible plus l'ampli sera capable de délivrer un fort courant de sortie. Le tout, sans chute importante de tension, ou sans surchauffe. J'ai donc choisi un ampli audio de classe D car celui-ci est réputé pour avoir un rendement de plus de 90%.

Figure 7. Photo d'un PAM8610 vue du dessus avec légende

f. ALIMENTATION DES COMPOSANTS

Pour l'alimentation, j'ai utilisé un module déjà présent dans mon Starter Kit Elegoo, un module prenant en tension d'entrée 12V et plusieurs canaux de sorties en 3.3V ou 5V. Il permet d'alimenter la bande LED, le potentiomètre, le capteur IR et les boutons poussoirs.

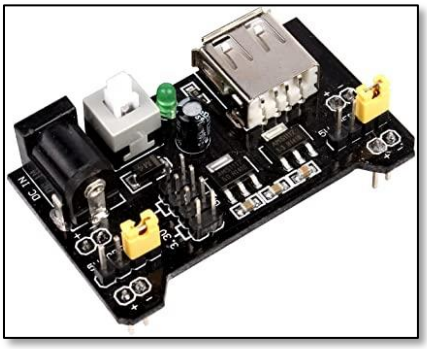


Figure 8. Power Supply Module du Starter Kit Elegoo

g. *RASPBERRY PI 3 MODEL B+*

Bien évidemment, n'oublions pas le plus important, tous ces composants doivent être contrôlés pour l'envoient ou les traitements des données reçues par les capteurs. Pour cela, j'ai utilisé un Raspberry Pi 3 model B+ sur lequel nous brancherons les composants sur les GPIOs. Maintenant que nous avons vu le rôle et le fonctionnement des chaque composants à l'intérieur du miroir, voici un schéma résumant le câblage de tous l'ensemble du projet réalisé jusqu'à maintenant.

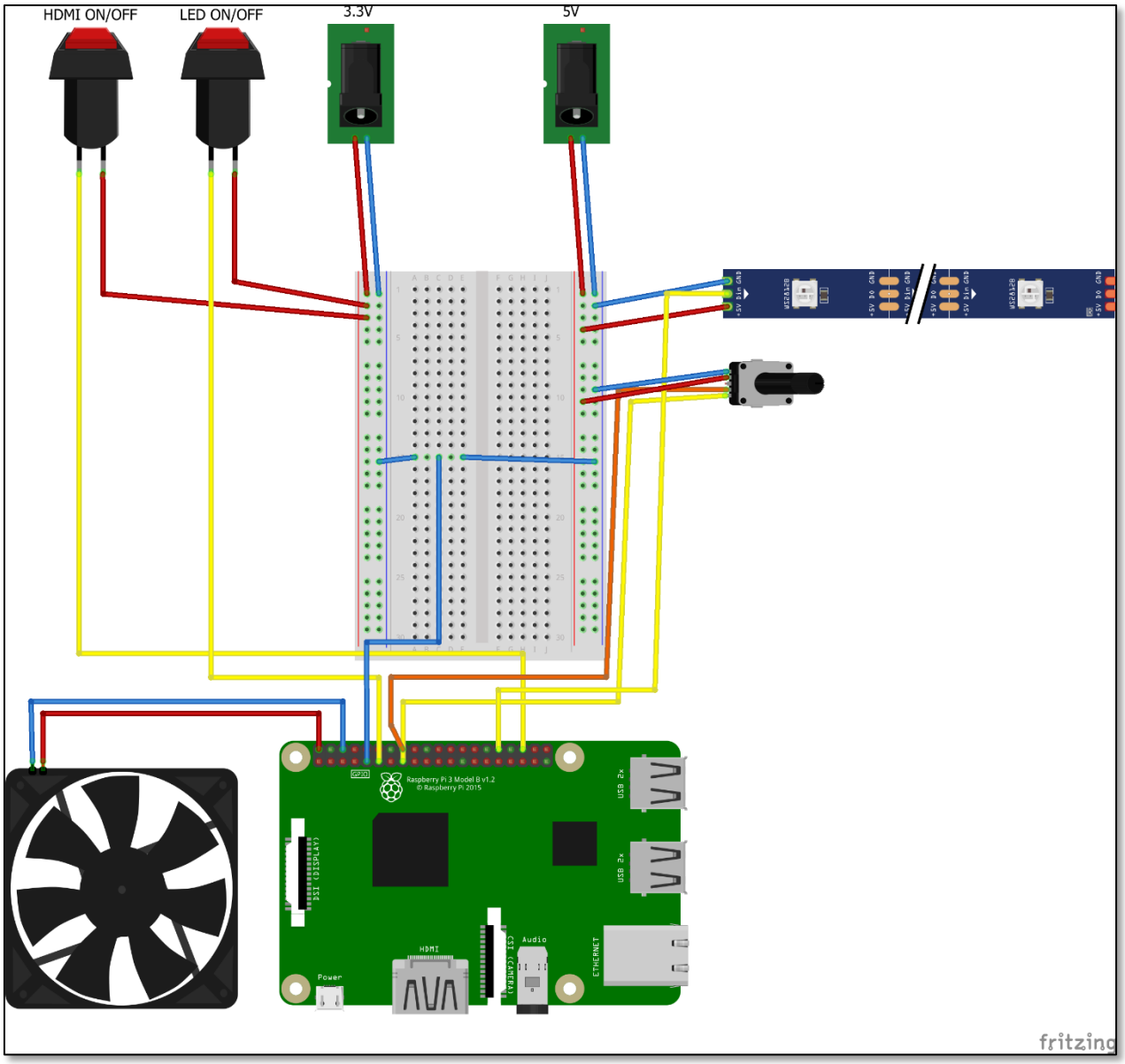


Figure 9. Schéma câblage du projet

a. *MAGIC MIRROR*

Pour le projet, nous utiliserons le populaire projet sur Raspberry Pi qui est Magic Mirror, un projet open source et modulable codé en JavaScript et basé sur NodeJS. C'est un programme qui permet d'afficher des modules sur un fond noir comme la date, l'heure, la météo et pleins d'autres encore, et ce l'allumage du Raspberry Pi. Pour cela, il suffit d'installer le projet provenant du git avec des commandes terminales, pour ma part j'ai utilisé le manuel d'installation sur le GitHub : *https://github.com/sdetweil/MagicMirror_scripts*.

Ce projet Magic Mirror est basé sur des modules, des modules de bases apparaitront lors de l'exécution du programme, les voici.

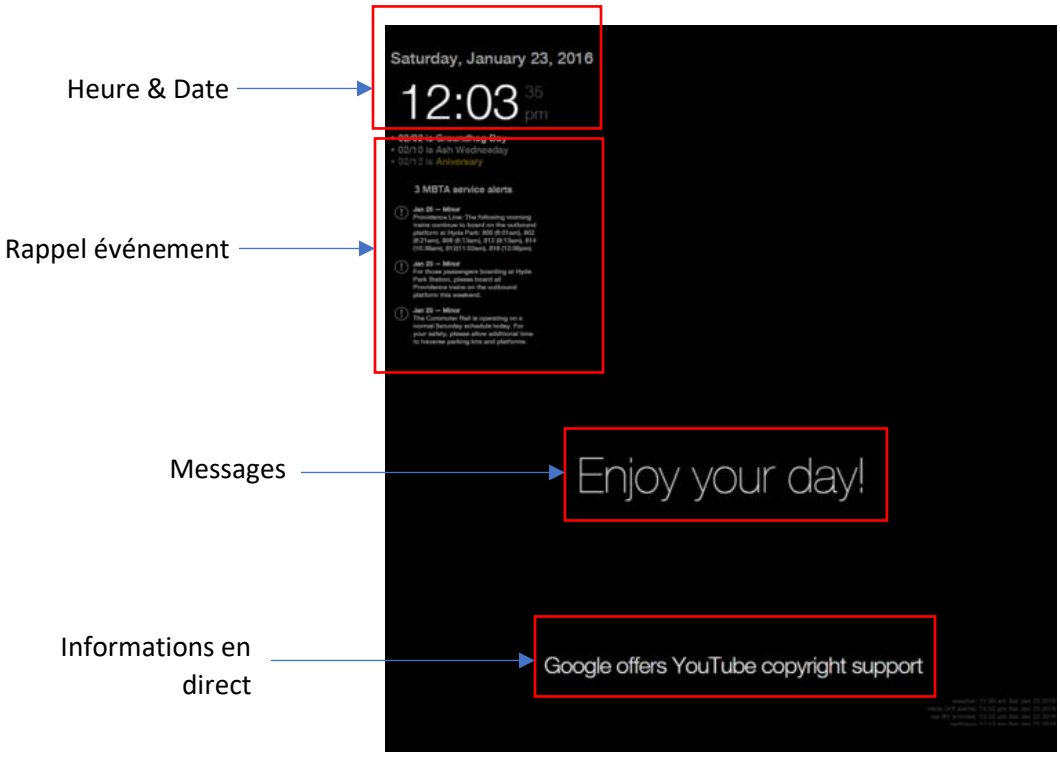


Figure 10. Exemple de modules de bases

J'ai donc modifié les modules afin que l'apparence me convienne d'une part, et que je puisse créer mon propre module d'une autre part, car effectivement, les utilisateurs peuvent eux-mêmes créer leurs propres modules et il en existe des centaines d'autres créé par les internautes. Pour ma part, je souhaite créer mon propre module sur lequel figure les horaires de passages des trams, j'ai déjà commencé à regarder comment et par où commencer. L'entreprise Keolis propose une API à travers lequel nous pouvons récolter les données nécessaires sous format XML. Il me suffit de traiter ce fichier XML à

travers du JavaScript pour pouvoir en faire mon module et du CSS pour la gestion de l'affichage à l'écran.

b. PYTHON

Le langage de programmation principal du Raspberry Pi est le Python, c'est pourquoi les scripts responsables du fonctionnement des composants électroniques sont codés en Python. Vous retrouverez les scripts en annexe 4 et 5.

Le script principal « *led.py* » doit être exécuté à la main, pour mon cas la commande :

```
sudo python3 Desktop/MM-LED/led.py
```

Or, je veux que les LEDs s'allume dès le démarrage du Raspberry Pi, pour cela j'ai utilisé *crontab* qui est un programme qui permet aux utilisateurs des systèmes Unix d'exécuter automatiquement des scripts. Pour commencer, j'ai créé un script Shell « *launcher.sh* » (avec les droits d'exécution « *chmod 755 launcher.sh* ») dans le même dossier que le script python voulu, à l'intérieur de celui-ci :

```
#!/bin/sh

cd /
cd home/pi/Desktop/MM-LED
sudo python led.py
cd /
```

Ensuite, il faut modifier le fichier *crontab* en ajoutant à la toute fin cette ligne permettant d'exécuter un script à un moment précis, ici, lors du démarrage :

```
sudo crontab -e
```

```
@reboot sh /home/pi/bbt/launcher.sh >/home/pi/logs/cronlog 2>&1
```

Il suffit de redémarrer le Raspberry Pi pour voir si cela fonctionne. La commande ci-dessous est très utile pour voir quel script Python tourne en arrière-plan :

```
ps -aef | grep python
```

Le processus est identique pour le script 1 « *hdmi_controller.py* » qui permet d'éteindre et d'allumer l'écran avec un bouton.

PROBLEMES RENCONTRES

Bien évidemment, le déroulement du projet ne s'est pas entièrement passé comme prévu, il y a eu de l'énervement et de l'incompréhension totale. J'ai eu de problèmes au niveau matériel et au niveau des composants. Tous ce qui suit se déroule dans l'ordre chronologique.

Pour commencer, le manque d'outil et l'impossibilité d'aller aux magasins pour y remédier. Heureusement que la maison familiale se situait à moins de 100km ce qui m'a permis de construire le cadre en bois avec l'aide de mon père, qui lui possédait tous les outils nécessaires (perceuse, scie circulaire, scie sauteuse, serre-joint, peinture et outil de peinture).

Une fois le cadre terminé, retour à Dijon pour l'assemblage des composants internes. Tout s'est bien déroulé jusqu'au moment où mon Raspberry Pi s'éteignit brusquement lorsque je faisais les branchement GPIO. Après plusieurs heures et gouttes sur le front à chercher pourquoi, j'en ai conclu que je l'ai court-circuité. Effectivement, il n'y avait plus de tension de sortie 3.3V sur les GPIOs et apparemment, c'est un signe mortel pour le Raspberry Pi. Erreur de manipulation de ma part lors des branchages GPIO, et évidemment je n'avais pas fait de sauvegarde donc tous mes programmes ont disparu. J'ai donc décidé de racheter un Raspberry Pi, mais cette fois-ci, j'ai choisi le model 4 b et désormais je regarde 7 fois avec mes yeux avant de brancher quoique ce soit.

Ensuite, la bande LED cessa de fonctionner et le capteur IR également, tous étaient bien branchés, je ne voyais pas d'où provenait le problème jusqu'à ce que je décide de vérifier les tensions délivrées par le module Power Supply du Starter Kit Elegoo et c'est à ce moment où la sortie 5V s'est transformé en 11.8V. J'en ai conclu qu'il était défectueux, j'en ai donc recommandé un.

Et enfin, le dernier et le plus récent des problèmes, on a décidé avec mon père de remplacer la plaque de Plexiglass avec le film miroir sans tain par un vrai miroir sans tain car on estimait qu'il y avait trop de défaut (rayures et bulles) et que l'image reflétée était déformée et floue. Nous avons reçu le miroir sans tain qui joue parfaitement son rôle. Cependant, le miroir laisse passer la lumière comme voulu, mais pas assez pour voir clairement l'écran derrière. Ce qui fait que pour la présentation finale, soit je le présente avec le miroir sans tain mais en assombrissant suffisamment la pièce pour voir mon travail, soit je remets la plaque de Plexiglass sans avoir la finition d'un vrai miroir. En attendant, je suis à la recherche d'écran 22" qui possède une luminosité suffisante pour être visible à travers le miroir.

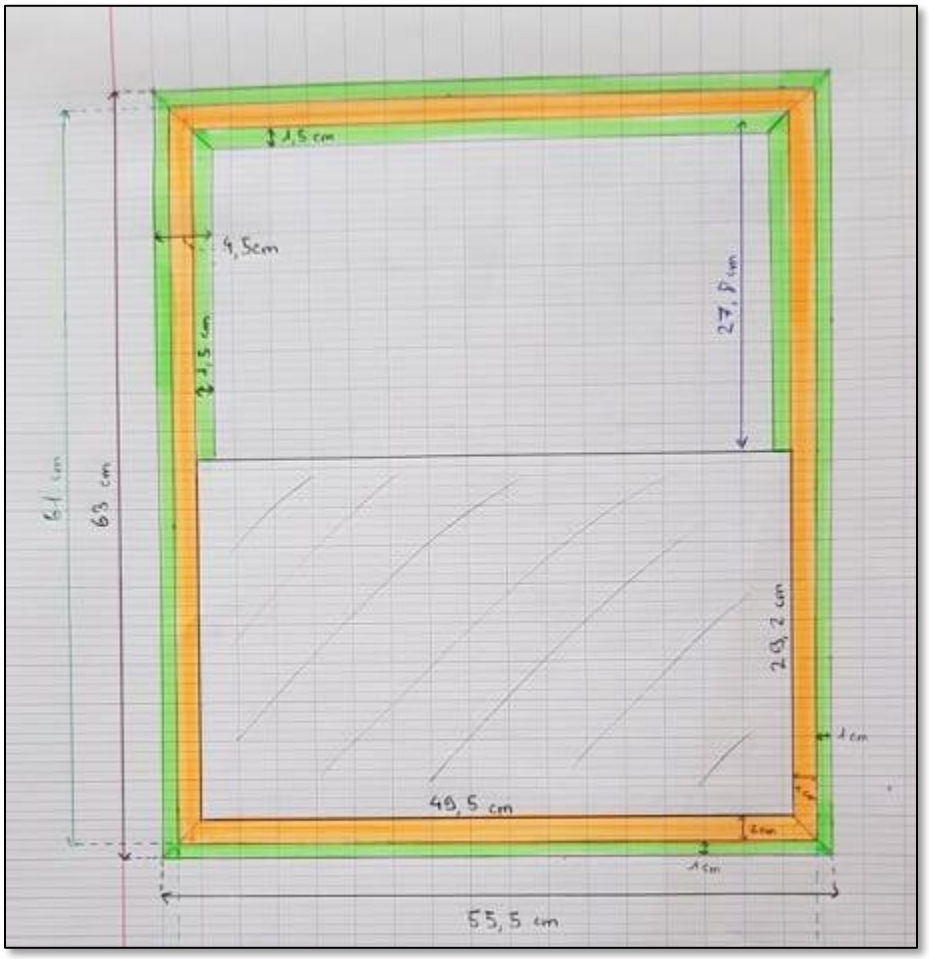
CONCLUSION

Malgré tous les problèmes rencontrés, j'en tire de ce projet une excellente expérience d'un point de vue conceptuel au niveau de l'agencement de tous les composants allant du cadre en bois jusqu'à la breadboard. De plus, j'ai aimé faire ce miroir qui me tenait à cœur depuis bien longtemps, désormais je le vois tous les jours chez moi et je peux y ajouter toutes améliorations que je souhaite.

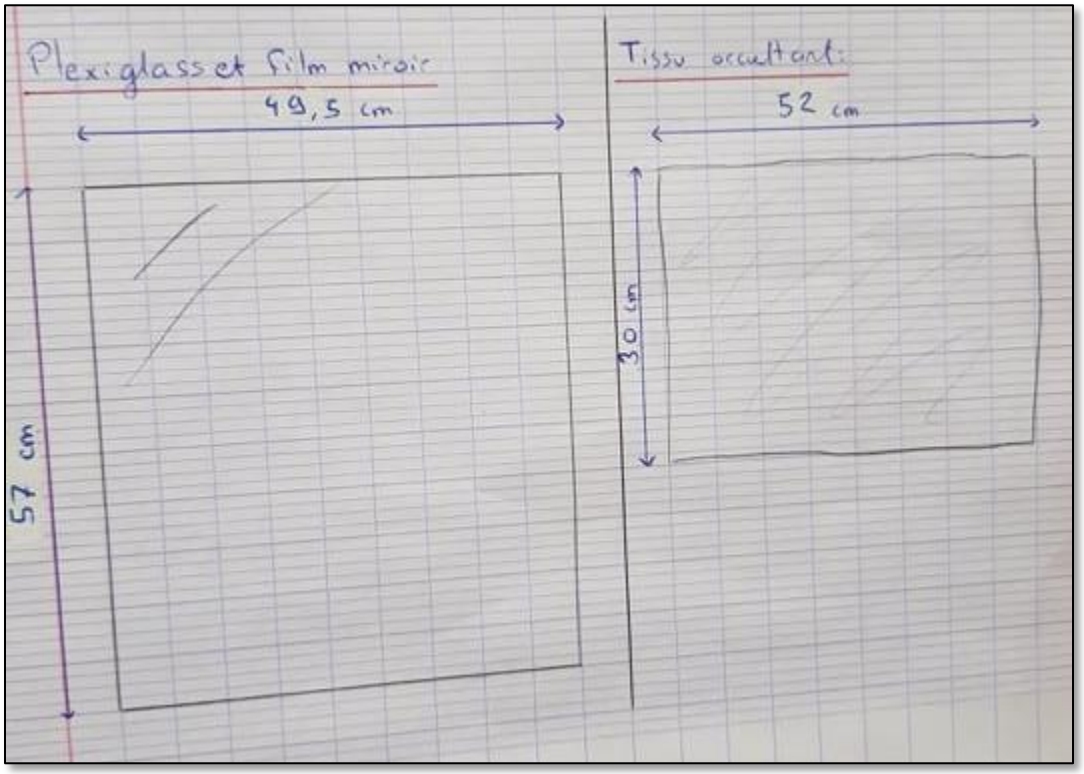
Je n'ai malheureusement pas pu accomplir tous mes objectifs. En effet, je n'ai pas intégré Alexa et mon module Keolis par manque de temps mais c'est dans mes projets dans un futur très proche. Le capteur PIR est fonctionnel mais pas efficace. Mais qu'importe, le BrightMirror fait partis d'un des plus beaux projets que j'ai effectués, je remercie encore mes parents de m'avoir financé le projet, le coût final (sans le Raspberry Pi 4 qui valait une centaine d'euros) de mon projet est d'environ 250 euros avec un poids final d'environ 10kg.

Vous trouverez des photos du rendu final du BrightMirror en annexe 6 à 9.

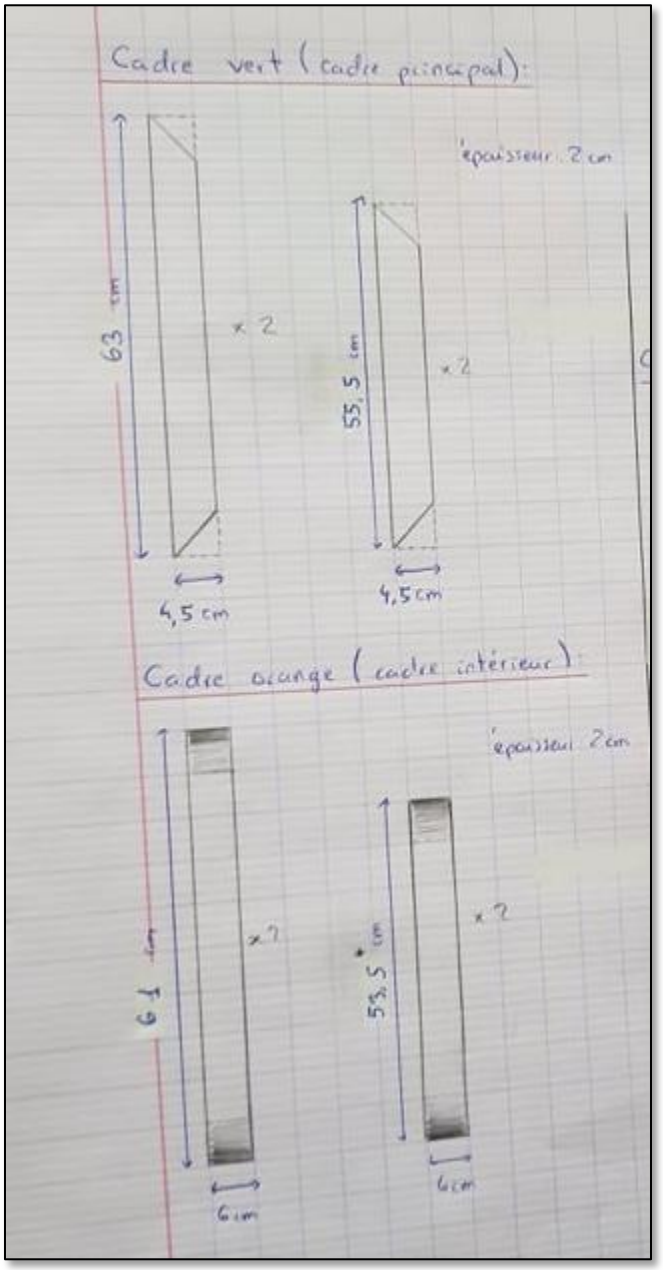
Annexe 1 : Dimensions des cadres en bois en fonction de l'écran	Page 6



Annexe 2 : Dimensions de la plaque Plexiglass et du tissu occultant	Page 6



Annexe 3 : Dimensions des tasseaux de bois	Page 6



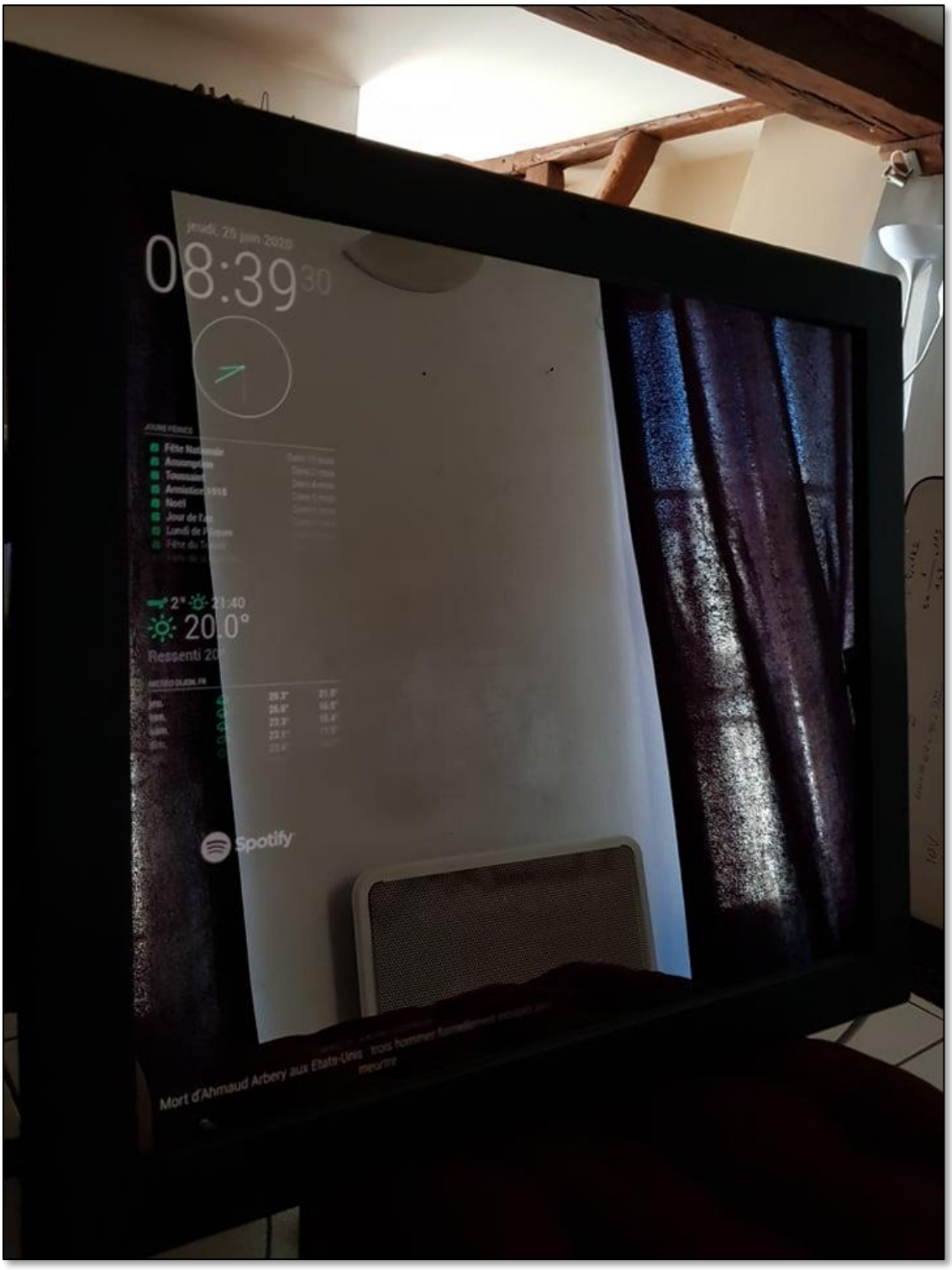
```
from RPi import GPIO
from time import *
from rpi_ws281x import *
import os
import subprocess


# Configuration des LED-------------------------------------------------
LED_COUNT      = 60      # Number of LED pixels.
LED_PIN        = 12    # GPIO pin connected to the pixels (18 uses PWM!).
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA        = 10      # DMA channel to use for generating signal (try 10)
LED_BRIGHTNESS = 100     # Set to 0 for darkest and 255 for brightest
LED_INVERT     = False   # True to invert the signal (when using NPN
transistor level shift)
LED_CHANNEL    = 0       # set to '1' for GPIOs 13, 19, 41, 45 or 53

# Variable donnant les valeurs des pixels des LED-----------------------
RED = 0
GREEN = 0
BLUE = 0
LED_STATE = 0

# Configuration des capteurs--------------------------------------------
clk = 23
dt = 22
button = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(clk, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(dt, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(button, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)

# En cas de modification de ce programme décommenter ces deux lignes
# et reboot-------------------------------------------------------------
# ~ GPIO.cleanup()
# ~ exit()

# Initialisation des variables et LED-----------------------------------
strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
LED_INVERT, LED_BRIGHTNESS, LED_CHANNEL)
strip.begin()
for i in range(0, LED_COUNT):
      strip.setPixelColor(i, Color(RED, GREEN, BLUE))
strip.show()
counter = 0
clkLastState = GPIO.input(clk)

# Toutes les fonctions--------------------------------------------------
def colorLED(red, green, blue):
      RED = red
      GREEN = green
      BLUE = blue
      if LED_STATE == 1:
            for i in range(0, LED_COUNT):
                  strip.setPixelColor(i,Color(RED,GREEN,BLUE))
```

```
            strip.show()
      elif LED_STATE == 0:
            for i in range(0, LED_COUNT):
                  strip.setPixelColor(i,Color(0,0,0))
            strip.show()

# Debut du main-------------------------------------------------------------
if __name__ == '__main__':
      print ('Ctrl-C pour quitter.')
      try:
          while True:
                clkState = GPIO.input(clk)
                dtState = GPIO.input(dt)
                buttonState = GPIO.input(button)
                if clkState != clkLastState:
                      if dtState != clkState and counter < 310:
                            counter += 2
                      elif dtState == clkState and counter > -10:
                            counter -= 2
                clkLastState = clkState
                if counter > 0 and counter < 100:
                      RED = 0
                      GREEN = counter
                      BLUE = 100-counter
                      colorLED(RED,GREEN,BLUE)
                if counter >= 100 and counter <= 2*100:
                      RED = counter-100
                      GREEN = 100-(counter-100)
                      BLUE = 0
                      colorLED(RED,GREEN,BLUE)
                if counter > 2*100 and counter <= 3*100:
                      RED = 300-counter
                      GREEN = 0
                      BLUE = counter-200
                      colorLED(RED,GREEN,BLUE)
                if counter > 305 or counter < -5:
                      colorLED(0,0,0)
                sleep(0.005)
                LED_STATE = buttonState
                print(counter, LED_STATE, RED ,GREEN,BLUE)

      except KeyboardInterrupt:
          for i in range(0, LED_COUNT):
                strip.setPixelColor(i, Color(0,0,0))
          strip.show()
          GPIO.cleanup()
          print('bye')
```

Annexe 5 : Script « *hdmi_controller.py* »	Page 14

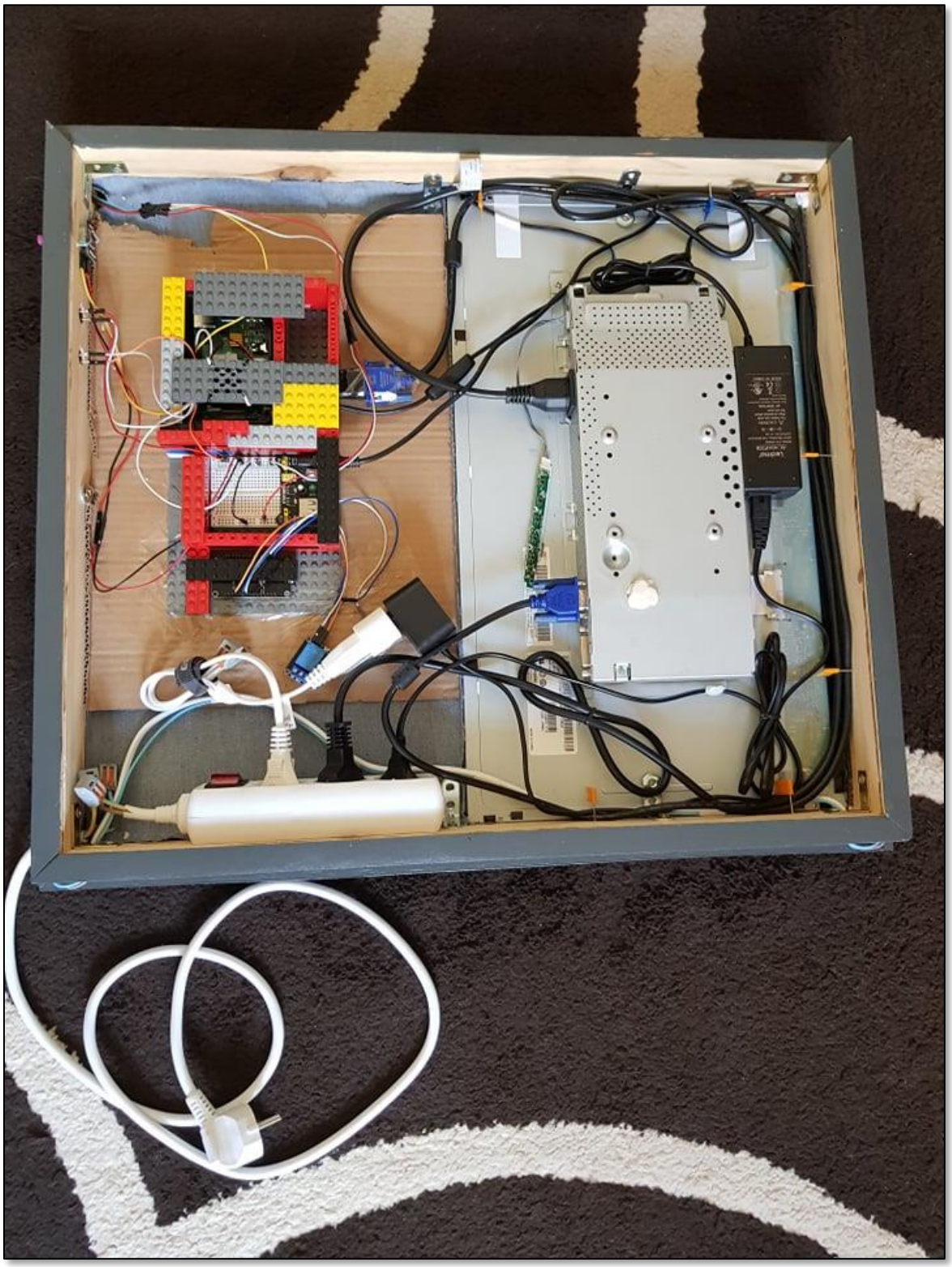
```
import RPi.GPIO as GPIO
import time
import os
import subprocess

buttonHDMI = 16
GPIO.setmode(GPIO.BCM)
GPIO.setup(buttonHDMI, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)

lasttimemotion = True

while True:
    button_state = GPIO.input(buttonHDMI)

    if button_state == 1:
        #print('motion!')
        if lasttimemotion == False:
            subprocess.call('vcgencmd display_power 1',shell=True)
            lasttimemotion = True
            #print('new motion!')
            time.sleep(1)
        time.sleep(1)
    else:
        if lasttimemotion == True:
            #print('no motion!')
            lasttimemotion = False
            subprocess.call('vcgencmd display_power 0',shell=True)
            time.sleep(1)
```

Annexe 6 : Vue du côté gauche du BrightMirror	Page 16



Annexe 7 : Vue du côté droit du BrightMirror	Page 16

Annexe 8 : Vue de l'intérieur du BrightMirror	Page 16



Annexe 9 : Vue l'arrière du BrightMirror (avec cache)	Page 16