Pour init un nouveau Raspberry (La page web afin de se connecter : 192.168.4.1)

Raspberry :

* Sur l’application RASPBIAN : uploader le modèle de la Raspberry sur la micro USB

Installation de librairies :

* Installer hostapd, dnsmasq, dhcpcd, git

Sudo apt install hostapd dnsmasq dhcpcd github

* Démasquer et enable hostapd

Sudo systemctl unmask hostapd

Sudo systemctl enable hostapd

* Fichier de configuration : /etc/hostapd/hostapd.conf

interface=wlan0

ssid=PING²

wpa\_passphrase=12345678

country\_code=FR

wpa=2

wpa\_key\_mgmt=WPA-PSK

wpa\_pairwise=TKIP CCMP

wpa\_pairwise=CCMP

driver=nl80211

hw\_mode=g

channel=6

wmm\_enabled=0

macaddr\_acl=0

auth\_algs=1

ignore\_broadcast\_ssid=0

* Fichier de configuration : /etc/dhcpcd.conf (à la fin du fichier)

interface wlan0

static ip\_address=192.168.4.1/24

nohook wpa\_supplicant

* Fichier de configuration : /etc/default/hostapd (décommenter et modifier)

DAEMON\_CONF="/etc/hostapd/hostapd.conf"

* Fichier de configuration : /etc/resolv.conf

nameserver 8.8.8.8

nameserver 8.8.4.4

* Fichier de configuration : /etc/dnsmasq.conf

interface=wlan0

dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h

GitHub :

* Cloner le dépôt dans Documents :

cd /Documents/

git clone <https://github.com/2PING2/PING2.git>

Python :

* Créer un environnement Python :

Python3 -m venv ~/Documents/python\_environnement

* Installer flask, esptool :

Pip install flask, esptool

* Créer un fichier init\_rasp.py dans Documents (c’est ce fichier qui sera exécuté au démarrage de la Raspberry et qui ensuite lancera le fichier main du jeu):

Cd Documents

Sudo nano init\_rasp.py

* Modifier le fichier : /home/pi/Documents/init\_rasp.py

import os

import subprocess

import time

import uuid

from datetime import datetime

from threading import Thread

from flask import Flask, request, send\_from\_directory

app = Flask(\_\_name\_\_)

# Paths and files

pathDirectory = '/home/pi/python\_environnement/bin/activate' # python environment path

pathRepoGithub = '/home/pi/Documents/PING2' # Github repository path

# Define paths for HTML and CSS files

pathHTML = '../Desktop/dossier/dossier/index.html' # Update with the correct path

pathCSS = '../Desktop/dossier/dossier/styles.css'  # Update with the correct path

# Define files to check for Github

filesToCheck = ["README.md", "firmware.bin"] # Update with the correct files (all files to check)

pathPrincipalMain = '/path/to/your/main.py'  # Update with the correct path (start the game)

timeForWifi = 60  # Time to wait for Wi-Fi connection in seconds

# Remove files of network configuration

os.system('sudo rm -rf /etc/NetworkManager/system-connections/\*')

# Activate the virtual environment

subprocess.run(['bash', '-c', f'source {pathDirectory}'])

# Function to check Wi-Fi connectivity

def check\_wifi():

    try:

        subprocess.run(['ping', '-c', '1', '-W', '1', 'google.com'], check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

        return True

    except subprocess.CalledProcessError:

        return False

# Function to update Git files

def update\_git():

    os.chdir(pathRepoGithub)

    subprocess.run(['git', 'fetch', 'origin'], check=True)

    for file in filesToCheck:

        if subprocess.run(['git', 'diff', '--name-only', 'origin/main'], stdout=subprocess.PIPE).stdout.decode().strip().find(file) != -1:

            # Cloning in progress...

            subprocess.run(['git', 'checkout', 'origin/main', '--', file], check=True)

            if file == "firmware.bin":

                subprocess.run(['esptool.py', '--port', '/dev/ttyUSB0', 'write\_flash', '-z', '0x0000', 'firmware.bin'], check=True)

# Flask route for Wi-Fi configuration page

@app.route("/", methods=["GET", "POST"])

def index():

    if request.method == "POST":

        # Retrieve network credentials from form

        ssid = request.form["ssid"]

        password = request.form["password"]

        connection\_uuid = str(uuid.uuid4())

        # Create NetworkManager configuration for the network

        ssid = ssid + ' '

        config\_content = f"""

[connection]

id={ssid}

uuid={connection\_uuid}

type=wifi

autoconnect=true

[wifi]

ssid={ssid}

mode=infrastructure

[wifi-security]

key-mgmt=wpa-psk

psk={password}

[ipv4]

method=auto

[ipv6]

method=ignore

"""

        config\_path = f"/etc/NetworkManager/system-connections/{ssid}.nmconnection"

        with open(config\_path, 'w') as f:

            f.write(config\_content)

        os.chmod(config\_path, 0o600)

        # Restart NetworkManager

        time.sleep(5)

        os.system("sudo systemctl restart NetworkManager")

        startingTime = datetime.now()

        #Work with dynamic check loop

        currentTime = datetime.now()

        watchdogTime = 5

        while check\_wifi() == 0 and ((currentTime - startingTime).seconds <= watchdogTime):

            currentTime = datetime.now()

        outputLatch = ((currentTime - startingTime).seconds > watchdogTime)

        if outputLatch:

            start\_services()

        return

    return send\_from\_directory(os.path.dirname(pathHTML), os.path.basename(pathHTML))  # Serve the HTML file

@app.route("/styles.css")

def styles():

    return send\_from\_directory(os.path.dirname(pathCSS), os.path.basename(pathCSS))

# Function to start services for Wi-Fi setup

def start\_services():

    global timeout

    timeout = time.time() + timeForWifi  # Reset the 60-second timer

    os.system('sudo systemctl stop hostapd')

    os.system('sudo systemctl stop dnsmasq')

    os.system('sudo systemctl start hostapd')

    os.system('sudo systemctl start dnsmasq')

    Thread(target=monitor\_services).start()  # Run service monitoring in a separate thread

# Function to stop services after a delay if not connected

def monitor\_services():

    global should\_stop, timeout

    while time.time() < timeout:

        if check\_wifi():

            should\_stop = True

            stop\_services()  # Stop services if connected

            shutdown\_server()  # Stop the Flask server

            return

        time.sleep(5)  # Check connection

    # Stop services if no connection after timeout

    if not check\_wifi():

        stop\_services()

        shutdown\_server()  # Stop the Flask server

# Function to stop services immediately

def stop\_services():

    os.system('sudo systemctl stop hostapd')

    os.system('sudo systemctl stop dnsmasq')

# Function to shutdown the Flask server gracefully

def shutdown\_server():

    os.\_exit(0)  # Forcefully stop the Flask server and end the program

# Main function to handle network logic

def main():

    global should\_stop

    should\_stop = False

    if check\_wifi():

        update\_git()  # Update Git files

        subprocess.run(['python3', pathPrincipalMain])  # Run main.py

    else:

        start\_services()  # Start services for Wi-Fi setup

        app.run(host='0.0.0.0', port=80)  # Start Flask server

if \_\_name\_\_ == "\_\_main\_\_":

    main()

* Rendre le Script Exécutable :

chmod +x /home/pi/Documents/init\_rasp.py

* Configurer afin d’exécuter le Script au Démarrage :

crontab -e

* Ajoute la ligne suivante à la fin du fichier :

@reboot bash /home/pi/Documents/init\_rasp.py

**Initialiser les ports USB de chaque manette :**

* Vérifier l’adresse Kernel du port :

udevadm info --name=/dev/ttyUSB0 --attribute-walk

* Ecrire dans le fichier suivant :

sudo nano /etc/udev/rules.d/10-usb-serial.rules

* Copier-coller le code suivant en modifiant les paramètres :

KERNEL=="ttyUSB[0-9]\*", KERNELS=="1-1.2.1", SYMLINK+="player4"

KERNEL=="ttyUSB[0-9]\*", KERNELS=="1-1.2.2", SYMLINK+="player3"

KERNEL=="ttyUSB[0-9]\*", KERNELS=="1-1.2.3", SYMLINK+="player2"

KERNEL=="ttyUSB[0-9]\*", KERNELS=="1-1.2.4", SYMLINK+="player1"

* Donner les accès :

sudo chmod 666 /dev/player\*

* Charger les paramètres :

sudo udevadm control --reload

sudo udevadm trigger

A présent les différentes ports USB seront toujours pour le même joueur !

**Réaliser une image de la Raspberry :**

