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
```

Réaliser une image de la Raspberry :

Méthode avec `dd` (Linux / macOS)

1. **Connectez la carte SD** de la Raspberry Pi à votre ordinateur avec un lecteur de carte SD.
2. **Identifiez le disque** en utilisant la commande :

   ```bash
   sudo fdisk -l
   ```

   Notez l'identifiant du disque (par exemple, `/dev/sdb`).
3. **Créer l'image** de la carte SD :

   ```bash
   sudo dd if=/dev/sdX of=/chemin/vers/sauvegarde.img bs=4M status=progress
   ```

   Remplacez `/dev/sdX` par l'identifiant de votre carte SD et `/chemin/vers/sauvegarde.img` par le chemin où vous voulez enregistrer l'image.
4. **Attendez la fin du processus**, puis éjectez la carte SD.