

Projet Réseau



Membres du groupe :

Dyhia Medjouti

Dalia Sellami

Ahmed Baaroun

Introduction :

Ce projet a pour but de créer une application qui se connecte à un réseau ADHOC en utilisant des raspberryPi et une architecture client/serveur, dans cette optique, nous avons choisi de développer un jeu : le morpion. Le but est d'avoir un raspberryPi qui soit un serveur qui *écoute* en attendant des connexions des clients et deux autres raspberryPi qui se connectent au serveur en tant que clients.

Le concept du jeu est très simple : il se pratique à deux joueurs, le but est de créer le premier un alignement sur une grille, les caractères utilisés pour remplir les grilles sont : X et O.

L'application a été codée en python en utilisant plusieurs bibliothèques dont *socket* ainsi que plusieurs structures pour sauvegarder les informations de connexion (par exemple le dictionnaire ou plus communément appelés les tables de hachage).

Problèmes rencontrés :

- Problème avec un RaspberryPi : notre tout premier problème était que l'un de nos raspberryPi n'avait pas de carte mémoire, et qu'il n'y en avait plus en stock, nous avons donc dû en acheter une.
- La plus grande difficulté que nous avons eu c'est la gestion du temps d'envoi des messages entre le serveur et des clients, au début les messages soit fusionnaient, soit ne s'envoyaient pas, on a donc du faire des pauses entre le laps de temps ou les différents messages s'envoient, en mettant des *time.sleep(1)* partout où l'erreur occure.
- Nous avons eu une légère difficulté à configurer les raspberryPi ainsi qu'à créer le point d'accès wifi depuis le serveur, néanmoins après quelques recherches nous avons pu trouver un tuto qui a permis de nous aider.

[Qu'est ce qu'un RaspberryPi ?](#)

Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM de la taille d'une carte de crédit conçu par des professeurs du département informatique de l'université de Cambridge dans le cadre de la fondation Raspberry Pi. Le Raspberry Pi fut créé afin de démocratiser l'accès aux ordinateurs et au digital making.

Analyse globale :

Cette partie sera consacrée pour détailler les différentes fonctionnalités de notre application ainsi que nos choix de programmations.

Code Serveur :

```
# Port > 1024
PORT = 8887
# IP ou on se connecte
SERVER = 'localhost' #"10.3.141.1"
ADDRESS = (SERVER, PORT)
# Format d'encodage des messages
FORMAT = 'utf-8'
# Cree le socket pour le server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDRESS)

# Dictionnaire ou on va stocker les connections
# et les roles des joueurs
clients = dict()
```

La première partie du code est destinée a la définition des différents éléments qui vont nous permettre d'établir la connexion, en effet, on crée un port qu'on choisit supérieur a 1024, puis on choisit un serveur pour créer l'adresse, ensuite le format d'encodage des messages et enfin on crée l'objet socket en lui allouant l'adresse pour ouvrir la connexion.

Class Morpion :

Cette class définit la logique du jeu, c'est le code qui permet de jouer, elle définit le rôle des joueurs, affiche la grille et attribut les rôles, elle permet également de désigner le gagnant, on y trouve ainsi plusieurs méthodes :

- `initializeBoard()` : elle initialise la grille avec des cases vides, on a ensuite la fonction `printBoard()` qui l'affiche.
- `whosStarting()` : détermine qui commence entre le X ou le O après que la fonction `randRole` ait attribué les rôles de façon aléatoire.
- `changeCurrentPlayer()` : elle switch d'un joueur a un autre quand le joueur actuel a bien effectué un coup valide.
- `isWinning()` : c'est la fonction qui calcule le coup gagnant, elle vérifie qu'il y a une ligne, une colonne ou une diagonale remplie par le meme élément et renvoie vrai si c'est le cas.

```

# Code du Morpion
class Morpion:

    def __init__(self):
        self.board = []
        self.currentPlayer = None
        self.startGame = False
        self.changement = False

    # rempli le board de case vide "-"
    def initializeBoard(self):
        for i in range(3):
            row = []
            for j in range(3):
                row.append("-")
            self.board.append(row)

    # Renvoie un role aleatoire
    def randRole(self):
        if random.randint(0, 1) == 0:
            return "X"
        else:
            return "O"

```

Reste du code :

C'est la partie comportant le code du serveur qui traite les requetes du client et lui envoie une réponse, il écoute en attendant les réponses des clients, on y trouve les fonctions suivantes :

- `start(connection, address)` : c'est la fonction qui s'occupe d'écouter les méthodes, elle prend en paramètre l'adresse et une variable connexion, puis vérifie si une partie a été lancée, si ce n'est pas le cas, elle lance une partie en choisissant un joueur qui commence de façon aléatoire, elle attend ensuite la réponse du joueur, puis envoie à qui le tour aux deux joueurs, envoie également qui a gagné et prévient si le match est nul.

```

def start(connection, address):
    # attend que les deux joueurs soient presents
    while not morpion.startGame:
        if len(clients) == 2:
            morpion.startGame = True

    # ecoute des messages
    while morpion.startGame:
        # afin de faire attendre le joueur qui ne joue pas
        morpion.changement = False
        # si la connection appartient au joueur dont s'est le tour
        if clients[connection] == morpion.currentPlayer:
            time.sleep(1)
            print('Current player : ' + morpion.currentPlayer)
            # envoie a qui de jouer a tous les joueurs
            sendAll(f'Player {morpion.currentPlayer} turn'.encode(FORMAT))
            # message contenant le coup joue par le joueur
            message = connection.recv(1024).decode(FORMAT)
            # coup de la forme "int int"
            # split le message au niveau de l'espace pour recquerer les deux
            # et les stock dans des variables
            split = message.split()
            row = int(split[0])
            col = int(split[1])
            # joue le coup du joueur
            morpion.play(row, col)

```

- `begin(morpion)` : elle initialise le serveur et attribue un rôle a chaque joueur, elle prend en paramètre un objet morpion. Elle dessine la grille en la remplissant de cases vides, puis décide qui commence, lorsque c'est fait, elle récupère l'adresse du client et le socket de connexion, elle récupère aussi le rôle du client, ensuite les associe dans le dictionnaire.

```

def begin(morpion):
    print(f'Démarrage du server sur [{SERVER}]')
    # Le server "ecoute" les nouvelles entrees
    server.listen()
    # Remplie le board de cases vides
    morpion.initializeBoard()
    # Defini qui commence
    morpion.whosStarting()
    while True:
        # Recquere le socket de connection et l'adresse du nouveau client
        connection, address = server.accept()
        # creation d'un role pour l'attribution aleatoire
        role = None
        # si 0 client dans le dictionnaire
        if len(clients) == 0:
            # premier role attribue de facon aleatoire
            role = morpion.randRole()
            # envoie du role au client
            connection.send(f'{role}'.encode(FORMAT))
            # print dans le chat du server le role, l'adresse et indique qu'il est
            print(f'[{address}] {role} - Connected')
            # associe le role et la connection dans le dictionnaire
            clients[connection] = role
            # creation du thread pour gerer le client
            thread = threading.Thread(target=start, args=(connection, address))
            thread.start()

```

Code du client :

Le début du code est simplement la définition de l'adresse, du port et du socket, nous avons ensuite la class morpion qui définit la logique du jeu et qui permet de jouer des coups, et de mettre à jour les roles et les tours.

Fonction start () : la fonction crée le jeu, initialise la grille, verifie les messages et se deconnecte si jamais le serveur est complet, elle verifie egalement quel est le role du joueur, puis quand c'est le tour du joueur, définit que c'est son tour et lui demande de jouer un coup valide (tant que le coup n'est pas valide elle demande au joueur de rejouer un coup valide), si la partie est fini on affiche le résultats sinon on dit que c'est à l'autre joueur de jouer.

```
76 def start():
77     # creation du jeu
78     morpion = Morpion()
79     # initialise le board
80     morpion.initializeBoard()
81
82     # temoin de connection
83     connected = True
84     while connected:
85         # message reçu par le client
86         message = client.recv(1024).decode(FORMAT)
87
88         # la suite du code est composee de verification des messages
89         # deconnecte le client si le server est complet
90         if message == '[SERVER FULL]':
91             print(message + '\n[DECONNECTION]')
92             connected = False
93         # interpretation du message donnant le role du joueur
94         elif len(message) == 1:
95             morpion.setPlayer(message)
96             text = f'You are the player {morpion.player}'
97             box(text)
98         # lorsqu'il s'agit du tour du joueur
99         elif message == f'Player {morpion.player} turn':
100             # defini le joueur comme le joueur dont c'est le tour
101             morpion.setCurrentPlayer(morpion.player)
```

Configuration du réseau :

Qu'est ce qu'un réseau AD-HOC ?

Un **réseau sans fil ad hoc**¹ ou WANET (Wireless Ad Hoc Network) ou MANET (Mobile Ad Hoc Network, du nom du groupe de travail qui a créé ce protocole) est un type de réseau sans fil décentralisé^{2,3}. Le réseau est ad hoc car il ne s'appuie pas sur une infrastructure préexistante, comme des routeurs dans les réseaux filaires ou des points d'accès dans les réseaux sans fil administrés⁴. Au lieu de cela, chaque nœud participe au routage en retransmettant les données aux autres nœuds, de façon que le choix du nœud qui va transmettre les données est opéré dynamiquement sur la base de la connectivité du réseau et de l'algorithme de routage utilisé⁵

Cette partie a été celle qui a demandé le plus de recherche et de travail, en effet, nous avons d'abord essayé des méthodes qui n'avaient pas marché pour configurer le réseau AD-HOC et avoir un serveur DHCP, puis nous avons trouvé un tuto qui nous a permis de créer un point d'accès WIFI et qui installe un serveur **DHCP** pour distribuer des adresses aux machines qui se connectent, le tout avec des commandes à exécuter sur le terminal du raspberryPI.

Après avoir donc créé un hotspot WIFI sur l'un des raspberryPI, on a un serveur DHCP qui distribue bien des adresses aux clients qui se connectent et donc on a bien l'architecture souhaitée.

Les grandes étapes de la configuration :

- Préparer le raspberryPI : il faut le mettre à jour, et créer un double du fichier de configuration originale du WIFI.
- Installer raspAP-webgui : qui est une interface Web simple et réactive pour contrôler le wifi, hostapd et les services associés sur le Raspberry Pi, il faut aussi redémarrer le RaspberryPI après l'installation.
- Configurer les accès RaspAP : on paramètre RaspAP et on récupère les informations liées à notre réseau.

Documentation:

Cette partie est dédiée à expliquer comment faire fonctionner le réseau et effectuer un test sur nos RaspberryPi.

Il convient tout d'abord de brancher chaque raspberryPi à un écran, il faut ensuite connecter les deux raspberryPi clients au serveur ; on change d'abord le port sur le code (attention doit être supérieur à 1024), on change également l'adresse en décommentant l'adresse mise en commentaire devant le local host sur chaque code.

Après quoi, on ouvre un terminal sur chaque écran et on appelle le code python pour l'exécuter, les commandes sont :

- Pour le serveur : Python3 server.py
- Pour les clients : Python3 client.py

Il faut veiller à avoir les bons chemins.

Pour jouer on tape des int et le format est : lignes colonnes.

Attention :

- ne pas toucher le terminal pendant que le code s'exécute ou lorsqu'il enregistre un coup, sinon le code crash.
- Une erreur de distribution des rôles peut parfois survenir, à ce moment-là il suffit de fermer le terminal et le relancer.

Conclusion:

Ce projet a été très instructif d'un point de vue technique, il nous a fallu beaucoup de documentation, d'heures de recherches et de patience, ce fut également l'occasion de développer une maîtrise plus fluide de python et des notions vues en réseau.

Ressources:

- <https://www.framboise314.fr/raspap-creez-votre-hotspot-wifi-avec-un-raspberry-pi-de-facon-express/>
- <https://static.cinay.eu/2019/08/04/RaspAP-interface-Web-pour-controler-wifi-hostapd-services-sur-RaspberryPi.html>
- https://fr.wikipedia.org/wiki/R%C3%A9seau_ad_hoc
- <https://www.clubic.com/raspberry-pi/article-849782-1-raspberry-pi-introduction-nano-ordinateur.html>