

# Rapport Réseau

## Morpion à l'aveugle

Noël Combarieu

Dimanche 11 novembre 2022

# 1 Présentation du projet

## 1.1 Fonctionnement du morpion à l'aveugle

Le morpion aveugle est une variante de ce jeu dans laquelle les joueurs ne voient pas les coups joués par l'adversaire. Si un joueur essaie de marquer une case déjà marquée par l'adversaire, le joueur est informé que cette case est prise et il doit marquer une autre case.

Vision/coups du joueur 1	Etat réel du jeu	Vision/coups du joueur 2																											
case 4																													
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td><b>X</b></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>					<b>X</b>					<table><tr><td></td><td></td><td></td></tr><tr><td></td><td><b>X</b></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>					<b>X</b>					<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>									
	<b>X</b>																												
	<b>X</b>																												
		case 2																											
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>					X					<table><tr><td></td><td></td><td><b>O</b></td></tr><tr><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>			<b>O</b>		X					<table><tr><td></td><td></td><td><b>O</b></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>			<b>O</b>						
	X																												
		<b>O</b>																											
	X																												
		<b>O</b>																											
case 6																													
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td></tr><tr><td><b>X</b></td><td></td><td></td></tr></table>					X		<b>X</b>			<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td><b>X</b></td><td></td><td></td></tr></table>			O		X		<b>X</b>			<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>			O						
	X																												
<b>X</b>																													
		O																											
	X																												
<b>X</b>																													
		O																											
		case 4																											
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>					X		X			<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>			O		X		X			<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td><b>X</b></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>			O		<b>X</b>				
	X																												
X																													
		O																											
	X																												
X																													
		O																											
	<b>X</b>																												
		case 8																											
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>					X		X			<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td><b>O</b></td></tr></table>			O		X		X		<b>O</b>	<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td><b>O</b></td></tr></table>			O		X				<b>O</b>
	X																												
X																													
		O																											
	X																												
X		<b>O</b>																											
		O																											
	X																												
		<b>O</b>																											
case 2																													
<table><tr><td></td><td></td><td><b>O</b></td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>			<b>O</b>		X		X			<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td>O</td></tr></table>			O		X		X		O	<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td>O</td></tr></table>			O		X				O
		<b>O</b>																											
	X																												
X																													
		O																											
	X																												
X		O																											
		O																											
	X																												
		O																											
case 0																													
<table><tr><td><b>X</b></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>	<b>X</b>		O		X		X			<table><tr><td><b>X</b></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td>O</td></tr></table>	<b>X</b>		O		X		X		O	<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td></td><td></td><td>O</td></tr></table>			O		X				O
<b>X</b>		O																											
	X																												
X																													
<b>X</b>		O																											
	X																												
X		O																											
		O																											
	X																												
		O																											
		case 5																											
<table><tr><td>X</td><td></td><td>O</td></tr><tr><td></td><td>X</td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>	X		O		X		X			<table><tr><td>X</td><td></td><td>O</td></tr><tr><td></td><td>X</td><td><b>O</b></td></tr><tr><td>X</td><td></td><td>O</td></tr></table>	X		O		X	<b>O</b>	X		O	<table><tr><td></td><td></td><td>O</td></tr><tr><td></td><td>X</td><td><b>O</b></td></tr><tr><td></td><td></td><td>O</td></tr></table>			O		X	<b>O</b>			O
X		O																											
	X																												
X																													
X		O																											
	X	<b>O</b>																											
X		O																											
		O																											
	X	<b>O</b>																											
		O																											

Figure 1: Fonctionnement d'un morpion à l'aveugle

## 1.2 Informations relatives au projet

Le projet est disponible sur mon GitHub : <https://github.com/NCombarieu/Morpion>  
Le projet a été réalisé en Python 3.9.1

## 2 Fonctionnalités du morpion à l'aveugle

Dans ce morpion à l'aveugle, il y a plusieurs fonctionnalités que j'ai implémenté.

### 2.1 Jouer en réseau

Le joueur peut jouer en réseau avec un autre joueur. Il suffit de démarrer le serveur et de lancer le client en renseignant l'IP et le port à utiliser.

Dans notre cas IP = localhost et PORT = 50000

Le serveur et le client communiquent via des sockets en TCP.

Les données du serveur vers le client sont transmises via un protocole que j'ai mis en place .

Le serveur envoie des variable mot clef au client pour lui indiquer ce qu'il doit executer.

J'ai ainsi le fichier constants.py qui contient les variables mot clef.

Exemple : le serveur attend le tire du joueur actuel, le serveur envoie au client le mot clef "GET\_CLIENT.SHOT" :

```
#!/usr/bin/python3
# PATH: server.py
from constants import *

def player_shot(current_player):
    """ Ask the player to play and return the case number """
    sc = clients[current_player-1]
    shot = -1
    while shot < 0 or shot >= NB_CELLS:
        sc.send(str(GET_CLIENT.SHOT).encode())
        shot = int(sc.recv(1024).decode())
    return shot
```

Le client reste à l'ecoute de ce que le serveur lui envoie avec une boucle infinie.

Si il reçoit le mot clef "GET\_CLIENT.SHOT", il rentre dans la condition correspondant.

```
#!/usr/bin/python3
# PATH: client.py
from constants import *

while True:
    """ Receive the message from the server """
    # receive action from server
    action = sc.recv(1)

    if action == GET_CLIENT.SHOT:
        sc.send(input("Quelle_case_voulez-vous_jouer_?").encode("utf-8"))
```

## 2.2 Possibilité de rejouer et de quitter

Une fois la partie terminée, les joueurs peuvent rejouer sans avoir à relancer le programme.

Il suffit de répondre par "Y" lorsque le programme demande si on veut rejouer en fin de partie.

Si on répond par autre chose, le programme se ferme.

Le protocole mis en place envoie au client le mot clef "REPLAY" pour lui demander s'il veut rejouer et c'est le client qui pose la question. Le serveur retourne VRAI ou FAUX en fonction de la réponse du client.

```
#!/usr/bin/python3
# PATH: server.py
def request_replay():
    """ Ask the players if they want to replay """
    for player in [J1, J2]:
        sc = clients[player-1]
        sc.send(str(REPLAY).encode())
        if sc.recv(1).decode() != "Y":
            return False
    return True
```

## 2.3 Affichage du score

Le score est affiché a la fin de la partie.

Il est incrémenté à chaque fois qu'un joueur gagne une partie.

```
#!/usr/bin/python3
# PATH: server.py
def send_score(Score):
    """ Send the score to the players """
    for player in [J1, J2]:
        sc = clients[player-1]
        sc.send(str(SCORE).encode())
        score = "Score_: _J1_=_ " + str(Score[J1-1]) + " _J2_=_ " + str(Score[J2-1]) + "\n"
        sc.send(score.encode())
```

## 3 Présentation du code

### 3.1 Fonctionnement du code

Le code est divisé en plusieurs fichiers.

Le fichier **server.py** contient le code du serveur.

Le fichier **client.py** contient le code du client.

Le fichier **constants.py** contient les variables utilisés pour définir mon protocole d'envoi de données entre le client et le serveur.

Le fichier **grid.py** contient le code de la grille.

#### 3.1.1 server.py

J'ai tout d'abord mis en place la création d'une socket d'écoute pour le serveur avec de la gestion d'erreur.

J'ai ensuite initialisé une liste vide qui contiendra les sockets des clients.

J'ai ensuite mis en place une boucle infinie qui va permettre d'accepter la connexion de deux clients afin de pouvoir lancer la partie.

```
# Create a TCP/IP socket
try:
    ss = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    ss.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
except socket.error as msg:
    print("Erreur de création de la socket : " + str(msg))
    exit()

HOST = 'localhost'
PORT = 50000

# Bind the socket to the port
try:
    ss.bind((HOST, PORT))
    ss.listen(2)
except socket.error as msg:
    print("Erreur de bind ou de listen : " + str(msg))
    exit()

# Define a list to store the clients sockets
clients = []

print("Serveur en attente de connexion...")
while len(clients) < 2:
    """ Wait for 2 clients """
    readl, _, _ = select.select(clients+[ss], [], [])
    for s in readl:
        if s is ss:
            client, addr = s.accept()
            clients.append(client)
            print("Client connecté : " + str(addr))
            s.send(str(NEW_PLAYER).encode())
```

Une fois les deux clients connectés, j'ai mis en place une boucle qui va permettre de lancer la partie tant que les joueurs souhaitent rejouer.

```
while rejouer:
    """Main loop"""
    grids = [grid(), grid(), grid()]
    current_player = J1
```

```

while grids[0].gameOver() == -1:
    """ Game loop """
    player_send_grid(current_player, grids[current_player])
    shot = player_shot(current_player)
    if (grids[0].cells[shot] != EMPTY):
        grids[current_player].cells[shot] = grids[0].cells[shot]
        player_send_grid(current_player, grids[current_player])
    else:
        grids[current_player].cells[shot] = current_player
        grids[0].play(current_player, shot)
        player_send_grid(current_player, grids[current_player])
        current_player = current_player%2+1

    print(" Grille_0\n")
    grids[0].display()

for player in [J1, J2]:
    if grids[0].gameOver() == player:
        score[player-1] += 1
        send_win_to_player(player)
    else:
        send_loose_to_player(player)

send_score(current_player, score)
rejouer = request_replay()

print(" Partie termin e , les joueurs ne veulent pas rejouer\n")

```

Si les joueurs ne souhaitent pas rejouer, je ferme la connexion avec les clients.

```

# fermer les socket clients
print("Fermeture_des_sockets_clients")
for client in clients:
    client.close()
# fermer la socket serveur
ss.close()

```

### 3.1.2 client.py

Lorsque le joueur lance le programme, il doit rentrer l'adresse IP et le port du serveur. Il y a une gestion d'erreur lors du lancement du programme en vérifiant le nombre de parametre rentrer.

```

#!/usr/bin/python3
# PATH: client.py
import socket
import sys
from constants import *

if len(sys.argv) != 3:
    print(" Usage: _python3_client.py_<ip>_<port>")
    exit()

HOST = sys.argv[1]
PORT = int(sys.argv[2])

```

Ensuite, je crée une socket pour le client et je me connecte au serveur. Il y a aussi une gestion d'erreur en cas d'erreur lors de la création de la socket ou bien de la connexion au serveur si host ou port n'est pas correct.

```

# Create a TCP/IP socket
try:
    sc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

```

```

except socket.error as msg:
    print("Erreur de creation de la socket: " + str(msg))
    exit()

# Connect the socket to the port where the server is listening
try:
    sc.connect((HOST, PORT))
except socket.error as msg:
    print("Erreur de connexion: " + str(msg))
    exit()

```

Une fois le client connecté au serveur, j'ai créé une boucle infinie qui va permettre de recevoir les messages du serveur.

Ainsi je pourrais recevoir la grille du joueur, le message de demande de tirer, le message de fin de partie, le score, le message de fin de partie ou encore vérifier si le serveur n'est pas déconnecté (le cas où je ne reçois rien).

```

while True:
    """ Receive the message from the server """
    # receive action from server
    action = sc.recv(1)
    # if action is empty, the server has closed the connection
    if action == b"":
        break
    # parse action to int and execute the corresponding action
    action = int(action.decode())

    if action == SHOW_GRID:
        grid = sc.recv(98).decode()
        print(grid)
    elif action == GET_CLIENT_SHOT:
        sc.send(input("Quelle case voulez-vous jouer ?").encode("utf-8"))
    elif action == WINNER:
        print("You WIN!")
    elif action == LOOSER:
        print("You Loose noob!")
    elif action == SCORE:
        score = sc.recv(22).decode()
        print(score)
    elif action == REPLAY:
        sc.send(input("Voulez-vous rejouer ? (Y/N)\n").encode("utf-8"))

```

### 3.1.3 constants.py

```

#!/usr/bin/python3
# PATH: constants.py
symbols = ['_', 'O', 'X']
EMPTY = 0
J1 = 1
J2 = 2
NB_CELLS = 9

# Actions
GET_CLIENT_SHOT = 3
SHOW_GRID = 4
WINNER = 5
LOOSER = 6
REPLAY = 7
SCORE = 8

```

J'ai dupliqué la fonction `display()` que j'ai appelé `sendDisplay()` de la grille pour pouvoir l'envoyer correctement au client avec la méthode `send()`.

```

def sendDisplay(self):
    output = "—————\n"
    for i in range(3):
        output += "|_ " + symbols[self.cells[i*3]] + "_|" + symbols[self.cells[i*3+1]] +
        "_|" + symbols[self.cells[i*3+2]] + "_|\n"
        output += "—————\n"
    return output

```

## 4 Conclusion

Ce projet m'a permis de découvrir le réseau et de me familiariser avec les sockets. J'ai pu apprendre à utiliser les sockets en python et à créer un serveur et un client ainsi que gérer plusieurs clients à la fois.

J'ai eu quelques difficultés tel que l'utilisation de git avec lequel je suis de plus en plus à l'aise, mais ou je n'arrive pas encore découpé mes commits pour qu'ils soient plus clairs et concis.

J'ai aussi eu quelques difficultés avec la mise en place du protocole d'envoi de données entre le client et le serveur car je ne savais pas exactement comment le faire. Néanmoins, celui que j'ai mis en place semble correct.

De plus, le travail en groupe fut compliqué car nous n'avions pas de communication régulière et même plus aucune communication à la fin du projet.

En revanche j'ai pu produire une version fonctionnelle du jeu du morpion à l'aveugle en réseau avec quelques fonctionnalités de base.