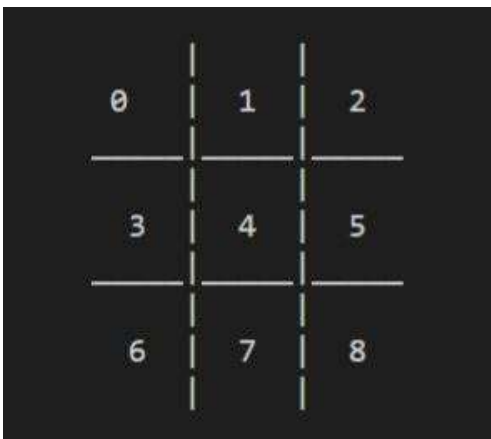


Le Jeu:

Le Morpion est un jeu à deux joueurs, qui est normalement joué sur une grille 3x3 mais peut-être aussi joué sur un 4x4 ou 5x5. Chaque joueur occupe une case par tour, avec le but de réaliser une des combinaisons gagnantes. Un premier joueur utilisera le 'x' comme sa marque, alors que le deuxième joueur utilisera le 'o'.

Etape 1: Design du grille



Quand un joueur veut marquer une case particulière, il doit entrer le numéro correspondant indiqué dans la grille. Supposons que nous souhaitions occuper le bloc central, alors nous entrerons 5 dans le terminal.

Cette grille peut être générée par le code:

```
def print_tic_tac_toe(values):
    print("\n")
    print("\t\t | \t\t |")
    print("\t {} | {} | {}".format(values[0], values[1], values[2]))
    print('\t_____|_____|_____')

    print("\t\t | \t\t |")
    print("\t {} | {} | {}".format(values[3], values[4], values[5]))
    print('\t_____|_____|_____')

    print("\t\t | \t\t |")

    print("\t {} | {} | {}".format(values[6], values[7], values[8]))
    print("\t\t | \t\t |")
    print("\n")
```

Dans le code ci-dessus, la fonction crée notre jeu selon les valeurs livrées en argument. Ici, l'argument `values` est une liste contenant le statut de chaque cellule de la grille.

Etape 2: Créer table de score

La table de score est stockée sous forme de dictionnaire, où les clés sont les noms des joueurs et les valeurs sont leur nombre de victoires.

```
def print_scoreboard(score_board):

    print("\t-----")

    print("\t\t\t\t\tSCOREBOARD\t\t\t")

    print("\t-----")


    players = list(score_board.keys())

    print("\t\t", players[0], "\t\t", score_board[players[0]])

    print("\t\t", players[1], "\t\t", score_board[players[1]])

    print("\t-----\n")
```

Pour afficher la table de score, nous avons besoin des noms des joueurs. Les clés sont extraites à l'aide de la fonction `.keys()` puis converti en liste, afin qu'il puisse être indexé lors de l'affichage des scores.

Résultat:

```
-----
                SCOREBOARD
-----
oumaima                0
sohaib                 0
-----
```

Step 3: Vérifier si c'est une victoire ou une partie nulle

Après chaque mouvement, nous devons vérifier si un joueur a gagné le jeu ou si c'est une partie nulle. Il peut être vérifié par le code :

```
# Function to check if any player has won
def check_win(player_pos, cur_player):

    # All possible winning combinations
    soln = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [1, 4, 7], [2, 5, 8], [3, 6, 9], [1, 5, 9], [3, 5, 7]]

    # Loop to check if any winning combination is satisfied
    for x in soln:
        if all(y in player_pos[cur_player] for y in x):

            # Return True if any winning combination satisfies
            return True

    # Return False if no combination is satisfied
    return False

# Function to check if the game is drawn
def check_draw(player_pos):
    if len(player_pos['X']) + len(player_pos['O']) == 9:
        return True
    return False
```

Si n'importe quel joueur gagne, alors la fonction **single_game()** retourne le joueur courant, qui a fait le mouvement. Dans le cas où, c'est une partie nulle (draw), "D" est renvoyé.

check_win() – La fonction a toutes les combinaisons gagnantes. Tout ce qu'elle fait est de vérifier si l'une des combinaisons gagnantes est satisfaite par les positions du joueur actuel. Si elle le fait, il retourne True. Si aucune des combinaisons n'est satisfaite, alors la fonction retourne False.

check_draw() - La condition du "draw" est assez simple, si les neuf positions sont prises c'est une partie nulle ou un Draw.

Step 4: Storage d'information

À tout moment, nous avons besoin de deux informations cruciales :

- **État de la grille** – Nous devons avoir une structure de données qui stocke l'état de chaque cellule, c'est-à-dire si elle est occupée ou vacante.
- **Mouvements de chaque joueur** – Nous devons en quelque sorte connaître les mouvements passés et présents de chaque joueur, c'est-à-dire les positions occupées par « X » et « O ».

```
# Function for a single game of Tic Tac Toe
def single_game(cur_player):

    # Represents the Tic Tac Toe
    values = [' ' for x in range(9)]

    # Stores the positions occupied by X and O
    player_pos = {'X':[], 'O':[]}
```

Le statut de la grille est géré par une liste de caractères, qui peut avoir trois valeurs possibles: ' ', 'X' et 'O' .

Les mouvements de chaque joueur sont stockés comme un dictionnaire d'une liste de nombres entiers. Les clés sont 'X' et 'O' pour chaque joueur respectif. Leurs listes correspondantes contiennent les numéros donnés aux cellules de la grille qu'elles occupent.

Step 5: Boucle de jeu

Chaque jeu a une sorte de boucle de jeu, qui se déroule jusqu'à ce que certains joueurs gagnent ou le jeu se termine par un draw.

Dans morpion, chaque itération de boucle se réfère à un seul mouvement que n'importe quel joueur fait.

```
# Game Loop for a single game of Tic Tac Toe
while True:
    print_tic_tac_toe(values)
```

Step 6: Vérification des positions

Afin de faire avancer le jeu de morpion, il nous faut vérifier que la position choisie par le joueur soit une position, dite "correcte". Pour cela, on a utilisé les blocs suivants :

- **try** : lorsque le joueur va choisir sa position et si la position (**move**) n'est pas correcte, on va envoyer un message d'erreur.

```
# Try exception block for MOVE input

    try:

        print("Player ", cur_player, " turn. Which box? : ", end="")

        move = int(input())

    except ValueError:

        print("Wrong Input!!! Try Again")

        continue
```

Il y a différentes possibilités pour que la position ne soit pas correcte : soit la position n'est pas comprise dans le *board* ou soit la position a été déjà prise. Pour pallier ce problème, nous avons réalisé des boucles *if*.

- Si la position n'est pas comprise dans le board, on renvoie un message d'erreur et on retourne au début de la loop

```
# Sanity check for MOVE inout

    if move < 1 or move > 9:

        print("Wrong Input!!! Try Again")

        continue
```

- Pour vérifier si la position a déjà été utilisée, il nous faut vérifier qu'il n'y pas de *str* dans les values. Dans le cas positif, on renvoie un message d'erreur.

```
- # Check if the box is not occupied already
-
-     if values[move-1] != ' ':
-
-
-         print("Place already filled. Try again!!")
-
-
-         continue
```

Il nous faut mettre à jour les données de positions à chaque tour, pour cela, nous mettons à jour les variables concernant les positions.

```
# Update game information

# Updating grid status

values[move-1] = cur_player

# Updating player positions

player_pos[cur_player].append(move)
```

On stocke les positions dans un tableau pour après les comparer aux positions gagnantes.

Pour renvoyer le gagnant, on réalise une condition qui retourne le gagnant, soit le joueur qui vient de mettre sa marque (current player)

```
# Function call for checking win

    if check_win(player_pos, cur_player):

        print_tic_tac_toe(values)

        print("Player ", cur_player, " has won the game!!")

        print("\n")

        return cur_player
```

Lors d'une égalité, on va retourner un message qui nous indique qu'il y a une égalité, donc 0 vainqueurs.

```
if check_draw(player_pos):

    print_tic_tac_toe(values)

    print("Game Drawn")

    print("\n")

    return 'D'
```

Enfin, il nous faut faire changer de joueur à chaque tour, donc à la fin de la boucle while, on implémente une condition qui fait modifier le joueur actuel.

```
# Switch player moves

    if cur_player == 'X':

        cur_player = 'O'

    else:

        cur_player = 'X'
```

Step 7 : la fonction main

Au départ de la fonction principale, il nous faut mettre en place les noms des joueurs :

```
if __name__ == "__main__":

    print("Player 1")

    player1 = input("Enter the name : ")

    print("\n")

    print("Player 2")

    player2 = input("Enter the name : ")

    print("\n")
```


On va affecter à la variable *cur_player* la joueur 1 :

```
# Stores the player who chooses X and O

cur_player = player1
```

On va stocker le joueur correspondant à la marque respective (X ou O) dans un dictionnaire :

```
# Stores the choice of players

player_choice = {'X' : "", 'O' : ""}
```

On stocke les différentes marques dans un tableau :

```
# Stores the options

options = ['X', 'O']
```

On initialise l'historique :

```
# Stores the scoreboard

score_board = {player1: 0, player2: 0}

print_scoreboard(score_board)
```

Step 7 - 1 : Plusieurs jeu de morpions :

Nous allons faire une boucle pour lancer le jeu et la suite (si choisie) et qui va s'arrêter lorsque les joueurs décideront de quitter le jeu.

Le joueur a 3 différent choix au début de la partie : choisir X, choisir O ou bien quitter le jeux :

```
# Game Loop for a series of Tic Tac Toe

# The loop runs until the players quit

while True:

    # Player choice Menu

    print("Turn to choose for", cur_player)

    print("Enter 1 for X")

    print("Enter 2 for O")

    print("Enter 3 to Quit")
```

Lorsque que le joueur ne rentre pas un chiffre correct (i.e ne correspondant pas aux différents choix), on renvoie un signal d'erreur:

```
# Try exception for CHOICE input

try:

    choice = int(input())

except ValueError:

    print("Wrong Input!!! Try Again\n")

    continue
```

Par la suite, il nous suffit de rajouter les conditions selon le choix du joueur :

- Choix 1 : le joueur veut être X

```
# Conditions for player choice

    if choice == 1:

        player_choice['X'] = cur_player

        if cur_player == player1:

            player_choice['O'] = player2

        else:

            player_choice['O'] = player1
```

- Choix 2 : le joueur veut être O

```
elif choice == 2:

    player_choice['O'] = cur_player

    if cur_player == player1:

        player_choice['X'] = player2

    else:

        player_choice['X'] = player1
```

- Choix 3 : le joueur ne veut plus jouer :

On renvoie l'historique des matchs joués :

```
elif choice == 3:

    print("Final Scores")

    print_scoreboard(score_board)

    break

else:

    print("Wrong Choice!!!! Try Again\n")
```

Step 7 - 2 : Choix du niveau de difficulté

Le joueur, ayant choisi sa marque (X ou O) a donc le choix entre 3 niveaux de difficultés (Easy : grille 3x3; Medium : grille 4x4; Hard : grille 5x5) :

```
# Player choice of difficulties (3x3,4x4,5x5)

    print("Turn to choose the difficulty for", cur_player)

    print("Enter 1 for 3x3 (Easy)")

    print("Enter 2 for 4x4 (Medium)")

    print("Enter 3 for 5x5 (Hard)")
```

Lorsque le joueur rentre un mauvais chiffre (i.e différents de 1,2 ou 3), un message d'erreur va s'afficher. On va alors inviter le joueur à bien choisir cette fois-ci.

```
#Try exception for player level choice

    try:

        level_choice = int(input())

    except ValueError:

        print("Wrong Input!!! Try Again\n")

        continue
```

Pour les différents choix, nous allons ainsi lancer une partie de morpion, selon la difficulté choisie, et enregistrer le vainqueur de la partie :

```
# Conditions for player choice

    if level_choice == 1:

        winner = single_game3x3(options[choice-1])

    elif level_choice == 2:

        winner = single_game4x4(options[choice-1])

    elif level_choice == 3:

        winner = single_game5x5(options[choice-1])
```

Step 7 - 3 : Mise à jour de l'historique

Lorsqu'il y a eu un vainqueur (pas d'égalité), on incrémente le score du joueur gagnant et on affiche l'historique :

```
# Edits the scoreboard according to the winner

    if winner != 'D' :

        player_won = player_choice[winner]

        score_board[player_won] = score_board[player_won] + 1

    print_scoreboard(score_board)
```

Enfin, il nous suffit de faire tourner les joueurs, c'est-à-dire, d'inverser les marques X et O :

```
# Switch player who chooses X or O

    if cur_player == player1:

        cur_player = player2

    else:

        cur_player = player1
```