Morpion +

Felipe Thibaut Robert

10 Janvier 2024

1 Introduction

1.1 Explication

1.2 Exemple

2 Code python

2.1 Explication

1 Introduction

1.1 Explication

Voici un exemple d'un tableau de jeu

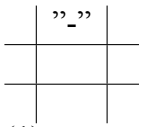
1	2	3
4	5	6
7	8	9

Comme vous pouvez le voir, le Morpion + est composé de neuf cases, tout comme le Morpion classique, il se joue aussi tour par tour en plaçant son élément respectif (ici, le "|" et le "-") et se gagne lorsque l'on arrive à mettre 3 éléments identiques côte à côte, que se soit en les alignant sur l'une des colonnes, lignes et/ou diagonales. Là où cette version donne plus de variété au partie aux parties, c'est en permettant de placer son éléments SUR celui de l'adversaire pour en créer un nouveau le "+" ! Si 3 "+" sont alignés, la personne qui gagne est celle qui à placé le dernier "+". Cependant il existe une règle supplémentaire, il est impossible de placer son élément sur celui de l'adversaire 2 tours consécutifs et si la personne qui joue n'a pas d'autre option, elle perd don tour. (Cette règle existe pour éviter de laisser le 2ème joueur seulement créer des "+" après le coup de son adversaire et ainsi gagner à tous les coups).

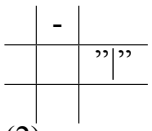
En résumé :

- Chaque adversaire a son propre élément le "|" ou le "-".
- Le jeu se déroule en tour par tour en plaçant son élément respectif.
- Il est possible de placer son élément sur celui de l'adversaire pour créer un "+".
- Il est impossible de placer son élément sur celui de l'adversaire 2 tours consécutifs.
- Si un joueur ne peut ni placer son éléments ni créer de "+", il perd son tour.
- Le 1er joueur a avoir aligné 3 éléments identiques gagne !

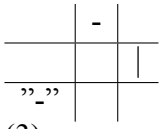
1.2 Exemple de partie



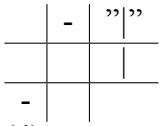
(1)



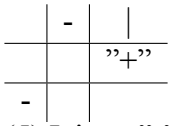
(2)



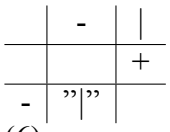
(3)



(4)



(5) Ici, un ”-” a été placé par dessus le ”|” au milieu à droite.



(6)

”-”	-	|
	|	+
-	| |	

(7)

”+”	-	|
	|	+
-	| |	

(8) Un ”|” a été placé par dessus un ”-” en haut à gauche.

+	-	|
	”-”	+
-	|	

(9)

+	-	|
	-	+
-	| |	”|”

(10)

+	-	|
	-	+
-	| |	”+”

(11) Un ”-” a été placé par dessus un ”|” en bas à droite.

”+”	-	|
	”+”	+
-	|	”+”

(12) Un ”|” a été placé par dessus un ”-” au milieu. Cela implique que le joueur ”|” a gagné !

2 Code python

2.1 Explication

Nous commençons par importer les librairies dont nous avons besoin, dans ce cas, pygame

```
import pygame
```

Ensuite nous déclarons quelques variables constantes, comme la taille de la fenêtre, quelques couleurs et la taille des lignes

```
# Constant variables
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
LINE_WIDTH = 12
```

Ensuite nous déclarons quelques variables que nous allons utiliser et changer pendant le déroulement du jeu comme ”current_player” qui change entre 1 et -1 en fonction de quel jour doit jouer, ”theme” qui décide la couleur du font d’écran et des traits en fonction de ce que le jouer choisit comme thème au début du programme, et ”played_plus_last_round”, une liste qui se rappelle si le joueur a fait un ”plus” la dernière fois qu’il a joué. Elle est composée de 3 indexes même si il n’y a que 2 joueurs afin de simplifier le code car le joueur 1 prend la position 1 de la liste et le joueur -1 prend la dernière position de la liste, donc la position 2. La position 0 n’est jamais utilisée.

```
# Other variables
current_player = 1
theme = 0
played_plus_last_round = [False, False, False]
```

Ensuite nous déclarons quelques fonctions comme :
”DrawGrid” qui est appelle chaque frame et a le role de remplir l’ecran avec la couleur du background (que nous alons definir plus tard) et de dessiner le ”terrain” soir les 3 lignes verticales et les 3 lignes horizontaux qui constituent le zone de jeu

```
def DrawGrid():
    screen.fill(background)
    for x in range(1, 3):
        pygame.draw.line(screen, draw_color, (x * SCREEN_WIDTH / 3, 0), (x * SCREEN_WIDTH / 3, SCREEN_HEIGHT), LINE_WIDTH)
        pygame.draw.line(screen, draw_color, (0, x * SCREEN_HEIGHT / 3), (SCREEN_WIDTH, x * SCREEN_HEIGHT / 3), LINE_WIDTH)
```

”DrawPlayers” qui a comme paramètre ”board”, une liste qui contient toutes les positions ou les joueurs ont joue. Elle est appelle chaque frame et a le rôle de remplir la zone de jeu avec les signes que les joueurs ont joue. Elle parcourt tout simplement la liste ”board” et si elle trouve un 1 ça veut dire que le joueur 1 a pose un -, si elle trouve un -1 ça veut dire que le joueur 2 a joue un |, et finalement si elle trouve 2 ça veut dire que les deux joueurs ont joue dans la même position et il faut dessiner un +. Pour dessiner les lignes le code utilise la taille de la fenêtre que nous avons déclare précédemment et donc même si on change la taille de la fenêtre, les positions et les tailles des symboles vont s’adapter. J’aimerais bien aller un peu plus en détail sur les maths mais honnêtement ça fait 2 mois que j’ai fait ce code et je n’ai pas envie de me rappeler de cette partie car je me rappelle avoir passe beaucoup trop de temps :)

```
# Draw the players' moves (j'ai passe bcp trop de temps a faire les calcules j'en ai marre)
def DrawPlayers(board):
    x_pos = 0
    for x in board:
        y_pos = 0

        for y in x:
            if y == 1:
                pygame.draw.line(screen, draw_color,
                                 ((x_pos * 2 + 1) * (SCREEN_WIDTH / 6), (y_pos * (SCREEN_HEIGHT / 3)) + SCREEN_HEIGHT / 18),
                                 ((x_pos * 2 + 1) * (SCREEN_WIDTH / 6), ((y_pos + 1) * (SCREEN_HEIGHT / 3)) - SCREEN_HEIGHT / 18), LINE_WIDTH)
            elif y == -1:
                pygame.draw.line(screen, draw_color,
                                 ((x_pos * (SCREEN_WIDTH / 3) + SCREEN_WIDTH / 18), (y_pos * 2 + 1) * (SCREEN_HEIGHT / 6)),
                                 (((x_pos + 1) * (SCREEN_WIDTH / 3) - SCREEN_WIDTH / 18), (y_pos * 2 + 1) * (SCREEN_HEIGHT / 6)), LINE_WIDTH)
            elif y == 2:
                pygame.draw.line(screen, draw_color,
                                 ((x_pos * 2 + 1) * (SCREEN_WIDTH / 6), (y_pos * (SCREEN_HEIGHT / 3)) + SCREEN_HEIGHT / 18),
                                 ((x_pos * 2 + 1) * (SCREEN_WIDTH / 6), ((y_pos + 1) * (SCREEN_HEIGHT / 3)) - SCREEN_HEIGHT / 18), LINE_WIDTH)
                pygame.draw.line(screen, draw_color,
                                 ((x_pos * (SCREEN_WIDTH / 3) + SCREEN_WIDTH / 18), (y_pos * 2 + 1) * (SCREEN_HEIGHT / 6)),
                                 (((x_pos + 1) * (SCREEN_WIDTH / 3) - SCREEN_WIDTH / 18), (y_pos * 2 + 1) * (SCREEN_HEIGHT / 6)), LINE_WIDTH)
            y_pos += 1

        x_pos += 1
```

”HandleMouseClick” qui a comme paramètres pos qui est une liste contenant la position x et y du souris, ”board” la même liste que pour la fonction précédente, ”player” qui est le même que ”current_player” déclaré précédemment, et ”played_plus” qui est le même que ”played_plus_last_round”. Elle est appelle a chaque fois que le joueur appuye sur le clic gauche. Elle trouve le carre ou le joueur a clique, vérifie si il est occupe et si possible de faire un plus. Elle renvoyé un 1 si le clic est invalide et le joueur ne peut pas jouer cette position et -1 si le clic est valide et le joueur a joue cette position. A la fin (mais avant de changer le ”player”) elle vérifie si le joueur courent a gagne en appellant ”CheckWin”.

```
# Handles the mouse click (checks position, checks board and updates the board)
def HandleMouseClick(pos, board, player, played_plus):
    cell_x = pos[0]
    cell_y = pos[1]

    # The // 3 is used to get the cell's position in the board (divides the screen into 3x3 cells)
    if board[cell_x // (SCREEN_WIDTH // 3)][cell_y // (SCREEN_HEIGHT // 3)] == 0:
        board[cell_x // (SCREEN_WIDTH // 3)][cell_y // (SCREEN_HEIGHT // 3)] = player
        played_plus[player + 1] = False

    # elif for the "plus" mechanic
    elif (board[cell_x // (SCREEN_WIDTH // 3)][cell_y // (SCREEN_HEIGHT // 3)] != player
          and board[cell_x // (SCREEN_WIDTH // 3)][cell_y // (SCREEN_HEIGHT // 3)] != 2
          and (not played_plus[player + 1] or 0 not in board)):
          # The previous line checks if player has already played a plus last round but if no other places available skip his turn (in reality just lets him play whatever he wants instead of skipping 2 turns)
        board[cell_x // (SCREEN_WIDTH // 3)][cell_y // (SCREEN_HEIGHT // 3)] = 2
        played_plus[player + 1] = True

    else:
        return 1

    CheckWin(board, player)
    return -1
```

”CheckWin” qui a comme paramètres ”board” la même liste que pour la fonction précédente et ”player” qui est le même que ”current_player”. Elle est appelle a chaque fois que un joueur place un symbole. Elle vérifie ”manuellement” chaque possibilité de gagner et si elle trouve une qui est vraie elle arrête la boucle du jeu et écrit dans la console le joueur qui a gagne. Cette manière n’est pas très efficace mais après avoir passe des heures a faire les maths pour bien afficher les symboles et les faire adaptable aux différents tailles, c’est tout ce que j’avais l’énergie de faire

```
def CheckWin(board, player):
    global run
    if (
            board[0][0] == board[0][1] == board[0][2] != 0 or
            board[1][0] == board[1][1] == board[1][2] != 0 or
            board[2][0] == board[2][1] == board[2][2] != 0 or
            board[0][0] == board[1][0] == board[2][0] != 0 or
            board[0][1] == board[1][1] == board[2][1] != 0 or
            board[0][2] == board[1][2] == board[2][2] != 0 or
            board[0][0] == board[1][1] == board[2][2] != 0 or
            board[0][2] == board[1][1] == board[2][0] != 0
    ):
        if player == -1:
            player = 2
        print(f"Player {player} won!")
        run = False
```

Ensuite le ”script” commence par demander a l’utilisateur la thème qu’il veut et calcule les couleur des variables ”background” et ” draw_color”

```
# Get the user's desired theme
theme = input("Enter theme (light/dark): ").lower()
while theme not in ["light", "dark", "l", "d"]:
    theme = input("Theme doesn't exist, please enter light(l) or
    dark(d): ").lower()

if theme == "light" or theme == "l":
    theme = 1
else:
    theme = 0

    # Set the background and draw color according to the theme
background = (255 * theme, 255 * theme, 255 * theme)
draw_color = (255 * (1 - theme), 255 * (1 - theme), 255 * (1 -
   theme))
```

Ensuite nous initialisons pygame et la fenêtre a qui nous donnerons un titre et une icône

```
# Initialize pygame
pygame.init()
# Create the screen (window)
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the title of the window
pygame.display.set_caption("TicTacToe +")
pygame.display.set_icon(pygame.image.load("icon.png"))
```

Ensuite nous déclarons une variable appelle ”game_board” qui, comme nous avons préciseé antérieurement est une liste a deux dimensions (soit une liste de listes) qui a les 9 positions du tableu du jeu

```
game_board = \
    [
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]
    ]
```

Et pour finir nous avons la boucle du jeu qui appelle les fonctions ”DrawGrid” et ”DrawPlayers” déclares antérieurement et s’occupe des évènements comme ”pygame.QUIT” qui arrive quand l’utilisateur essaye de fermer la fenetre et ”pygame . MOUSEBUTTONDOWN” qui arrive quand le joueur appuye sur le clic gauche. Nous multiplions ”current_player” par le resultat de la fonction ”HandleMouseClick” car la fonction renvoye un -1 si le clic est valid et il faut donc changer le joeur (et quand on multiplie 1 ou -1 par -1 on obtient l’autre) et un 1 si le clic n’est pas valide et donc rien ne change

```
# Game loop
run = True
while run:
    # Draw the grid and the players' moves
    DrawGrid()
    DrawPlayers(game_board)

    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

        if event.type == pygame.MOUSEBUTTONDOWN:
            current_player *= HandleMouseClick(pygame.mouse.
    get_pos(), game_board, current_player, played_plus_last_round
    )

    # Update game state
    pygame.display.update()

pygame.quit()
```