Contexte :

Projet : Jeu de Morpion
Langage : Python
Dépendances : Le code nécessite la bibliothèque NumPy pour fonctionner. Vous devez avoir un environnement Python configuré avec NumPy installé pour exécuter ce jeu.
Objectif: comprendre le code fournis, les éventuels problèmes et produire une version du jeu dans un nouveau langage.

Erreurs relevées :

Inversion Colonne/Ligne :

L'une des erreurs notables est une inversion entre les termes "colonne" et "ligne" dans le code. Cela affecte principalement la saisie des coordonnées par l'utilisateur. Lorsque l'utilisateur est invité à choisir une colonne, le code utilise ces valeurs comme indices pour les lignes, et vice versa. Cela peut entraîner une confusion pour les joueurs. La correction de cette erreur consiste à inverser les termes dans les instructions d'entrée utilisateur et les appels de fonction Result.

Rétro-ingénierie :

Les variables:

Bien sûr, voici une explication des variables utilisées dans le code :

grille: Une matrice (tableau bidimensionnel) 3x3 représentant la grille de jeu du Morpion. Les cases vides sont représentées par des zéros (0), les cases remplies par l'IA par des uns (1), et les cases remplies par le joueur humain par des deux (2).

jIa: Une variable qui stocke le numéro du joueur IA. Dans ce code, le numéro de l'IA est 1.

jHumain: Une variable qui stocke le numéro du joueur humain. Dans ce code, le numéro du joueur humain est 2.

nbparties: Un compteur qui enregistre le nombre de parties jouées.

score1 et score2: Deux variables qui enregistrent le score de l'IA (score1) et du joueur humain (score2).

currentplayer: Une variable qui indique le joueur en cours de jeu. Initialement, c'est l'IA qui commence.

compteur: Un compteur qui garde une trace du nombre de cases remplies dans la grille. Il est utilisé pour détecter un match nul.

meilleurscore: Une variable temporaire utilisée dans l'algorithme Minimax pour stocker la meilleure valeur trouvée lors de l'exploration des mouvements possibles.

decision: Une variable qui stocke la décision prise par l'IA lors de l'algorithme Minimax. Elle contient les coordonnées (ligne, colonne) du meilleur coup.

kkk: Une variable de débogage utilisée pour compter le nombre d'appels à la fonction min_max_value.

Les fonctions:

AfficherGrille(g): Cette fonction affiche la grille du jeu. Elle parcourt la grille (une matrice 3x3) et imprime des "❌" pour le joueur 1 (jIa), "⭕" pour le joueur 2 (jHumain), et "." pour les cases vides.

Terminal_Test(g): Cette fonction vérifie si le jeu est terminé. Elle recherche des victoires sur les diagonales, les lignes et les colonnes, ainsi que si la grille est complètement remplie (match nul). Elle renvoie un booléen indiquant si le jeu est terminé, le gagnant s'il y en a un, et le nombre de cases restantes.

Utility(g): Cette fonction calcule l'utilité d'une configuration de grille donnée. Elle utilise le résultat de Terminal_Test pour déterminer si le jeu est terminé et attribue des valeurs en conséquence : 1 pour la victoire de l'IA, -1 pour la victoire du joueur humain et 0 pour un match nul.

Actions(g): Cette fonction renvoie les actions possibles à partir d'une configuration de grille donnée. Elle recherche les cases vides dans la grille et renvoie leurs coordonnées.

Result(g, pos): Cette fonction met à jour la grille après qu'un joueur ait joué un coup à une position donnée (pos). Elle vérifie d'abord si le jeu est terminé, puis si le coup est valide, puis effectue le coup et passe au joueur suivant.

min_max_value(grille, prof, maxi): Cette fonction implémente l'algorithme Minimax pour évaluer les positions possibles dans le jeu. Elle utilise une approche récursive pour explorer les arbres de jeu en fonction du joueur courant (maxi pour l'IA, et non maxi pour le joueur humain). Elle renvoie la valeur minimax de la position.

MiniMax_Decision(grille): Cette fonction utilise l'algorithme Minimax pour décider du meilleur coup à jouer par l'IA. Elle explore toutes les actions possibles et choisit celle qui maximise la valeur minimax.

Jeu(g): Cette fonction est le moteur principal du jeu. Elle gère les tours des joueurs (IA et humain), vérifie si le jeu est terminé, affiche le résultat et permet aux joueurs de décider s'ils veulent rejouer.

Le code présente un jeu de Morpion où l'IA (jIa) utilise l'algorithme Minimax pour prendre ses décisions. L'utilisateur (jHumain) peut jouer contre l'IA. Le code est actuellement conçu pour être exécuté en mode console.

Nouvelle production:

Le code fourni en pièce jointe implémente un jeu de morpion en JavaScript où un joueur humain joue contre une intelligence artificielle (IA).
Comme le script précédent le jeu se lance dans la console, pour cela il suffit de faire un initialisation node sur le projet (npm init) et de lancer l'instruction npm start.
Les prompts de la console sont possibles grâce au module readline embarqué par nodejs.

Voici une explication de la logique mise en place:

Le jeu utilise une grille de morpion 3x3, représentée par un tableau board contenant 9 éléments (représentant chaque case). Le tableau board est initialisé avec des espaces vides.
Il y a deux joueurs : 'X' (joueur humain) et 'O' (l'IA). Le joueur qui commence est choisi au hasard avec Math.random(). Si la valeur générée est inférieure à 0.5, 'X' commence, sinon, 'O' commence.
Les combinaisons gagnantes possibles sont définies dans le tableau winningCombos.
Regroupant les indices des cases qui forment des lignes, des colonnes ou des diagonales gagnantes.
Le jeu commence avec l'appel à startNewGame(). Si le premier joueur est l'IA ('O'), l'IA effectue son premier mouvement immédiatement, sinon le joueur humain commence.

Les fonctions:

printBoard(board) :affiche la grille actuelle dans la console, avec des espaces vides pour les cases non jouées et les numéros de cases de 0 à 8 pour les cases disponibles.

checkWin(board, player): vérifie si un joueur a gagné en parcourant les combinaisons gagnantes possibles. Si un joueur a rempli l'une de ces combinaisons, la fonction renvoie true, sinon false.

isBoardFull(board): vérifie si le plateau est complètement rempli. Si toutes les cases sont occupées (sans espaces vides), elle renvoie true, ce qui signifie que le match est nul.

makeMove(board, move, player): permet à un joueur de faire un mouvement en mettant à jour le tableau board avec le symbole du joueur (soit 'X' soit 'O') à l'indice spécifié. Si la case est déjà occupée, elle renvoie false, sinon true.

playAgain(): affiche le score actuel du joueur humain et de l'IA, puis demande si les joueurs veulent rejouer. En fonction de la réponse (0 pour non, 1 pour oui), elle redémarre le jeu ou ferme l'application.

startNewGame(): initialise un nouveau jeu en réinitialisant le tableau board, en choisissant aléatoirement le premier joueur, et en lançant le jeu. Elle utilise également switchPlayer pour basculer entre 'X' et 'O' à chaque tour.

makeAIMove(): permet à l'IA de faire un mouvement. Elle commence par vérifier si elle peut compléter une combinaison gagnante. Si c'est le cas, elle fait le mouvement gagnant. Sinon, elle joue aléatoirement dans une case vide.

playGame(): gère le déroulement du jeu. Elle affiche le tableau, demande au joueur en cours de faire un mouvement, vérifie si le jeu est terminé (gagné ou nul), puis bascule entre les joueurs humains et l'IA.

Conclusion:

Ce code met en place un jeu de morpion où un joueur humain joue contre une IA qui peut rechercher des mouvements gagnants et jouer de manière aléatoire lorsque nécessaire. La logique de basculement entre les joueurs, de vérification de la victoire et de gestion du jeu est mise en œuvre pour créer une expérience de jeu complète.
La notion de coordonnées à également été retirée pour éviter l'erreur qui à été repéré dans le script Python au profit d'un numéro de case.