ISE224: Bonus Assignment

Due: May 9th, 11:59 PM (ET)

Introduction to Othello: (https://youtu.be/zFrlu3E18BA?si=b9ULKWgCpmqme72)

Othello, also known as Reversi, is a two-player strategy board game played on an 8x8 grid. The game pieces, called discs, are colored white on one side and black on the other. The objective of the game is to have the majority of discs showing your color when the game ends.

Players take turns placing discs on the board, with the restriction that a disc must be placed in a position that "outflanks" one or more of the opponent's discs. A disc outflanks another disc if it is placed in a line (horizontally, vertically, or diagonally) with one or more discs of the opponent's color followed by a disc of the current player's color. After placing a disc, any of the opponent's discs that are outflanked are flipped to the current player's color.

The game ends when neither player can make a valid move, and the player with the most discs of their color on the board wins.

Assignment:

Your task is to implement the Othello game in Python. The game should allow two players to play against each other, with the board displayed in the console. The game should enforce the rules of Othello and determine the winner at the end of the game.

Function Descriptions:

1. print_board(board):
    - This function takes the game board as input and prints it in a readable format.
    - The board is represented as a 2D list, where 0 represents an empty cell, 1 represents a white disc, and 2 represents a black disc.
    - The function should print the board with clear cell separators.
2. initialize_board():
    - This function initializes the game board with the starting position.
    - It should create an 8x8 2D list filled with zeros.
    - The four central cells should be initialized with the starting discs: two white discs (1) and two black discs (2) are listed diagnostically on the board.
    - The function should return the initialized board.
3. valid_moves(board, player):
    - This function takes the current board state and the current player (1 for white, 2 for black) as input.
    - It should return a list of valid moves for the current player.
    - A move is valid if it outflanks at least one of the opponent's discs.
    - The function should check all possible moves on the board and return a list of valid move positions as tuples (row, col).
4. make_move(board, move, player):

    - This function takes the current board state, the selected move position, and the current player as input.
    - It should update the board by placing the current player's disc at the selected position and flipping the outflanked discs.
    - Parameters:
        1. board: the game board as a 2D list.
        2. move: a tuple indicating the position to place the disk (row, column).
        3. player: the current player (1 for white and 2 for black).
    - The function should modify the board in-place and return True if the move is valid and successfully made, False otherwise.
5. player_input(board, player):
    - This function handles the player's input for making a move.
    - It should prompt the player to enter the row and column of their desired move.
    - The function should validate the input and ensure it is a valid move by calling the valid_moves function.
    - If the input is valid, it should return the selected move as a tuple (row, col). Otherwise, it should prompt the player to enter a valid move.
6. check_winner(board):
    - This function takes the final board state as input and determines the winner of the game.
    - It should count the number of white and black discs on the board.
    - If there are more white discs, it should return "White wins!". If there are more black discs, it should return "Black wins!". If there is an equal number of discs, it should return "It's a tie!".
7. play_game():
    - This function represents the main game loop.
    - It should initialize the board by calling initialize_board().
    - It should alternate between the two players, allowing them to make moves until the game ends.
    - In each turn, it should display the current board state, prompt the current player for a move using player_input(), and update the board using make_move().
    - The game ends when neither player has any valid moves left.
    - After the game ends, it should display the final board state and the winner by calling check_winner().

Additional Instructions:

- Use appropriate data structures to represent the game board and moves.
- Handle edge cases and invalid inputs gracefully.
- Provide clear instructions to the players on how to enter their moves.
- Test your code thoroughly to ensure it follows the rules of Othello and produces the correct results.
- Player 2 can set the computer as player, which uses the first available valid move to play the game.
- Add comments to your code to explain the functionality of each function and any important logic.
- Some example steps:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 2 0 0 0
0 0 0 2 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0


Player 1's turn (White).
Enter your move as row,col (0-indexed): 5,3
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 2 0 0 0
0 0 0 1 1 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0


Player 2's turn (Black).
AI making a move...
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 2 0 0 0 0
0 0 0 1 2 0 0 0
0 0 0 1 1 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0


Player 1's turn (White).
Enter your move as row,col (0-indexed):
```

Submission:

- Submit a single Python file named **othello.py** containing your implementation of the Othello game.
- Score: 2 bonus points of overall score (you have to answer it fully correctly to get this bonus points).

Good luck, and have fun implementing the Othello game!