Tic-Tac-Toe game

The Python code implemented in tictactoe.py outlines the logic for a Tic-Tac-Toe game, incorporating a Minimax algorithm to power an AI opponent. The project involves creating a Tic-Tac-Toe game with a graphical user interface (GUI) using the Pygame library, where players can challenge an optimally playing AI opponent.

The **initial_state** function initializes the game board as a 3x3 grid of **None** values, representing an empty board at the start of the game. The **player** function determines which player's turn it is based on the count of X and O on the board, with X always starting the game. The **actions** function returns all possible moves available on the board, represented as tuples (i, j) indicating row and column positions. The **result** function calculates the board state resulting from a player's move, ensuring the move is valid and creating a new board state without altering the original. The **winner** function checks for a winning condition by evaluating rows, columns, and diagonals for three matching symbols. The **terminal** function determines if the game has ended, either by a win or a draw. The **utility** function assigns a numerical value to the end state of the game: 1 for a win by X, -1 for a win by O, and 0 for a draw.

The core of the AI, the **minimax** function, decides the optimal move for the AI by evaluating all possible future game states and selecting the move that maximizes the AI's chance of winning while minimizing the opponent's chance.

Upon testing the AI implementation in a series of games, a report was created to document the outcomes, especially focusing on the performance difference when the user plays as 'X' versus when playing as 'O'. Over ten games, the results were as follows:

- When the user played as 'X', all 6 games resulted in a tie. This demonstrates the Minimax algorithm's effectiveness in ensuring that the AI can secure a draw at minimum, given optimal play by both sides.
- Conversely, when the user played as 'O', the outcome of 4 games were 2 ties for the user and 2 loses. This indicates that while playing as 'O' presents more of a challenge due to 'X' having the first move advantage, optimal play can still not lead to favorable outcomes against the AI.

Below is a tabular summary of the game outcomes, clearly showing the distribution of wins, losses, and ties based on the player's initial choice of symbol (X or O):

Player	**Total Games**	**Wins**	**Ties**	**Loses**
X	6	0	6	0
O	4	0	2	2

This table not only highlights the AI's performance but also underscores the importance of strategic play. The documentation and this report together fulfill the project's requirement to analyze and understand adversarial search algorithms through practical application.