**Tic-Tac-Toe AI Game**

This implementation of Tic-Tac-Toe features an AI that uses the minimax algorithm to make optimal choices while playing as "O." Built with Python's pygame library, the game provides a graphical interface, and the AI's decision-making relies on recursive exploration of the game tree through minimax.

### Key Concepts and AI Flow

1. **Overview of Minimax Algorithm**:
   * Minimax is a decision-making technique commonly applied in two-player zero-sum games, where one player's gain equals the other's loss (like Tic-Tac-Toe). The goal is to maximize the player's score while minimizing the opponent's score.
   * In this game, we have two players:
     + **Maximizer (X)**: Aims to maximize their score (prefers to win).
     + **Minimizer (O)**: Aims to minimize the opponent's score (wants the opponent to lose).
2. **AI as the Minimizer (O)**:
   * The AI functions as the Minimizer, striving to decrease the score by putting the player in losing positions.
   * **Terminal States**: The game can end with a win for "X," a win for "O," or a tie. The AI identifies when to stop evaluating deeper into the game tree (i.e., when the game has concluded).
3. **Utility Function**:
   * The utility function assigns numeric values to terminal game states:
     + 1 if "X" (the opponent) wins.
     + -1 if "O" (the AI) wins.
     + 0 if the game ends in a tie.
   * This function aids the minimax algorithm in determining which path in the game tree yields the best outcome for the current player.
4. **Game Flow**:
   * The game starts with the player choosing to be either "X" or "O."
   * The board begins empty, and players take turns.
     + **Player's Turn**: The player makes a move using the graphical interface. After the player's move, the AI (as "O") calculates its best possible move using the minimax algorithm.
     + **AI's Move**: The AI invokes the minimax function, which employs the Max\_Value and Min\_Value functions to recursively assess potential game states and selects the move that provides the best outcome for itself.
5. **Decision-Making Process**:
   * For each board state, the minimax algorithm assesses all possible moves and returns the one that maximizes the player's chances of winning (or minimizes the opponent’s score).
   * The algorithm simulates every potential series of moves, with the AI focusing on reducing the player's chances of winning by making the best possible response.

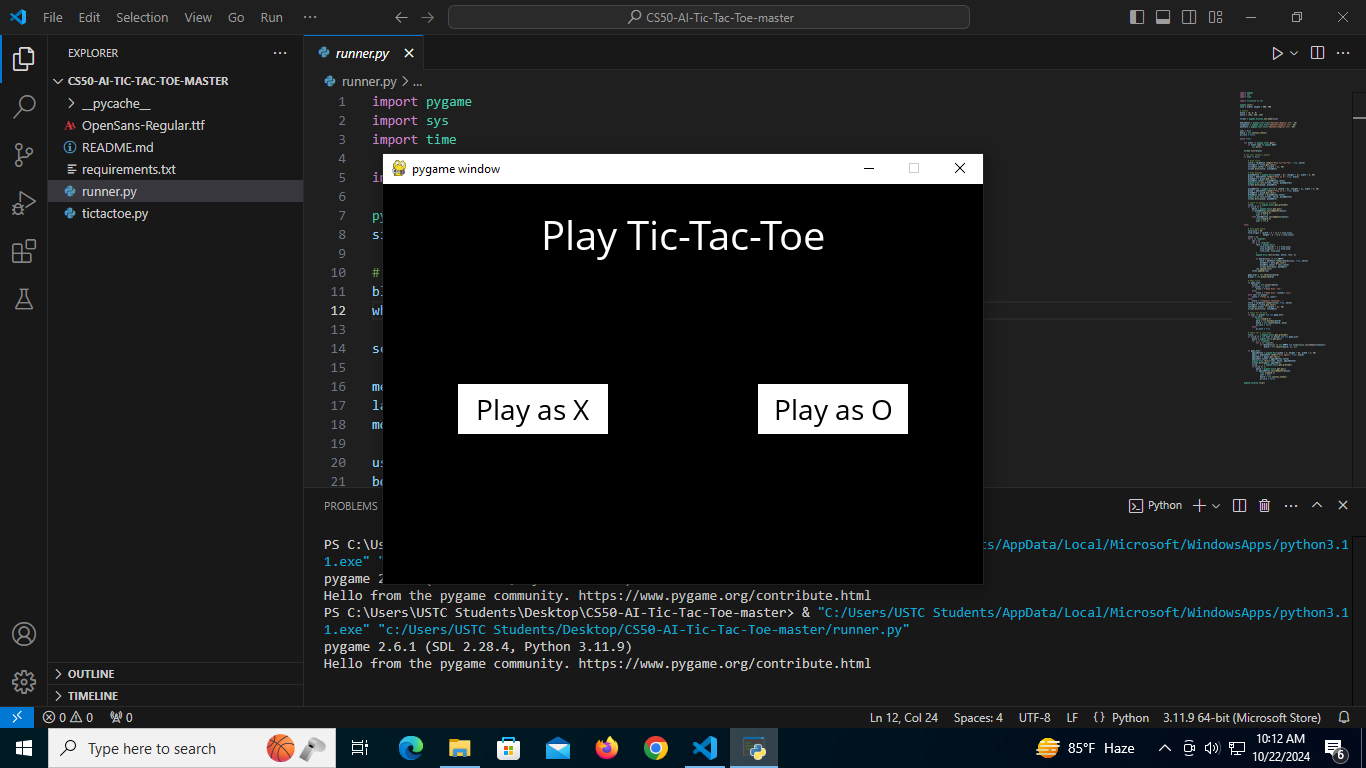
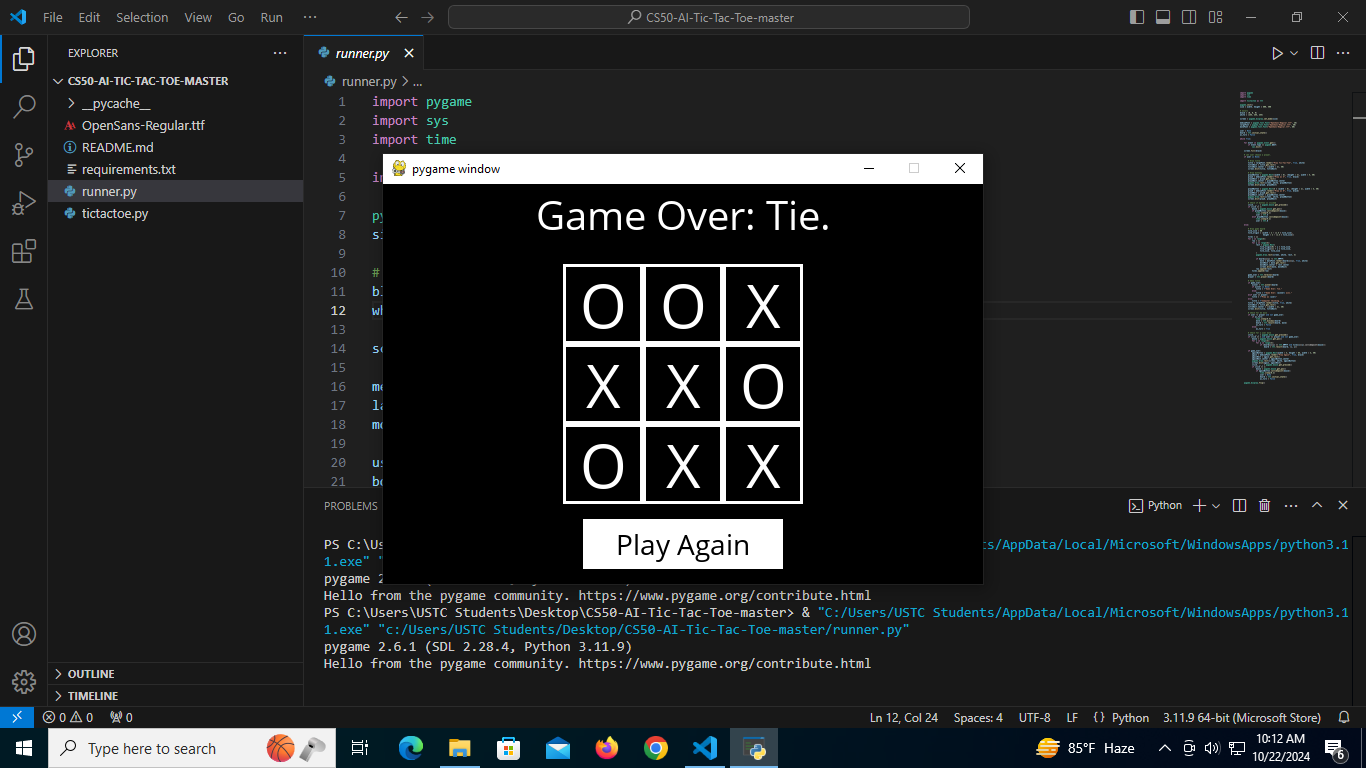
**How the AI Operates**

* The minimax algorithm constructs a tree representing all possible board configurations from the current state.
* At each node in this tree, it evaluates the board from the perspective of the current player:
  + If it's the AI's turn (Minimizer), it selects the move that minimizes the potential score (optimizing for the worst outcome for the player).
  + If it's the player's turn (Maximizer), the AI evaluates the potential score to maximize its own score (optimizing for the best outcome).

**Key Observations:**

* The AI will always choose the best move if it has enough time to explore the entire game tree.
* If the AI faces a perfect player, the game will always end in a tie.
* The algorithm's time complexity is O(bd)O(b^d)O(bd), where bbb is the branching factor (the number of possible moves per state), and ddd is the depth of the game tree (the maximum number of turns).

**Output:**

** **

**Summary**: This implementation's AI utilizes the minimax algorithm to assess the game board following each move. By recursively analyzing all potential moves and their outcomes, the AI determines the best decision to secure a win or force a tie, while reducing the human player's chances of winning. The algorithm alternates between the maximizer (the player) and the minimizer (the AI), simulating the game's progression and ensuring optimal play by the AI.