Quaid I Azam University Islamabad



**Assignment # 02 (Artificial Intelligence “**CS-414**”)**

**Instructor:**

**Dr. Saqib**

**Assignment submitted by**

* Shayan Danish 04072212006

**Report: Tic-Tac-Toe Game with Minimax Algorithm**

**Overview:**

This Python script implements a console-based Tic-Tac-Toe game where a human player competes against the computer. The computer uses the Minimax algorithm with Alpha-Beta pruning to determine the optimal move.

**Main Components**

1. **Board Initialization and Display**
   * **create\_board()**: Initializes a 3x3 game board represented as a list of lists, with each cell containing a space character (' ').
   * **print\_board(board)**: Displays the current state of the board in a readable format, with rows separated by lines.
2. **Game Mechanics**
   * **is\_winner(board, player)**: Checks if the specified player ('X' or 'O') has won the game. It evaluates the board against predefined winning conditions: rows, columns, and diagonals.
   * **is\_draw(board)**: Determines if the game has ended in a draw by checking if all cells are occupied.
   * **get\_available\_moves(board)**: Returns a list of available moves (empty cells) on the board.
3. **Minimax Algorithm with Alpha-Beta Pruning**
   * **minimax(board, depth, is\_maximizing, alpha, beta)**: Implements the Minimax algorithm to evaluate possible moves. The function uses Alpha-Beta pruning to optimize the search by eliminating branches that will not influence the final decision:
     + **Base Cases**: If a player has won or the game is a draw.
     + **Maximizing Player (Computer's turn)**: The computer (playing 'O') seeks to maximize its score.
     + **Minimizing Player (Human's turn)**: The human (playing 'X') seeks to minimize the computer's score.
   * **best\_move(board)**: Determines the optimal move for the computer by iterating through all possible moves and selecting the one with the highest Minimax value.
4. **Game Loop**
   * **play\_game()**: Manages the flow of the game:
     + Initializes the board and displays it.
     + Alternates turns between the human and the computer.
     + Validates human input to ensure valid moves.
     + Displays the board after each move.
     + Checks for a winner or a draw after each turn.

**Key Features**

* **Input Validation**: Ensures that human players enter valid moves within the board's range and that the selected cell is unoccupied.
* **Artificial Intelligence**: The computer employs a robust decision-making process via the Minimax algorithm, ensuring it plays optimally.
* **User Interaction**: The game is played interactively via the console, with prompts for user input and real-time feedback after each move.

**How It Works**

1. The game starts with an empty board.
2. The human player makes the first move by selecting a cell.
3. The computer calculates the best possible move using the Minimax algorithm and updates the board accordingly.
4. The game alternates between the human and computer until one player wins or the game ends in a draw.

**Conclusion**

This script demonstrates a classic implementation of the Tic-Tac-Toe game with a strong AI opponent. The use of the Minimax algorithm with Alpha-Beta pruning ensures the computer plays optimally, making it a challenging opponent for the human player. The code is well-structured, with clear separation between game logic and AI decision-making processes.

Output:





