Tic-Tac-Toe Game Documentation

Overview:

This program implements a console-based Tic-Tac-Toe game in Python. Players can choose to play in single-player mode (against the computer) or multiplayer mode (two players). The computer uses the Minimax algorithm with Alpha-Beta pruning to make optimal decisions.

Modules and Dependencies

The program uses the following module:

- random: Used to generate random moves for the computer in "easy" difficulty mode.

Functions

1. ConstBoard(board)
    - **Description**: Draws the current state of the Tic-Tac-Toe board.
    - **Parameters**:
        - board (list): A list of size 9 representing the board. Values are:
            - 0: Empty spot.
            - 1: O (computer's move).
            - -1: X (player's move).
    - **Output**: Prints the current board to the console.
2. UserTurn(board, player)
    - **Description**: Handles the user's move, validating input and updating the board.
    - **Parameters**:
        - board (list): The current board state.
        - player (int): Player identifier (-1 for X, 1 for O).
    - **Input**: Prompts the user for a position between 1 and 9.
    - **Output**: Updates the board with the user's move.
3. analyzeboard(board)
    - **Description**: Analyzes the board to determine if there is a winner or a draw.
    - **Parameters**:
        - board (list): The current board state.
    - **Returns**:

        - 1: If O (computer) wins.
        - -1: If X (user) wins.
        - 0: If the game is a draw.
        - None: If the game is ongoing.

4. minimax(board, depth, is_maximizing, alpha, beta, player)

   Description: Implements the Minimax algorithm with Alpha-Beta pruning to determine the best move.
   Parameters:

   board (list): The current board state.
   depth (int): The current depth of the recursion (used for limiting search depth).
   is_maximizing (bool): True if the current player is maximizing (computer).
   alpha (float): The best score for the maximizing player.
   beta (float): The best score for the minimizing player.
   player (int): 1 for computer (O), -1 for user (X).

   Returns:

   The score of the board state.

5. CompTurn(board, difficulty)

- **Description**: Determines and executes the computer's move based on the selected difficulty level.
- **Parameters**:
    - board (list): The current board state.
    - difficulty (str): Difficulty level ("easy", "medium", "hard").
- **Output**: Updates the board with the computer's move.

6. main()
    - **Description**: The main function to run the game loop. Handles game initialization, mode selection, and turn-taking.
    - **Flow**:
        1. Prompts the user to select single-player or multiplayer mode.
        2. Initializes the board as a list of 9 zeros.
        3. For single-player mode:
            - Allows the user to choose difficulty and who plays first.

        - Alternates turns between the user and the computer until the game ends.

    4. For multiplayer mode:
        - Alternates turns between two players until the game ends.
    5. Analyzes the board for a winner or draw after each move.
    6. Displays the final result.
- **Output**: Prints the winner or indicates a draw.

How to Run the Game

1. Execute the script in a Python environment.
2. Follow the prompts to:
    - Select game mode (single-player or multiplayer).
    - If single-player, choose difficulty (easy, medium, or hard).
    - If single-player, decide who plays first.
3. Input moves as prompted (1-9 positions corresponding to the board).

Board Representation

The board is represented as a list of size 9, with positions mapped as follows:

```
1 | 2 | 3
---------
4 | 5 | 6
---------
7 | 8 | 9
```

Values in the list:

- 0: Empty spot.
- 1: O (computer's move).
- -1: X (player's move).

Game Modes

1. **Single-Player**:

   - Play against the computer.
   - Choose difficulty:
     - **Easy**: Random moves by the computer.
     - **Medium**: Minimax with a depth limit.
     - **Hard**: Full Minimax with Alpha-Beta pruning.
2. **Multiplayer**:
   - Two players alternate moves.

Output Examples

1. **Board Display**:

```
Current State of Board:
X | O | -
---------
- | X | -
---------
- | - | O
```

2. **Result**:
   - "Computer (O) wins!"
   - "You (X) win!"
   - "It's a draw!"