TITLE: CodTech IT Solutions Internship - Task Documentation: “TIC-TAC-TOE” AI project

INTERN INFORMATION:

Name: SAI TEJA BOMMA

ID: ICOD7167

INTRODUCTION

The classic game of Tic-Tac-Toe, also known as Noughts and Crosses, is a simple yet engaging two-player game where players take turns marking cells in a 3x3 grid with their respective symbols, typically 'X' and 'O', with the objective of getting three of their symbols in a row, column, or diagonal. While Tic-Tac-Toe has a relatively small game space compared to more complex games, it serves as an excellent platform for exploring game theory and basic search algorithms.

In this project, we aim to implement an AI agent that plays Tic-Tac-Toe against a human player. The AI agent will employ algorithms like Minimax with or without Alpha-Beta Pruning to ensure optimal gameplay and make the AI player unbeatable. By developing this AI agent, we will gain insights into game theory concepts such as optimal decision-making and exploring the game tree, as well as practical experience with basic search algorithms for adversarial games.

Through this project, we will delve into the fundamental principles of game theory and explore how these concepts can be applied to design intelligent agents capable of playing games strategically. Additionally, we will gain a deeper understanding of algorithms like Minimax and Alpha-Beta Pruning, which are foundational techniques in the field of artificial intelligence and have broad applications beyond just Tic-Tac-Toe. Overall, this project will provide valuable insights into game theory and basic search algorithms while offering an opportunity to develop a sophisticated AI player for Tic-Tac-Toe.

IMPLEMENTATION

- **Initialize the Board:** Start by creating a 3x3 grid to represent the Tic-Tac-Toe board. You can use a nested list or a two-dimensional array to store the state of the board. Initially, all cells are empty.
- **Print the Board:** Write a function to print the current state of the board to the console. This function will display the grid with the current markings (symbols) of the players.
- **Check for Win or Draw:** Implement a function to check if there is a winner or if the game has ended in a draw. This function will examine the rows, columns, and diagonals of the board to determine if any player has achieved three in a row, column, or diagonal, or if all cells are filled without a winner.
- **Generate Possible Moves:** Write a function to generate all possible moves (next board states) for a given player. This function will iterate through the empty cells on the board and create new board configurations by placing the player's symbol in each empty cell.
- **Minimax Algorithm:** Implement the Minimax algorithm, a recursive search algorithm used in decision-making for two-player zero-sum games like Tic-Tac-Toe. The Minimax algorithm explores the game tree, alternating between maximizing and minimizing players, to determine the best move for the AI player.
- **Alpha-Beta Pruning (Optional):** Enhance the Minimax algorithm with Alpha-Beta Pruning, a technique to reduce the number of nodes evaluated in the game tree. Alpha-Beta Pruning eliminates branches of the game tree that are guaranteed to be worse than previously examined branches, making the search more efficient.
- **AI Move:** Create a function for the AI player to make a move on the board. This function will use the Minimax algorithm (optionally with Alpha-Beta Pruning) to determine the optimal move for the AI player.
- **Player Input:** Implement user input functionality to allow the human player to make moves on the board. Prompt the user to input the row and column where they want to place their symbol ('X').
- **Game Loop:** Create a loop to alternate between the AI player and the human player making moves until the game is over. Check for win/draw conditions after each move.
- **Game Over:** Display the final state of the board and announce the winner or declare a draw.

CODE EXPLAINATION

1. **Print Board Function:**

This function prints the current state of the Tic-Tac-Toe board to the console. It iterates through each row of the board, joins the elements with "|" to represent the columns, and separates each row with horizontal lines.

```
def print_board(board):
    for row in board:
        print("|".join(row))
    print("-" * 5)
```

2. Check Winner Function:

This function checks if a player (either 'X' or 'O') has won the game by examining all rows, columns, and diagonals on the board**.**

```
def check_winner(board, player):
    # Check rows, columns, and diagonals
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or \
           all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or \
       all(board[i][2-i] == player for i in range(3)):
        return True
    return False
```

3.Is Full Function:

This function checks if the board is full, meaning all cells are occupied by either 'X' or 'O', indicating that the game has ended in a tie.

```
def is_full(board):
    return all(all(cell != " " for cell in row) for row in board)
```

4.Evaluate Function:

This function evaluates the current state of the board and assigns a score: 1 if 'X' wins, -1 if 'O' wins, and 0 for a tie or ongoing game.

```
def evaluate(board):
    if check_winner(board, 'X'):
        return 1
    elif check_winner(board, 'O'):
        return -1
    else:
        return 0
```

5. Minimax Algorithm:

This function implements the Minimax algorithm to determine the best move for the AI player. It recursively explores the game tree, alternating between maximizing and minimizing players, to find the optimal move.

```
def minimax(board, depth, maximizing_player):
    if check_winner(board, 'X'):
        return -1
    elif check_winner(board, 'O'):
        return 1
    elif is_full(board):
        return 0

    if maximizing_player:
        max_eval = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = 'O'
                    eval = minimax(board, depth + 1, False)
                    board[i][j] = " "
                    max_eval = max(max_eval, eval)
        return max_eval
```

6. Best Move Function:

This function finds the best move for the AI player by evaluating all possible moves using the Minimax algorithm and selecting the move with the highest score.

```
def best_move(board):
    best_eval = -math.inf
    best_move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
```

```
            board[i][j] = 'O'
            eval = minimax(board, 0, False)
            board[i][j] = " "
            if eval > best_eval:
                best_eval = eval
                best_move = (i, j)
    return best_move
```

7. Play Function:

This function orchestrates the gameplay loop, allowing the human player to make moves and then making moves for the AI player until the game is over. It also prints the board after each move and announces the winner or a tie at the end.

```
def play():
    board = [[" " for _ in range(3)] for _ in range(3)]
    print_board(board)
    while True:
        x, y = map(int, input("Enter your move (row col): ").split())
        if board[x][y] != " ":
            print("Invalid move. Try again.")
            continue
        board[x][y] = 'X'
        print_board(board)
        if check_winner(board, 'X'):
            print("You win!")
            break
        if is_full(board):
            print("It's a tie!")
            break
        print("Computer's move:")
        move = best_move(board)
        board[move[0]][move[1]] = 'O'
        print_board(board)
        if check_winner(board, 'O'):
```

```
            print("Computer wins!")
            break
        if is_full(board):
            print("It's a tie!")
            break

play()
```

USAGE

The Tic-Tac-Toe project offers users a platform to engage in the classic game against an AI opponent utilizing the Minimax algorithm. Here's a concise summary of its usage:

- **Gameplay Initiation**:
- Users can initiate the game by running the provided code.
- The AI ('O') and the user ('X') take turns making moves on a 3x3 grid.
- **Interacting with the Game:**
- Users input their moves by specifying the row and column for their symbol placement.
- The game displays the updated board after each move, facilitating easy tracking of game progress.
- **Game Outcome Determination:**
- The game continues until a player wins, the board is filled (resulting in a tie), or the user chooses to exit.
- Announcements are made for wins, ties, or the user's exit from the game.
- **AI Logic Understanding:**
- Users gain insight into AI decision-making processes through observing the Minimax algorithm in action.
- The AI strives to make strategic moves to secure victory or force a draw.

CONCLUSION

The Tic-Tac-Toe project presents a user-friendly platform for playing the classic game against an AI opponent powered by the Minimax algorithm. Through intuitive gameplay and interactive decision-making, users can explore fundamental concepts of game theory and AI logic. With the ability to customize and extend the project, users can experiment with different strategies, furthering their understanding of AI development. Serving as an educational resource, this project caters to learners of all levels, providing practical insights into search algorithms while offering an enjoyable gaming experience.

OUTPUT

```
| |
-----
| |
-----
| |
-----
Enter your move (row col):  2 2
| |
-----
| |
-----
| |X
-----
Computer's move:
| |
-----
|O|
-----
| |X
-----
Enter your move (row col):  2 1
| |
-----
|O|
-----
|X|X
-----
Computer's move:
| |
-----
|O|
-----
O|X|X
-----
Enter your move (row col):  0 2
| |X
-----
|O|
-----
O|X|X
-----
Computer's move:
| |X
-----
|O|O
-----
O|X|X
-----
Enter your move (row col):  1 1
Invalid move. Try again.
Enter your move (row col):  1 1
Invalid move. Try again.
Enter your move (row col):  0 1
 |X|X
-----
 |O|O
```

```
-----
O|X|X
-----
Computer's move:
 |X|X
-----
O|O|O
-----
O|X|X
-----
Computer wins!
```