Tic Tac Toe

A Tic Tac Toe expert system with minimax algorithm.

WEI ZHAO

Spring 2023 CS5001

Tic-Tac-Toe Introduction and Basic Rules

- Tic-Tac-Toe, also known as "Noughts and Crosses" or "Xs and Os," is a classic two-player paper-and-pencil game. The objective of the game is for one player to achieve three of their symbols (either Xs or Os) in a row, either horizontally, vertically, or diagonally, on a 3x3 grid. Players take turns placing their symbols in empty cells on the grid, with one player using Xs and the other using Os.

- I wanted to build an interactive version of this game, allowing a human player to compete against an AI opponent. This project explore how the minimax algorithm and recursion can be effectively utilized to develop an AI with strategic decision-making abilities.

Algorithm Deep Dive: get_ai_move()

The main purpose of the get_ai_move() function is to find the best board move for the AI and utilizes the minimax algorithm by exploring all possible moves to select the optimal one for the AI.

```
def get_ai_move():
    # Uses the minimax algorithm to determine the best mov
    best_score = float('-inf')
    best_move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'O'
                score = minimax(board, 0, False)
                board[i][j] = ' '
                if score > best_score:
                    best_score = score
                    best_move = (i, j)
    return best_move
```

Algorithm Deep Dive: Minimax()

```python
def minimax(board, depth, is_maximizing):
    # Uses the minimax algorithm to determine the best move for the AI
    if check_win('O'):
        return 1
    elif check_win('X'):
        return -1
    elif ' ' not in [cell for row in board for cell in row]:
        return 0

    if is_maximizing:
        best_score = float('-inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    score = minimax(board, depth+1, False)
                    board[i][j] = ' '
                    best_score = max(score, best_score)
        return best_score

    else:
        best_score = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    score = minimax(board, depth+1, True)
                    board[i][j] = ' '
                    best_score = min(score, best_score)
        return best_score
```

Use recursion to find the optimal solution

1. the function checks whether there is already a winner, and if there is, it returns the corresponding score based on the winner.
2. if the board is full and there is no winner, the function returns a score of 0 to indicate a draw.
3. It determines whether the current layer is a maximizing or minimizing layer based on the is maximizing parameter. In the maximizing layer, the function seeks to find the highest score for the AI; in the minimizing layer, it aims to find the lowest score for the player. To achieve this, the function iterates through each empty cell in the board, recursively calls the minimax function, and updates the best score.

Role of Recursion in AI Algorithm

Recursion is an essential concept in the two functions, get_ai_move() and minimax().

- The function starts by placing the AI‘s symbol (’O‘) in an empty cell on the game board, then calls the minimax() function to calculate the score of this move.
- The minimax() function itself utilizes recursion to explore all possible moves and their outcomes for both the AI and the human player. At each level of the game tree, the function is called recursively with alternating values for the is maximizing parameter, allowing it to evaluate the best moves for both players. If is_maximizing is set to True, the function seeks the maximum score for the AI, while if it’s set to False, the function seeks the minimum score for the human player.
- The recursion continues until a terminal state is reached, either when a player wins or the game ends in a draw. This recursive approach allows the minimax algorithm to make well-informed decisions, leading to a challenging and engaging gameplay experience.

Conclusion and Future Developments

- Throughout this project, we learned how to use the minimax algorithm and recursion to create an AI player.

- One limitation of our current implementation is the potential for slow performance when dealing with large game trees.

Citations:

- Python documentation (https://docs.python.org/3/)
- Minimax algorithm tutorial (https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/)

THANK YOU FOR WATCHING