**TIC-TAC-TOE**

Aarthi Ganesan ([ganesanaarthi@cityuniversity.edu](mailto:ganesanaarthi@cityuniversity.edu))

**Introduction**

An Implementation of a Python console-based Tic-Tac-Toe game with an AI opponent utilizing the minimax algorithm. The game allows two players, ‘X’ and ‘O’, to take turns making moves on a 3x3 grid, where ‘X’ is the player and ‘O’ is the computer (AI). The code includes functionalities such as displaying the game board, making moves, saving moves to a file, checking for a winner, determining if the board is full, and implementing an AI opponent using the minimax algorithm.

**Minimax Algorithm**

The minimax algorithm is a decision-making algorithm commonly used in two-player, turn-based games with perfect information. The goal of the algorithm is to determine the best move for a player by considering all possible moves and their outcomes.

The term “minimax” comes from the algorithm’s strategy of minimizing the possible loss for a worst-case scenario (minimizing) while maximizing the potential gain (maximizing) for the current player.

The below Python code defines a ‘minimax’ function used in this Tic-Tac-Toe game:

|  |
| --- |
| def minimax(self, board, depth, is\_maximizing\_player):          scores = {'X': -1, 'O': 1, 'tie': 0}          if self.is\_winner('X'):              return scores['X']          if self.is\_winner('O'):              return scores['O']          if self.is\_board\_full():              return scores['tie']          if is\_maximizing\_player:              max\_score = float('-inf')              for move in self.get\_possible\_moves():                  board[move] = 'O'                  score = self.minimax(board, depth + 1, False)                  board[move] = ' '                  max\_score = max(score, max\_score)              return max\_score          else:              min\_score = float('inf')              for move in self.get\_possible\_moves():                  board[move] = 'X'                  score = self.minimax(board, depth + 1, True)                  board[move] = ' '                  min\_score = min(score, min\_score)              return min\_score |

Here’s a breakdown:

The function takes three parameters: ‘board” (the current state of the Tic-Tac-Toe board), ‘depth’ (the depth of the recursion, representing the number of moves ahead to look), and ‘is\_maximizing\_player’ (a Boolean indicating whether the current player is maximizing or minimizing.

The ‘scores’ dictionary represents the scores for different outcomes. ‘X’ gets a score of -1, ‘O’ gets a score of 1, and a tie gets a score of 0.

The function first checks if ‘X’ or ‘O’ has won or if the board is full. If any of these conditions are met, it returns the corresponding score.

If the current player is maximizing, the function iterates over all possible moves, simulates each move, and recursively calls itself with the updated board and depth. It then updates the maximum score found so far.

If the current player is minimizing, the function does the same but minimizes the score instead.

The final score returned by the function represents the best score the maximizing player (‘O’) can achieve at the current state of the game.

**Code Overview**

Class Structure:

The code defines a ‘TicTacToe’ class with the following key methods:

1. **\_\_init\_\_(self)**: Initializes the game by creating an empty game board, setting the current player to ‘X’, and initializing an empty list to store moves.
2. **display\_board(self):** Display the current state of the Tic-Tac-Toe board.
3. **make\_move(self, move):** Allows a player to make a move on the board, checks for validity, and updates the board accordingly.
4. **save\_move\_to\_file(self):** Saves the sequence of moves to a text file named ‘tictactoe.txt’
5. **is\_winner(self, player):** Checks if the specified player has won the game
6. **is\_board\_full(self):** Checks if the Tic-Tac-Toe board is full, indicating a tie.
7. **get\_possible\_moves(self):** Returns a list of possible moves on the current board.
8. **get\_best\_move(self):** Implements the minimax algorithm to find the best move for the AI opponent.
9. **minimax(self, board, dept, is\_maximizing\_player):** Recursive function for the minimax algorithm
10. **play\_game(self):** Manages the game loop, allowing players to make moves and determining the winner or a tie.

Main Execution:

The code concludes with the instantiation of a ‘TicTacToe’ object and the execution of the game using the ‘play\_game’ method.

**Conclusion**

The provided Python code successfully implements a console-based Tic-Tac-Toe game with a basic AI opponent. The code is well-organized, making use of object-oriented programming principles, and includes error handling for invalid inputs. The minimax algorithm is employed to provide a challenging opponent for the player. The code also includes a function to save the game moves to a file.

Overall, the code provides an interactive and enjoyable implementation of the classic Tic-Tac-Toe game.

**References**

Anand, Aprataksh. “Tic-Tac-Toe Using Python.” *AskPython*, 16 Feb. 2023, www.askpython.com/python/examples/tic-tac-toe-using-python.

Aradhya, Akshay L. “Finding Optimal Move in Tic-Tac-Toe Using Minimax Algorithm in Game Theory.” *GeeksforGeeks*, GeeksforGeeks, 20 Feb. 2023, www.geeksforgeeks.org/finding-optimal-move-in-tic-tac-toe-using-minimax-algorithm-in-game-theory/.

Surma, Greg. “Tic Tac Toe - Creating Unbeatable AI.” *Medium*, Medium, 24 June 2021, gsurma.medium.com/tic-tac-toe-creating-unbeatable-ai-with-minimax-algorithm-8af9e52c1e7d.

Ying, Kylie. “Coding an Unbeatable Tic Tac Toe Ai (Game Theory Minimax Algorithm Explained).” *YouTube*, YouTube, 20 Jan. 2020, www.youtube.com/watch?v=fT3YWCKvuQE.