

**Department of Computer Science**

**Engineering (AI)**

**ARTIFICAL INTELLIGENCE**

**Project Report**

**On**

**EMPLOYEE SALARY ANALYSIS**

**By**

ANURAG

**Date**: 09/01/2025



**Report: Noughts and Crosses (Tic-Tac-Toe) with Alpha-Beta Pruning**

**Introduction:**

This project implements the classic game of **Noughts and Crosses** (also known as **Tic-Tac-Toe**) in Python, where a human player (Player O) plays against an AI (Player X). The AI uses the **Minimax algorithm with Alpha-Beta Pruning** to make optimal moves, ensuring that it plays perfectly against the human player. This report will describe the key components of the code, the logic behind the implementation, and the results of running the game.

**Game Overview:**

* **Players:**
  + Player X (AI): The AI, which makes optimal moves using the Minimax algorithm with Alpha-Beta pruning.
  + Player O (Human): The human player inputs their moves manually.
* **Objective:**
  + The goal is to get three of one's marks (either 'X' or 'O') in a row, column, or diagonal.
  + The game ends when one player wins, or if the board is filled without a winner (resulting in a draw).

**Key Components:**

**1. TicTacToe Class:**

This class handles the core logic of the game, including:

* **Board Representation:** The board is represented as a 3x3 grid using a list of lists in Python. Each cell can be either X, O, or empty (' ').
* **Win Check:** The check\_win method checks whether a player has won the game. It evaluates all rows, columns, and diagonals for three consecutive marks of the same player.
* **Move Management:** The make\_move and undo\_move methods allow players to place or remove marks on the board.
* **Game State Management:** The is\_full method checks if the board is full, indicating a draw condition. The game\_over method returns True if the game has ended.

**2. Alpha-Beta Pruning with Minimax Algorithm:**

The **Minimax algorithm** is a decision-making algorithm used in game theory and artificial intelligence. It recursively evaluates all possible moves, assuming both players play optimally.

* **Maximizing Player:** Player X (AI) is the maximizing player, aiming to maximize the score.
* **Minimizing Player:** Player O (human) is the minimizing player, aiming to minimize the score.
* **Alpha-Beta Pruning:** Alpha-Beta pruning is an optimization technique that eliminates the need to evaluate certain moves in the game tree, based on the assumption that they won't affect the final decision. This significantly speeds up the decision-making process.

The minimax function evaluates the game tree recursively:

* **Maximizing Player's Turn (AI):** The AI selects the move that maximizes its chances of winning.
* **Minimizing Player's Turn (Human):** The human selects the move that minimizes the AI's chances of winning.

**3. Best Move Selection:**

The best\_move function iterates through all available moves, evaluates them using the minimax function, and selects the move that provides the highest evaluation score for the AI.

**4. Game Loop:**

The play\_game function manages the game's flow:

* **Player X's Turn (AI):** The AI selects the best move using the best\_move function and makes the move on the board.
* **Player O's Turn (Human):** The human player is prompted to enter their move in the form of row and column indices.
* The game alternates between players until a winner is found or the game ends in a draw.

**Evaluation Function:**

The **evaluation function** is central to the Minimax algorithm and is used to assign a value to the board based on the current state:

* **Score of +1**: Player X (AI) wins.
* **Score of -1**: Player O (Human) wins.
* **Score of 0**: The game is a draw or ongoing.

**Game Flow:**

1. The game begins with an empty 3x3 board.
2. Player X (AI) makes the first move, using the Minimax algorithm with Alpha-Beta Pruning to select the best move.
3. Player O (Human) enters a move by specifying a row and column.
4. The game continues alternately between Player X and Player O until:
   * One player wins (three consecutive marks in a row, column, or diagonal).
   * The board is full, resulting in a draw.
5. The game ends with a message indicating the winner or a draw.

**Sample Gameplay:**

| |

-----

| |

-----

| |

Player X plays: (1, 1)

Player O's move:

Enter row and column (0-2) separated by space: 0 0

| O |

-----

| X |

-----

| |

Player X plays: (2, 2)

...

Player X wins!

**Code Optimization:**

* **Alpha-Beta Pruning:** The inclusion of Alpha-Beta pruning drastically reduces the number of nodes evaluated in the game tree. This makes the AI decision process faster, especially in more complex games with deeper game trees.
* **Evaluation Function:** A simple evaluation function is used to quickly determine the outcome of the game. This function is sufficient for the Tic-Tac-Toe game, as the game tree is small.

**Conclusion:**

This project demonstrates the application of the **Minimax algorithm** with **Alpha-Beta Pruning** in a simple 3x3 **Tic-Tac-Toe** game. The AI plays optimally by calculating the best possible moves using these techniques, ensuring it always plays to win or force a draw. The code provides a clean and simple interface for a human player to play against the AI, with easy-to-understand logic and clear output of game results.

**Possible Improvements:**

* **User Interface:** Implementing a graphical user interface (GUI) could improve the user experience.
* **Difficulty Levels:** Adding multiple levels of difficulty for the AI by limiting the depth of the Minimax search.
* **Enhanced Evaluation Function:** More complex evaluation functions could be used in future implementations to handle more advanced board states and strategies.

**End of Report.**