**Tic Tac Toe - Final Report**

## **Project Structure**

### Code Components

1. Classes:
   * Player: Represents a human player with a label and color.
   * ComputerPlayer: Inherits from Player and includes methods for processing moves using AI algorithms.
   * Move: A NamedTuple to represent a move with row, column, and label attributes.
   * TicTacToeGame: Manages the game logic, including move validation, game state checks, and switching players.
   * TicTacToeBoard: A tk.Tk class that manages the GUI elements of the game, including the board display and handling player interactions.
2. Functions:
   * main(): The entry point of the game. It initializes the game difficulty, players, and starts the main game loop.

### **Methodology:**

#### Game Initialization

* Player Input: The game begins by asking the user to input a difficulty level (1 or 2).
* Player Setup: Two players are initialized: one human (Player) and one computer (ComputerPlayer).

#### Game Logic

* Move Validation: The is\_valid\_move method in TicTacToeGame checks if a move is allowed based on the current state of the board.
* Game State Checks: The game checks for winners with has\_winner and for ties with is\_tied after each move.
* Player Toggle: The toggle\_player method switches the current player between human and computer.

#### AI Algorithms

* Depth-First Search (DFS): Used for difficulty level 1. The computer evaluates possible moves recursively to find the best move.
* Minimax Algorithm with Alpha-Beta Pruning: Used for difficulty level 2. This algorithm evaluates the game tree to a given depth, using alpha-beta pruning to optimize performance and reduce computation time.

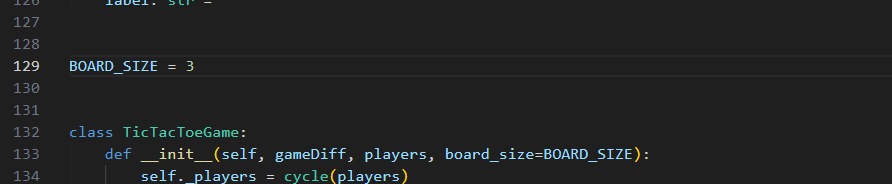
### GUI Interaction

* Tkinter GUI: The TicTacToeBoard class creates the GUI, including the grid for the board and display labels.
* Event Handling: The play method handles both human and computer moves. It updates the board display and checks the game state after each move.

## **Result / Outcome / Observations**

### Functionality

* The game allows for a human player to compete against a computer player.
* Players take turns to place their markers ('X' for human, 'O' for computer) on a 3x3 grid (flexible).
* The game checks for a winner or a tie after each move.
* The computer player's move is determined by either DFS or the Minimax algorithm, based on the selected difficulty level.
* You can expand the grid by changing the BOARD\_SIZE



### Performance

* DFS Algorithm: Suitable for easier difficulty, providing longer move decisions and potentially less optimal moves.
* Minimax Algorithm: Provides more strategic and optimal moves, taking a shorter computation time.

### Observations

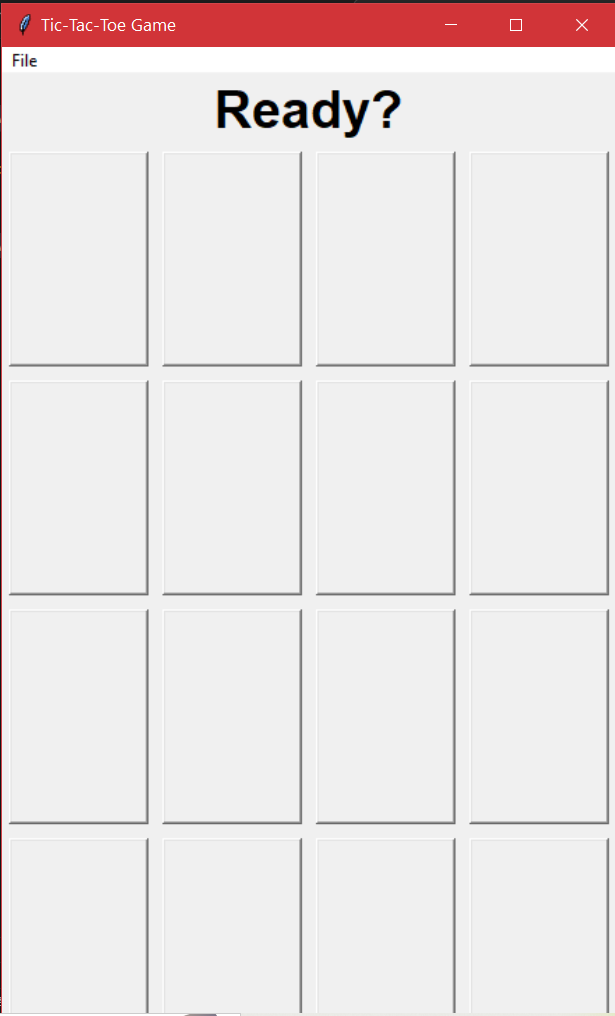
* The game performs well in terms of responsiveness and accuracy in determining the winner or a tie.
* DFS vs Minimax: The DFS algorithm is slower and may result in less optimal moves. The Minimax algorithm, especially with alpha-beta pruning, takes shorter to compute and generally results in a more challenging opponent.
* Both algorithms ensure that the computer plays intelligently, but the difference in their performance is noticeable in terms of the time taken to make a move.
* The use of Tkinter provides a simple yet effective interface for playing the game, with clear indications of game states and player turns.

## Conclusion

This project successfully implements a functional and engaging Tic-Tac-Toe game using Python and Tkinter. The integration of AI algorithms for the computer player enhances the gameplay experience by offering varying levels of difficulty. The clear structure and modular design of the code facilitate easy understanding and potential future enhancements, such as adding more difficulty levels or improving the GUI.

For future implementation, we can enhance both algorithms to make sure the difficulty is different in terms of challenge and not just the time.

**NOTE:** Grid examples down below.

**4 x 4 Grid:**  


**6 x 4 Grid**

