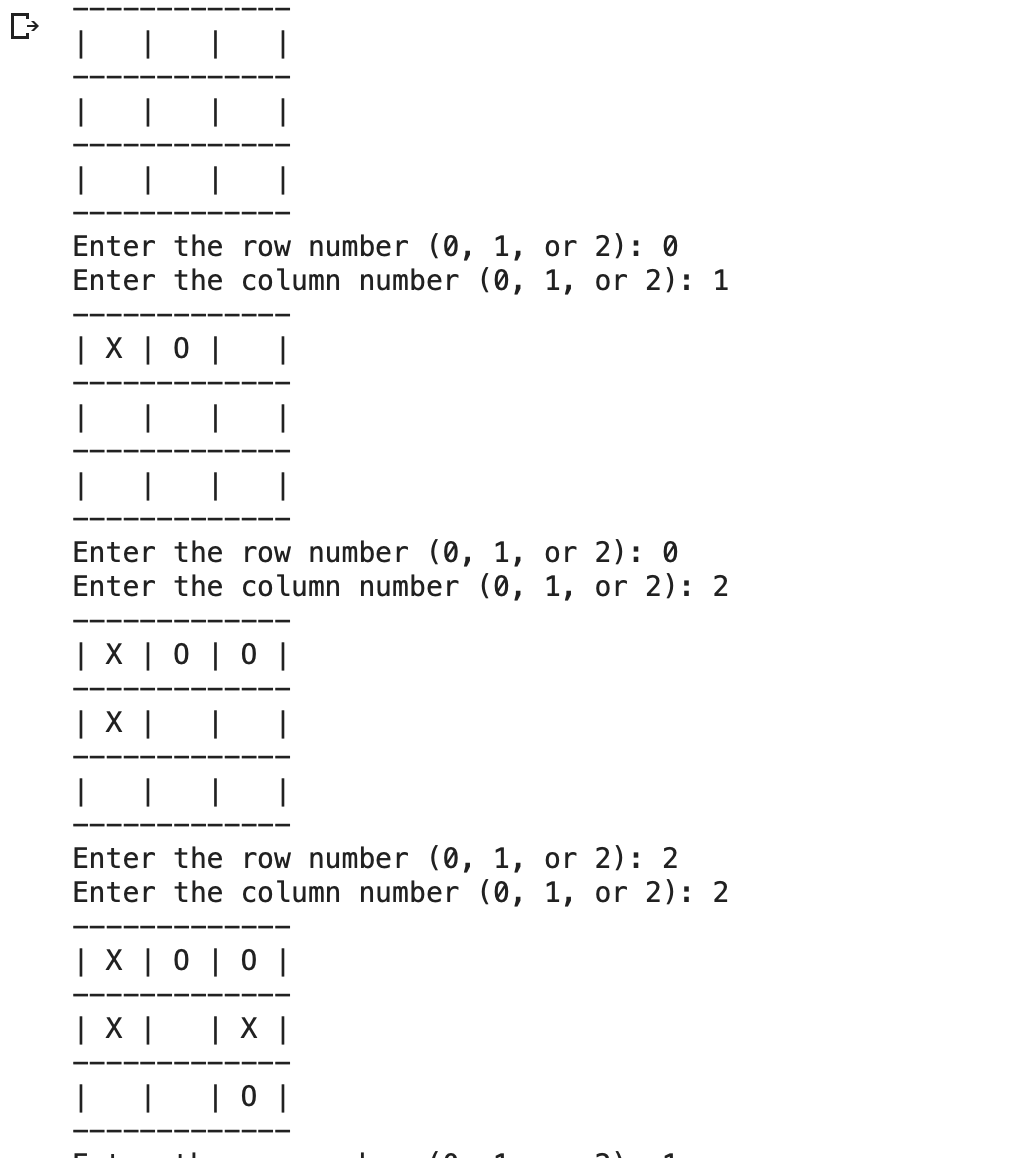
**Features of the Tic Tac Toe Game:**

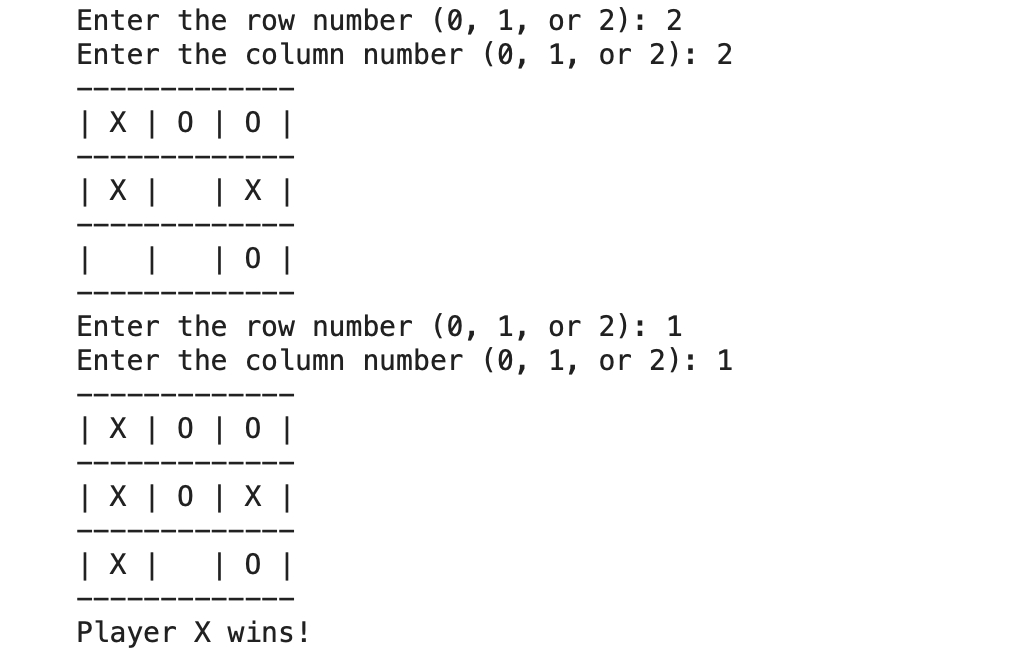
1. The **print\_board(board) function** prints the current state of the Tic Tac Toe board.
2. The check\_win(board, player) function checks if the specified player has won the game by checking all the possible winning conditions.
3. The **game\_over(board) function** checks if the game is over by calling the check\_win function for both players and also checks if there are any empty spaces left on the board.
4. The **evaluate(board) function** evaluates the current state of the board and returns a score: 1 if "X" has won, -1 if "O" has won, or 0 for a tie.
5. The **minimax(board, depth, alpha, beta, is\_maximizing) function** implements the Minimax algorithm with Alpha-Beta Pruning. It recursively explores all possible moves on the board up to a certain depth and returns the best score for the AI player ("X") or the opponent player ("O").
6. The **make\_ai\_move(board) function** determines the best move for the AI player by calling the minimax function for all possible moves and selecting the move with the highest evaluation score.
7. The **play\_game() function** initializes the game by creating an empty board and starts the game loop. It alternates between the human player ("O") and the AI player ("X") until the game is over. It takes input from the human player, updates the board, checks for a win or a tie, and makes the AI move accordingly.
8. Finally, the game starts by calling the **play\_game() function**.

**How to play our tic tac toe game:**

1. The game starts by displaying an empty Tic Tac Toe board, which consists of a 3x3 grid.
2. Each cell in the grid is represented by a row and column number. The rows and columns are numbered from 0 to 2.
3. The game begins with the human player ("O") taking the first turn.
4. The current state of the board is displayed, showing the empty cells as blank spaces.
5. When it's the human player's turn, you will be prompted to enter the row number (0, 1, or 2) and the column number (0, 1, or 2) for your move.
6. Enter the row number and column number where you want to place your "O" symbol. For example, if you want to place your symbol in the top-right cell, you would enter row 0 and column 2.
7. If the chosen cell is empty, your move is valid, and the board will be updated with your "O" symbol in the chosen cell.
8. After your move, the code checks if you have won the game or if the game is a tie. If either condition is met, the game ends, and the result is displayed.
9. If the game is not over, it's now the AI player's turn.
10. The AI player (represented by "X") uses the Minimax algorithm to determine the best move. It evaluates the possible moves and selects the one with the highest chance of winning.
11. The AI player's move is automatically made, and the board is updated with the AI player's "X" symbol in the chosen cell.
12. The code checks if the AI player has won the game or if the game is a tie. If either condition is met, the game ends, and the result is displayed.
13. Steps 5 to 12 are repeated until there is a winner or a tie.
14. If the game ends in a tie, the message "It's a tie!" will be displayed.
15. If the human player wins, the message "Player O wins!" will be displayed.
16. If the AI player wins, the message "Player X wins!" will be displayed.
17. The game loop continues until there is a winner or a tie, and the players can choose to play again by running the game again.

**Output of the Game:**





Player X - AI Player

Player O - Human

In the provided code, the names of the two players are "O" and "X". The human player is represented by "O", and the AI player is represented by "X".

**How was Minimax with Alpha Beta Pruning algorithm used to aid the AI PLAYER in winning the tic tac toe game?**

1. The AI player uses a smart strategy to decide its moves. It wants to find the best possible move that gives it the highest chance of winning.
2. The Minimax algorithm is like a smart decision-making process that the AI player follows. It explores all possible moves on the game board and evaluates the resulting game states.
3. The algorithm pretends to be both players, the AI player ("X") and the opponent ("O"). It looks ahead and tries to figure out how the game might unfold based on different moves.
4. The algorithm assigns a score to each possible move. If a move leads to a win for the AI player, it gets a high score. If it leads to a win for the opponent, it gets a low score. Ties get a neutral score.
5. By considering all possible moves and their scores, the AI player can make an informed decision about which move is most likely to lead to a win.
6. Alpha-Beta Pruning is a clever technique that helps the AI player search through the moves more efficiently. It allows the algorithm to quickly discard moves that are guaranteed to be worse than other moves it has already considered.
7. Imagine the game as a big tree, where each node represents a possible move and its resulting game state. The algorithm tries to explore the tree but cuts off branches that are clearly not worth exploring, saving time and computational resources.
8. By using the Minimax algorithm with Alpha-Beta Pruning, the AI player can "think ahead" and make smarter decisions based on the evaluation of possible future game states. This increases the AI player's chances of winning the game.