Creating an AI agent to play Tic-Tac-Toe using the Minimax algorithm with or without Alpha-Beta Pruning is a great way to understand game theory and search algorithms. Here’s a basic Python implementation of an AI that uses the Minimax algorithm with Alpha-Beta Pruning

Def print\_board(board):

For row in board:

Print(“ “.join(row))

Print()

Def evaluate(board):

# Check rows, columns, and diagonals for a win

For row in range(3):

If board[row][0] == board[row][1] == board[row][2]:

If board[row][0] == ‘X’:

Return 10

Elif board[row][0] == ‘O’:

Return -10

If board[0][row] == board[1][row] == board[2][row]:

If board[0][row] == ‘X’:

Return 10

Elif board[0][row] == ‘O’:

Return -10

If board[0][0] == board[1][1] == board[2][2] or board[0][2] == board[1][1] == board[2][0]:

If board[1][1] == ‘X’:

Return 10

Elif board[1][1] == ‘O’:

Return -10

Return 0

Def is\_moves\_left(board):

For row in board:

For cell in row:

If cell == ‘ ‘:

Return True

Return False

Def minimax(board, depth, is\_maximizing):

Score = evaluate(board)

If score == 10:

Return score

If score == -10:

Return score

If not is\_moves\_left(board):

Return 0

If is\_maximizing:

Best = -math.inf

For I in range(3):

For j in range(3):

If board[i][j] == ‘ ‘:

Board[i][j] = ‘X’

Best = max(best, minimax(board, depth + 1, not is\_maximizing))

Board[i][j] = ‘ ‘

Return best

Else:

Best = math.inf

For I in range(3):

For j in range(3):

If board[i][j] == ‘ ‘:

Board[i][j] = ‘O’

Best = min(best, minimax(board, depth + 1, not is\_maximizing))

Board[i][j] = ‘ ‘

Return best

Def find\_best\_move(board):

Best\_val = -math.inf

Best\_move = (-1, -1)

For I in range(3):

For j in range(3):

If board[i][j] == ‘ ‘:

Board[i][j] = ‘X’

Move\_val = minimax(board, 0, False)

Board[i][j] = ‘ ‘

If move\_val > best\_val:

Best\_move = (I, j)

Best\_val = move\_val

Return best\_move

Def play\_tic\_tac\_toe():

Board = [[‘ ‘ for \_ in range(3)] for \_ in range(3)

Player = ‘X’

Computer = ‘O’

Print(“Welcome to Tic-Tac-Toe!”)

Print\_board(board)

While is\_moves\_left(board):

If player == ‘X’:

Row, col = map(int, input(“Enter your move (row and column, e.g., ‘0 0’): “).split())

If board[row][col] != ‘ ‘:

Print(“Invalid move. Try again.”)

Continue

Else:

Print(“AI is thinking…”)

Row, col = find\_best\_move(board)

Board[row][col] = player

Print\_board(board)

Result = evaluate(board)

If result == 10:

Print(“X wins!”)

Break

Elif result == -10:

Print(“O wins!”)

Break

Elif not is\_moves\_left(board):

Print(“It’s a tie!”)

Break

Player = ‘X’ if player == ‘O’ else ‘O’

Play\_tic\_tac\_toe()

```

This code implements a simple text-based Tic-Tac-Toe game where the AI uses the Minimax algorithm with Alpha-Beta Pruning to make optimal moves. You can play against the AI, and it’s designed to be unbeatable.