# Tic-Tac-Toe Game with Minimax and Alpha-Beta Pruning

# The board is represented as a list of strings where each index corresponds to a position on the board.

# 'X' represents the human player, 'O' represents the AI, and an empty string represents an empty space.

board = [" " for \_ in range(9)]

# Function to print the Tic-Tac-Toe board

def print\_board(board):

print(board[0] + "|" + board[1] + "|" + board[2])

print("-+-+-")

print(board[3] + "|" + board[4] + "|" + board[5])

print("-+-+-")

print(board[6] + "|" + board[7] + "|" + board[8])

# Function to check for available moves

def empty\_cells(board):

return [i for i, cell in enumerate(board) if cell == " "]

# Function to check if the game is over

def game\_over(board):

# Check for a win

for combo in [(0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4, 6)]:

if board[combo[0]] == board[combo[1]] == board[combo[2]] != " ":

return board[combo[0]]

# Check for a tie

if " " not in board:

return "Tie"

return None

# Minimax function with Alpha-Beta Pruning

def minimax(board, depth, maximizing\_player, alpha, beta):

if game\_over(board) == "O":

return 1

if game\_over(board) == "X":

return -1

if game\_over(board) == "Tie":

return 0

if maximizing\_player:

max\_eval = float("-inf")

for move in empty\_cells(board):

board[move] = "O"

eval = minimax(board, depth - 1, False, alpha, beta)

board[move] = " "

max\_eval = max(max\_eval, eval)

alpha = max(alpha, eval)

if beta <= alpha:

break

return max\_eval

else:

min\_eval = float("inf")

for move in empty\_cells(board):

board[move] = "X"

eval = minimax(board, depth - 1, True, alpha, beta)

board[move] = " "

min\_eval = min(min\_eval, eval)

beta = min(beta, eval)

if beta <= alpha:

break

return min\_eval

# Function to find the best move for the AI using Minimax and Alpha-Beta Pruning

def best\_move(board):

best\_move = -1

best\_eval = float("-inf")

for move in empty\_cells(board):

board[move] = "O"

eval = minimax(board, 9, False, float("-inf"), float("inf"))

board[move] = " "

if eval > best\_eval:

best\_eval = eval

best\_move = move

return best\_move

# Main game loop

while True:

print\_board(board)

if " " not in board or game\_over(board):

result = game\_over(board)

if result == "Tie":

print("It's a Tie!")

else:

print(f"{result} wins!")

break

try:

user\_move = int(input("Enter your move (0-8): "))

if board[user\_move] != " ":

print("Invalid move. That cell is already taken.")

continue

board[user\_move] = "X"

except ValueError:

print("Invalid input. Please enter a number between 0 and 8.")

continue

ai\_move = best\_move(board)

board[ai\_move] = "O"