

**Artificial Intelligence Systems**

Lab Report # 05

**Submitted by:** Ali Hamza

**Registration No:** B23F0063AI106

**Submitted to:** Dr. Mohsin

Date: 25th February , 2025

# Lab Task:

# The pseudocode for score function in given in the Fig. 7.9. You are required to implement it.

# Complete implementation of minimize () function given in section 7.3

# Complete implementation of maximize () function given in section 7.3

# Implement of minmax () function given in section 7.3

# Test tic tac toa game for following outcome

# Computer wins

# Draw

# Human wins

*# The score() Function  
def* score(board):  
 *'''  
 Evaluate the board state and returns a score.  
 '''* winning\_combinations = [  
 (0, 1, 2), (3, 4, 5), (6, 7, 8), *# Rows* (0, 3, 6), (1, 4, 7), (2, 5, 8), *# Columns* (0, 4, 8), (2, 4, 6) *# Diagonals* ]  
 *for* (x, y, z) *in* winning\_combinations:  
 *if* board[x] == board[y] == board[z] *and* board[x] != " ":  
 *return* 1 *if* board[x] == "X" *else* -1 *# AI wins (+1) / Human wins (-1)  
  
 if* " " *not in* board: *# if no spaces left, it's a draw  
 return* 0  
  
 *return None # Game is still ongoing.  
  
# The minimize() Function  
def* minimize(board):  
 *'''  
 Finds the best move for the human player by minimizing AI's score.  
 '''  
 if* score(board) *is not None*:  
 *return* score(board)  
  
 best\_score = float("inf") *# Start with the highest possible score  
 for* move *in* range(9):  
 *if* board[move] == " ":  
 board[move] = "O" *# Human plays* best\_score = min(best\_score, maximize(board)) *# Recursive call* board[move] = " " *# Undo move  
  
 return* best\_score  
  
*# The maximize() Function  
def* maximize(board):  
 *"""  
 Finds the best move for the AI player by maximizing AI's score.  
 """  
 if* score(board) *is not None*:  
 *return* score(board)  
  
 best\_score = float("-inf") *# Start with the lowest possible score  
 for* move *in* range(9):  
 *if* board[move] == " ":  
 board[move] = "X" *# AI plays* best\_score = max(best\_score, minimize(board)) *# Recursive call* board[move] = " " *# Undo move  
  
 return* best\_score  
  
*# The minimax() Function  
def* minimax(board):  
 *"""Finds the best move for the AI using Minimax."""* best\_score = float("-inf")  
 best\_move = -1  
  
 *for* move *in* range(9):  
 *if* board[move] == " ":  
 board[move] = "X" *# AI plays* move\_score = minimize(board) *# AI tries to minimize human's score* board[move] = " " *# Undo move  
  
 if* move\_score > best\_score:  
 best\_score = move\_score  
 best\_move = move  
  
 *return* best\_move *# Returns the best move AI can make  
  
# Print the Board  
def* print\_board(board):  
 *"""Prints the Tic-Tac-Toe board in a user-friendly format."""* print("\n")  
 print(f"{board[0]} | {board[1]} | {board[2]}")  
 print("--+---+--")  
 print(f"{board[3]} | {board[4]} | {board[5]}")  
 print("--+---+--")  
 print(f"{board[6]} | {board[7]} | {board[8]}")  
 print("\n")  
  
*# Play the Game  
def* play\_game():  
 *"""Runs the Tic-Tac-Toe game."""* board = [" "] \* 9  
 human\_turn = *True # Human starts first  
  
 while True*:  
 print\_board(board)  
  
 *if* score(board) == 1:  
 print("AI Wins!")  
 *break  
 elif* score(board) == -1:  
 print("Human Wins!")  
 *break  
 elif* score(board) == 0:  
 print("It's a Draw!")  
 *break  
  
 if* human\_turn:  
 *try*:  
 move = int(input("Enter your move (0-8): "))  
 *if* board[move] == " ":  
 board[move] = "O"  
 human\_turn = *False  
 else*:  
 print("Invalid move! Try again.")  
 *except* (ValueError, IndexError):  
 print("Please enter a number between 0-8.")  
 *else*:  
 print("AI is making a move...")  
 move = minimax(board)  
 *if* move != -1:  
 board[move] = "X"  
 human\_turn = *True  
 else*:  
 print("Error: AI cannot make a move!")  
  
*# Run the Game*play\_game()

# 