**Practical no.03**

import math

MAX\_PLAYER = 'X'

MIN\_PLAYER = 'O'

EMPTY = '\_'

class TicTacToe:

def \_\_init\_\_(self):

# Initialize an empty 3x3 board

self.board = [

[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY]

]

def is\_moves\_left(self, board):

# Check if there are empty cells left on the board

for row in board:

if EMPTY in row:

return True

return False

def evaluate(self, board):

# Check rows, columns, and diagonals for a winner

for row in range(3):

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

if board[row][0] == MAX\_PLAYER:

return 10

elif board[row][0] == MIN\_PLAYER:

return -10

for col in range(3):

if board[0][col] == board[1][col] == board[2][col]:

if board[0][col] == MAX\_PLAYER:

return 10

elif board[0][col] == MIN\_PLAYER:

return -10

if board[0][0] == board[1][1] == board[2][2]:

if board[0][0] == MAX\_PLAYER:

return 10

elif board[0][0] == MIN\_PLAYER:

return -10

if board[0][2] == board[1][1] == board[2][0]:

if board[0][2] == MAX\_PLAYER:

return 10

elif board[0][2] == MIN\_PLAYER:

return -10

return 0

def minimax(self, board, depth, is\_maximizing, alpha, beta):

score = self.evaluate(board)

if score == 10:

return score - depth

if score == -10:

return score + depth

# If no more moves and no winner (tie)

if not self.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] == EMPTY:

# Make the move

board[i][j] = MAX\_PLAYER

# Recur and choose the maximum value

best = max(best, self.minimax(board, depth + 1, False, alpha, beta))

board[i][j] = EMPTY

alpha = max(alpha, best)

if beta <= alpha:

break

return best

else:

best = math.inf

for i in range(3):

for j in range(3):

if board[i][j] == EMPTY:

# Make the move

board[i][j] = MIN\_PLAYER

best = min(best, self.minimax(board, depth + 1, True, alpha, beta))

board[i][j] = EMPTY

# Update beta

beta = min(beta, best)

# Alpha-Beta Pruning

if beta <= alpha:

break

return best

def find\_best\_move(self):

best\_val = -math.inf

best\_move = (-1, -1)

# Traverse all cells, evaluate minimax function for each empty cell, and return the best move

for i in range(3):

for j in range(3):

if self.board[i][j] == EMPTY:

# Make the move

self.board[i][j] = MAX\_PLAYER

# Compute evaluation function for this move

move\_val = self.minimax(self.board, 0, False, -math.inf, math.inf)

# Undo the move

self.board[i][j] = EMPTY

# If the value of the current move is better than the best value, update best move

if move\_val > best\_val:

best\_move = (i, j)

best\_val = move\_val

return best\_move

def print\_board(self):

for row in self.board:

print(" | ".join(row))

print("-" \* 9)

game = TicTacToe()

game.board = [

['X', 'O', 'X'],

['O', 'O', '\_'],

['\_', '\_', 'X']

]

print("Current board:")

game.print\_board()

best\_move = game.find\_best\_move()

print(f"\nThe best move for 'X' is: {best\_move}")