## Indian Institute of Information Technology Surat

****

# Lab Report on

# Artificial Intelligence (CS 701) Practical

**Submitted by**

### [RAHUL KUMAR SINGH] (UI21CS44)

**Course Faculty**

### Dr. Ritesh Kumar

### Mrs. Archana Balmik

## Department of Computer Science and Engineering

## Indian Institute of Information Technology Surat

## Gujarat-394190, India

**Aug-2024**

## Lab No: 11

## Aim:

Implement Alpha-Beta Search and DFID for efficient decision-making in the tic-tac game.

## Description:

* 3x3 board used for Tic-Tac-Toe game implementation.
* Enhances minimax by pruning branches, reducing time complexity from O(b^d) to O(b^{d/2}).
* Measures efficiency in terms of explored nodes and time saved through intelligent pruning.
* Game ends when the player wins, the AI wins, or the draw occurs.
* DFID combines DFS and BFS techniques, leading to memory efficient and optimal result.

## Code:

**import math**

**def print\_board(board):**

**for row in board:**

**print("| " + " | ".join(row) + " |")**

**print("-" \* 13)**

**def check\_winner(board):**

**for row in board:**

**if row[0] == row[1] == row[2] != ' ':**

**return row[0]**

**for col in range(3):**

**if board[0][col] == board[1][col] == board[2][col] != ' ':**

**return board[0][col]**

**if board[0][0] == board[1][1] == board[2][2] != ' ':**

**return board[0][0]**

**if board[0][2] == board[1][1] == board[2][0] != ' ':**

**return board[0][2]**

**return None**

**def is\_draw(board):**

**for row in board:**

**if ' ' in row:**

**return False**

**return True**

**def alpha\_beta(board, depth, alpha, beta, is\_ai):**

**winner = check\_winner(board)**

**if winner == 'X':**

**return -1**

**elif winner == 'O':**

**return 1**

**elif is\_draw(board):**

**return 0**

**move = 'O' if is\_ai else 'X'**

**best\_score = -math.inf if is\_ai else math.inf**

**func = None**

**aMax = None**

**bMin = None**

**if is\_ai:**

**func = lambda x,y: max(x,y)**

**aMax = lambda x,y: max(x,y)**

**bMin = lambda x,y: x**

**else:**

**func = lambda x,y: min(x,y)**

**aMax = lambda x,y: x**

**bMin = lambda x,y: min(x,y)**

**for i in range(3):**

**for j in range(3):**

**if board[i][j] == ' ':**

**board[i][j] = move**

**score = alpha\_beta(board, depth + 1, alpha, beta, not is\_ai)**

**board[i][j] = ' '**

**best\_score = func(score, best\_score)**

**alpha = aMax(alpha, best\_score)**

**beta = bMin(beta, best\_score)**

**return best\_score**

**def find\_best\_move\_ab(board):**

**best\_score = -math.inf**

**move = None**

**alpha = -math.inf**

**beta = math.inf**

**for i in range(3):**

**for j in range(3):**

**if board[i][j] == ' ':**

**board[i][j] = 'O'**

**score = alpha\_beta(board, 0, alpha, beta, False)**

**board[i][j] = ' '**

**if score > best\_score:**

**best\_score = score**

**move = (i, j)**

**return move**

**def dfid(board, depth, is\_ai):**

**winner = check\_winner(board)**

**if winner == 'X':**

**return True, -1**

**elif winner == 'O':**

**return True, 1**

**elif is\_draw(board):**

**return True, 0**

**if depth == 0:**

**return False, 0**

**move = 'O' if is\_ai else 'X'**

**best\_score = -math.inf if is\_ai else math.inf**

**func = max if is\_ai else min**

**found = False**

**for i in range(3):**

**for j in range(3):**

**if board[i][j] == ' ':**

**board[i][j] = move**

**\_, score = dfid(board, depth - 1, not is\_ai)**

**board[i][j] = ' '**

**best\_score = func(best\_score, score)**

**found = True**

**return found, best\_score**

**def find\_best\_move\_dfid(board, max\_depth):**

**best\_move = None**

**for depth in range(1, max\_depth + 1):**

**for i in range(3):**

**for j in range(3):**

**if board[i][j] == ' ':**

**board[i][j] = 'O'**

**found, score = dfid(board, depth, False)**

**board[i][j] = ' '**

**if found and score >= 0:**

**best\_move = (i, j)**

**return best\_move**

**return best\_move**

**def main():**

**board = [[' ' for \_ in range(3)] for \_ in range(3)]**

**player\_turn = True**

**ch = int(input("Enter your choice for AI (0-> Alpha-beta || 1-> DFID): "))**

**while True:**

**print\_board(board)**

**if check\_winner(board) == 'X':**

**print("Player wins!")**

**break**

**elif check\_winner(board) == 'O':**

**print("AI wins!")**

**break**

**elif is\_draw(board):**

**print("It's a draw!")**

**break**

**if player\_turn:**

**row, col = map(int, input("Enter your move (row col): ").split())**

**if board[row][col] == ' ':**

**board[row][col] = 'X'**

**player\_turn = False**

**else:**

**print("Invalid move. Try again.")**

**else:**

**print("AI is making its move...")**

**if ch==0: move = find\_best\_move\_ab(board)**

**else: move = find\_best\_move\_dfid(board,100)**

**if move:**

**board[move[0]][move[1]] = 'O'**

**player\_turn = True**

**else:**

**print("AI is unable to make a move due to technical errors.")**

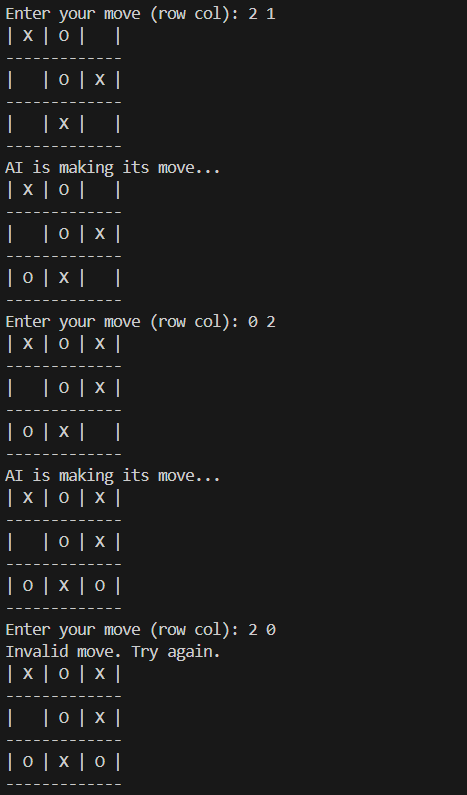
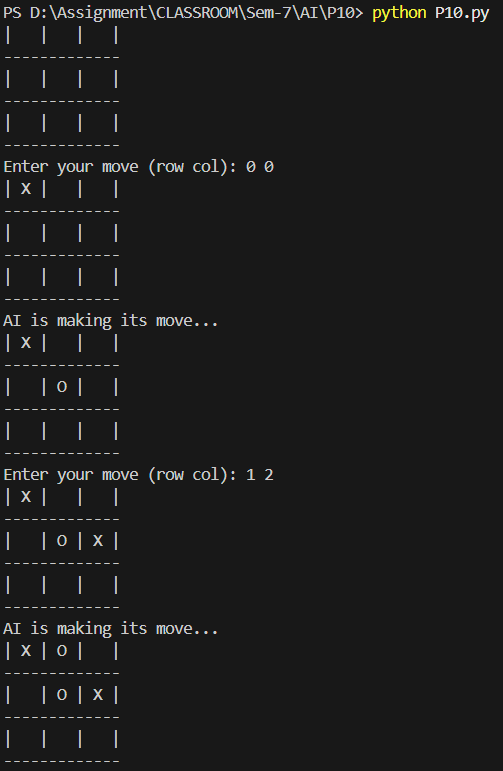
**exit()**

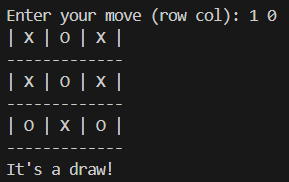
**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

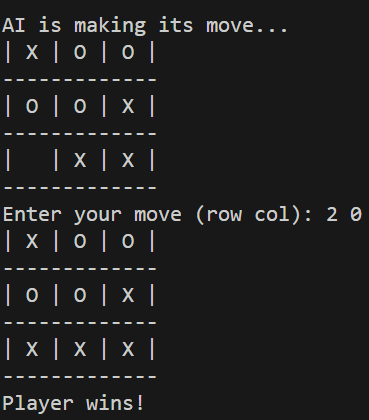
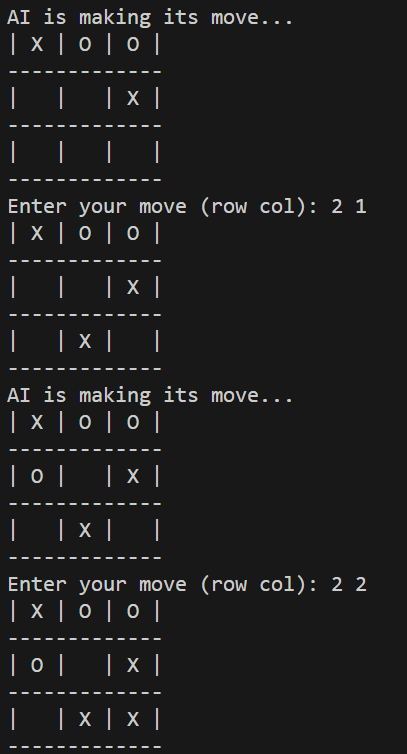
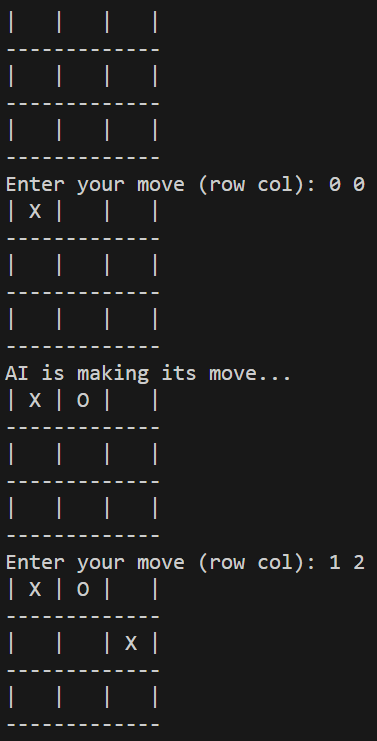
## Output:

**Alpha-Beta:**





**DFID:**

****

## Conclusion:

* In DFID, AI makes locally optimal moves up to a set depth, allowing the player a chance to win.
* In alpha-beta, AI ensures a win or draw, guaranteeing the player can never achieve a victory.
* AI guarantees optimal play using Minimax (alpha-beta) for unbeatable strategy.
* Player's challenge lies in preventing AI from winning.