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.

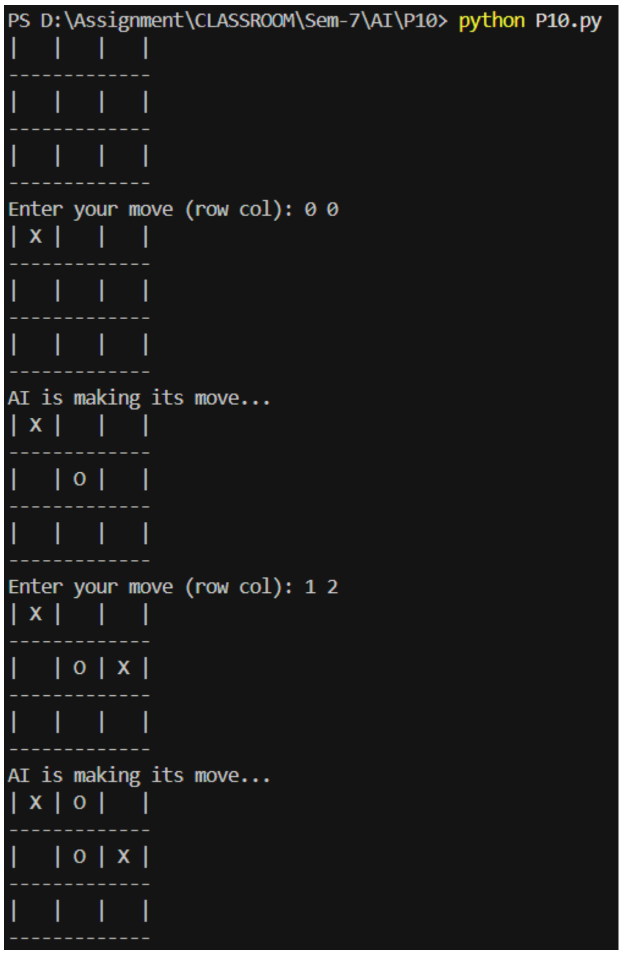
Code:

```python
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:
```

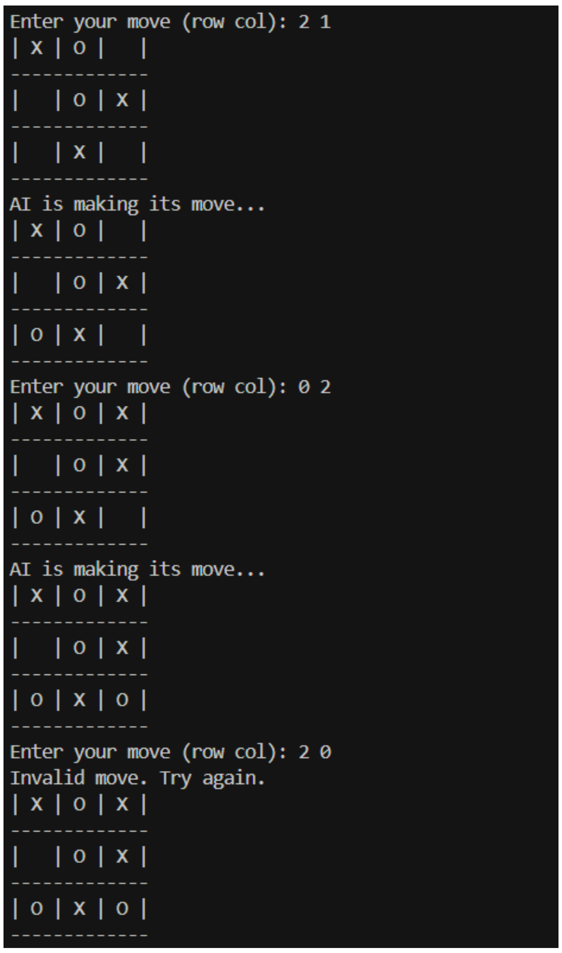
```python
                print("Invalid move. Try again.")
        else:
            print("AI is making its move...")
            if ch==0: move = find_best_move_ab(board)
            else: move = find_best_move_dfid(board,100)
            if move:
                board[move[0]][move[1]] = 'O'
                player_turn = True
            else:
                print("AI is unable to make a move due to technical errors.")
                exit()

if __name__ == "__main__":
    main()
```

Output:

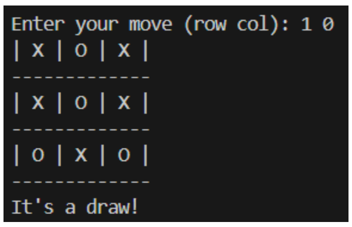
Alpha-Beta:



```
PS D:\Assignment\CLASSROOM\Sem-7\AI\P10> python P10.py
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Enter your move (row col): 0 0
| X |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
AI is making its move...
| X |   |   |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
Enter your move (row col): 1 2
| X |   |   |
-------------
|   | O | X |
-------------
|   |   |   |
-------------
AI is making its move...
| X | O |   |
-------------
|   | O | X |
-------------
|   |   |   |
-------------
```

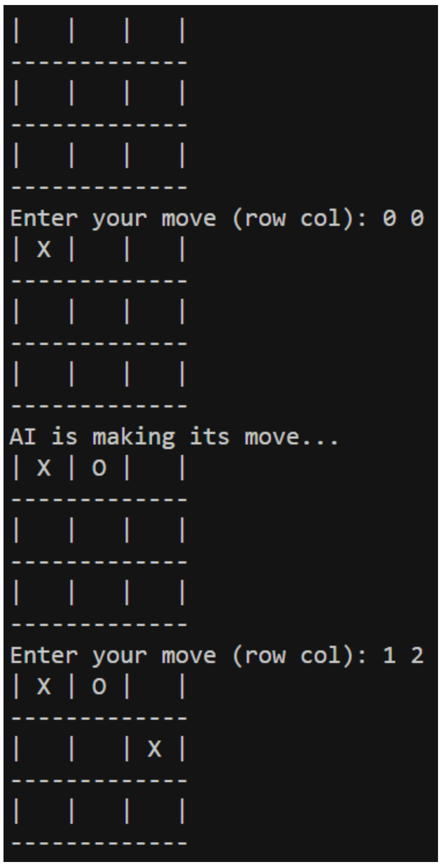


```
Enter your move (row col): 2 1
| X | O |   |
-------------
|   | O | X |
-------------
|   | X |   |
-------------
AI is making its move...
| X | O |   |
-------------
|   | O | X |
-------------
| O | X |   |
-------------
Enter your move (row col): 0 2
| X | O | X |
-------------
|   | O | X |
-------------
| O | X |   |
-------------
AI is making its move...
| X | O | X |
-------------
|   | O | X |
-------------
| O | X | O |
-------------
Enter your move (row col): 2 0
Invalid move. Try again.
| X | O | X |
-------------
|   | O | X |
-------------
| O | X | O |
-------------
```

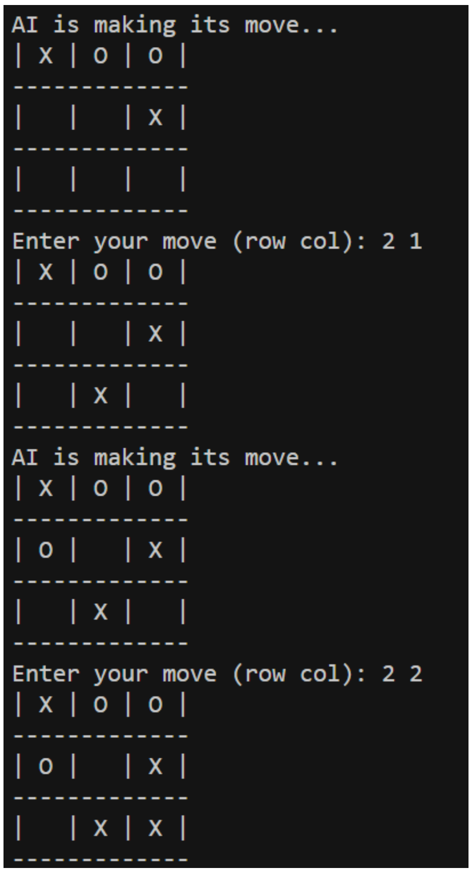


```
Enter your move (row col): 1 0
| X | O | X |
-------------
| X | O | X |
-------------
| O | X | O |
-------------
It's a draw!
```

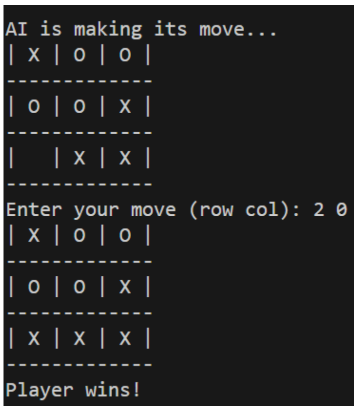
DFID:



```
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Enter your move (row col): 0 0
| X |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
AI is making its move...
| X | O |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Enter your move (row col): 1 2
| X | O |   |
-------------
|   |   | X |
-------------
|   |   |   |
-------------
```



```
AI is making its move...
| X | O | O |
-------------
|   |   | X |
-------------
|   |   |   |
-------------
Enter your move (row col): 2 1
| X | O | O |
-------------
|   |   | X |
-------------
|   | X |   |
-------------
AI is making its move...
| X | O | O |
-------------
| O |   | X |
-------------
|   | X |   |
-------------
Enter your move (row col): 2 2
| X | O | O |
-------------
| O |   | X |
-------------
|   | X | X |
-------------
```



```
AI is making its move...
| X | O | O |
-------------
| O | O | X |
-------------
|   | X | X |
-------------
Enter your move (row col): 2 0
| X | O | O |
-------------
| O | O | X |
-------------
| X | X | X |
-------------
Player wins!
```

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.