AI PROJECT REPORT(AI101B)

EVEN SEMESTER
SESSION 2024-25

NOUGHTS AND CROSS WITH ALPHA BETA PRUNING

SHALENDRA SHARMA(202410116100191)
RISHU AGARWAL(202410116100168)
RITIK(202410116100169)
SANIDHYA GARG(202410116100182)

PROJECT SUPERVISOR

KOMAL SALGOTRA
ASSISTANT PROFESSOR

INTRODUCTION

Tic-tac-toe also known as noughts and crosses is a paper and pencil game for two players, who take turns marking the spaces in a 3 x 3 grid traditionally. The player who succeeds in placing three of their marks in a horizontal, vertical or diagonal row wins the game. It is a zero-sum of perfect information game. This means that it is deterministic, with fully observable environments in which two agents act alternately and the utility values at the end of the game are always equal and opposite. Because of the simplicity of tic-tac-toe, it is often used as a pedagogical tool in artificial intelligence to deal with searching for game trees. The optimal move for this game can be gained by using minimax algorithm, where the opposition between the utility functions makes the situation adversarial, hence requiring adversarial search supported by a minimax algorithm with alpha beta pruning concept in artificial intelligence.

OBJECTIVES

To develop Artificial intelligence-based tic-tac-toe game for human Vs AI by implementing minimax algorithm with adversarial search concept. To analyze the complexity of the minimax algorithm through 3x3 tic tac toe game. To study and implement alpha-beta pruning concept for improved speed of searching the optimal choice in tic-tac toe game. To study optimizing methods for alpha-beta pruning using heuristic evaluation function.

AGENTS

An agent perceives its environment through sensors and acts on the environment through actuators. Its behavior is described by its function that maps the percept to action. In tic tac toe there are two agents. the computer system and humans. In the AI-based tic-tac-toe same the function is mapped using the minimax algorithm alongside alpha beta pruning. The agent can be further described by following factors:

Performance Measure: Number of wins or draws.

Environment: A 3x3 grid with 9 cells respectively, with opponent (human agent). The cell once occupied by a mark cannot be used again. Task environment is deterministic, fully observable, static, multi-agent, discrete, sequential.

Actuators: Display, Mouse (human agent)

Sensors: The opponent's (human agent) input as a mouse pointer on a cellFurthermore. if is a unity-cased agent since it uses heuristics and pruning concept as utility function that measures its preferences among the states and chooses the action that leads to the best expected utility for example, winning or tie condition for the computer system agent. The Utility function is taken as +10 for winning situation of maximizing agent or AI, -10 for winning situation of minimizing agent or human player and 0 for draw condition.

AGENT ENVIRONMENT

The basic environment for tic tac toe game agent is a 3x3 grid with 9 cells respectively with the opponent as human agent. The cell once occupied by a mark cannot be used again. The agent has access to the complete state of the environment at any point and can detect all aspects that are relevant to the choice of action making the environment fully observable. Also, the next state of the environment can be completely determined by current state and action executed which means the environment is deterministic. It is a two-player game with the opponent as a human agent, so it is a multi-agent environment. The game environment has a finite number of states, percepts and actions, making the environment static and discrete. Also, the environment is sequential since the current choice of cell could affect all future decisions. Thus, the task environment for the tic tac toe system agent is sequential, deterministic, fully observable, static, discrete and a multi-agent.

PROBLEM SPECIFICATION

Tic tac toe game has 9 cells for a 3x3 grid. The two players with their respective marks as 'X' and 'O' are required to place their marks in their turns one by one. Once the cell is occupied by a mark it cannot be used again. The game is won if the agent is able to make a row or column or a diagonal occupied completely with their respective marks. The game terminates once the winning situation is gained or the cells are fully occupied. The problem specification for this game is given below:

Problem: Given a 3x3 grid, the agents have to find the optimal cell to fill with respective marks.

Goals: To find the optimal cell to fill with respective marks and in order to win the game, the cell must be filled such that one of the following criteria is satisfied:

A row is completely filled by a mark 'X' or 'O'.

A diagonal is completely filled by a mark 'X' or 'O'.

A column is completely filled by a mark 'X' or 'O'.

If these criteria are not satisfied by both the agents, the game is terminated with a tie situation.

Constraints:

Once the cell is occupied by a mark, it cannot be reused.

Agents place the mark alternatively. So, consecutive moves from any agent are not

Allowed.

Methodology for Noughts and Crosses using Alpha-Beta Pruning

1. Game Representation

Represent the Tic-Tac-Toe board as a 3×3 matrix.

Each cell can have three values:

X (Player 1)

O (Player 2)

Empty (Available move)

2. Minimax Algorithm with Alpha-Beta Pruning

Minimax Algorithm is a recursive function that explores all possible moves and selects the optimal move for the AI.

Alpha-Beta Pruning reduces unnecessary evaluations by skipping branches that won't affect the final decision.

3. Implementation Steps

Step 1: Define the Evaluation Function

The function assigns scores to board states:

+10 → If AI wins (X or O, depending on the AI's symbol)

-10 → If the opponent wins

0 → If it’s a draw

Step 2: Implement Minimax Algorithm

If the current board state is a win, loss, or draw, return the evaluation score.

Otherwise, recursively explore all possible moves and choose the best one.

Step 3: Apply Alpha-Beta Pruning

Introduce two variables:

alpha (best already found for the maximizing player)

beta (best already found for the minimizing player)

Prune branches when beta ≤ alpha, as further exploration is unnecessary.

4. Algorithm Steps

If the game is over (win/loss/draw), return the evaluation score.

If the current player is AI (maximizing), initialize maxEval = -∞:

For each possible move:

Apply the move

Call the Minimax function recursively

Update maxEval = max(maxEval, result)

Update alpha = max(alpha, maxEval)

If beta ≤ alpha, break (pruning)

If the current player is the opponent (minimizing), initialize minEval = +∞:

For each possible move:

Apply the move

Call the Minimax function recursively

Update minEval = min(minEval, result)

Update beta = min(beta, minEval)

If beta ≤ alpha, break (pruning)

5. Choosing the Best Move

The AI iterates through all available moves, calls the Minimax function with Alpha-Beta Pruning, and selects the move with the best score.

Code :

```
import math



def print_board(board):

    for row in board:
```

```
        print(" ".join(row))

    print()



def is_moves_left(board):

    return any(" " in row for row in board)



def evaluate(board):

    for row in board:

        if row[0] == row[1] == row[2] and row[0] != " ":

            return 10 if row[0] == 'X' else -10



    for col in range(3):

        if board[0][col] == board[1][col] == board[2][col] and
board[0][col] != " ":

            return 10 if board[0][col] == 'X' else -10



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

        return 10 if board[0][0] == 'X' else -10

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

        return 10 if board[0][2] == 'X' else -10
```

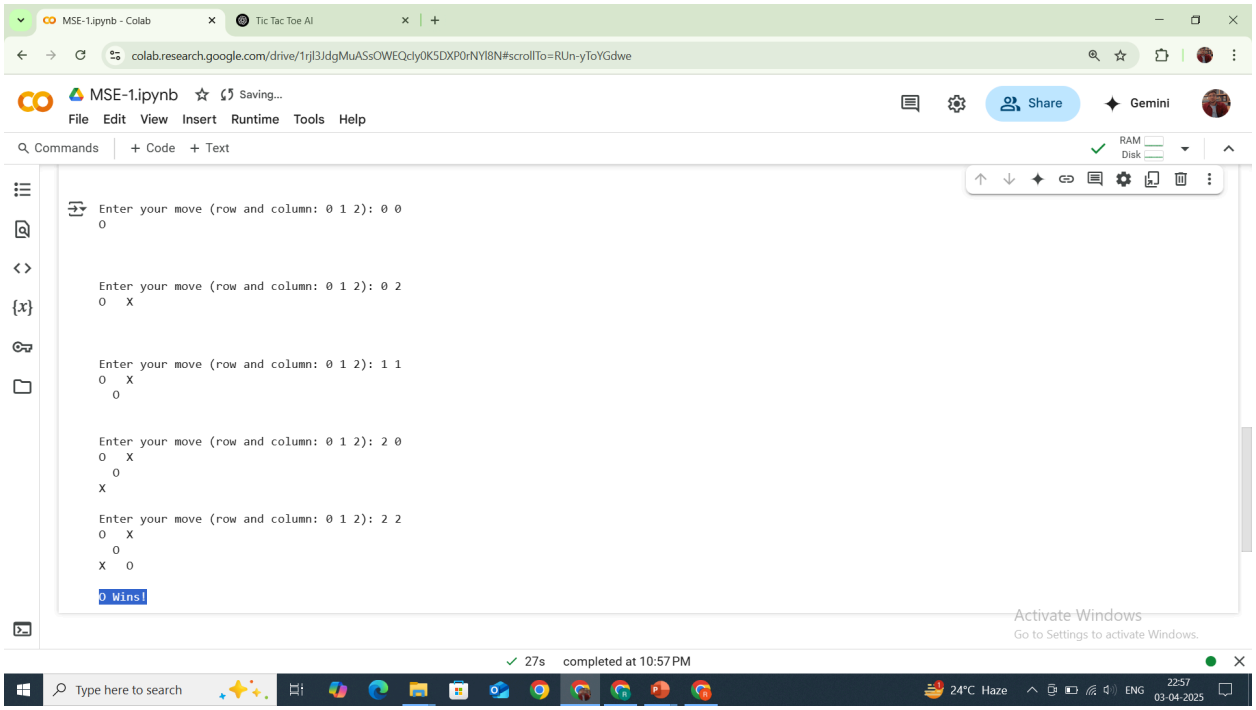
```
    return 0



def minimax(board, depth, is_max, alpha, beta):

    score = evaluate(board)

    if score == 10 or score == -10:

        return score - depth if score == 10 else score + depth

    if not is_moves_left(board):

        return 0



    if is_max:

        best = -math.inf

        for i in range(3):

            for j in range(3):

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

                    board[i][j] = 'X'

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

                    board[i][j] = " "

                    alpha = max(alpha, best)

                    if beta <= alpha:

                        break
```

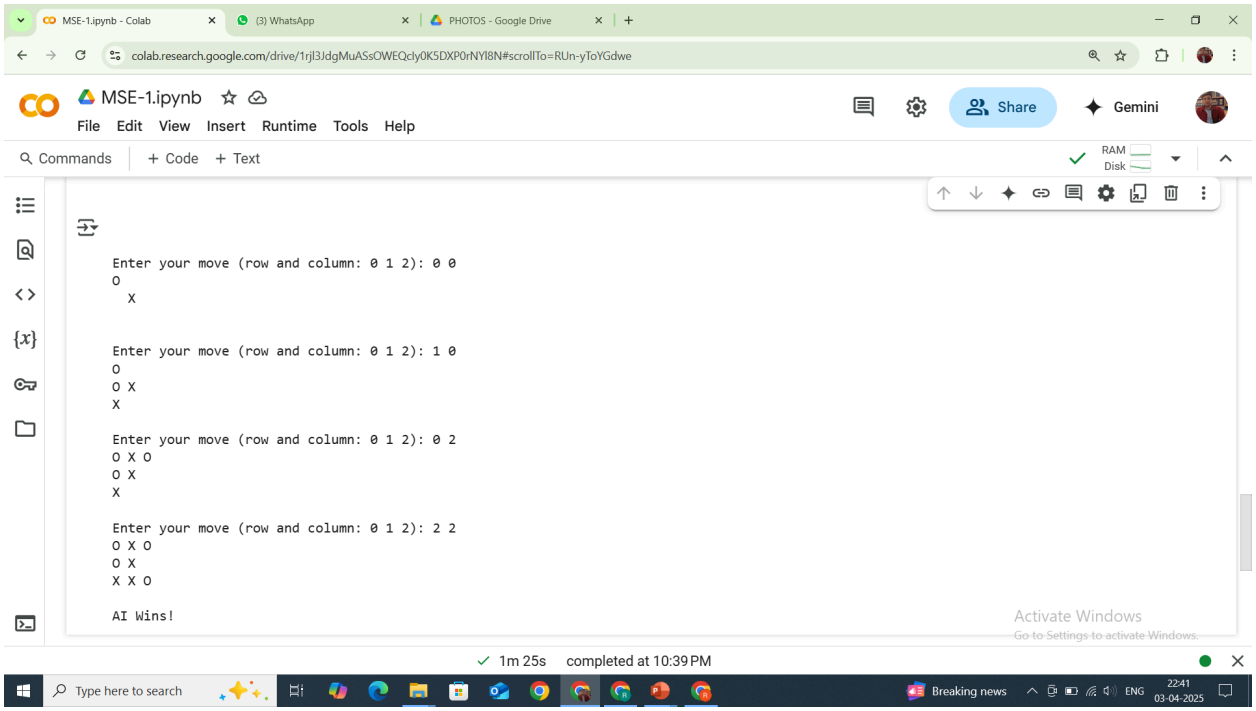
```
        return best
    else:
        best = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = 'O'
                    best = min(best, minimax(board, depth + 1, True,
alpha, beta))
                    board[i][j] = " "
                    beta = min(beta, best)
                    if beta <= alpha:
                        break
        return best


def find_best_move(board):
    best_val = -math.inf
    best_move = (-1, -1)


    for i in range(3):
        for j in range(3):
```

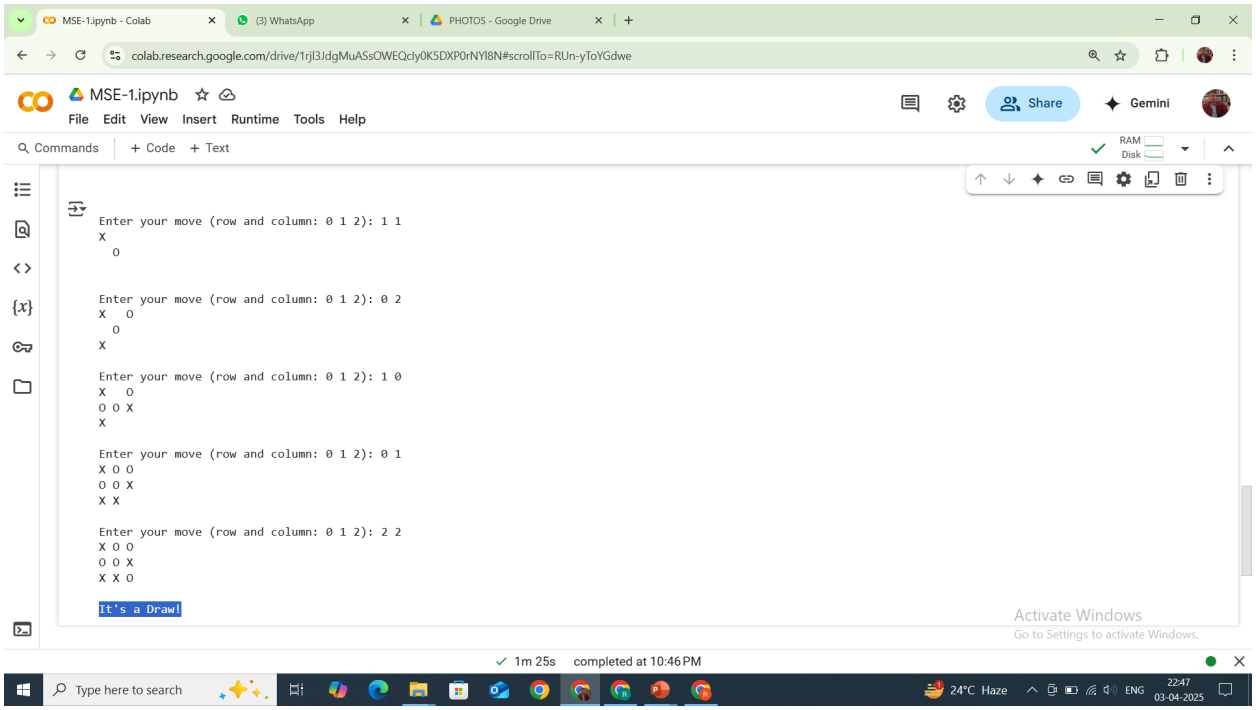
```
            if board[i][j] == " ":
                board[i][j] = 'X'
                move_val = minimax(board, 0, False, -math.inf, math.inf)
                board[i][j] = " "
                if move_val > best_val:
                    best_val = move_val
                    best_move = (i, j)
    return best_move


def play_game():
    board = [[" " for _ in range(3)] for _ in range(3)]
    while is_moves_left(board) and evaluate(board) == 0:
        print_board(board)
        row, col = map(int, input("Enter your move (row and column: 0 1
2): ").split())
        if board[row][col] != " ":
            print("Invalid move! Try again.")
            continue
        board[row][col] = 'O'
        if evaluate(board) != 0 or not is_moves_left(board):
            break
```

```
        ai_move = find_best_move(board)
        board[ai_move[0]][ai_move[1]] = 'X'
    print_board(board)
    result = evaluate(board)
    if result == 10:
        print("AI Wins!")
    elif result == -10:
        print("You Win!")
    else:
        print("It's a Draw!")


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

Output :


MSE-1.ipynb - Colab
Tic Tac Toe AI
colab.research.google.com/drive/1rjl3JdgMuASsOWEQcly0K5DXP0rNYl8N#scrollTo=RUn-yToYGdwe
MSE-1.ipynb
Saving...
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
RAM
Disk
Enter your move (row and column: 0 1 2): 0 0
O
Enter your move (row and column: 0 1 2): 0 2
O X
Enter your move (row and column: 0 1 2): 1 1
O X
O
Enter your move (row and column: 0 1 2): 2 0
O X
O
X
Enter your move (row and column: 0 1 2): 2 2
O X
O
X O
O Wins!
Activate Windows
Go to Settings to activate Windows.
27s completed at 10:57 PM
Type here to search
24°C Haze
ENG
22:57
03-04-2025


MSE-1.ipynb - Colab
(3) WhatsApp
PHOTOS - Google Drive
colab.research.google.com/drive/1rjl3JdgMuASsOWEQcly0K5DXP0rNYl8N#scrollTo=RUn-yToYGdwe
MSE-1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
RAM
Disk
Enter your move (row and column: 0 1 2): 0 0
O
X
Enter your move (row and column: 0 1 2): 1 0
O
O X
X
Enter your move (row and column: 0 1 2): 0 2
O X O
O X
X
Enter your move (row and column: 0 1 2): 2 2
O X O
O X
X X O
AI Wins!
Activate Windows
Go to Settings to activate Windows.
1m 25s completed at 10:39 PM
Type here to search
Breaking news
ENG
22:41
03-04-2025


MSE-1.ipynb - Colab
(3) WhatsApp
PHOTOS - Google Drive
colab.research.google.com/drive/1rjl3JdgMuASsOWEQcly0K5DXP0rNYl8N#scrollTo=RUn-yToYGdwe
MSE-1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
RAM
Disk
Enter your move (row and column: 0 1 2): 1 1
X
O
Enter your move (row and column: 0 1 2): 0 2
X O
O
X
Enter your move (row and column: 0 1 2): 1 0
X O
O O X
X
Enter your move (row and column: 0 1 2): 0 1
X O O
O O X
X X
Enter your move (row and column: 0 1 2): 2 2
X O O
O O X
X X O
It's a Draw!
Activate Windows
Go to Settings to activate Windows.
1m 25s completed at 10:46 PM
Type here to search
24°C Haze
ENG
22:47
03-04-2025