DATE:08/08/2025

EXP:05

ALPHA BETA PRUNING

AIM:

Problem statement

Design a Tic Tac Toe game using Min-Max search algorithm with alpha-beta Pruning

ALGORITHM:

Algorithm: ALPHA BETA PRUNING

```
function MINIMAX(node, depth, alpha, beta, maximizingPlayer):
    // Base case: depth limit or terminal state
    if depth == 0 OR node is a terminal node:
        return EVALUATE(node)      // static evaluation of board
    // Case 1: Maximizer's turn
    if maximizingPlayer == True:
        maxEval = -∞
        for each child in SUCCESSORS(node):
            eval = MINIMAX(child, depth - 1, alpha, beta, False)
            maxEval = max(maxEval, eval)
            alpha = max(alpha, maxEval)
            if beta <= alpha:      // prune
```

```
            break
        return maxEval
    // Case 2: Minimizer's turn
    else:
        minEval = +∞
        for each child in SUCCESSORS(node):
            eval = MINIMAX(child, depth - 1, alpha, beta, True)
            minEval = min(minEval, eval)
            beta = min(beta, minEval)
            if beta <= alpha:      // prune
                break
        return minEval
```

CODE:

```
import math
import random

# Board positions are 0..8 (row-major)
# Board cells: 'X', 'O', or ' ' (space) for empty

WIN_COMBINATIONS = [
```

```
    (0,1,2), (3,4,5), (6,7,8),  # rows
    (0,3,6), (1,4,7), (2,5,8),  # cols
    (0,4,8), (2,4,6)            # diags
]

def print_board(board):
    def cell(i):
        return board[i] if board[i] != ' ' else str(i+1)
    print()
    print(f" {cell(0)} | {cell(1)} | {cell(2)} ")
    print("---+---+---")
    print(f" {cell(3)} | {cell(4)} | {cell(5)} ")
    print("---+---+---")
    print(f" {cell(6)} | {cell(7)} | {cell(8)} ")
    print()

def available_moves(board):
    return [i for i, c in enumerate(board) if c == ' ']

def check_winner(board):
    for a,b,c in WIN_COMBINATIONS:
        if board[a] == board[b] == board[c] and board[a] != ' ':
            return board[a]  # 'X' or 'O'
```

```
    return None

def is_full(board):
    return all(c != ' ' for c in board)

def game_over(board):
    winner = check_winner(board)
    if winner:
        return True, winner
    if is_full(board):
        return True, None
    return False, None

# Minimax with Alpha-Beta pruning
def minimax(board, depth, alpha, beta, maximizing, ai_player, human_player):
    over, winner = game_over(board)
    if over:
        if winner == ai_player:
            return 10 - depth, None   # prefer faster win
        elif winner == human_player:
            return depth - 10, None   # prefer slower loss
        else:
            return 0, None            # draw
```

```
if maximizing:
    max_eval = -math.inf
    best_move = None
    for move in available_moves(board):
        board[move] = ai_player
        eval_score, _ = minimax(board, depth+1, alpha, beta, False, ai_player, human_player)
        board[move] = ' '
        if eval_score > max_eval:
            max_eval = eval_score
            best_move = move
        alpha = max(alpha, eval_score)
        if beta <= alpha:
            break  # beta cut-off
    return max_eval, best_move
else:
    min_eval = math.inf
    best_move = None
    for move in available_moves(board):
        board[move] = human_player
        eval_score, _ = minimax(board, depth+1, alpha, beta, True, ai_player, human_player)
        board[move] = ' '
        if eval_score < min_eval:
```

```
            min_eval = eval_score
            best_move = move
        beta = min(beta, eval_score)
        if beta <= alpha:
            break  # alpha cut-off
    return min_eval, best_move


def ai_move(board, ai_player, human_player):
    # If board is empty, pick a random corner (tiny speed boost / variety)
    if board.count(' ') == 9:
        return random.choice([0,2,6,8])
    score, move = minimax(board, depth=0, alpha=-math.inf, beta=math.inf,
                          maximizing=True, ai_player=ai_player, human_player=human_player)
    return move


def human_turn(board, human_player):
    while True:
        user = input(f"Your move ({human_player}). Enter cell 1-9: ").strip()
        if not user.isdigit():
            print("Please enter a number 1-9.")
            continue
        idx = int(user) - 1
        if idx < 0 or idx > 8:
```

```
            print("Index out of range. Choose 1-9.")
            continue
        if board[idx] != ' ':
            print("Cell already taken. Pick another.")
            continue
        return idx

def play_game():
    print("Tic-Tac-Toe with Minimax (Alpha-Beta pruning)\n")
    human_player = ''
    while human_player not in ('X','O'):
        human_player = input("Choose your symbol (X goes first): X or O? ").strip().upper()
    ai_player = 'O' if human_player == 'X' else 'X'
    board = [' '] * 9
    current = 'X'  # X always starts

    print_board(board)

    while True:
        over, winner = game_over(board)
        if over:
            if winner:
                print_board(board)
```

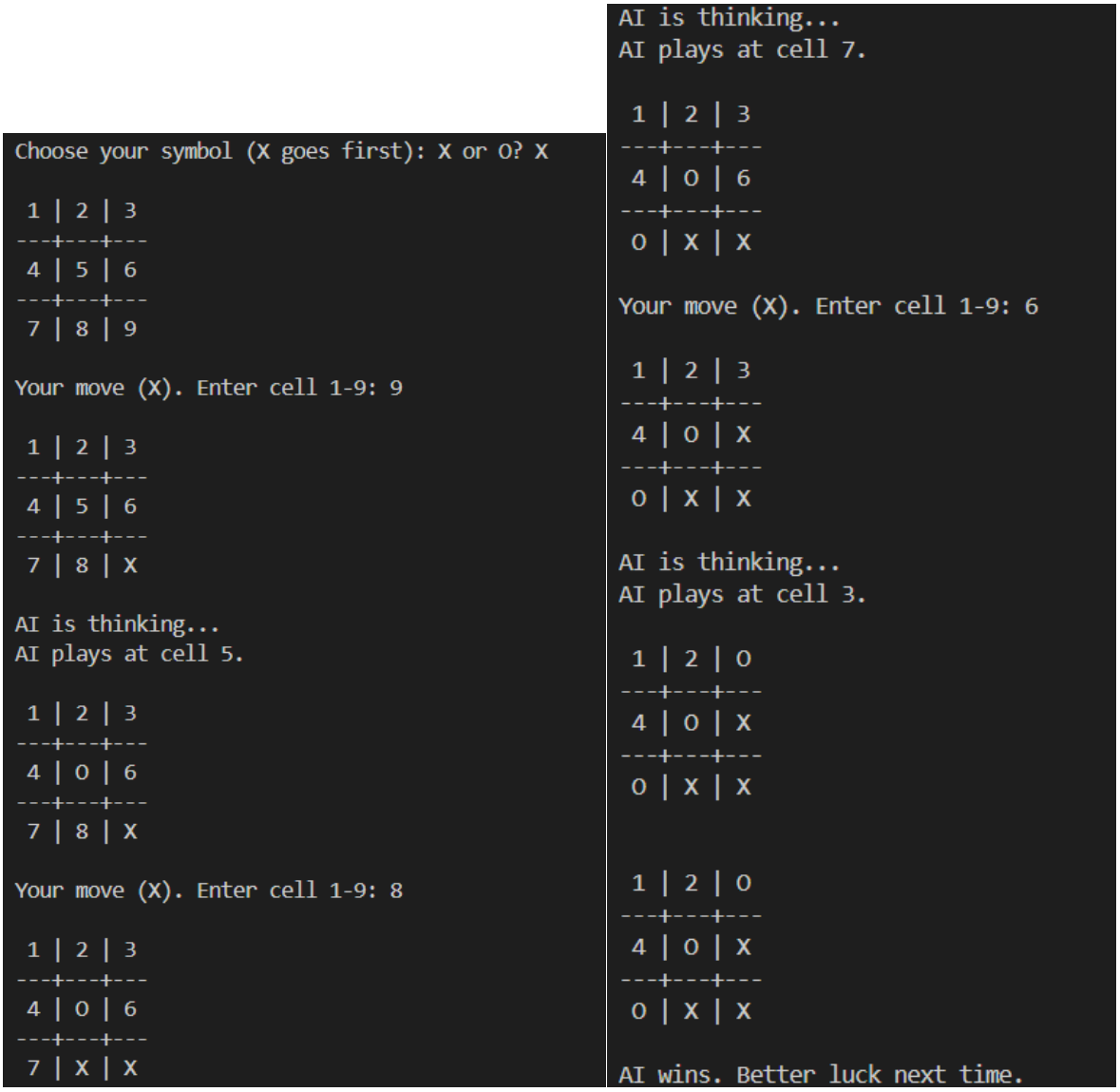
```
            if winner == human_player:
                print("You win! 🥕")
            else:
                print("AI wins. Better luck next time.")
        else:
            print_board(board)
            print("It's a draw.")
        break

    if current == human_player:
        idx = human_turn(board, human_player)
        board[idx] = human_player
    else:
        print("AI is thinking...")
        idx = ai_move(board, ai_player, human_player)
        board[idx] = ai_player
        print(f"AI plays at cell {idx+1}.")

    print_board(board)
    current = ai_player if current == human_player else human_player

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

OUTPUT:



```
Choose your symbol (X goes first): X or O? X

 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9

Your move (X). Enter cell 1-9: 9

 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | X

AI is thinking...
AI plays at cell 5.

 1 | 2 | 3
---+---+---
 4 | O | 6
---+---+---
 7 | 8 | X

Your move (X). Enter cell 1-9: 8

 1 | 2 | 3
---+---+---
 4 | O | 6
---+---+---
 7 | X | X

AI is thinking...
AI plays at cell 7.

 1 | 2 | 3
---+---+---
 4 | O | 6
---+---+---
 O | X | X

Your move (X). Enter cell 1-9: 6

 1 | 2 | 3
---+---+---
 4 | O | X
---+---+---
 O | X | X

AI is thinking...
AI plays at cell 3.

 1 | 2 | O
---+---+---
 4 | O | X
---+---+---
 O | X | X


 1 | 2 | O
---+---+---
 4 | O | X
---+---+---
 O | X | X

AI wins. Better luck next time.
```

RESULT:

The programs have been completed and the outputs have been verified.