1

Page No.
Date

Devanshu Surana
PC-23, 1032210755
Panel C, Batch C1

AIES Lab Assignment 2

Aim: Solve Tic-Tac-Toe using Minimax algorithm.

Objective: To study and implement Minimax algorithm for Tic-tac-Toe.

Theory:
Adversarial Search: It is a method applied to a situation where you are planning while another actor prepares against you. It is used in AI to model a competition between two individuals. Adversarial Search is often used in two-person games such as chess, tic-tac-toe, go, etc. In these games, the players can see the moves of opponents.

- Tic-tac-toe simply involves playing the game strategically to ensure a win or draw.
Steps for tic-tac-toe
1. Understand the rules of Tic-tac-toe
2. Focus on making winning moves when possible
3. If winning isn't possible, block your opponent from winning.
4. Prioritize center and corner positions for your moves.
5. If you can't win or block, aim for a draw by preventing your opponent from winning.

6. Keep adapting your strategy based on the gam
progress.

Data structures and other details about Minimax algo. excluding algorithm.

1. Game tree: Represent the game state as a tree structure, where nodes are game positions and edges are legal moves.
2. Nodes: Each node contains current game ~~based~~ board state player's turn and level.
3. Evaluation | Heuristic function: Used to estimate the desirability of a game state if its not terminal state. It assigns numerical value to position.
4. Alpha-beta pruning:
   - α = Maximizing player score
   - β = Minimizing player score
5. Depth limit:
   To prevent the algorithm from exploring the entire tree.

Minimax Algorithm:

```
function Minimax-Decision (state) returns an action
    v ← Max-Value (state)
    return the action in Successors (state) with
    value v.
function Max-Value (state) returns a utility value
    if Terminal-Test (state) then return Utility
                                         (state).
    v ← - ∞
    for a,s in successors (state) do
        v ← Max (v, Min-Value (s))
    return v
```

```
function Min-Value (state) returns a utility value
  If Terminal - Test (state) then return utility (state)
  v ← ∞
    for a,s in successors (state) do
      v ← Min ( v, Max - Value (s))
    return v
```

FAQ's :

1. Compare Informed search and adversarial search.

Ans. Informed Search :

1. Uses heuristic and domain specific knowledge.
2. Aims to find solutions efficiently.
3. Applicable in various problem solving domains.
4. Can guarantee optimality with admissible and consistent heuristic
5. Ex: A* , Greedy, BFS

Adversarial Search:

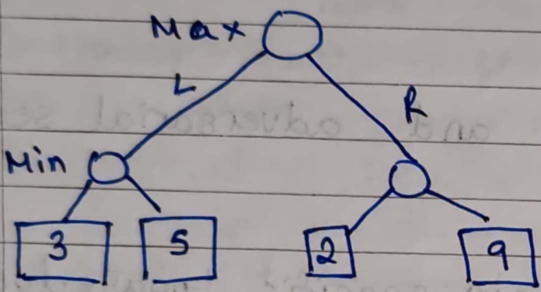
1. Designed for competitive scenarios like games.
2. Considers opponents moves focuses on finding optimal strategies.
3. May not guarantee absolute best outcome due to game complexity.
4. Ex : Minimax with Alpha-Beta pruning, MCTS, etc.

2. Explain Minimax algorithm with an example.

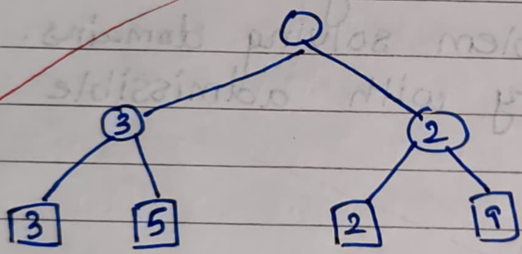
Ans. Minimax is a kind of a backtracking algorithm that is used ~~to~~ in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used

Page No.
Date

in two player turn-based games such as Tic-tac-toe, chess, go, etc.
eg: consider a game which has 4 final states, and paths to reach final state. Assume you are maximizin-
-g player and you get 1st chance to move then which move you would make as a maximizing player.



Max goes left: It is minimizers turn. It will choose 3 (min)
Max goes Right: Again minimizers turn. It will choose 2. After Being Maximizer it will choose 3 from (3, 2)



The tree shows two possible scores when maximizer makes left and right.

3. Explain alpha-Beta pruning.
Ans. It is modified version of Minimax algorithm
Alpha-Beta pruning can be applied at any depth of tree, and it not only prune the tree leaves but also entire sub-tree.
The two parameter can be defined as:
Alpha: (highest value) choise we have found so far at any point along the path of Maximizer. The initial value of alpha is $-\infty$.

Page No.
Date

Beta : (lowest value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.

The Alpha-Beta pruning to a standard minimax algo. returns the same move as the standard algorithm does, but it removes all the nodes, which are not really affecting the final decision but making algorithm slow.

✓

C

[illegible]
31/10/23

AIES_Assignment_2

```
def isMovesLeft(board):
    for i in range(3):
        for j in range(3):patiadkvnaslvnakvsc
            if board[i][j] == '_':
                return True
    return False

def evaluate(b):
    for row in range(3):
        if b[row][0] == b[row][1] == b[row][2]:
            if b[row][0] == player:
                return 10
            elif b[row][0] == opponent:
                return -10
    for col in range(3):
        if b[0][col] == b[1][col] == b[2][col]:
            if b[0][col] == player:
                return 10
            elif b[0][col] == opponent:
                return -10
    if b[0][0] == b[1][1] == b[2][2]:
        if b[0][0] == player:
            return 10
        elif b[0][0] == opponent:
            return -10
    if b[0][2] == b[1][1] == b[2][0]:
        if b[0][2] == player:
            return 10
        elif b[0][2] == opponent:
            return -10
    return 0

def minimax(board, depth, isMax):
    score = evaluate(board)
    if score == 10:
        return score
    if score == -10:
        return score
    if not isMovesLeft(board):
        return 0

    if isMax:
        best = -1000
        for i in range(3):
            for j in range(3):
                if board[i][j] == '_':
                    board[i][j] = player
                    best = max(best, minimax(board, depth + 1, not
isMax))
                    board[i][j] = '_'
        return best
    else:
        best = 1000
        for i in range(3):
            for j in range(3):
                if board[i][j] == '_':
                    board[i][j] = opponent
                    best = min(best, minimax(board, depth + 1, not 
isMax))
                    board[i][j] = '_'
        return best

def findBestMove(board):
    bestVal = -1000
    bestMove = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == '_':
                board[i][j] = player
                moveVal = minimax(board, 0, False)
                board[i][j] = '_'
                if moveVal > bestVal:
                    bestMove = (i, j)
                    bestVal = moveVal
    return bestMove

player, opponent = 'x', 'o'

board = [
    ['x', 'o', 'o'],
    ['x', 'x', 'o'],
    ['_', '_', '_']
]
bestMove = findBestMove(board)
print("The Optimal Move is :")
print("ROW:", bestMove[0], " COL:", bestMove[1])
```

```
The Optimal Move is :
ROW: 2  COL: 0
```