Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2025-26

Class:	**TE**	**Semester:**	**V**
Course Code:	**CSL502**	**Course Name:**	**Artificial Intelligence Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**51**
Experiment No.:	**6**
Title of the Experiment:	**Implementation of Adversarial Search using mini-max algorithm.**
Date of Performance:	**12/08/2025**
Date of Submission:	**09/09/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mrs. Rujuta Vartak

Signature :

Date:

Aim: Implementation of Adversarial Search using mini-max algorithm.

Objective: To study the mini-max algorithm and its implementation for problem solving.

Theory:

Adversarial Search

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

There might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.

The environment with more than one agent is termed as multi-agent environment, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.

So, Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.

Mini-Max Algorithm in Artificial Intelligence

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Pseudo-code for MinMax Algorithm:

```
1. function minimax(node, depth, maximizingPlayer) is
2. if depth ==0 or node is a terminal node then
3.      return static evaluation
of node 4.
```

```
if MaximizingPlayer then    // for Maximizer Player
maxEva= -infinity
 for each child of node do
 eva= minimax(child, depth-1, false)
maxEva= max(maxEva,eva)    //gives Maximum of the values
    return
maxEva
else                    // for Minimizer player
 minEva= +infinity
 for each child of node do
 eva= minimax(child, depth-1, true)
 minEva= min(minEva, eva)    //gives minimum of the values
 return minEva
```

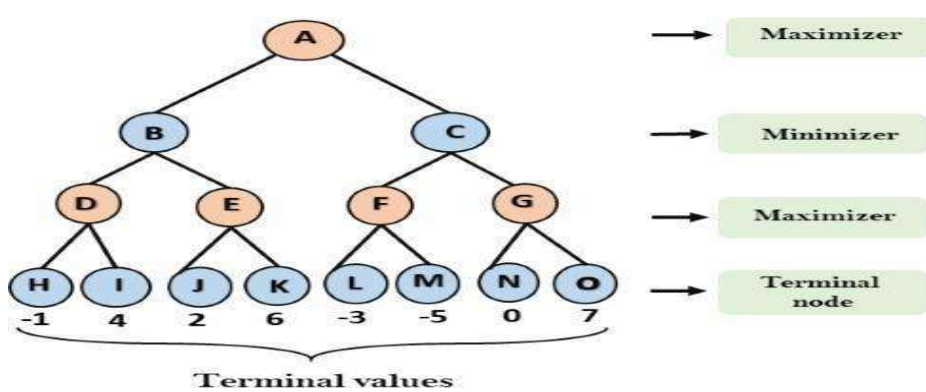
Initial call:

Minimax(node, 3, true)

Working of Min-Max Algorithm:

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two- player game tree:

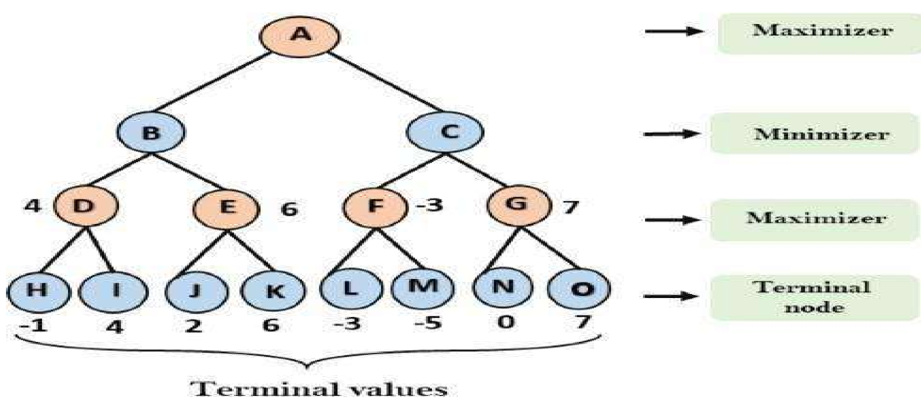
Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =-infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial

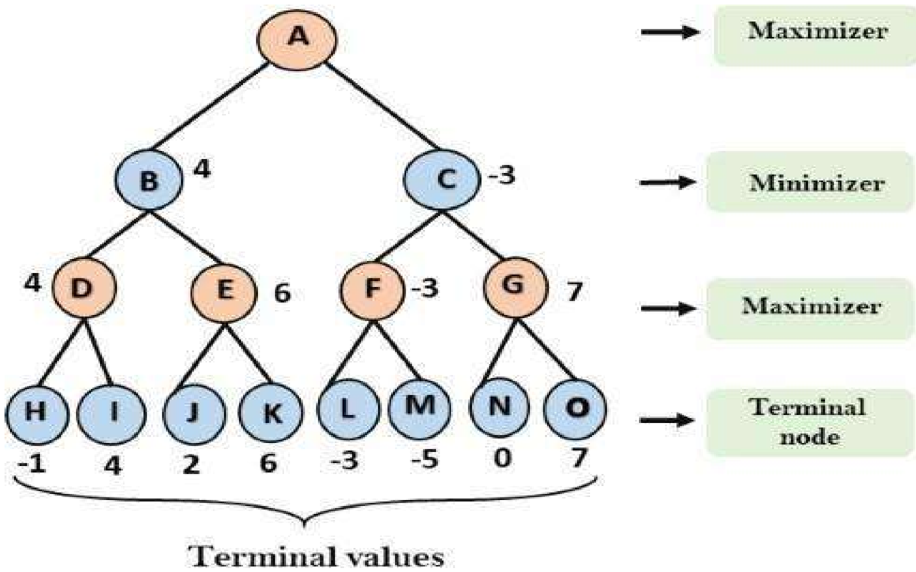
value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



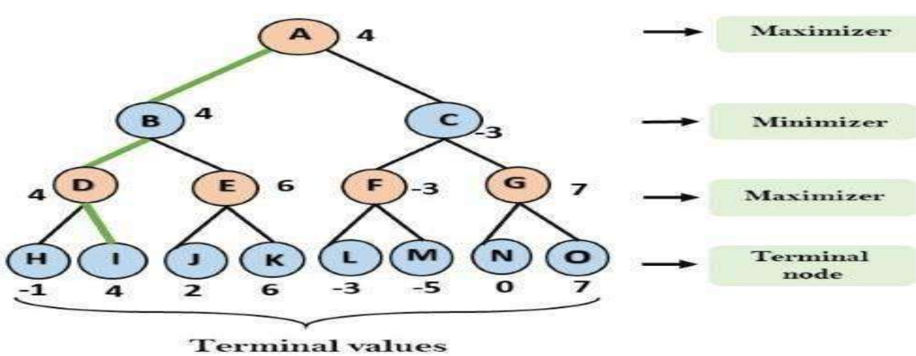
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **O(b^m)**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.

- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is **O(bm)**.

PROGRAM-

```
import math

# Define the tree (MAX and MIN nodes)
tree = {
    "A": ["B", "C", "D"],  # Root MAX
    "B": ["E", "F", "G"],  # MIN
    "C": ["H", "I", "J"],  # MIN
    "D": ["K", "L", "M"]   # MIN
}

# Terminal node values
values = {
    "E": 3, "F": 5, "G": 2,
    "H": 9, "I": 1, "J": 6,
    "K": 0, "L": 4, "M": 8
}

# Minimax function
def minimax(node, is_max):
    if node in values:  # Terminal node
        return values[node], [f"{node}={values[node]}"]

    if is_max:  # Maximizer
        best_val = -math.inf
        best_path = []
        for child in tree[node]:
            val, path = minimax(child, False)
            if val > best_val:
                best_val, best_path = val, [node] + path
        return best_val, best_path
    else:  # Minimizer
        best_val = math.inf
        best_path = []
        for child in tree[node]:
            val, path = minimax(child, True)
            if val < best_val:
                best_val, best_path = val, [node] + path
        return best_val, best_path

# Run Minimax from root A (MAX)
optimal_value, optimal_path = minimax("A", True)
```

```
print("Optimal Value:", optimal_value)
print("Optimal Path:", " -> ".join(optimal_path))
```

Output

```
Output

Optimal Value: 2
Optimal Path: A -> B -> G=2

=== Code Execution Successful ===
```

Conclusion:

The Minimax algorithm effectively provides a strategy for making optimal decisions in two-player, zero-sum games. By recursively evaluating potential future game states, the algorithm determines the best possible move for the maximizing player while minimizing the potential loss against an optimal opponent. The implementation showcases its ability to navigate through a binary tree of possible moves, returning the highest score achievable for the maximizing player at the root node. This foundational approach can be easily adapted to more complex games, enabling strategic decision-making in various competitive scenarios. Overall, the Minimax algorithm is a powerful tool in game theory, demonstrating its significance in areas such as artificial intelligence and game development.