TASK:4

Implementation of Mini-Max algorithm using recursion to search through the Game tree

Implementation of Mini-Max algorithm uses recursion to search through the game-tree using python by applying following constraints.

•In this algorithm two players play the checker's 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.

Tools : Python

PROBLEM STATEMENT: CO2 S3

Develop an AI opponent for a turn-based strategy game called "Battle Stones." In this simplified two-player game, players take turns selecting stones from a shared pool, where each stone has a hidden score value—some positive, some negative. The goal for the AI player (MAX) is to maximize its final score, while the human opponent (MIN) tries to minimize the AI's advantage. The game is represented as a binary tree with a depth of 3, where each leaf node corresponds to a possible outcome score. To make intelligent decisions, the student implements the Minimax algorithm in Python

MINI-MAX ALGORITHM

AIM

To implement the Mini-Max algorithm using recursion to search through the Game - tree using python

ALGORITHM

1. Start with the root of the game tree at depth 0.
2. Check if the current node is a leaf node (i.e., depth equals the maximum depth of the tree):
   - If yes, return the score associated with that node.
3. If it's the MAX player's turn (AI):
   - Recursively call the minimax function for the left child and right child of the current node.
   - Return the maximum of the two values.
4. If it's the MIN player's turn (opponent):
   - Recursively call the minimax function for the left child and right child of the current node.
   - Return the minimum of the two values.
5. Repeat this process recursively until the root node receives its final evaluated value.
6. Return the final value at the root, which is the best guaranteed outcome for the MAX player.

PROGRAM

Minimax in Battle Stones

```
# Minimax algorithm for a simple turn-based game: Battle Stones

def minimax(depth, node_index, is_maximizing, scores, max_depth):
    # Base case: reached a leaf node
    if depth == max_depth:
        return scores[node_index]
```

```
    # If it's MAX's turn
    if is_maximizing:
        return max(
            minimax(depth + 1, node_index * 2, False, scores, max_depth),
            minimax(depth + 1, node_index * 2 + 1, False, scores, max_depth)
        )
    # If it's MIN's turn
    else:
        return min(
            minimax(depth + 1, node_index * 2, True, scores, max_depth),
            minimax(depth + 1, node_index * 2 + 1, True, scores, max_depth)
        )

# Example leaf scores (could represent the result of different sequences of stone choices)
scores = [4, 6, -3, 5, 2, -1, 0, 7]  # 8 leaf nodes (2^3 for depth 3)

# Tree depth
max_depth = 3  # Levels: Root -> Level 1 -> Level 2 -> Leaf

# Starting the game at root node (index 0) with MAX player's turn
optimal_value = minimax(0, 0, True, scores, max_depth)

print("Optimal value for MAX (AI) player:", optimal_value)
```

OUTPUT

```
========== RESTART: C:/Users/gvlr4/OneDrive/Documents/vtu26455 task4 output.py ==========
Optimal value for MAX (AI) player: 5
```

RESULT

Thus the implementation of **Mini-Max algorithm** using recursion to search through the Game tree using python was successfully executed and output was verified.