Ex. No: 2	Min Max Algorithm

Aim: To find the best possible outcome in a game tree by implementing the Minimax algorithm. In this approach, two players take turns, with one aiming to maximize the score and the other aiming to minimize it.

Algorithm:

1. **Evaluate Leaf Nodes:** If the depth of the current node equals maxDepth, it is a leaf node. Return the node's value as no further moves are possible at this level.
2. **Logic for the Maximizing Player:** If it's the maximizing player's turn: Set an initial value for best as negative infinity. Recursively evaluate both child nodes of the current node. Update best by selecting the maximum value between best and the returned value from each child node.
3. **Logic for the Minimizing Player:** If it's the minimizing player's turn: Set an initial value for best as positive infinity. Recursively evaluate both child nodes of the current node. Update best by choosing the minimum value between best and the returned value from each child node.
4. **Return the Optimal Score**: After recursively evaluating all nodes, return the best value calculated at each step, representing the optimal choice.
5. **Resulting Optimal Value:** The value at the root node after all calculations represents the optimal move for the maximizing player, based on the entire game tree.

Code:

```
import math

def minimax(depth, nodeIndex, isMaximizingPlayer, values, maxDepth):
  # Base case: Leaf node reached
  if depth == maxDepth:
    print(f"Leaf node reached at depth {depth}, returning value: {values[nodeIndex]}")
    return values[nodeIndex]

  # Maximizing player's move
  if isMaximizingPlayer:
    best = -math.inf
    print(f"Maximizer at depth {depth}")

    # Evaluate both child nodes
    for i in range(2):
      value = minimax(depth + 1, nodeIndex * 2 + i, False, values, maxDepth)
      print(f"Maximizer at depth {depth}, comparing value: {value} with best: {best}")
      best = max(best, value)

    print(f"Maximizer at depth {depth}, selected best: {best}")
    return best
  # Minimizing player's move
  else:
    best = math.inf
    print(f"Minimizer at depth {depth}")

    # Evaluate both child nodes
    for i in range(2):
      value = minimax(depth + 1, nodeIndex * 2 + i, True, values, maxDepth)
      print(f"Minimizer at depth {depth}, comparing value: {value} with best: {best}")
      best = min(best, value)
```

```
        print(f"Minimizer at depth {depth}, selected best: {best}")
        return best

# The depth of the game tree
maxDepth = 3
# Leaf node values
values = [-1, 4, 2, 6, -3, -5, 0, 7]
optimalValue = minimax(0, 0, True, values, maxDepth)
print("\nThe optimal value is:", optimalValue)
```

Output:

```
Maximizer at depth 0
Minimizer at depth 1
Maximizer at depth 2
Leaf node reached at depth 3, returning value: -1
Maximizer at depth 2, comparing value: -1 with best: -inf
Leaf node reached at depth 3, returning value: 4
Maximizer at depth 2, comparing value: 4 with best: -1
Maximizer at depth 2, selected best: 4
Minimizer at depth 1, comparing value: 4 with best: inf
Maximizer at depth 2
Leaf node reached at depth 3, returning value: 2
Maximizer at depth 2, comparing value: 2 with best: -inf
Leaf node reached at depth 3, returning value: 6
Maximizer at depth 2, comparing value: 6 with best: 2
Maximizer at depth 2, selected best: 6
Minimizer at depth 1, comparing value: 6 with best: 4
Minimizer at depth 1, selected best: 4
Maximizer at depth 0, comparing value: 4 with best: -inf
Minimizer at depth 1
Maximizer at depth 2
Leaf node reached at depth 3, returning value: -3
Maximizer at depth 2, comparing value: -3 with best: -inf
Leaf node reached at depth 3, returning value: -5
Maximizer at depth 2, comparing value: -5 with best: -3
Maximizer at depth 2, selected best: -3
Minimizer at depth 1, comparing value: -3 with best: inf
Maximizer at depth 2
Leaf node reached at depth 3, returning value: 0
Maximizer at depth 2, comparing value: 0 with best: -inf
Leaf node reached at depth 3, returning value: 7
Maximizer at depth 2, comparing value: 7 with best: 0
Maximizer at depth 2, selected best: 7
Minimizer at depth 1, comparing value: 7 with best: -3
Minimizer at depth 1, selected best: -3
Maximizer at depth 0, comparing value: -3 with best: 4
Maximizer at depth 0, selected best: 4

The optimal value is: 4
```

Result:

Therefore the given algorithm has been successfully coded and verified.