

**BS Artificial Intelligence**

**Name:** Hafiz M. Muneeb Akbar

**Roll No:** SU92-BSAIM-F24-048

**Subject:** Artificial Intelligence (Lab)

**Lab Task 6**

**Lab Task — Step-by-Step Explanation of Code (Minimax Algorithm)**

This lab task implements the Minimax Algorithm, which is a decision-making algorithm used in Artificial Intelligence, especially for games such as Chess, Tic-Tac-Toe, and Checkers. It explores all possible moves to determine the optimal one for a player assuming the opponent also plays optimally.

**Function Definition**

The function named **minimax(curDepth, nodeIndex, maxTurn, scores, targetDepth)** is defined to recursively evaluate the game tree. It takes five parameters:

**- curDepth:** The current level **(depth)** of the tree.  
**- nodeIndex:** Index of the current node in the scores list.  
**- maxTurn:** A boolean flag indicating whether it is the maximizing player's turn.  
**- scores:** A list of terminal node values representing possible outcomes.  
**- targetDepth:** The maximum depth of the tree.

**Base Case Check**

If the current depth equals the target depth, the recursion stops, and the score value at the corresponding node is returned. This represents reaching a leaf node in the game tree.

**Recursive Evaluation**

If it is the maximizing player's turn **(maxTurn=True),** the function returns the maximum value between the two child nodes, simulating the maximizing player choosing the best possible move.  
  
If it is the minimizing player's turn **(maxTurn=False),** the function returns the minimum of the two child node values, simulating the opponent trying to minimize the maximizing player's outcome.

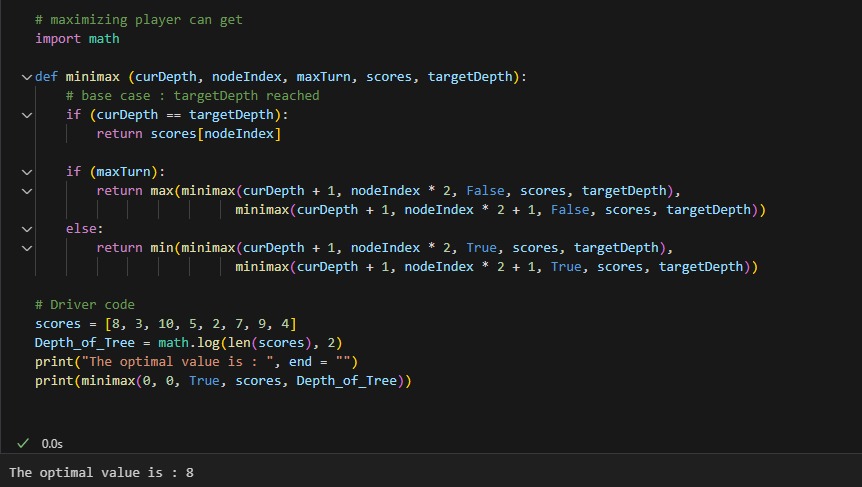
**Driver Code**

The scores list represents the final outcomes or payoffs at the leaf nodes of the game tree:  
**scores = [8, 3, 10, 5, 2, 7, 9, 4]**  
  
The depth of the tree is calculated using the logarithm **(base 2)** of the number of scores, since the tree is assumed to be a complete binary tree.  
  
Finally, the minimax function is called as: **minimax(0, 0, True, scores, Depth\_of\_Tree)**and the optimal value (best achievable outcome for the maximizing player) is printed.

**Example Execution**

When executed, the function explores all possible game states recursively. For each level of the tree, the algorithm alternates between maximizing and minimizing decisions. The final output displays the optimal value that the maximizing player can achieve given both players play optimally.  
  
**Output:** The optimal value is **: 7**

**Output Screenshot**



**Conclusion**

This implementation demonstrates how recursive algorithms like Minimax can simulate intelligent decision-making in games by exploring every possible outcome. The key takeaway is understanding how recursion and alternating decision layers work together to produce an optimal strategy.