

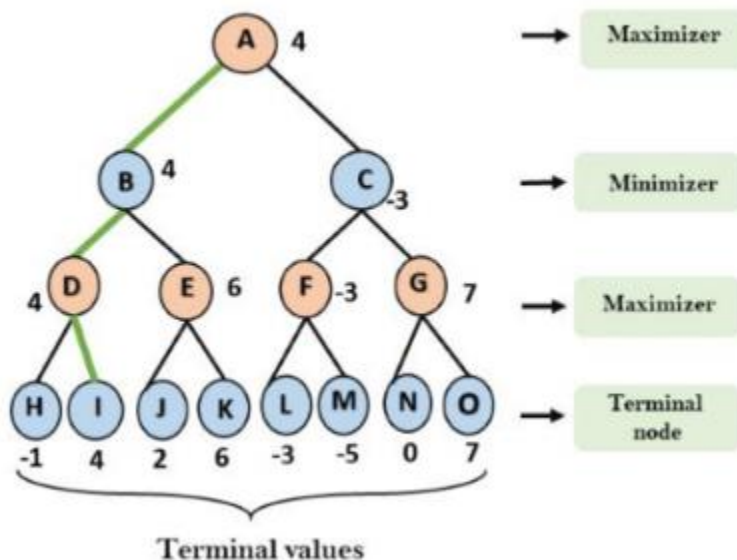
EX.NO:4

DATE:4/9/2024

Reg.no:220701026

### MINIMAX ALGORITHM

- A simple example can be used to explain how the minimax algorithm works. We've included an example of a game-tree below, which represents a two-player game.
- There are two players in this scenario, one named Maximizer and the other named Minimizer.
- Maximizer will strive for the highest possible score, while Minimizer will strive for the lowest possible score.
- Because this algorithm uses DFS, we must go all the way through the leaves to reach the terminal nodes in this game-tree.
- The terminal values are given at the terminal node, so we'll compare them and retrace the tree till we reach the original state.



## CODE:

```
def minimax(depth, nodeIndex, isMaximizingPlayer, scores, targetDepth):

    if depth == targetDepth:
        return scores[nodeIndex]

    if isMaximizingPlayer:
        return max(minimax(depth + 1, nodeIndex * 2, False, scores,
targetDepth),
                    minimax(depth + 1, nodeIndex * 2 + 1, False, scores,
targetDepth))
    else:

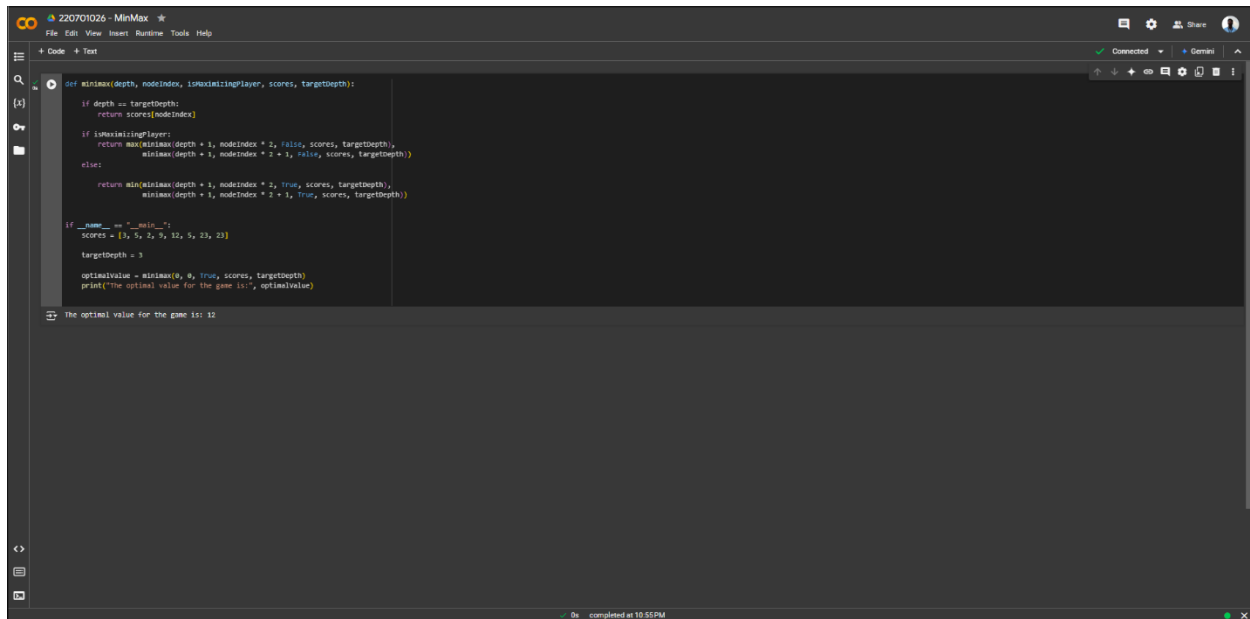
        return min(minimax(depth + 1, nodeIndex * 2, True, scores,
targetDepth),
                    minimax(depth + 1, nodeIndex * 2 + 1, True, scores,
targetDepth))

if __name__ == "__main__":
    scores = [3, 5, 2, 9, 12, 5, 23, 23]

    targetDepth = 3

    optimalValue = minimax(0, 0, True, scores, targetDepth)
    print("The optimal value for the game is:", optimalValue)
```

## OUTPUT:



The screenshot shows a code editor window titled "220701026 - MinMax". The code implements a MinMax algorithm for a game. The function `minimax` takes parameters `depth`, `nodeIndex`, `isMaximizingPlayer`, `scores`, and `targetDepth`. It recursively evaluates the game state, alternating between maximizing and minimizing players. The base case is reached when `depth == targetDepth`, where the score is returned. The recursive calls alternate between `max` and `min` based on the `isMaximizingPlayer` flag. The code also includes a `__name__ == "__main__":` block that initializes the `scores` array to `[1, 7, 5, 7, 12, 5, 21, 23]`, sets `targetDepth = 3`, and calls `minimax(0, 0, True, scores, targetDepth)` to find the optimal value. The output of the program is displayed in the console: "The optimal value for the game is: 12".

```
def minimax(depth, nodeIndex, isMaximizingPlayer, scores, targetDepth):  
    if depth == targetDepth:  
        return scores[nodeIndex]  
  
    if isMaximizingPlayer:  
        return max(minimax(depth + 1, nodeIndex + 1, False, scores, targetDepth),  
                    minimax(depth + 1, nodeIndex + 2 + 1, False, scores, targetDepth))  
    else:  
        return min(minimax(depth + 1, nodeIndex + 1, True, scores, targetDepth),  
                    minimax(depth + 1, nodeIndex + 2 + 1, True, scores, targetDepth))  
  
if __name__ == "__main__":  
    scores = [1, 7, 5, 7, 12, 5, 21, 23]  
    targetDepth = 3  
    optimalValue = minimax(0, 0, True, scores, targetDepth)  
    print("The optimal value for the game is:", optimalValue)  
  
The optimal value for the game is: 12
```

completed at 10:55 PM