Ai Applications & Ethics (TC-7)

LAB 6

Name: - Hitesh Choudhary

PRN: - 20200802146

DS1

> **Aim: -** Program to implement, game playing: Minimax, alpha-beta pruning.
>
> **Software Required:** Google Colab.

> **Algorithm:**

1. **Define the Game Tree:**

- Represent the game state as a tree where each node represents a possible game state.
- Nodes at even depths (0, 2, 4, etc.) represent the maximizing player's turn.
- Nodes at odd depths (1, 3, 5, etc.) represent the minimizing player's turn.

2. **Evaluate Terminal Nodes:**

- Assign values to terminal nodes based on the game outcome. For example, +1 for a win, -1 for a loss, and 0 for a draw.

3. **Recursively Evaluate Non-terminal Nodes:**

- Use a recursive approach to evaluate non-terminal nodes by considering the values of their children.
- If it's the turn of the maximizing player, choose the child with the maximum value.
- If it's the turn of the minimizing player, choose the child with the minimum value.

4. **Return the Best Move or Value:**

- At the root of the tree (initial game state), choose the move or value that leads to the best outcome for the maximizing player.

> **Pseudo Code:**

```
plaintext
function minimax(node, depth, maximizingPlayer):
if depth is 0 or node is a terminal node:
return the heuristic value of node
if maximizingPlayer:
```

```
maxEval = -∞
for each child of node:
eval = minimax(child, depth - 1, False)
maxEval = max(maxEval, eval)
return maxEval
else: # minimizing player
minEval = +∞
for each child of node:
eval = minimax(child, depth - 1, True)
minEval = min(minEval, eval)
return minEval
// Initial call
minimax(initialNode, depth, True)
```

```
import math
class Node:
 def __init__(self, value, children=None):
  self.value = value
  self.children = children if children else []
def min_max(node, depth, is_maximizing):
  if depth == 0 or not node.children:
    return node.value

  if is_maximizing:
    max_eval = -math.inf
    for child in node.children:
      eval = min_max(child, depth - 1, False)
      max_eval = max(max_eval, eval)
    return max_eval

  else:
    min_eval = math.inf
    for child in node.children:
      eval = min_max(child, depth - 1, True)
      min_eval = min(min_eval, eval)
    return min_eval

def build_custom_graph():
 # Define the custom game graph with provided values
 node_g = Node(-1)
 node_f = Node(8)
 node_e = Node(-3)
 node_d = Node(-1, [node_g])
 node_c = Node(2)
 node_b = Node(1, [node_e, node_f])
 root_node = Node(4, [node_b, node_c, node_d])

 return root_node

root_node = build_custom_graph()
depth = 4
result = min_max(root_node, depth, True)
print(f"Best value at root node is: {result}")
```

```
print("Best value at root node is:", result)
```

```
Best value at root node is: 2
```

Relative Application: -

- **Chess, Checkers, and Go:** Minimax is widely used to create AI opponents in board games. It helps the computer player decide the best move by considering possible future moves and outcomes.
- **Card Games like Poker:**Minimax can be applied to poker games to create strategic AI players that evaluate different possible actions and select the one with the highest expected utility.
- **Network Routing:** Minimax can be applied to optimize network routing, where the goal is to find the path that minimizes potential delays or maximizes data throughput.
- **Adversarial Scenarios:** In the field of cybersecurity, Minimax can be used to model the decision-making process of both attackers and defenders in an adversarial environment.
- **General Game Playing:** Minimax is a fundamental concept in game theory and general game playing, where the objective is to create agents that can play a wide variety of games effectively

Conclusion:

We have successfully implemented game playing: Minimax, alpha-beta pruning using python.