10 . Alpha-Beta Pruning

```
#alpha beta pruning
class Node:
    def __init__(self, value=None, children=None):
        self.value = value
        self.children = children or []

def alpha_beta(node, depth, alpha, beta, maximizing_player):
    if depth == 0 or not node.children:
        return node.value
    if maximizing_player:
        max_eval = -float('inf')
        for child in node.children:
            eval = alpha_beta(child, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            print(f"Maximizing: alpha={alpha}, beta={beta}")
            if beta <= alpha:
                print(f"Pruning at node with value {node.value}")
                break
        return max_eval
    else:
        min_eval = float('inf')
        for child in node.children:
            eval = alpha_beta(child, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            print(f"Minimizing: alpha={alpha}, beta={beta}")
            if beta <= alpha:
                print(f"Pruning at node with value {node.value}")
                break
        return min_eval
root = Node(children=[
    Node(children=[Node(value=10), Node(value=9)]),
    Node(children=[Node(value=14), Node(value=18)]),
    Node(children=[Node(value=5), Node(value=4)]),
    Node(children=[Node(value=50), Node(value=3)])
])
final_value = alpha_beta(root, 3, -float('inf'), float('inf'), True)
print(f"Final Value at MAX node: {final_value}")
```

Output:

```
Minimizing: alpha=-inf, beta=10
Minimizing: alpha=-inf, beta=9
Maximizing: alpha=9, beta=inf
Minimizing: alpha=9, beta=14
Minimizing: alpha=9, beta=14
Maximizing: alpha=14, beta=inf
Minimizing: alpha=14, beta=5
Pruning at node with value None
Maximizing: alpha=14, beta=inf
Minimizing: alpha=14, beta=50
Minimizing: alpha=14, beta=3
Pruning at node with value None
Maximizing: alpha=14, beta=inf
Final Value at MAX node: 14
```

17/12/24 Tuesday

Lab - 10

Analyse Alpha-Beta pruning method and implement the same to write alpha-beta value generated to identify final value of MAX node and subtrees pruned for a given game tree.

Algorithm:

```
def alpha_beta(node, depth, alpha, beta,
        maximizing_player):
    if depth == 0 or is_terminal(node):
        return Evaluate(node)
    if maximizing_player:
        max_eval = -float("inf")
        for child in node.children:
            eval = alpha_beta(child, depth-1, alpha,
                beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = float("inf")
        for child in node.children:
            eval = alpha_beta(child, depth-1, alpha,
                beta, True)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval
```

Output:

minimizing : alpha = -inf, beta = 10
minimizing : alpha = -inf, beta = 9
maximizing : alpha = 9, beta = inf
minimizing = alpha = 9, beta = 14
minimizing = alpha = 9, beta = 14
maximizing : alpha = 14, beta = inf
minimizing : alpha = 14, beta = 5
pruning at node with value none
maximizing : alpha = 14, beta = inf
minimizing : alpha = 14, beta = 50
minimizing : alpha = 14, beta = 5
pruning at node with value none
maximizing : alpha = 14, beta = inf
Final value at MAX node : 14