**SOFTWARE LABORATORY 1**

**GROUP B – EXPERIMENT 3**

**TITLE:**

Implement Alpha-Beta Tree search for any game search problem.

**CODE:**

class Node:

def \_\_init\_\_(self, name, value=None, children=None):

self.name = name # Node identifier

self.value = value # Evaluation score if leaf node

self.children = children or [] # List of child nodes

# Alpha-Beta pruning algorithm

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)

if beta <= alpha:

# Prune the remaining branches

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)

if beta <= alpha:

# Prune the remaining branches

break

return min\_eval

# Example game tree

E = Node("E", 3)

F = Node("F", 5)

G = Node("G", 6)

H = Node("H", 9)

I = Node("I", 1)

B = Node("B", children=[E, F])

C = Node("C", children=[G, H])

D = Node("D", children=[I])

A = Node("A", children=[B, C, D])

# Run Alpha-Beta Pruning on this tree

best\_value = alpha\_beta(A, depth=3, alpha=float('-inf'), beta=float('inf'), maximizing\_player=True)

print(f"Best evaluated value for the root node A: {best\_value}")

**Output:**

[Running] python -u "e:\SL1\slb3.py"

Best evaluated value for the root node A: 6

[Done] exited with code=0 in 0.095 seconds

