**ANS TO THE QUESTION NO : 1**

class TREE:

def \_\_init\_\_(self):

self.tree = {}

self.root = None

def addNode(self, node, parent, value=None):

if not self.tree:

self.root = node

self.tree[node] = [parent, value, None, None]

elif parent in self.tree:

if self.tree[parent][2] is None:

self.tree[parent][2] = node

else:

self.tree[parent][3] = node

self.tree[node] = [parent, value, None, None]

def getValue(self, node):

return self.tree[node][1] if node in self.tree else None

def getChildren(self, node):

return [child for child in self.tree[node][2:] if child is not None]

def getTree(self):

return self.tree

def alpha\_beta\_pruning(tree, node, alpha, beta, depth, maximizePlayer):

if depth == 0:

return tree.getValue(node)

if maximizePlayer:

value = float('-inf')

for child in tree.getChildren(node):

value = max(value, alpha\_beta\_pruning(tree, child, alpha, beta, depth - 1, False))

alpha = max(alpha, value)

if alpha >= beta:

break

return value

else:

value = float('inf')

for child in tree.getChildren(node):

value = min(value, alpha\_beta\_pruning(tree, child, alpha, beta, depth - 1, True))

beta = min(beta, value)

if beta <= alpha:

break

return value

def create\_game\_tree():

tree = TREE()

tree.addNode("A", None)

tree.addNode("B", "A")

tree.addNode("C", "A")

tree.addNode("D", "B")

tree.addNode("E", "B")

tree.addNode("F", "C")

tree.addNode("G", "C")

tree.addNode("H", "D", -1)

tree.addNode("I", "D", 1)

tree.addNode("J", "E", -1)

tree.addNode("K", "E", 1)

tree.addNode("L", "F", -1)

tree.addNode("M", "F", 1)

tree.addNode("N", "G", -1)

tree.addNode("O", "G", 1)

return tree

def main():

starter = int(input("Enter 0 for Scorpion or 1 for Sub-Zero to start: "))

rounds = 3

players = ["Scorpion", "Sub-Zero"]

tree = create\_game\_tree()

result = alpha\_beta\_pruning(tree, "A", alpha=float('-inf'), beta=float('inf'), depth=3, maximizePlayer=(starter == 1))

game\_winner = "Sub-Zero" if result == 1 else "Scorpion"

round\_winners = []

current\_player = starter

for i in range(1, rounds + 1):

round\_result = alpha\_beta\_pruning(tree, "A", alpha=float('-inf'), beta=float('inf'), depth=3, maximizePlayer=(current\_player == 1))

if round\_result == 1:

round\_winners.append("Sub-Zero")

else:

round\_winners.append("Scorpion")

current\_player = 1 - current\_player

print(f"Game Winner: {game\_winner}")

print(f"Total Rounds Played: {rounds}")

for i, winner in enumerate(round\_winners, start=1):

print(f"Winner of Round {i}: {winner}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**ANS TO THE QUESTION NO 2**

class GameTree:

def \_\_init\_\_(self):

self.nodes = {}

self.root\_node = None

def add\_node(self, node\_id, parent\_id=None, value=None):

if not self.nodes:

self.root\_node = node\_id

self.nodes[node\_id] = [parent\_id, value, None, None]

elif parent\_id in self.nodes:

if self.nodes[parent\_id][2] is None:

self.nodes[parent\_id][2] = node\_id

else:

self.nodes[parent\_id][3] = node\_id

self.nodes[node\_id] = [parent\_id, value, None, None]

def get\_value(self, node\_id):

return self.nodes[node\_id][1] if node\_id in self.nodes else None

def get\_children(self, node\_id):

return [child for child in self.nodes[node\_id][2:] if child is not None]

def display\_tree(self):

return self.nodes

def ab\_pruning(tree, current\_node, alpha, beta, depth\_level, is\_maximizing\_player):

if depth\_level == 0:

return tree.get\_value(current\_node)

if is\_maximizing\_player:

max\_eval = float('-inf')

for child\_node in tree.get\_children(current\_node):

max\_eval = max(max\_eval, ab\_pruning(tree, child\_node, alpha, beta, depth\_level - 1, False))

alpha = max(alpha, max\_eval)

if alpha >= beta:

break

return max\_eval

else:

min\_eval = float('inf')

for child\_node in tree.get\_children(current\_node):

min\_eval = min(min\_eval, ab\_pruning(tree, child\_node, alpha, beta, depth\_level - 1, True))

beta = min(beta, min\_eval)

if beta <= alpha:

break

return min\_eval

def pacman\_simulation(magic\_cost):

game\_tree = GameTree()

game\_tree.add\_node("Root", None)

game\_tree.add\_node("Node1", "Root")

game\_tree.add\_node("Node2", "Root")

game\_tree.add\_node("Node3", "Node1")

game\_tree.add\_node("Node4", "Node1")

game\_tree.add\_node("Node5", "Node2")

game\_tree.add\_node("Node6", "Node2")

game\_tree.add\_node("Leaf1", "Node3", 3)

game\_tree.add\_node("Leaf2", "Node3", 6)

game\_tree.add\_node("Leaf3", "Node4", 2)

game\_tree.add\_node("Leaf4", "Node4", 3)

game\_tree.add\_node("Leaf5", "Node5", 7)

game\_tree.add\_node("Leaf6", "Node5", 1)

game\_tree.add\_node("Leaf7", "Node6", 2)

game\_tree.add\_node("Leaf8", "Node6", 0)

minimax\_without\_magic = ab\_pruning(game\_tree, "Root", alpha=float('-inf'), beta=float('inf'), depth\_level=3, is\_maximizing\_player=True)

max\_left\_subtree = max(game\_tree.get\_value("Leaf1"), game\_tree.get\_value("Leaf2"), game\_tree.get\_value("Leaf3"), game\_tree.get\_value("Leaf4"))

max\_right\_subtree = max(game\_tree.get\_value("Leaf5"), game\_tree.get\_value("Leaf6"), game\_tree.get\_value("Leaf7"), game\_tree.get\_value("Leaf8"))

score\_left\_magic = max\_left\_subtree - magic\_cost

score\_right\_magic = max\_right\_subtree - magic\_cost

if score\_left\_magic > score\_right\_magic:

best\_magic\_score = score\_left\_magic

direction\_choice = "left"

else:

best\_magic\_score = score\_right\_magic

direction\_choice = "right"

if minimax\_without\_magic < best\_magic\_score:

print(f"The new minimax value is {best\_magic\_score}. Pacman goes {direction\_choice} and uses dark magic")

else:

print(f"The minimax value is {minimax\_without\_magic}. Pacman does not use dark magic")

if \_\_name\_\_ == "\_\_main\_\_":

magic\_cost\_input = int(input("Enter the cost of dark magic: "))

pacman\_simulation(magic\_cost\_input)