# Alpha-Beta Pruning Algorithm

class Node:

def \_\_init\_\_(self, state, parent=None):

self.state = state

self.parent = parent

self.children = []

def alpha\_beta(node, depth, alpha, beta, maximizing\_player):

if depth == 0 or not node.children:

return evaluate(node.state)

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)

min\_eval = min(min\_eval, eval)

beta = min(beta, eval)

if beta <= alpha:

break

return min\_eval

def evaluate(state):

# Placeholder for evaluation function

return state

# Example usage

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

root = Node(state=0)

# Populate the tree with nodes and states

# root.children.append(Node(state=1, parent=root))

# root.children.append(Node(state=2, parent=root))

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

print("Best value:", best\_value)

OUTPUT:

