import math

# Define a simple game tree with alpha-beta pruning

def alpha\_beta(node, depth, alpha, beta, is\_maximizing):

if not node['children']:

return node['value']

if is\_maximizing:

best\_score = -math.inf

for child in node['children']:

score = alpha\_beta(child, depth + 1, alpha, beta, False)

best\_score = max(score, best\_score)

alpha = max(alpha, best\_score)

if beta <= alpha:

break # Beta cutoff

return best\_score

else:

best\_score = math.inf

for child in node['children']:

score = alpha\_beta(child, depth + 1, alpha, beta, True)

best\_score = min(score, best\_score)

beta = min(beta, best\_score)

if beta <= alpha:

break # Alpha cutoff

return best\_score

# Example game tree

game\_tree = {

'is\_maximizing': True, # Maximizing player starts

'value': None, # Not used in non-leaf nodes

'children': [

{'is\_maximizing': False, 'value': None, 'children': [

{'is\_maximizing': True, 'value': 1, 'children': []},

{'is\_maximizing': True, 'value': 2, 'children': []}

]},

{'is\_maximizing': False, 'value': None, 'children': [

{'is\_maximizing': True, 'value': 3, 'children': []},

{'is\_maximizing': True, 'value': 4, 'children': []}

]}

]

}

# Calculate the best score for the root node with alpha-beta pruning

best\_score = alpha\_beta(game\_tree, 0, -math.inf, math.inf, True)

print("Best score for the maximizing player with Alpha-Beta Pruning:", best\_score)

output:

