Lab 7 AI

# Name: Hassan Shahzad

# Class: BSCS 7C CMS ID: 211798

## Task

## Code:

!pip install --upgrade binarytree

from binarytree import Node, tree, build

from random import seed, random, randint

def minimax(position, depth, maximizingPlayer):

  children = []

  if(position.right == None):

    children.append(position.left)

  elif(position.left == None):

    children.append(position.right)

  elif(position.left == None and position.right == None):

    children.append(None)

  else:

    children.append(position.left)

    children.append(position.right)

  if(depth == 0):

    return  position.value

  if (maximizingPlayer):

    maxEval = float('-inf')

    for child in children:

      if(child == None):

        maxEval = position.value

        break

      eval = minimax(child, depth - 1, False)

      maxEval = max(maxEval, eval)

    return maxEval

  else:

    minEval = float('inf')

    for child in children:

      if(child == None):

        minEval = position.value

        break

      eval = minimax(child, depth - 1, True)

      minEval = min(minEval, eval)

    return minEval

def AlphaBeta(position, depth, alpha, beta, maximizingPlayer):

  children = []

  if(position.right == None):

    children.append(position.left)

  elif(position.left == None):

    children.append(position.right)

  elif(position.left == None and position.right == None):

    children.append(None)

  else:

    children.append(position.left)

    children.append(position.right)

  if(depth == 0):

    return position.value

  if(maximizingPlayer):

    maxEval = float('-inf')

    for child in children:

      if(child == None):

        maxEval = position.value

        break

      eval = AlphaBeta(child, depth - 1, alpha, beta, False)

      maxEval = max(alpha, eval)

      if (beta <= alpha):

        break

    return maxEval

  else:

    minEval = float('inf')

    for child in children:

      if(child == None):

        minEval = position.value

        break

      eval = AlphaBeta(child, depth - 1, alpha, beta, True)

      minEval = min(minEval, eval)

      beta = min(beta, eval)

      if(beta <= alpha):

        break

    return minEval

values = []

for x in range (15):

  values.append(randint(0, 10))

print(values)

tree = build(values)

print(tree)

optimalValue = minimax(tree[0], tree.height, True)

print("Optimal Value (Minimax): ", optimalValue)

optimalValue1 = AlphaBeta(tree[0], tree.height, float('-inf'), float('inf'), True)

print("Optimal Value (Alpha Beta): ", optimalValue1)

## Output:

