import random

import math

count = 0

def minimax(tree, branch, depth, alpha, beta, maximizing\_player):

global count

if depth == 0:

value = tree[count]

count += 1

return value

if maximizing\_player == True:

max\_val = -math.inf

for i in range(branch):

evaluated = minimax(tree, branch, depth-1, alpha, beta, False)

max\_val = max(max\_val, evaluated)

alpha = max(alpha, evaluated)

if alpha >= beta:

break

return max\_val

else:

min\_val = math.inf

for ind in range(branch):

evaluated = minimax(tree, branch, depth-1, alpha, beta, True)

min\_val = min(min\_val, evaluated)

beta = min(beta, evaluated)

if alpha >= beta:

break

return min\_val

def alpha\_beta\_pruning(student\_id, min\_value, max\_value):

global count

depth = 2 \* int(student\_id[0])

hp = int(student\_id[-2:][::-1])

branch = int(student\_id[2])

leafs = random.sample(range(min\_value, max\_value+1), pow(branch, depth))

tree = leafs

print('1. Depth and Branches ratio is '+str(depth)+":"+str(branch))

print('2. Terminal States (leaf node values) are', str(leafs)[1:-1])

damage = minimax(tree, branch, depth, -math.inf, math.inf, True)

print("3. Left life(HP) of the defender after maximum damage caused by the attacker is", hp - damage)

print('4. After Alpha-Beta Pruning Leaf Node Comparisons', count)

student\_ID = input("Enter your student ID:")

min\_value, max\_value = map(int, input(

"Minimum and Maximum value for the range of negative HP:").split())

alpha\_beta\_pruning(student\_ID, min\_value, max\_value)