import math

import random

traced = []

const = math.inf

def mini\_max(pos, flags, deepth, alpha, beta, maximizingPlayer):

if deepth == 0:

flags += 1

return traced[pos], flags

if maximizingPlayer is True:

isMaximizing = -const;

for leaf in range(bulletCost):

i, flagCount = mini\_max((pos \* bulletCost) + leaf,flags, deepth - 1, alpha, beta, False)

isMaximizing = max(isMaximizing, i)

alpha = max(alpha, i)

if alpha >= beta:

flagCount - 1

break

return isMaximizing, flags

if maximizingPlayer is False:

isMinimizing = const;

for leaf in range(bulletCost):

i, flagCount = mini\_max((pos \* bulletCost) + leaf,flags, deepth - 1, alpha, beta, True)

isMinimizing = min(isMinimizing, i)

beta = min(beta, i)

if alpha >= beta:

flags - 1

break

return isMinimizing, flags

inputID = input('Enter ID: ')

minimum, maximum = map(int,input().split(" "))

minimum = int(minimum)

maximum = int(maximum)

iniHP = [inputID[len(inputID) - 1], inputID[len(inputID) - 2]]

iniHP\_trim = int("".join(map(str, iniHP)))

turns = inputID[0]

deepth = 2 \* int(turns)

bulletCost = int(inputID[2])

nodeDist = bulletCost \*\* deepth

for i in range(nodeDist):

traced.append(random.randint(minimum, maximum))

max\_amount, flags = mini\_max(0,0, deepth, -const, const, True)

print('Depth and Branches ratio is ', deepth, ":", bulletCost)

print('Terminal States (leaf node values) are ', \*traced)

print('Left life (HP) of the defender after maximum damage caused by the attacker is ', iniHP\_trim - max\_amount)

print('After Alpha-Beta Pruning Leaf Node Comparisons ', flags)