Programming Assignment: 105 Lab 8

Team Name: delacruz-afra

Member 1: Tazkia Afra

Member 2: Marisleysis De La Cruz (Mari)

Log table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date/Time | Task | Time spent | Driver | Navigator | Notes |
| 11/15/19 10:30 | Worked on strategies for three player game and mutual recursion | 1 hour | Tazkia | Mari | P3won never showed up in any of the graphs, will work on that next time |
| 11/16/19  3-5pm | Coded unique paths | 2 hours | Mari | Tazkia | PrintPaths prints index of the paths rather than the actual number of the block |
| 11/17/19  6-7pm | Coded three player pebble game | 1 hour | Tazkia | Mari | The graphs do not seem very even |
| 11/18  7-8pm | Try test cases | 1 hour | Mari | Tazkia | Test cases for countpaths works. Test cases for printpaths do not pass because it print indexes, not the numbers |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

I understand that pair programming is a collaborative process where both partners work together. Each partner is expected to "drive" roughly 50% and “navigate” 50% of the time the team is working together. At most 25% of an individual’s effort for an assignment should be spent working alone. The object is to work together, learning from each other, not to divide the work into two pieces with each partner working on a different piece. By signing below, we certify that the above log is an accurate reflection of how I spent my time on this project

Member1: Tazkia Afra

Member2: Marisleysis De La Cruz

# YOU MUST MAKE USE OF RECURSION

# Read the TwoPlayerGame.py carefully

# After reading the script you may have figured out the following

# The program simulates a two player game

# The player gets alternate chance to play

# Game gets started by a randomly chosen player

# The strategy the players play the game with is given below:

# ----Player1 always removes two pebbles

# ----Player2 removes two pebbles if an even number of pebbles is on the table, and one otherwise

# A total of five sets of experiments run, during each experiment, 2000 games are played

# What you are supposed to do is the following:

# Keeping everything same, you have to extend the game for three players

# Each player after playing for their turn should handover the turn to the next player

# Mutually recursive calls will be made in the following order Player1->Player2->Player3->Player1

# The game should be started by a randomly chosen player -- already implemented

# The most important thing that you have to make sure that THE GAME IS NOT BIASED

# In an unbiased game, no matter who starts the game with whatever number of pebbles, the number of wins for all

# the players should be ALMOST SAME after playing significant number of games. This is the most important part of this problem

# In short, you have to come up with a STRATEGY (see the strategy for TwoPlayerGame) such that the game remains unbiased.

# It is also important to make sure that the games finishes at some point.

# If you do everything correctly, all five graphs will show similar (may not be the same) heights of bars for all possible cases.

# For TwoPlayerGame its four cases, P1StartsP1Wins, P1StartsP2Wins, P2StartsP1Wins, and P2StartsP2Wins. Do you already know

# how many cases would be there for ThreePlayerGame? Find out and make the changes accrodingly.

# Player1 always removes two pebbles

# Player2 removes two pebbles if an even number of pebbles is on the table, and one otherwise

# Note that the strategy of both players allow the number of pebbles to become zero

# Otherwise, the program will run into infinite loop if the number of pebbles never becomes "zero"

# importing random so we can start with random number of pebbles as well as randomly chose who starts the game

import random

from matplotlib import pyplot as plt

import sys

# Setting the recursion depth to 20000

sys.setrecursionlimit(20000)

# defining how player1 plays the game

def player1(pebble\_count):

# Strategy:

# Always picks 2

if pebble\_count == 0:

return "p3won" # player3 wins

else:

return player3(pebble\_count - 1) # handing over to player2 after picking 2 pebbles

# defining how player2 plays the game

def player2(pebble\_count):

# Strategy:

# pick 2 pebbles if number of pebbles is even otherwise pick 1

if pebble\_count == 0:

return "p1won" # player1 wins

elif pebble\_count % 2 == 0: # Checking for even

return player1(pebble\_count - 1) # handing over to player1 after picking 2 pebbles

else:

return player1(pebble\_count - 2) # # handing over to player1 after picking 1 pebbles

def player3(pebble\_count):

# Strategy:

# Always picks 2

if pebble\_count == 0:

return "p2won" #player 2 wins

elif pebble\_count == 1:

return player2(pebble\_count -1)

else:

return player2(pebble\_count - 2)

# Playing the game for desired number of times

# Also testing whether the game is biased or not

# Game would not be biased if both players win equal number of times regardless who starts with what number of pebbles

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

# Number of experiments

number\_of\_experiments = 5

for experiment\_id in range(number\_of\_experiments):

# Automate the game play

# Initializing some required variables

number\_of\_players = 3

howmanytimes\_per\_player = 1000

minimum = number\_of\_players \* 3

maximum = number\_of\_players \* howmanytimes\_per\_player

# generating a list of initial pebble counts to start with

initial\_pebble\_counts = random.sample(range(minimum, maximum), maximum - minimum)

# print initial\_pebble\_counts or debug to see what are initial pebble counts are

# Creating a blank list, which will keep tuples with elements who starts the game

# with howmany pebbles and who wins the game

results = []

for initial in initial\_pebble\_counts: # Play the game for every randomly generated pebble count

whostarts = random.randint(1, number\_of\_players) # decide who starts the game by generating a random number

if whostarts == 1: # In case player1 starts the game

results.append(('p1started', initial, player1(initial))) # storing who started with how many pebbles along with won the game

elif whostarts == 2: # In case player2 starts the game

results.append(('p2started', initial, player2(initial))) # storing who started with how many pebbles along with won the game

elif whostarts == 3: # In case player3 starts the game

results.append(('p3started', initial, player3(initial)))

else:

raise ValueError("No such player exists") # raise an error if player was not recognized

stat\_dict = {}

for result in results:

key = result[0] + result[2] # creating the key using who started and who won

if key not in stat\_dict:

stat\_dict[key] = 1

else:

stat\_dict[key] += 1

plt.figure('Experiment'+str(experiment\_id))

plt.bar(stat\_dict.keys(), stat\_dict.values())

plt.xlabel('Scenarios')

plt.xticks(rotation='vertical')

plt.ylabel("Number of wins")

plt.title('Game stats')

plt.tight\_layout()

plt.show()

Unique paths

def count\_paths(Start, End):

# <Feel free to write helper functions if you need to>

if Start == End:

return 1

elif Start[0]<End[0] and Start[1]<End[1]:

return count\_paths((Start[0]+1, Start[1]), End) + count\_paths((Start[0], Start[1]+1), End) + count\_paths((Start[0]+1, Start[1]+1), End)

elif Start[0] == End[0]:

return count\_paths((Start[0], Start[1]+1), End)

else:

return count\_paths((Start[0]+1, Start[1]), End)

pass

def print\_paths(Start, End): #BONUS QUESTION 15 points

# <Feel free to write helper functions if you need to>

def findpaths(Start, End):

if Start == End:

return [str(Start)]

elif Start[0] < End[0] and Start[1] < End[1]:

paths = findpaths((Start[0] + 1, Start[1]), End) + findpaths((Start[0], Start[1] + 1), End) + findpaths((Start[0] + 1, Start[1] + 1), End)

output = []

for path in paths:

output.append(str(Start)+ "->" + path)

return output

elif Start[0] == End[0]:

paths = findpaths((Start[0], Start[1] + 1), End)

output = []

for path in paths:

output.append(str(Start)+ "->" + path)

return output

else:

paths = findpaths((Start[0] + 1, Start[1]), End)

output = []

for path in paths:

output.append(str(Start) + "->" + path)

return output

print(findpaths(Start, End))

pass

Three player game

def player1(pebble\_count):

# Strategy:

# Always picks 2

if pebble\_count == 0:

return "p3won" # player3 wins

else:

return player2(pebble\_count - 2) # handing over to player2 after picking 2 pebbles

# defining how player2 plays the game

def player2(pebble\_count):

# Strategy:

# pick 2 pebbles if number of pebbles is even otherwise pick 1

if pebble\_count == 0:

return "p1won" # player1 wins

elif pebble\_count % 2 == 0: # Checking for even

return player3(pebble\_count - 2) # handing over to player1 after picking 1 pebbles

else:

return player3(pebble\_count - 1) # # handing over to player1 after picking 2 pebbles

def player3(pebble\_count):

# Strategy:

# Always picks 2

if pebble\_count == 0:

return "p2won" #player 2 wins

elif pebble\_count == 1:

return player1(pebble\_count -1)

else:

return player1(pebble\_count - 2)