Project 1 report

Creating Hanoi Tower Animation

20B11806
Saw Kay Khine Oo

1. Solving Hanoi Tower problem

The general idea of the program is that when given number of disks input by the user, we will print out "Start Configuration", followed by the first display, second display, and so on until the last display is printed out. In each display, we will print out 0,1,2 as three pegs and beside each peg will be a list of disks on that peg (numbers are important as they correspond to the size of the disk). Before each display, we will also print out which step it is right now and from which peg to which peg we are going to move the top disk. In the followings, the purpose of each function will be explained.

1.1 print_pegs function

This is the function to print out the display part. The main list pegs will have 3 sub-lists for 3 pegs and inside each sub-list, there will be all the disks corresponding to each peg. We use enumerate so that we can call out the peg and its corresponding list.

1.2 hanoi function

The main list pegs used in print_pegs function will actually be generated here. The first sub-list will also be first filled with number of disks given by the user and it will be the parameter for this function. Then, we can call the print_pegs function to print out the current display which is only the first one. In order to move the disks and print out another display, we need other functions.
*data list is for animation and will be explained in the animation section.

1.3 move_disk function

move_disk will take 3 parameters, the main pegs list, source peg (from 0,1,2) and destination peg (also from 0,1,2). The main purpose of the move_disk function is as the name suggests, to move the disks to play the Hanoi tower game. However, since there are certain rules in Hanoi tower game, we need to check if the disk is movable, in other words, obeying the rule or not. Firstly, check if source peg is assigned with the indexes from 0,1,2 because we only have 3 pegs. Secondly, check if destination peg is assigned with indexes from 0,1,2. Thirdly, check if the sub-list of source peg has disks in it. Before we check the last requirement, we need to assign a new variable to the disk we are moving. We will call it "disk" and this is taken from the top of the source peg. Lastly, check if the "disk" is smaller the existing disk on the destination peg. We can easily check this by inequation since smaller disk will have smaller number.

After successfully checking all the conditions, we will take the top disk (last element) from the source peg using list.pop() and add that disk to the destination peg using list.append(). We will complete moving the disk for one display now and we will print the print_pegs function.

Notice that this function needs to be looped for the number of steps needed to complete the game. So, we will put count+1 here to keep the number of loops.

1.4 move_tower function

This function requires 4 parameters, the main pegs list, number of disks, source peg index and destination peg index. We need to know the index of number of the spare peg which is not the source nor destination. We can get it by using "`spare = 3 - source - dest`".

Next, we will construct recursion loop here. The base case is when number of disks is 2, we will complete the Hanoi tower game in 3 steps. Firstly, move disk from source to spare peg. Secondly, move disk from source to destination peg. Finally, move the disk in spare to destination peg. Note that we do not need to consider whether the top disk will be moved or not because we have written the conditions in move_disk function.

Now, we need to think how the recursion can be written. The base case is already written with 2 disks (2d case). When we have 3 disks (3d case), we can move the upper 2 disks from source to the spare disk first by recurring the 2d case. Then, we move the biggest disk from source to destination peg. Now, again, we need to move 2 disks from the spare to the destination peg by recurring 2d case. In the same way, when dealing with 4d case, we recur 3d case and move the biggest disk from source to destination peg and then recur 3d case again.

1.5 Difficulties

I didn't find any difficulties in writing the recursion function, but I I wanted to add the number of counts or number of steps taken to complete the game. So, I have to add the count variable, which unfortunately doesn't work well in recursive function. If I set the count variable to zero inside the function, it won't work since the function will be recurring and setting it to zero again and again. So, I have to declare the variable outside the function. By using global variable, I can call the variable in the function.

2. Creating Animation for Hanoi Tower

Maybe because the pegs of Hanoi tower look like the bars in the bar chart, I developed an idea to make an animation of Hanoi tower with the bar chart. Initially, I thought it would be difficult, I wasn't thinking of differentiating between large and small disks. However, with the help from professor Bonnet, I was able to write the program I wanted.

2.1 Getting the necessary data

The number of steps to complete the game is 2**n-1. For the animation, a list containing 2**n-1 number of display sub-lists is required. In each display sub-list, another 3 peg sub-lists representing 3 pegs are required. Corresponding disks will be in each peg sub-list. I declared the data list in the hanoi function. In the move_disk function, the required data will be inserted into the list using the following lines.

```
data[0][0]=list(range(n-1,-1,-1))
data[count+1]= copy.deepcopy(data[count])
data[count+1][source].pop()
data[count+1][dest].append(disk)
```

It took me some time to figure out that using deepcopy is more appropriate here. I tried to use list.copy() which didn't turn out so well because of the sub-lists inside.

2.2 Creating animation

Now that I have the data list for any number of disks input by the user, I basically just edit a few of the codes sent by professor Bonnet (which was written for animation with only 3 disks). The main place that I focused on is to make the number of disks, bottom and the width of the disks arbitrary.

```
                for i in range(n):
                    widthlist.append(1.5*i+2)
disks = plt.bar([0]*n, 2, bottom=list(range(n-1, -1, -1)),
                      width=widthlist)
```