Tower of Hanoi

Shahriar Ahmad Fahim

Tokyo Institute of Technology

January 15, 2021

1 Introduction

This paper explains the solution for the problem 'Tower of Hanoi'. It has mainly two parts- i) the recursive solution of the problem, and ii) animation of how it is solved. The program solves for any number of disks. However, for the sake of simplicity, the animation is available up to 5 disks only.

2 Recursion algorithm for the problem: Tower of Hanoi

We are given with three pegs. In this problem, peg0, peg1 and peg2 are source peg, destination peg and spare or alternate peg respectively. Here n is the number of disks.

For the **base case** where n is equal to 1,

Step 0: We send the disk from source peg to the destination peg. (peg0 to peg1)

For the **recursion case** where n is greater than 1,

Step 1: We send (n - 1) disks to the spare peg **using recursion**. (n-1 disks from peg0 to peg2)

Step 2: We send nth disk to the destination peg. (nth disk from peg0 to peg1)

Step 3: We send the (n-1) disks from the spare peg to the destination peg now **using recursion**. (n-1 disks from peg2 to peg1)

3 Explanation of code

We explain the modifications done to the skeleton of code that was given.

3.1 print_peg function:

We add only one line to this function.

```
pegs_states.append(copy.deepcopy(pegs))
```

This is done to save the local variable pegs in another variable pegs_states which is global. pegs_states variable shall be used to run the loop to make the animation later in another function 3.3.

Here, deepcopy is used by importing copy module because otherwise saved value in the global variable pegs_states change as the value of local variable pegs change with repeated call of this function.

3.2 move_tower function:

This function takes four parameters- pegs, nb_disk, source, and dest. Here, nb_disk is the number of disk, source is the peg number of initial disk location, and dest is the peg number the disks need to be sent to.

This is the function where we implement the algorithm in 2.

```
spare = 3 - source - dest
```

As this function is called multiple times for recursion with new source and destination peg number, this spare variable initializes the alternate peg number each time.

This is the base case where number of disk is equal to 1.

```
  if nb_disk == 1:
    return move_disk(pegs, source, dest)
```

This is the STEP 0 of our algorithm. If the disk number is 1, we just move the (only) disk to the destination peg using the function move_disk.

This is the recursive case where number of disks is greater than 1.

```
  else: #recursive case
    #move n-1 disk to the spare peg
    move_tower(pegs, nb_disk - 1, source, spare)
```

This is the STEP 1. Considering there are n disks, we send (n-1) disks to the spare peg using the same function move_tower. This is where recursion occurs. If (n-1) is equal to 1, then base case will run. Otherwise, this else condition will run again.

```
    #move the nth peg to the destination peg
    move_disk(pegs, source, dest)
```

This is the STEP 2. After we have moved all the disks to the spare peg except the largest or the last one, we now just move the largest disk to the destination peg using the function move_disk.

```
    #move n-1 pegs to the destination peg
    move_tower(pegs, nb_disk - 1, spare, dest)
```

This is the STEP 3. Now, we again move the (n-1) number of disks from the spare peg to the destination peg through the function move_tower. This step works just like Step 1 except, our old spare peg (peg 2) will be now our source peg and old source peg (peg 0) will be now our spare peg.

3.3 hanoi_anime function:

This is a new function added to the code. We used pygame module for making the animation. This function takes only one parameter pegs_states which has all the states of pegs throughout the transfer of disks from source peg to destination peg as the recursion algorithm.

```
  red = (255, 0, 0)
  sky_blue = (0,255,255)
  yellow = (255, 255, 0)
```

We define three colours to use in the animation.

```
  red = (255, 0, 0)
  sky_blue = (0,255,255)
  yellow = (255, 255, 0)
  height_adjust = 100 - n*20
```

We define three colours to use in the animation. And we define another variable height_adjust to adjust the disk position in the display when disk number is less than 5.

```
  hello = pegs_states[j]
```

We run a loop to get each state of the pegs here. We save each state in a local variable named hello.

```
1     gameDisplay = pygame.display.set_mode((800, 600))
```

We initialize our our display in 800X600. Next, we draw the bases on which pegs stand in red and rectangle form.

We now look at the the following rough diagram to understand the dimension of our drawing in the coordinate system:

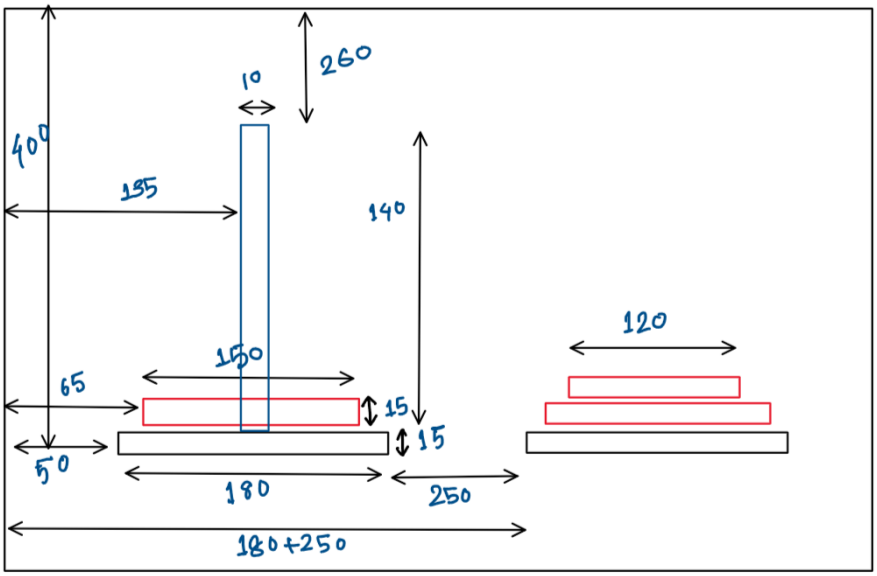

Figure 1: Rough diagram for dimension of peg, peg base and disks

Here, distance between one base to another base is 250. Distance between disks from peg to peg is 15 + 250 + 15 = 280. Each disk width decreases by 30 from larger to smaller. These dimensions will help to understand the rest of the code.

```
1     gameDisplay.fill(red, rect=[50, 400, 180, 15])
2         ... ... ...
3     gameDisplay.fill(yellow, rect=[135, 260, 10, 140])
4         ... ... ...
```

Line 1 draws the base of the peg. Two more bases were drawn similarly. Line 3 draws the peg. Two more pegs were drawn similarly. The values inside the third bracket can be understood comparing with the Figure 1.

```
    # FOR PEG 0
    if hello[0] == []:
      pass
    else: #otherwise
      for i in range(1, 6): #we check the number of disks
        if i in hello[0]:
```

If there is no disk, if condition becomes TRUE, nothing happens as line 3. Otherwise, if there are disks in the peg, each disk is denoted by a digit 1, 2, 3, etc. So we run a loop get different digits from 1 to 5 (5 is our maximum disk number for animation). And we check if the disk is present by variable i. This way, we can understand the size of the disk through this which will be helpful to draw the disk.

```
1         if i in hello[0]:
2           i=6-i
3           a1 = 65 + (i - 1) * 15
4           a2 = 380 - (i - 1) * 20 + height_adjust #when n<5, height
              ↪  is adjusted so that disks do not seem floating from
              ↪  the base
5           a3 = 150 - (i - 1) * 30
6           pygame.draw.rect(gameDisplay, sky_blue, [a1, a2, a3, 15])
              ↪  #15 is thickness of the disk
7           pygame.display.update()
```

We change the value of i in line 2 to iterate the right dimension of each disk. Variables a1, a2, and a3 correspond to the dimensions as mentioned in the Figure 1. As different disk with different value of i comes, we change the value of a1, a2 and a3. However, thickness of the disk remains unchanged as in the line 6. Line 7 updates the display. The same process occurs with peg1 and peg2 or, hello[1] and hello[2].

The key portions of the function anime_hanoi is thus explained.

4 Sample Output

We have included the animation output in this report only.

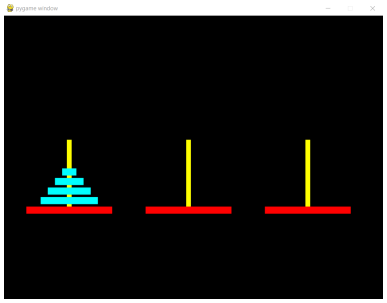

Figure 2: Screenshot before the transfer of disks began

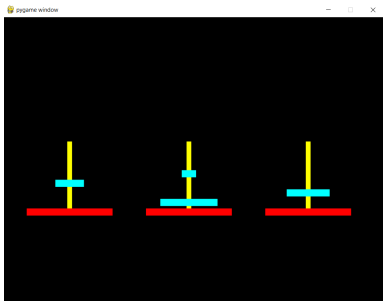

Figure 3: Screenshot during the transfer of disks

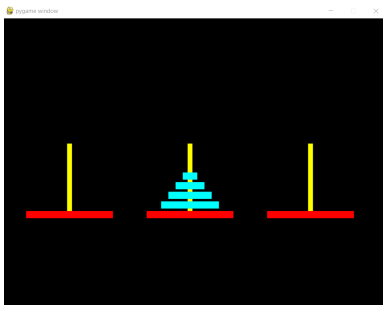

Figure 4: Screenshot after the transfer of disks is complete

These are screenshots for a sample output. In this case, the user entered n = 4 (number of disks is 4).

5 Conclusion

It is seen that Google Colab cannot show the pygame animation. So, another software PyCharm may be used to run this program to see the animation output.