TP 1 python - HMI and tkinter

the towers of Hanoi

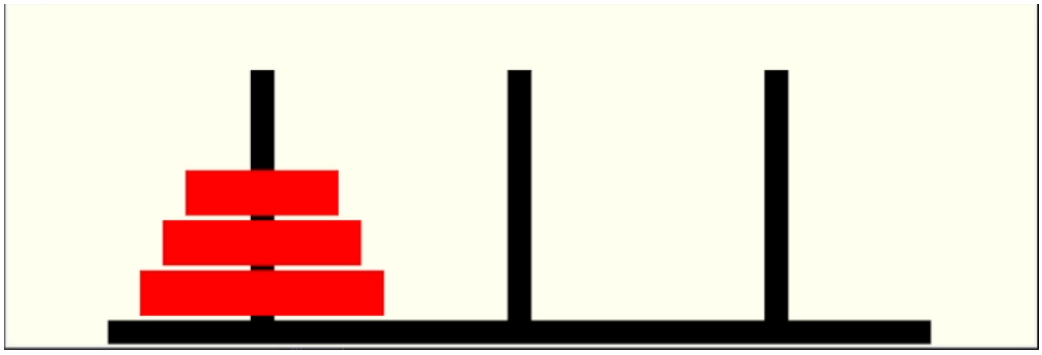
Name : KESKES Nazim
Group: A

2.2) The recursive solving algorithm:

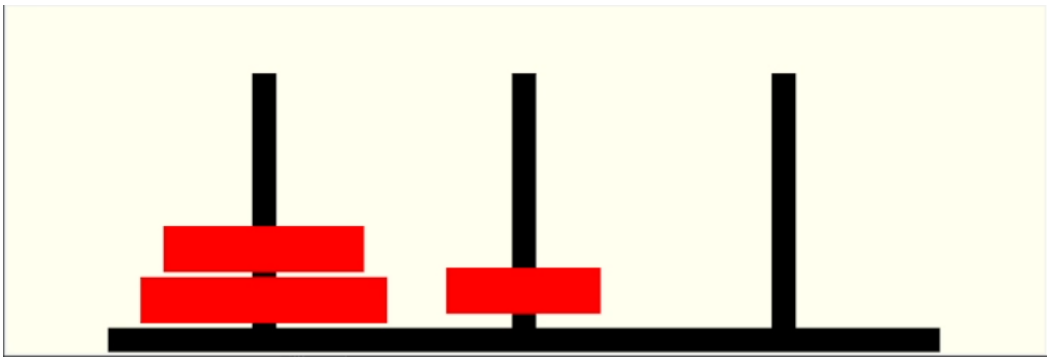
2.2.1) solving the problem for 3 disks:

In order to understand the principle of the algorithm, here are the different movements of the 3 disks:

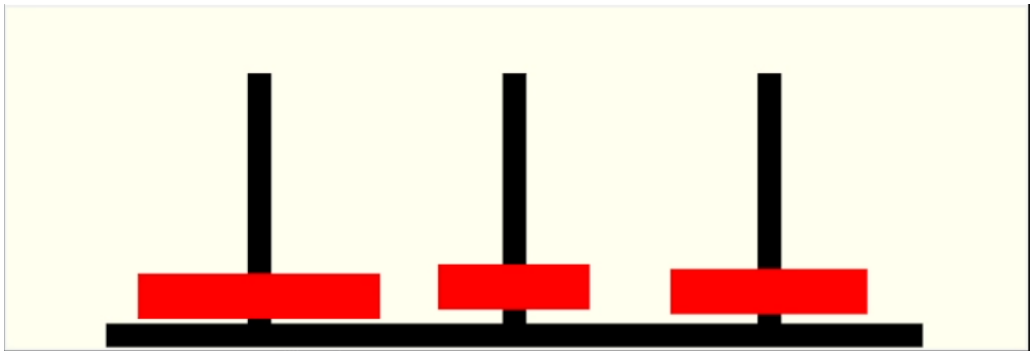
Initial case:



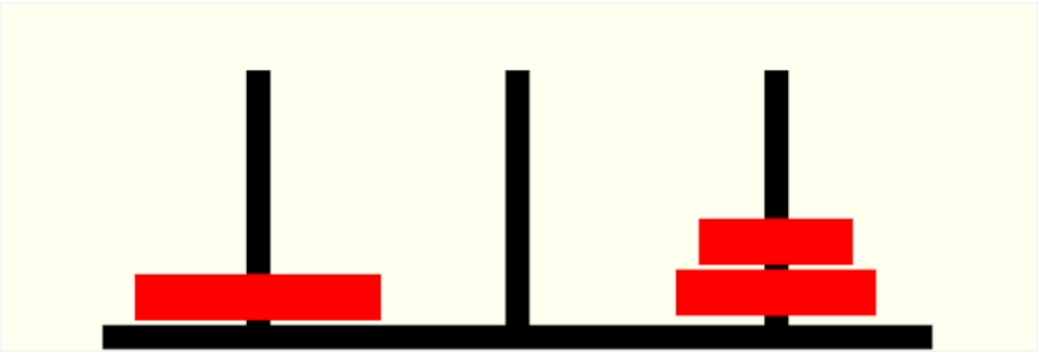
Displacement 1:



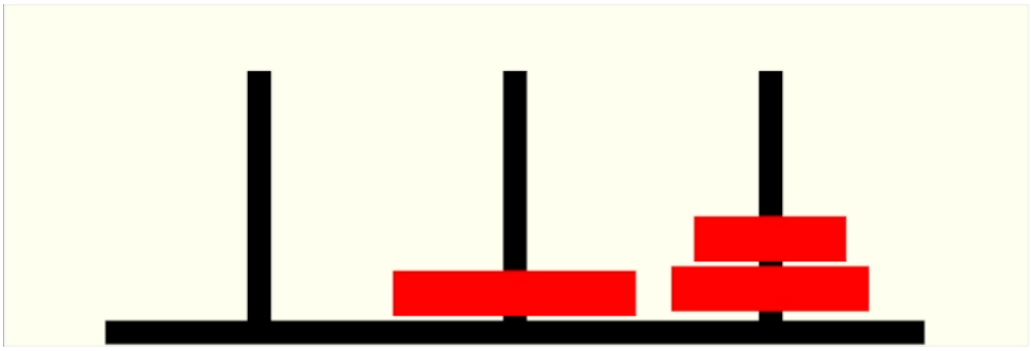
Displacement 2:



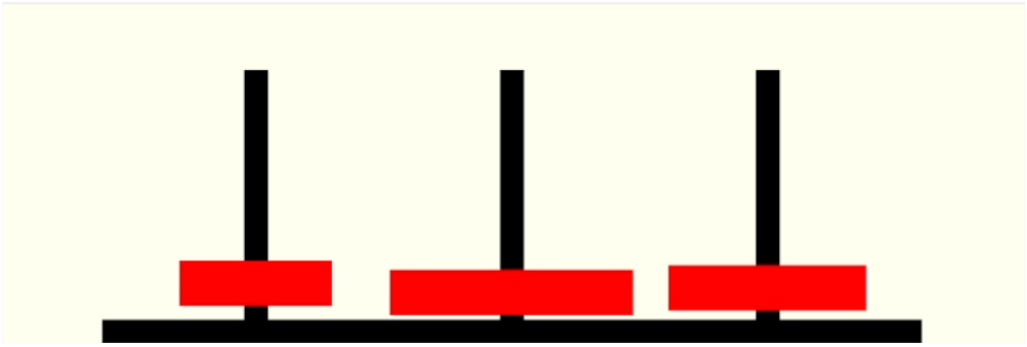
Move 3:



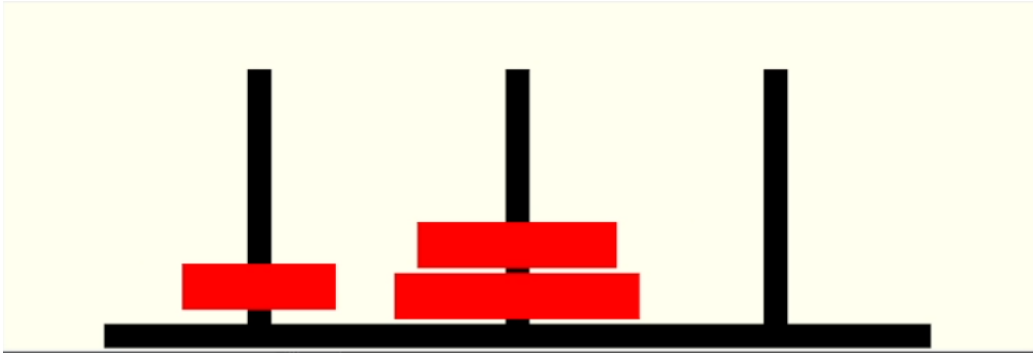
Displacement 4:



Move 5:



Trip 6:



Displacement 7:

We notice that there are 7 displacements.
We note the different stems by A, B and C. To understand how it works, we note each move by the tuple **(source, destination).** So our sequence is summarized by : (A,B) , (A,C) , (B,C) , (A,B) , (C,A) , (C,B) , (A,B) .

The intermediate rod is considered to be the C rod.
If we put **n** disks. We notice that the process takes place in 3 main phases:

- Move all but the last disc from the source rod to the intermediate rod.
- Move this last disk to its destination.
- Move the n - 1 discs from the intermediate rod to the destination rod.

In the 1st and 3rd step, to move the **n - 1** disks (i.e. except the last one), we notice that we solve the same problem, i.e. to move a certain number of disks from one stem to another by having the 3rd one as an intermediate stem.
Then, it is a recurring problem where each time, we apply the 3 previous steps and we move to a more restricted problem and finally we change the 3 stems (source, destination, intermediate).

2.2.2) the function hanoi(n,src,des,tmp) :

The function is attached in the **src** folder under the name hanoi.py.

```
def hanoi(n, src, des, tmp):
    if n > 0:
        hanoi(n - 1, src, tmp, des)
        print("src =", src, "\tdst =", des, "\n**************************")
        hanoi(n - 1, tmp, des, src)

hanoi(3, "A", "B", "C")
```

2.2.3) the execution of the function:

If we execute **hanoi(3, "A", "B", "C")** we get :

```
src = A     dstn = B
*************************
src = A     dstn = C
*************************
src = B     dstn = C
*************************
src = A     dstn = B
*************************
src = C     dstn = A
*************************
src = C     dstn = B
*************************
src = A     dstn = B
*************************

Process finished with exit code 0
```

One notices that one fell in the same sequence as that of the question 1. To generalize the case (see the code in the attachment):

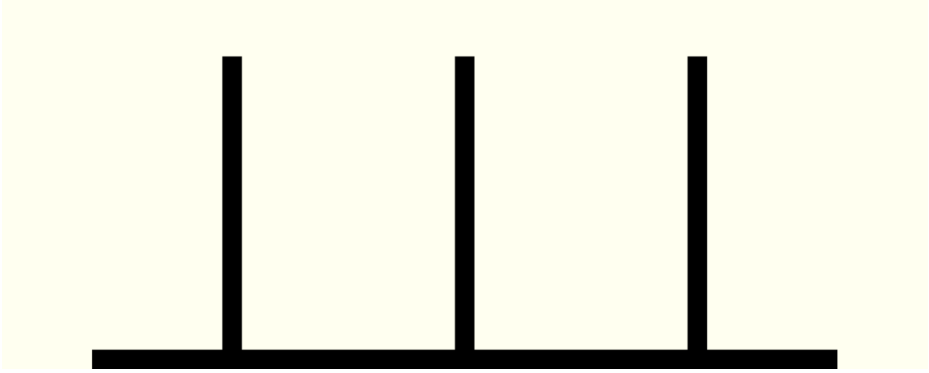
- we realize hanoi(n, X, Y, Z)
- when we move the n-1 rods from X to Y, we make the call hanoi(n - 1, X, Z, Y)
- when we move the n-1 rods from Z to the rod Y, we make the call hanoi(n - 1, Z, Y, X)

In the case arriving at n = 0, we start to go up in the recursion stack in this order:
hanoi(1, "A", "B", "C")
hanoi(2, "A", "C", "B")
hanoi(1, "B", "C", "A")
hanoi(3, "A", "B", "C")
hanoi(1, "C", "A", "B")
hanoi(2, "C", "B", "A")
hanoi(1, "A", "B", "C")
each time we execute the procedure **display(print)** , we will have : (A,B) , (A,C) , (B,C) , (A,B) , (C,A) , (C,B) , (A,B) .

2.3) Draw the scenery in Tkinter:

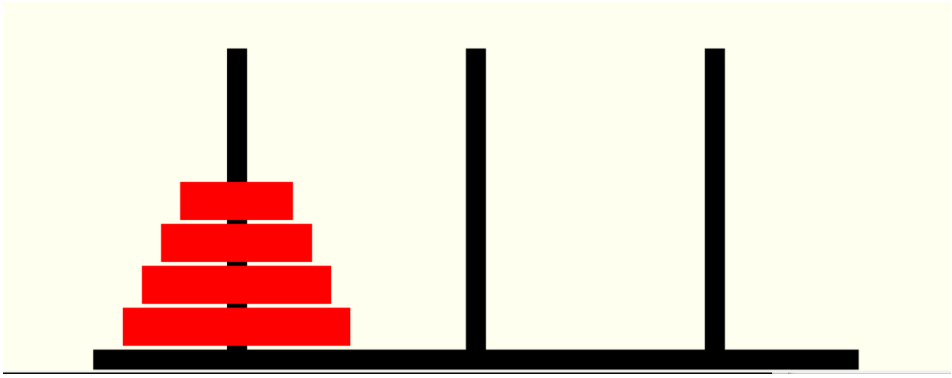
2.3.1) Draw the canvas, the three rods and the base:

The code is attached in the **src** folder under the name caneva.py. Here is the result of the execution:



2.3.2) place the discs on the left rod:

The code is attached in the **src** folder under the name caneva_avec_disque.py. Here is the result of the execution:

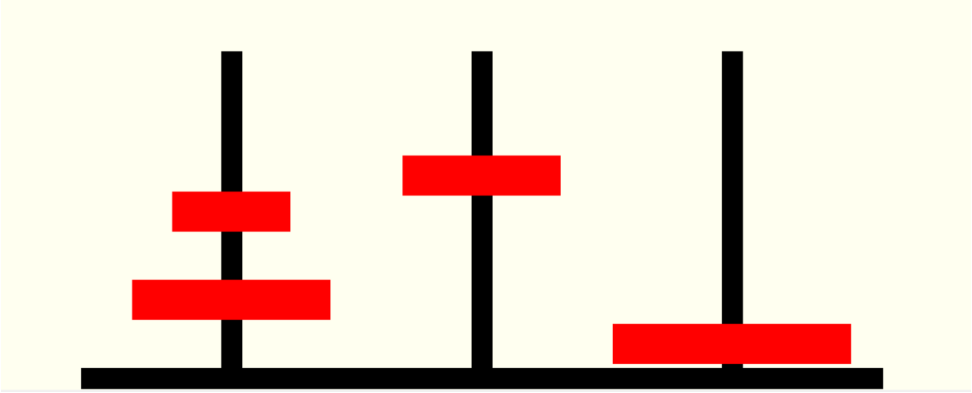


2.4) Move a disk move(nro,src,des):

```python
#Déplacement d'un disque
def move(nro, src, des):
    i, hi = src
    j, hj = des
    dx = (j - i) * Distance
    dy = -(hj - hi) * Height
    cnv.move(ids[nro], dx, dy)

#test1 deplacement
move(2, (0,2), (1, 4))
#test2 deplacement
move(0, (0,0), (2, 0))
```

The code is attached in the **src** folder under the name deplacement.py. Here is the result of the execution of a test by moving the disk[0] to the stem C with the same initial position and also the disk[2] to the stem B with the 4th position :



2.2.3) Adapting the algorithm for Tkinter :

2.5.1) The function hanoi(ids,source,destination,temp,heights):

```python
def hanoi(ids, source, destination, temp, hauteurs):
    if ids:
        hanoi(ids[1:], source, temp, destination, hauteurs)
        print(ids[0], (source, hauteurs[source] - 1),
              (destination, hauteurs[destination]))
        hauteurs[source] -= 1
        hauteurs[destination] += 1
        hanoi(ids[1:], temp, destination, source, hauteurs)

#test1 de l'algo
hanoi([0, 1, 2], 0, 2, 1, [3, 0, 0])
```

The code is attached in the **src** folder under the name hanoi_adapte.py.

2.5.2) the execution of the function:

If we execute **hanoi([0,1,2], 0, 2, 1, [3,0,0])** we get :

```
2 (0, 2) (1, 0)
1 (0, 1) (2, 0)
2 (1, 0) (2, 1)
0 (0, 0) (1, 0)
2 (2, 1) (0, 0)
1 (2, 0) (1, 1)
2 (0, 0) (1, 2)
```

Now instead of n (number of disks) we work directly with the list of disks since the rectangles are already built. What is more compared to the first version is the height in the stem to know where to place the disk after moving it

(on which level on the corresponding stem). To generalize the case (see the code in the attachment):

- we realize hanoi(ids, source, destination, temp, heights)
- when we move the n-1 stems from source to destination, we make the call hanoi(ids[1:], source, tmp, destination , heights)
- when we move the n-1 temp stems to the destination stem, we make the call hanoi(ids[1:], temp, destination, source, heights)

When a displacement is made, the height of the source rod is decreased and increased in the destination rod

In the case when the list of disks (ids) is empty, we start to go up in the recursion stack in this order:
hanoi([2],0,1,2,[3,0,0])
hanoi([1,2],0,2,1,[2,1,0])
hanoi([2],1,2,0,[1,1,1])
hanoi([0,1,2],0,1,2,[1,0,2])
hanoi([2],2,0,1,[0,1,2])
hanoi([1,2],2,1,0,[1,1,1])
hanoi([2],0,1,2,[1,2,0])

Thereafter, each time the procedure **display(print) is** executed, we will have: the target disk, its displacement from a stem with a height to another stem with its height.

2.6) Animation for n records :

2.6.1) the function hanoi(ids,source,destination,temp,heights,done):

The code is attached in the **src** folder under the name hanoi_version_final.py.

```python
#Realiser l'algorithme de hanoi
def hanoi(ids, source, destination, temp, hauteurs, done):
    if ids:
        A = hanoi(ids[1:], source, temp, destination, hauteurs, False)
        B = [(ids[0], (source, hauteurs[source] - 1), (destination,hauteurs[destination]))]
        hauteurs[source] -= 1
        hauteurs[destination] += 1
        C = hanoi(ids[1:], temp, destination, source, hauteurs, False)
        if done:
            moves = list(enumerate(A + B + C))
            for (i, (nro, src, des)) in moves:
                cnv.after(1000 * (i + 1), move, nro, src, des)
        return A + B + C
    return []

#test hanoi pour n = 4
hanoi(list(range(n)), 0, 1, 2, [n, 0, 0], True)
```

2.6.2) the execution of your function:

We notice that as a process, we do exactly the same thing as the improved version. Except that this time if the **hadoi** function returns the list of moves. So instead of displaying each move in an ascending way in the recursion like the previous version. Now we store its moves in a moves list. And for each move i (or displacement of a disk) we assign a time of launching that is: (i + 1) seconds. In this way, we guarantee that the displacements are going to be visualized graphically in a fluid way (i.e. each displacement unfolds in one second). If we leave the code of the improved version as it is, the moves will be done instantly. So, each move i will overlap with move i - 1.
Therefore, we do not respect the standards of the ergonomics of our animation.