TASK 4 DOCUMENTATION

Complete The Cube

Tanmay Amritkar (23ME10092)

I. INTRODUCTION

To implement a finite state machine using python to develop a color composing cube , using a relation between pixel intensities and rotation angles in a three- 8-bit integer based RGB system. In this project , we have developed this imaginary cube in which we have sixe faces(R,G,B,RG,GB,BR) and using pixel intensities of R,G,B face we pixelate the faces of RG,GB,BR.In this ,we also implement the concept of finite state machines , which simplify our program in terms of thinking state wise and reduces the redundancy of if-else statement and while/switch usages.To develop this FSM in python , I referred to an online resource mentioned in referennces. The main approach is to think of the problem state wise , taking advantage of the finite state machine concept , we can distinguish each case from another and focus on that itself , forgetting about the rest of the program. Then we need to think of cargo , which refers to the transfer of variable values, arrays and images from one state to another in the FSM implementation. Then we break the problem one by one into states of initialization , procession and display.

Initial Attempts : Initially I thought of implementing this using a switch condition statement , which I thought would complicate the scenario while going up the conditions.

I have not implemented the excessive row constriction due to lack of time but the logic is to define an array counting each row's input and according to the conditions refilling it or telling the user input to ask for a different row.

II. Problem Statement

This task presents the challenge of creating a color-combining device akin to a printer, which blends three primary colors—red, green, and blue—to generate composite images. At its core, the device operates as a structured system, orchestrating the flow of user inputs and actions through various stages. Users initiate the process by specifying coordinates for each color—designated as \(x \), \(y \), and \(z \)—within a 16x16x16 grid, representing the red, green, and blue faces, respectively. Additionally, users define rotational angles (0, 90, 180, 270 degrees) for combining color pairs, such as RG, GB, and BR faces. These inputs drive the subsequent image generation process.
The device employs a finite state machine (FSM) to manage its operation. The FSM consists of states and transitions, facilitating the sequential execution of tasks. It starts with an initialization state, where users input coordinates and rotation angles. This information is then passed to the next state, which processes the inputs to generate composite images. In the processing state, the device utilizes random permutations to assign pixel intensities to the red, green, and blue faces independently. Each face becomes a canvas for blending colors based on the specified coordinates and rotational angles. For instance, on the RG face, the intensity of the R-channel from the red face and the G-channel from the green face are combined, with the blue channel set to zero. Similarly, the GB face combines the G-channel from the green face and the B-channel from the blue face, and so on. Once the composite images are generated, the device moves to the display state, where it determines which images to showcase based on the user's payment amount. If the payment exceeds 60 units, all

three composite images are displayed. For payments between 20 and 60 units, the user selects one or two images to display. Any remaining amount after deducting the cost of the displayed images is returned to the user as change. To prevent repetitive use of the same color rows, the device tracks the frequency of row selections. If a row is selected more than three times, the user is prompted to choose a different row. Once all rows are exhausted, they reset to their initial state of three selections. In summary, the color-combining device operates as a structured system, leveraging a finite state machine to manage the flow of inputs and actions, ultimately producing composite images based on user specifications.

III. Related Work

We observe that this project resonates a lot with things in our daily life applications , for example the application of blending of images . In this task we observed the blending of colors of red , green and blue in a unique way . The case of alpha blending , color space transformations and CNN(Convolutional Neural Networks) are all related to these basic concepts of images.Furthermore, advancements in machine learning and computer vision have led to the development of automated color blending systems that can analyze input images and generate composite outputs based on predefined rules or learned patterns. These systems leverage deep learning models trained on large datasets of color images to accurately predict the blending parameters for achieving desired color effects.

IV. INITIAL ATTEMPTS

1.Switch Case Attempt : The main initial attempt in this code of mine was to implement the switch case condition , but the problem I later believed would lie in accessing another infinite while loop for it , with also difficulty traversing from different states to states , especially something like traversing back or initializing back.

V. Final Approach

Final Approach :

1.Visualizing,Drawing and Coding the Finite State Machine: We first order the different states of the state machine and develop how they will flow from one to another .

Initialization->Procession->Display-> and the loop continues with the option to break.

2.Initialization:We ask the user to enter three values (x,y,z) in the [0,15] range.Then we prompt the user to enter the rotation angles for each phase(rg_face , gb_face , br_face).We also ask the user the price which they would like to pay.

3.Procession: In this step , we process the inputted data and use it to pixelate and rotate the three faces(rg_face , gb_face , br_face).Firstly we develop 6 numpy zero arrays of shape(16,16, 3) of dtype:8bit. Then it is important to randomly permute the values of [0,255] among each pixel of r_face , g_face and b_face which we approach by using the numpy randint function.

```
r_face = np.zeros((16,16,3), dtype = np.uint8)
r_face[:,:,0]=np.random.permutation(np.arange(256)).re
shape((16, 16))
```

Now we need to pixelate the three faces using the explanation given in the problem statement which is : Suppose the pixel with location at (x,p) on R-face has intensity h (of R-channel) and pixel

With location at (y,q) on G-face has intensity k (of G-channel) with p and q varying in the range [0,15] ,then on RG-face, the pixel at location (p,q) should have intensities (R,G,B) as (h,k,0) before subjecting it to any rotations. Based on the inputs given by the user, you need to rotate the image and save the final image.

```
for p in range(16):
    for q in range(16):
      intensity_r = r_face[x, p, 0]
      intensity_g = g_face[y, q, 1]
      rg_face[p, q] = [intensity_r, intensity_g, 0]
```

After pixelating the three faces , we must rotate these three images from the user input of angle rotation(0,90,180,270) .For this we define a function called (def rotated_image) and call it whenever required.

```
def rotate_image(image, angle):
rotated_image = np.rot90(image, k=angle // 90) return
rotated_image
```

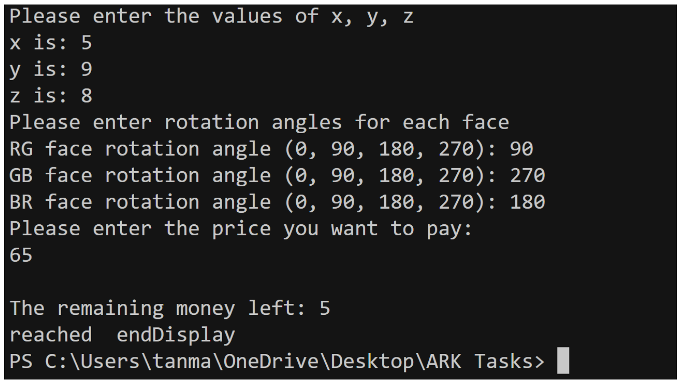
As an image is a numpy array we can rotate this using the np.array function.We then transfer (rg_face,gb_face,br_face,price) as cargo to the next state which is displayed.

4.Display:The main part of this state is to display the pixelated and rotated images with the given price provided by the user . Depending on the price , the number of images displayed will depend . As each image costs 20, for a price higher than 60 , all three faces will be displayed , with the remaining price displayed as a message. For prices between 40 and 60 , any two faces will be shown , which will be asked from the user and for between 20 and 40 , only one will be displayed and the one to be displayed will be from the user's input. We use matplotlib to display the image , that is convert the numpy array to display of pixels .

5.Running the FSM: In the end we call the State Machine and we keep adding states such Initialization , Procession , Display , endDisplay , and then finally define our start state and then run it

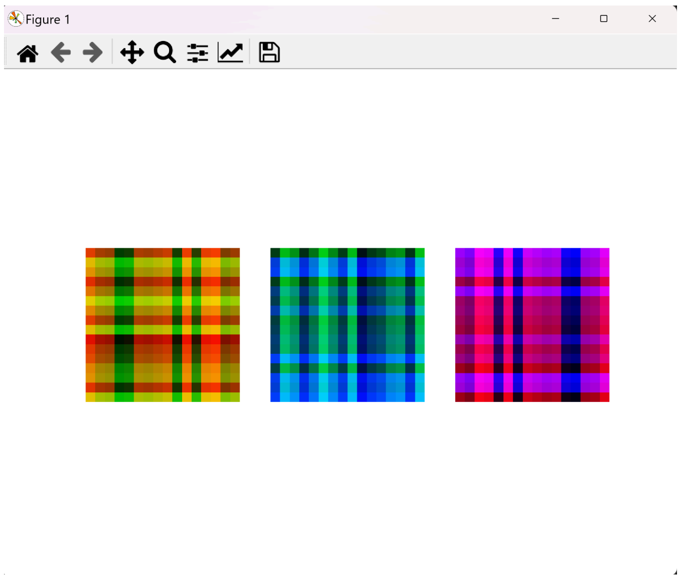
```
def main():
   fsm = StateMachine()
   fsm.add_state("Initialization", initialization_handler)
   fsm.add_state("Procession", procession_handler)
  fsm.add_state("Display",display_handler)
   fsm.add_state("endDisplay",end_display_handler,1)
   fsm.set_start("Initialization")
  F sm.run("Initial cargo")
```

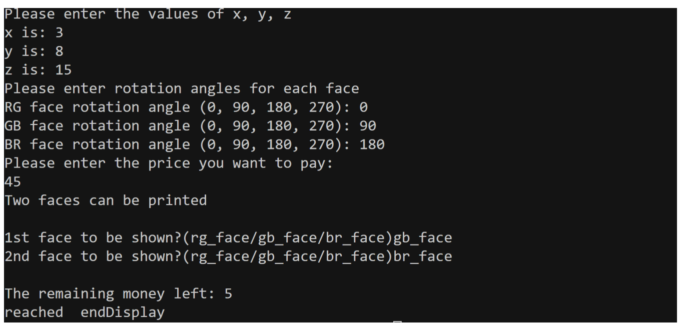
VI. Results and Observation



```
Please enter the values of x, y, z
x is: 5
y is: 9
z is: 8
Please enter rotation angles for each face
RG face rotation angle (0, 90, 180, 270): 90
GB face rotation angle (0, 90, 180, 270): 270
BR face rotation angle (0, 90, 180, 270): 180
Please enter the price you want to pay:
65

The remaining money left: 5
reached  endDisplay
PS C:\Users\tanma\OneDrive\Desktop\ARK Tasks>
```

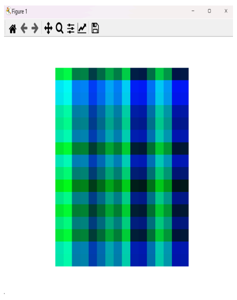


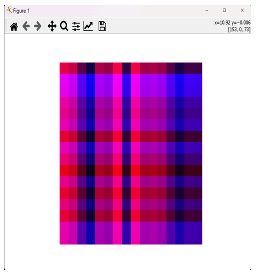


```
Please enter the values of x, y, z
x is: 3
y is: 8
z is: 15
Please enter rotation angles for each face
RG face rotation angle (0, 90, 180, 270): 0
GB face rotation angle (0, 90, 180, 270): 90
BR face rotation angle (0, 90, 180, 270): 180
Please enter the price you want to pay:
45
Two faces can be printed

1st face to be shown?(rg_face/gb_face/br_face)gb_face
2nd face to be shown?(rg_face/gb_face/br_face)br_face

The remaining money left: 5
reached  endDisplay
```





VII. Future Work

1.Implementing Bonus Kernel :

```
#Bonus Task :

  kernel = np.zeros((16,16,3), dtype = np.uint8)


  key = 0
  for key in range(256):
    a=b=c=d=e=f=0
    x1=y1=y2=z2=x3=z3=0
    #Finding r__face coordinates
    for a in range(16):
      for b in range(16):
        if ( r__face[a,b,0] == key):
          print(a)
          x1 = a
          y1 = b
        break

    for c in range(16):
      for d in range(16):
        if ( g__face[c,d,1] == key):
          y2 = c
          z2 = d
        break

    for e in range(16):
      for f in range(16):
        if ( r__face[e,f,0] == key):
          x3 = e
          z3 = f
        break

    kernel[(key/16) , key - (key/16),0] = 16*x1
    kernel[(key/16) , key - (key/16),1] = 16*y2
    kernel[(key/16) , key - (key/16),2] = 16*z3


  plt.imshow(kernel)
  plt.axis('off')
  plt.title("BONUS__KERNEL")
  plt.show()
```

I have implemented this code as an individual segment , but the future work remains to create this as an individual state and then integrate it in the entire program.

VIII.CONCLUSION

The problem at hand involved creating a color composing machine using a finite state machine approach. The machine needed to compose three faces (R, G, and B) based on user inputs, rotate them as required, and display them according to the user's budget and preferences. Each face had pixel intensities ranging from 0 to 255, and the machine had to ensure that no two pixels in the same face had the same intensity. Additionally, the machine had to handle cases where a particular row from each face could only be used a maximum of three timesWhile implementing the solution, I faced several challenges, including handling user inputs, managing image pixel data, ensuring proper rotation of images, and managing state transitions.In the context of Aerial Robotics, where drones and unmanned aerial vehicles (UAVs) are extensively used for various tasks such as surveillance, mapping, and environmental monitoring, the ability to process and analyze images in real-time is crucial. The color composing machine can aid in this process by allowing UAVs to compose and display images captured from different perspectives or sensors. For example, it can be used to overlay thermal imaging data onto visible light images for enhanced analysis of environmental conditions or to composite images from multiple cameras to generate a comprehensive aerial view of a given area..

References

[1]https://python-course.eu/applications-python/finite-state-machine.php