Complete The Cube

Samarth Barve

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 six 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 references. 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.

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

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.

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)).reshape((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. 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$_i mage) and call it whenever required. def rotate_image(image, angle) :$
$rotated_i mage = np.rot90(image, k = angle//90) return rotated_i mage As an image is a numpy array we can rotate this using the np.array function. We then transfer (rg\_face$
$The main part of this state is to display the pixelated and rotated images with the given price provided by the user. Depending on the price,$
$In the end, we call the StateMachine and we keep adding states such Initialization, Procession, Display, end Display, and then finally$
$fsm = StateMachine() fsm.add_state("Initialization", initialization_handler) fsm.add_state("Procession", procession_handler) f$