Lauren Bassett

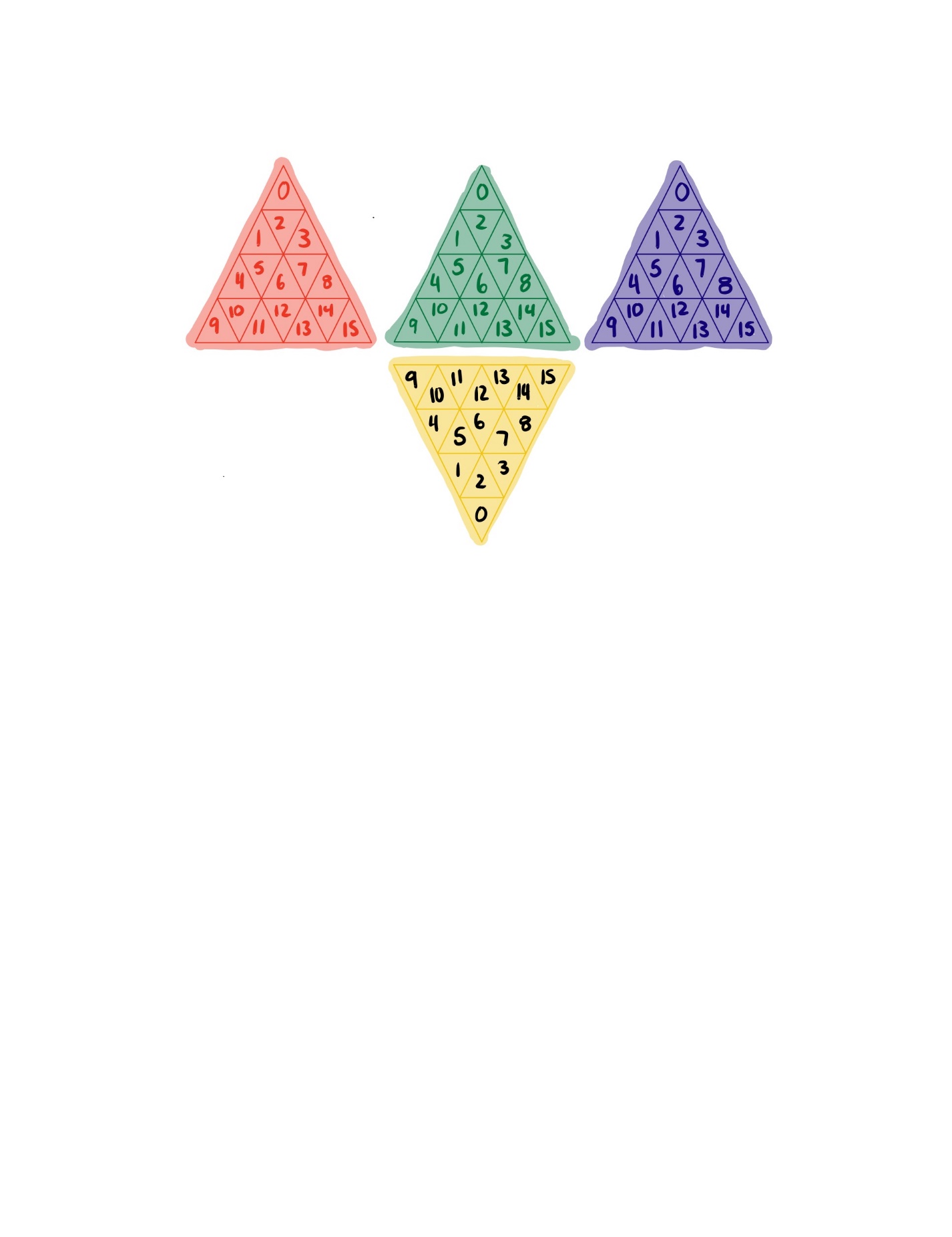
CS 463-G Fall 2020

Modeling the Pyraminx

# A description of the data structure:

For my data structure, I generated an array that represented the “cubies” in each of the four sides of the pyraminx. I knew that even as the pyraminx shifted, each swap would involve the same parts of the pyraminx every single time. While the content (color) of the indices change, the specific indices exchanging data do not.

Thus, I decided to model my arrays in this fashion, and kept my labeling constant in all functions of the program. This was my original sketch. The numbering convention did not change.



The code for these data structures is straightforward.

A close up of a keyboard

Description automatically generated

I implemented this data structure in two parts. The first you can see commented out, where I made the value of each index a letter indicating the color of the cubie and the index in the array. I did this to ensure that all of my swaps were working correctly within the pyraminx function. After I was sure that they worked correctly, I changed the name to a value indicating the color of the cubie in order to work with the GUI.

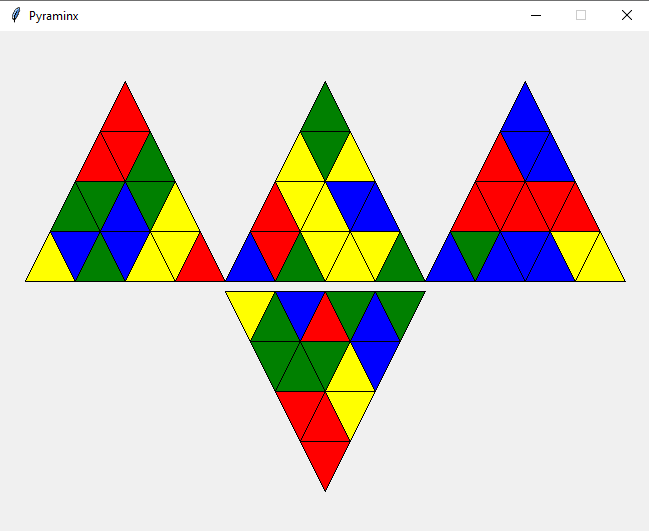
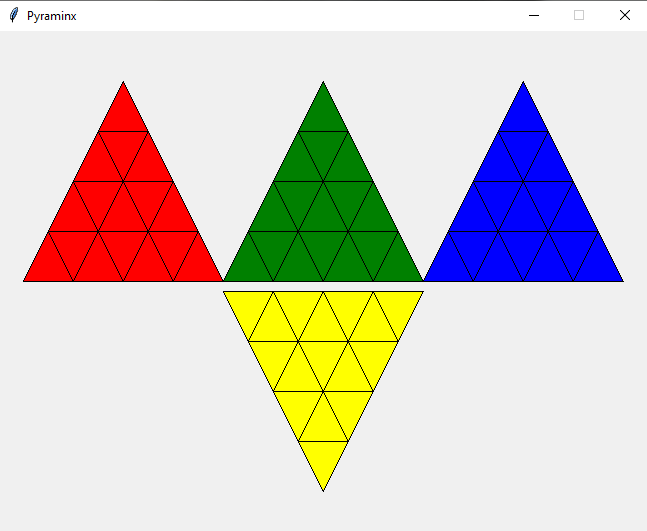
After compiling, the program should run using the command python3 4\_Layer\_Pyraminx.py There are no command line requirements for this program.

Once it starts, you will be prompted to give a value for the number of times to shuffle the pyraminx.

# The GUI

It will then show a window with the pre-swapped pyraminx. Simply click and the window will be replaced with the post swap pyraminx.

This is an example of what the GUI outputs look like pre and post swap.



# The Randomizer

The randomizer exists inside of a loop, which iterates for each swap requested by the user at the beginning of the program. In each iteration, a random number is generated. Because Python does not have switch statements, I use if/else statements in a similar format. Based on the random number generated, I call one of the 12 swap functions. This repeats for each swap designated by the user. To keep track of the swaps and check for errors, the swaps are entered into a file called “swaps.txt.”

\*\*\*I do account for counter-clockwise swaps, the rotation algorithm is called twice in a row to swap it. I didn’t program it because I thought it would be a waste of time.

# Heuristic

Heuristic:

My heuristic consists of counting the number of stickers that are out of place and dividing that number by the maximum number of stickers that would be moved in one rotation.

Why it is admissible: This will give me the lowest number of moves possible because it is calculating the total number of stickers out of place, and how they will move as the puzzle rotates to solve it.

It is admissible because this would generate the lowest number of moves to get all colors back to their original place.

# What I learned

I refreshed my memory on programming graphics in python.

I learned about heuristics and what makes them admissible or not.

I had to spend a lot of time ensuring my swaps were accurate based on how the device twists.