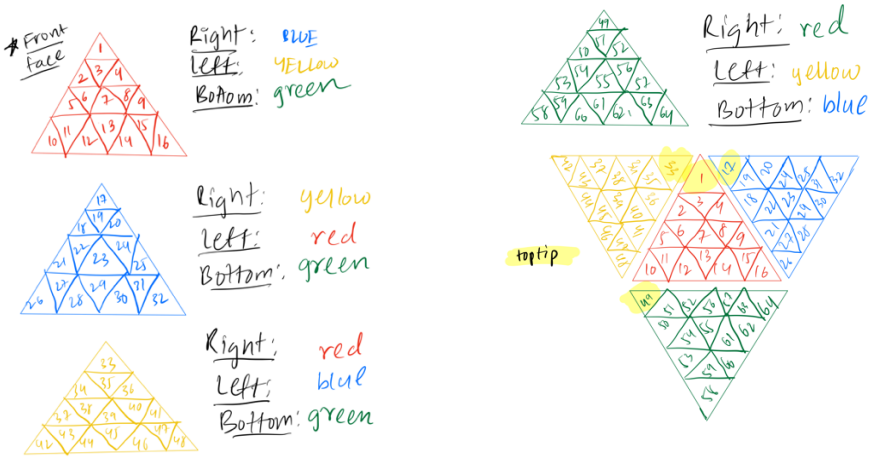
Description of Data Structure:

- Sticker is a class designed to represent each sticker on the pyraminx. It has two attributes: position and color.
- Then, we have a pyraminx class, which has the following attributes: red_face, blue_face, green_face, yellow_face, faces.
- To create an instance of this class, the user doesn't need to pass in any arguments.
- When the instance is created, an initialize method is called, which subsequently makes four calls to the create_faces method—one call per color in the pyraminx
- Judging by the color that is passed in, the create_faces method creates "sticker" objects for each sticker in the given face.
    - We have made the decision to keep red as our front face—to the left of red is yellow, to the right is blue and to the bottom is green.
    - Consequently, red faces' positions get assigned to be 1-16, blue faces' are 17-32, yellow faces' are 33-48, and green faces' are 48-64.
    - Once create_faces function executes successfully, the pyraminx class's face attributes are populated: each face is a list that has 4 elements—1 for each row of a face. Each of these elements are lists that have the stickers for that row—1 element in the first list, 3 in the second and so on to represent the pyraminx.
    - As an example, this is what you would do access the 4 sticker in the third row of the red face: self.red_face[2][3]
- The self.faces variable is an array of array of arrays: it was necessary for the print function and therefore was added later on. It simply stores a reference to the four different faces.
- Based on the rotations that take place, the sticker objects are moved from one list to another. For instance, horizontally rotating the tip of the front face counterclockwise will take self.red_face[0][0] and put it in self.blue_face[0][0], takes self.blue_face[0][0] and puts it in self.yellow_face[0][0] and self.yellow_face[0][0] gets put in the place of self.red_face[0][0].



Description of Randomizer:

- Our randomizer is divided into 2 functions:
    - randomizer
        - This function takes the number of moves the user wants to execute to scramble the Pyraminx. Then it calls the movePicker function as many times as the number of moves the user wanted. After executing them it returns the scrambled Pyraminx.
    - movePicker
        - This function takes the cube (Pyraminx) as input and calls a random move for it. To do so, it randomly picks a horizontal rotation (rotate_front_rows) or a diagonal rotation (rotate_diagonal_layer). Then it randomizes the layer it will happen in, as well as if it is clockwise or counterclockwise. It is noted that diagonal rotations have the additional argument of "diagonal" to determine which of the three lower tips of the puzzle to use as a reference for the rotation.

Instructions for Pyraminx:

These are brief instructions to run our code:

- To begin, run main.py in a compiler that supports python3 by typing in "python3 main.py" while in the pyraminx directory.
- The GUI that this activates will walk you through everything:
    - The first thing you're asked is if you want to start with the pyraminx in the solved state or if you would like it scrambled and if so, by how many moves
    - In both the cases, the pyraminx is printed out for you in its current state.
    - Then, you will be prompted to choose between Horizontal rotation, diagonal rotation, and a quit option.
- The quit option will exit you out of the simulation.
- If you choose the horizontal rotation, you will ultimately be moving the red, yellow, and blue faces(unless it's the 4^{th} layer which will rearrange green) of the pyraminx horizontally:
    - You will be prompted to choose the layer you want to move: the tip is layer 1, next would be layer 2 with 3 sticker, layer 3 with 5 stickers and layer 4 with 7 stickers.
    - Once you choose a layer, you will be asked to choose a direction: clockwise or counterclockwise.
    - After this, your pyraminx will be updated accordingly and printed.
    - Example: To move red tip to the right, you would choose Tip/first layer and then choose counterclockwise. This will bring yellow tip to the red face, red tip to the blue face and blue tip to the yellow face. Green will stay constant.
- If you choose the diagonal rotation, the faces you move will depend on the type of move you choose:
    - Choosing "tip in the lower left corner of red face" allows you to modify yellow, red, and green faces.
    - Choosing "tip in the lower right corner of red face" allows you to modify blue, red and green faces.

- Choosing “lower tip between yellow and blue” allows you to modify green, blue and yellow faces. Keep in mind that this lower tip between yellow and blue is actually the top tip of the green face.
- Once you chose a move, it worked similarly to the horizontal rotation in that you pick a layer and direction and your pyraminx will be updated accordingly and printed.
- Example: If you choose the diagonal move “tip in the lower left corner of red face,” and pick layer 1 and move it counterclockwise, the left corner of red will now be yellow. If you look at yellow face straight on, the right corner will be green. If you look at green face, it’s top tip will be red. Keep in mind that green face’s top tip is that corner.

Heuristic:

The heuristic we have chosen is to track the highest number of necessary moves for any given sticker to be returned to its initial/corresponding position. This means that our heuristic will equal the highest number of movements required by any sticker to return to its original position. If the heuristic value of all stickers equals zero, the puzzle is solved. This is an admissible heuristic, as the maximum number of movements required to solve the puzzle will never be less than the minimum number of moves needed to solve the puzzle. Thus, the heuristic value will always be at least the minimum number of moves required to solve the puzzle. To further illustrate this heuristic, let us use the case in which sticker A is four moves away from its starting position, and sticker B is one move away. The heuristic value would equal four moves, meaning at least four moves are needed to solve the puzzle.

Learning Outcomes: Joshna

This was our first time modeling any sort of a puzzle, and it was very helpful in demonstrating to us how important choosing a data structure can be. The data structure we used was the most intuitive way we could think of. We did discuss some other options, but settled on this because it readily made sense to us. That is not to say it doesn’t have its cons—accessing the stickers requires a long list index line each time—but my partner and I weighed the pros and cons and still chose this. In this sense, the project showed us how to present one’s idea to the other and compromise on what’s best for the team.

One of the other important things this project helped us recognize is that hard-coding is sometimes needed. Sometimes it’s the only way to go about a problem—as simple as that. And it also taught us to take a second to think about how we can make our code more efficient—even if it’s just going from 90 lines to 80 lines.

Lastly, it also showed us the importance of documenting the code. While I feel as though some parts of the code are hard to understand because of all the index mapping—that only makes sense when you visualize it somehow—you can also get a gist of what’s going on in each piece by looking at the doc strings and comments. We also made debugging a bit easier because each method’s argument types and return types are documented(even though python doesn’t restrict them).