Bo Hui Lu, Simon Tsui, Maxwell Vale APCS PD2
WeNeedAName

Rubik's Cube Visualizer

For our final project, we're planning to use Processing as a tool to visualize a Rubik's Cube. The cube will be interactive and the user will be able to move it in every way that it can be moved in real life. We know through research that both of these can be done with methods like box(), and utilizing the coordinates of the screen. With this, the user can attempt to solve one of the few preset cubes that we'll create manually beforehand. However, this simulator will also have an option for the user to input a specific arrangement of a Rubik's Cube(assuming its valid), onto a blank cube. After doing that, the user can once again try to solve it on their own.

The last functionality that we want to implement, but feel like would potentially cause a lot of trouble, is creating a Rubik's cube solver. After the user inputs their own Rubik's cube onto the blank cube, we hope to have something like a solve button in order to show a solution to the specific arrangement given. Though this may be difficult, all three of us know how to solve Rubik's Cubes, and that knowledge could prove helpful to implementing such solving algorithms in code. Not to mention, the use of 2D arrays for storing the sides, backtracking for the actual solving, and heaps for storing the numerous algorithms could be potential implementations.

If time permits, we may also add a 2x2 cube along with the standard 3x3 cube

Ideas for Concepts:

- 2D arrays will play a huge role: We can assign a number + letter to each "side" of the rubix cube. Using six 2D arrays with Strings, we can easily create an organized cube(not necessarily the one user sees in processing, it's just the one we'll use to interact with in code)
- If we decide to create an auto-solving algo, we can use a lot of backtracking: Like in the maze solving class, we can create steps for the cube to "walk" through. If suddenly a side breaks the color rules, backtrack and try another algo
- We learned quick methods to compare data in arrays. This would be useful especially when comparing sides of a rubix cube to check if its solved(all same color) or not.
- When solving a cube, there's sort of an order to doing so. We could potentially utilize stacks to store our algo steps for better progression throughout the solving process

To-Do List for Feature Implementation

1. . Learn about the 3D tools in Processing (how to work in 3D, specific built in methods for creating 3D objects)
2. Create the cube in Processing
    a. Overall shape using the built in box() method
    b. Coloring sides
    c. Dividing sides into individual pieces
        i. We are going to be making a 3x3 cube, but we may also try to implement other sizes, like the 2x2 or 4x4.
3. Move the cube in 3D space
    a. First, be able to rotate the cube using one of these methods
        i. Have the center of a face follow the mouse (AKA have the cube rotate around its center by following where the cursor is)
        ii. Have buttons on the screen that are dedicated to specific rotations of the cube (X, Y, Z)
        iii. Dedicate keyboard buttons to the rotation of the cube (again x, y, z)
    b. Next, work on turning the sides of the cube using on of these methods
        i. Use keyboard input to turn each side of the cube (for example, an "l" will turn the left side clockwise, while an "L" will turn the left side counterclockwise) *** Preferred
        ii. Create buttons on the screen to press to turn each side of the cube

4. Implement a scrambling method
    a. Want to be able to generate a random Rubik's Cube scramble so that the user may solve it.
5. **Possibly** allow for user input of the colors of each face of the cube and allow the program to solve the cube, showing the steps on the way
6. **Possibly** Create different sized cubes that can be selected