Rubik's Cube Assistant Report

Clarence Mariano

Link to Repository: https://github.com/rensrmari/Rubiks-Cube-Assistant

Background

My long-standing interest in the puzzle and desire to break down the solving process is the reason behind the idea to make a Rubik's cube assistant. I have come across many online solvers that, although providing the most efficient solution, simply hand to the user the operations that must be performed on the cube without further explanation. This program aims to hold the hand of the user by walking them through the various stages in the beginner's method, show them what moves to perform during each step, and what it does to the cube. The project also lends itself well to extensibility, for example allowing the implementation of more methods of solving, making it feasible for long-term investment. To me, the project seemed like it would be a blend of something I can enjoy working on for a while as well as something challenging enough for my skill level.

Design Decisions

One of the main aspects of the project that allow for variety in design is the representation of the Rubik's cube. For instance, I was presented with the choice of implementing it as individual, polymorphic cube objects that can be center, edge, or corner pieces. This significantly breaks down the problem of identifying the colors and types of pieces that are involved in solving the cube. However, I arrived at implementing it as a three-dimensional array of faces containing characters that represent colors. This way, fundamental operations on the cube, such as rotation and turning, are made more intuitive. For example, a right turn is just a shift in array values in a certain column. Locating the necessary pieces during the solving process was also made intuitive because it is similar to how people solve cubes in real life: locating a certain sticker and its adjacent stickers.

Moves are implemented as strings, as that is how they are represented in cubing. For example, a clockwise right turn is conventionally referred to as an "R", so such moves are direct carryovers. Additionally, it makes the inclusion of various Rubik's cube algorithms a trivial matter.

As I had implemented moves as strings, the logical thing for me to do when it came to storing the state of the cube is to store the strings of its scramble and applied moves. This ensures that the cube will always be valid. In hindsight, this design would not be efficient in scenarios where many moves have been applied. The long string would instead have to be checked for validity, and then the moves would need to be applied. A valid alternative would be storing the state as colors, as the cube only has a limited number of them, and then translating them to the array. This would entail checking if the cube is solvable, but it would be a considerable and far-reaching improvement in design.

C++ Concepts Used

The core C++ concepts that were used for this project are pointers, structs, streams, classes, searching algorithms, stacks, and templates. First, classes were used to separate the problem into easily tackle-able assignments; namely, file handling, cube-related operations, and solving. For my file handler class, heavy use of file streams was leveraged to interact with user-supplied text files, such as saving data to them and opening a file to gather cube-related information. File streams were also used to access files involved in testing the file handler class. String streams were employed for parsing the data in files as well as the sequences of moves to be done on the cube to ensure that all information is valid before processing it. Stacks came into use for implementing the undoing of moves, which involved processing the top of the stack. Linear search was used to locate certain pieces of the cube. Templates were used in the cube class to streamline the construction of strings that represent colors and faces, which involve maps of character-string and integer-string types respectively. Pointers were used by the assistant class to directly edit the state of the user's cube so that changes made by the assistant persist.

Challenges

The project's challenges lay in file handling and locating the cube's pieces. For file handling, there are many processes involved, such as validating the moves, tracking duplicate names, seeing if move counts match the moves. Because of these extensive responsibilities, I delegated some of them to the cube class, such as counting moves, through static member functions. As for locating the cube's pieces, finding them are integral parts of the solving process, and I found myself constantly needing to access a data member of the cube class from the assistant class. To resolve this, I implemented various generalized functions for finding a piece in the cube class. Henceforth, the process of getting a certain sticker, edge, or corner from a cube was simplified, allowing me to focus more on implementing the solving of the cube.

Insights Gained

One of the most insightful pieces of information that I gained from this project is unit testing. For my previous computer science projects, it sufficed to simply use the program to confirm that it is functional. However, unit testing adds another element of security, and I soon found out that it was not enough that the program had consistently run properly. By introducing tests for my classes' functions, I discovered some functions taking on more responsibilities than they should, making processes, albeit functional, tough to work with. This would be unpleasant to deal with if I decided to pursue the extension of the program in the future. Because of unit testing, I was able to ascertain that parts of my program run as expected and identify other parts where I could break down processes to make the program more manageable. When I came across a failing test, it often led to me needing to rewrite the implementation for related functions and sharpen their focus. Therefore, I also realized the importance of planning a function beforehand.