SPL-1 Project Report

PixelCube

Submitted by

Abb Muhaimeen

BSSE Roll No. : 1423

BSSE Session: 2021-2022

Submitted to

Kishan Kumar Ganguly

Assistant Professor

Institute of Information Technology

University of Dhaka

[illegible]

- - - - - - - - - - - - - - - - - -

Signature of the Supervisor

Institute of Information Technology

University of Dhaka

[17-12-2023]

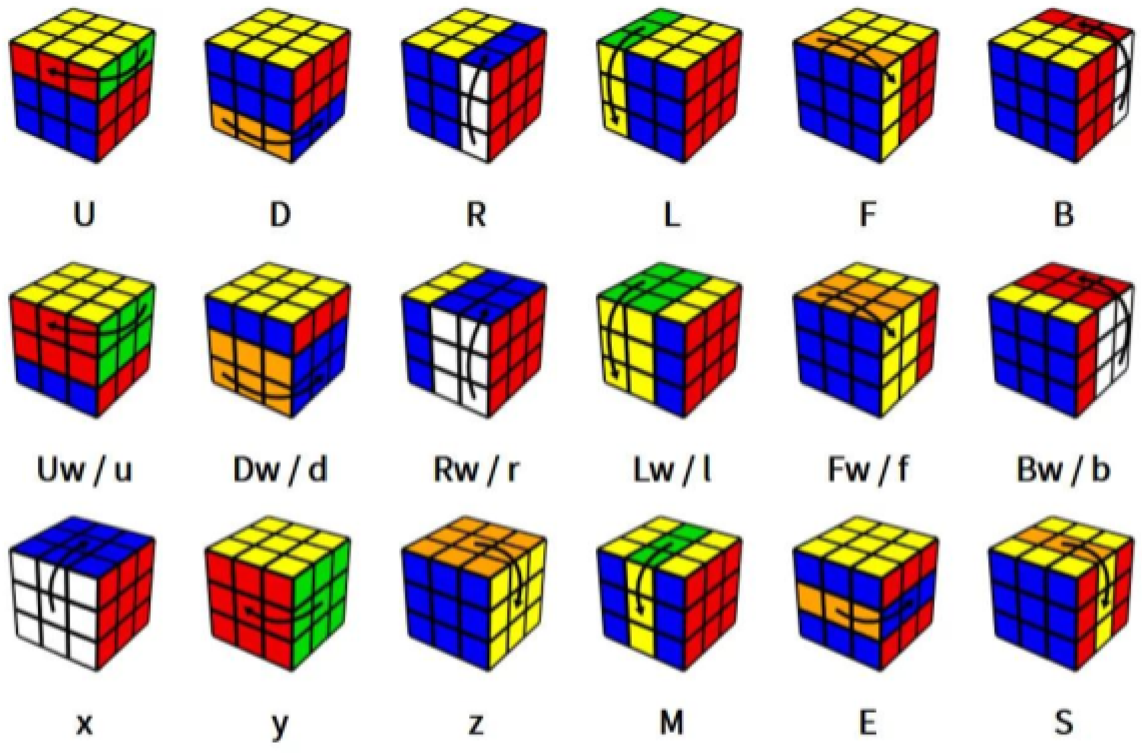
Table of Contents

1. Introduction

The PixelCube project works to solve the Rubik’s Cube, a combinatorial puzzle of 43 quintillion possible combinations by following a systematic approach called Layer-By-Layer method.PixelCube seeks to unravel the solution through algorithmic strategies and interactive solutions by being extremely beginner-friendly while keeping enthusiasts in mind.

2. Background of the Project

At the heart of the **PixelCube** project lies the Rubik's Cube, a three-dimensional combination puzzle that has captured the imagination of everyone since its invention by Ernő Rubik in 1974. The Rubik's Cube consists of six faces, each composed of nine colored stickers, with the challenge lying in restoring each face to a uniform color. With an astounding **43 quintillion** possible combinations, the Rubik's Cube stands as a shining flag of combinatorial complexity [1].



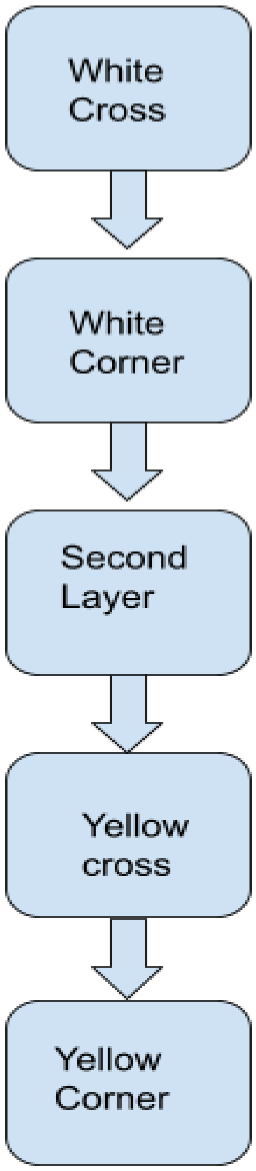
Move Sets of a Rubik's Cube

3. Description of the Project

PixelCube, a Rubik's Cube solver program, ventures into combinatorial puzzles and algorithmic problem-solving. At its core is the innovative layer-by-layer solving method, a systematic and approachable strategy for conquering the complexities of the Rubik's Cube. This description unfolds the whole mechanics of **PixelCube** step by step.

Layer-by-Layer Methodology

- **Breaking Down the Complexity**



The layer-by-layer method forms the crux of PixelCube's solving strategy. Unlike more advanced methods that tackle the entire cube simultaneously, the layer-by-layer approach simplifies the challenge by breaking it into manageable steps. Each layer is solved independently, progressively building toward a complete solution.

- **Solving the Cross (First Layer):**

In the first step, **PixelCube** aligns the edges of one face to form a cross pattern with a distinct color.**PixelCube** utilizes predefined algorithms to manipulate the cube's edge pieces strategically. The algorithms focus on moving and orienting the edge stickers to create a cross on the chosen face.

-Example Moves:

A typical move might involve rotations and turns such as F (Front), U (Upper), and R (Right) to align the edges correctly.

- **Filling in the Corners (First Layer):**

After that, It positions the corner pieces on the same face as the cross to complete the first layer. **PixelCube** employs algorithms specifically designed for corner manipulation. These algorithms ensure the proper placement of the corner stickers to create a cohesive first layer.

- Example Moves:

Sequences of moves like R U R' U' are executed to maneuver and orient the corner pieces.

- **Moving to Subsequent Layers:**

Then, it gradually builds layers on top of the completed first layer. **PixelCube** follows a systematic approach, addressing one layer at a time. Algorithms are applied iteratively to solve each layer, ensuring the preservation of previously solved layers.

- Example Moves:

For the second layer, moves like F R U R' U' F' might be used to position and align the edge pieces.

- **Completing the Final Layers:**

Last but not least, it extends the layer-by-layer approach until all six faces of the cube are solved. **PixelCube** orchestrates a sequence of moves for each layer, maintaining a coherent strategy. Algorithms for the last layers often involve more intricate sequences.

- Example Moves:

Moves like the **T-permutation** (a sequence of moves involving the U face) might be used to finalize the last layer.

4. Implementation

I am using a 2d representation of a 3D cube and the below figure shows how that is represented. I have used an array of 54 where each cube block represents a fixed array.

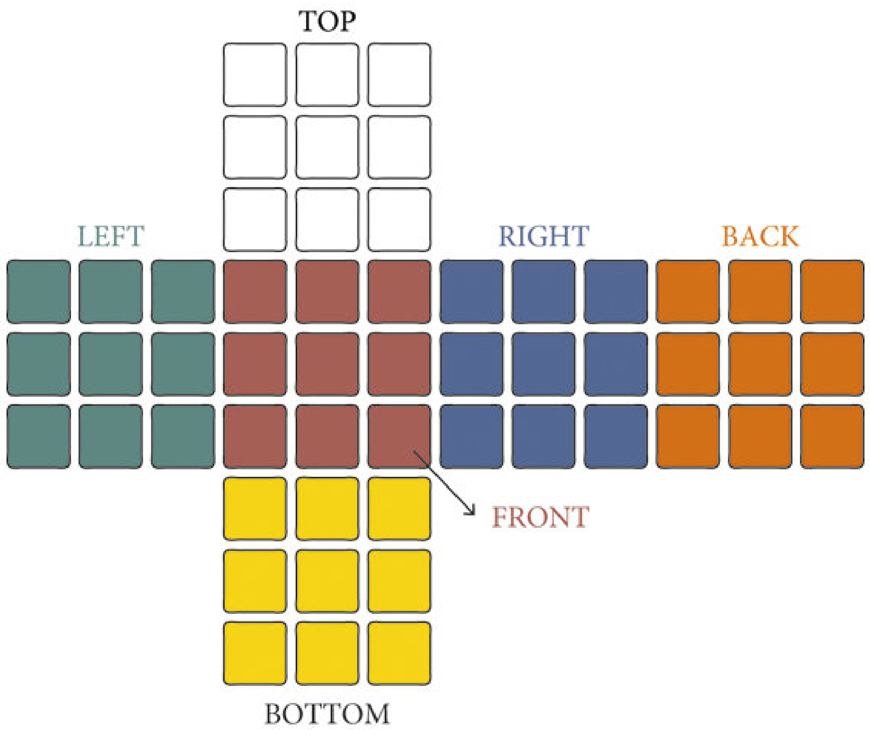
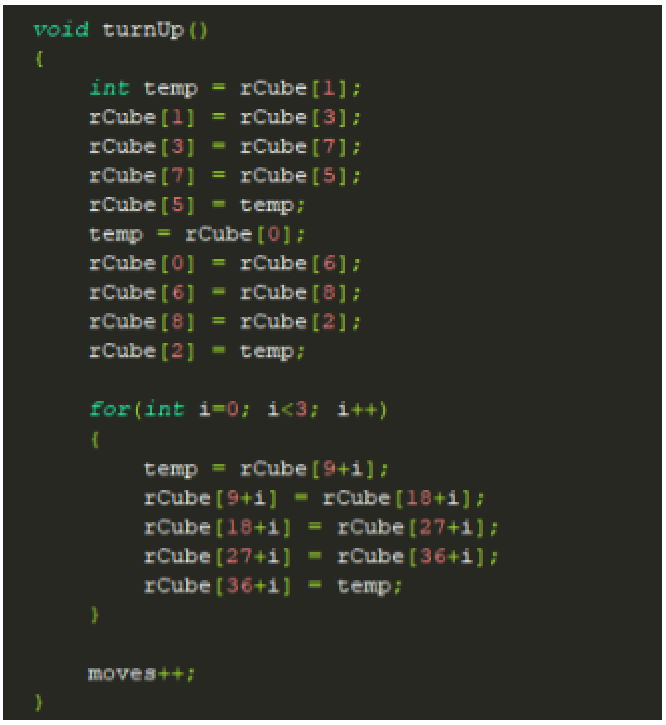


Fig: 2D representation of Rubik's Cube

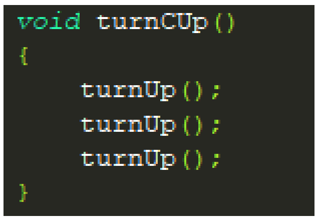
The Cube is printed out in this fashion to understand it clearly. Then I have initialized every possible move of the cube and tried to optimize it as much as possible. Example: The below figure shows how Up move is initialized:



```
void turnUp()
{
    int temp = rCube[1];
    rCube[1] = rCube[3];
    rCube[3] = rCube[7];
    rCube[7] = rCube[5];
    rCube[5] = temp;
    temp = rCube[0];
    rCube[0] = rCube[6];
    rCube[6] = rCube[8];
    rCube[8] = rCube[2];
    rCube[2] = temp;

    for(int i=0; i<3; i++)
    {
        temp = rCube[9+i];
        rCube[9+i] = rCube[18+i];
        rCube[18+i] = rCube[27+i];
        rCube[27+i] = rCube[36+i];
        rCube[36+i] = temp;
    }

    moves++;
}
```

Fig: Function denoting Turning the Upper layer once in clockwise

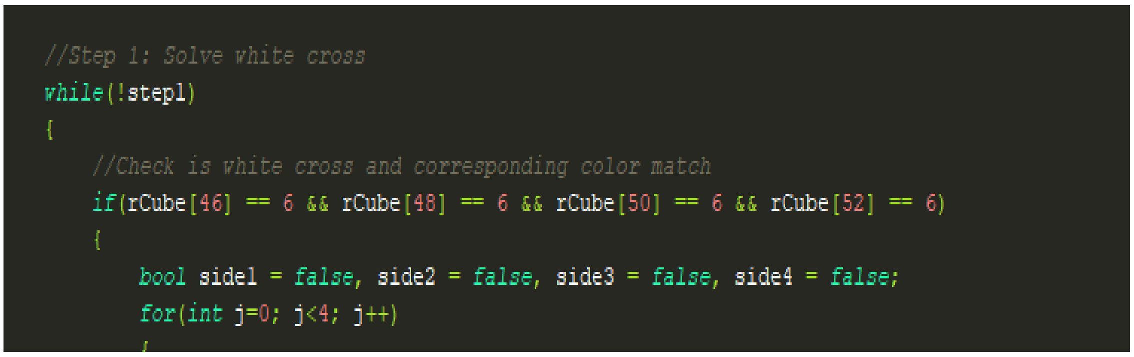
The following example shows how every move is optimized like turning the upper layer anticlockwise can be written as a separate function but I used UP function thrice which essentially does the same thing:



```
void turnCUp()
{
    turnUp();
    turnUp();
    turnUp();
}
```

Fig: AntiClockwiseUp

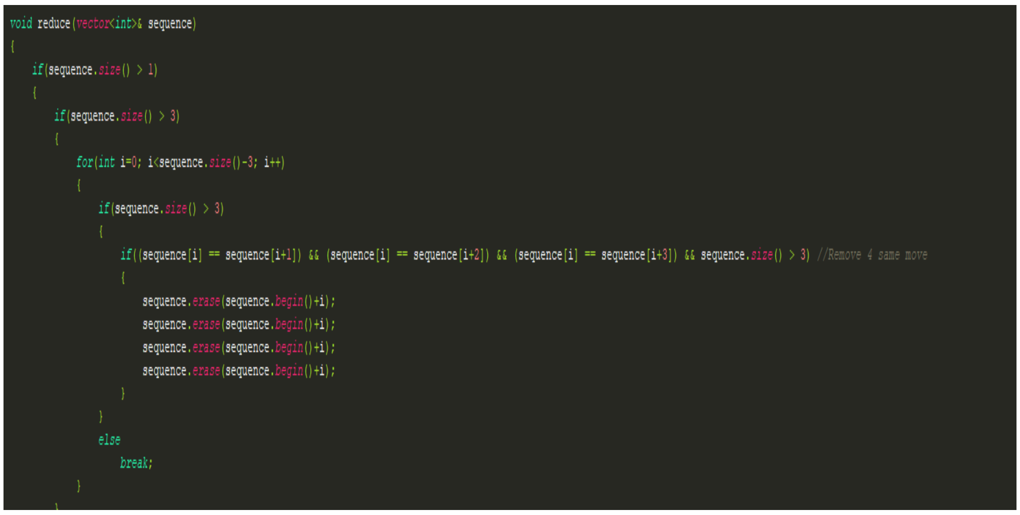
Solve function works on brute force method and checks every step if its complete and then proceeds to the next step.



```
//Step 1: Solve white cross
while(!step1)
{
    //Check is white cross and corresponding color match
    if(rCube[46] == 6 && rCube[48] == 6 && rCube[50] == 6 && rCube[52] == 6)
    {
        bool side1 = false, side2 = false, side3 = false, side4 = false;
        for(int j=0; j<4; j++)
```

Fig:Solve Function

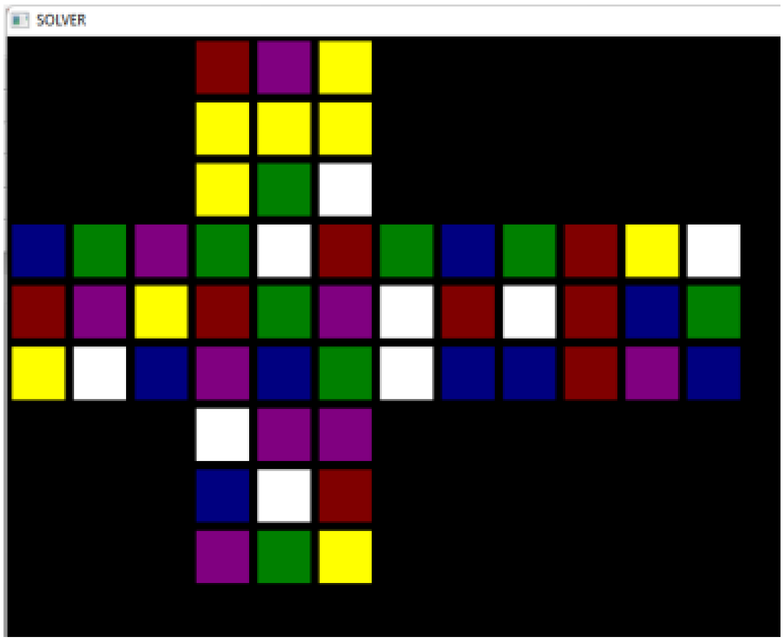
Reduce function searches for repetition of moves and then replaces them with much more optimal move:



```
void reduce(vector<int>& sequence)
{
    if(sequence.size() > 1)
    {
        if(sequence.size() > 3)
        {
            for(int i=0; i<sequence.size()-3; i++)
            {
                if(sequence.size() > 3)
                {
                    if((sequence[i] == sequence[i+1]) && (sequence[i] == sequence[i+2]) && (sequence[i] == sequence[i+3]) && sequence.size() > 3) //Remove 4 same move
                    {
                        sequence.erase(sequence.begin()+i);
                        sequence.erase(sequence.begin()+i);
                        sequence.erase(sequence.begin()+i);
                        sequence.erase(sequence.begin()+i);
                    }
                }
                else
                    break;
            }
```

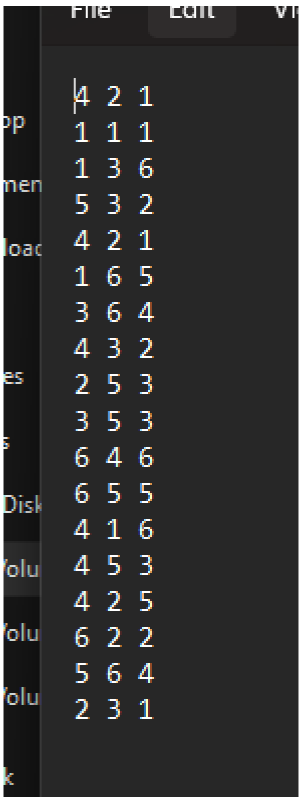
Fig: Reduce function

5. User Interface



Graphical Representation of the given cube

The following input from the file where each 1-6 digit represent a color:



```
4 2 1
1 1 1
1 3 6
5 3 2
4 2 1
1 6 5
3 6 4
4 3 2
2 5 3
3 5 3
6 4 6
6 5 5
4 1 6
4 5 3
4 2 5
6 2 2
5 6 4
2 3 1
```

Fig:Load cube from file

Starting Menu:

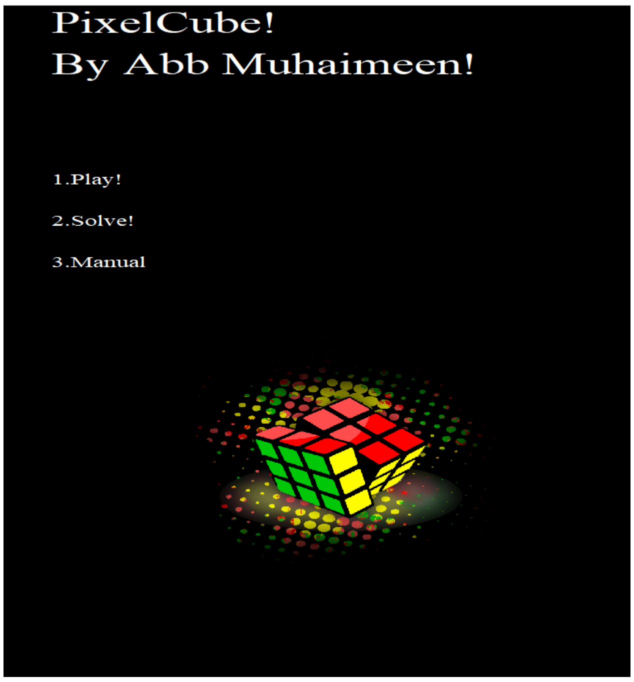
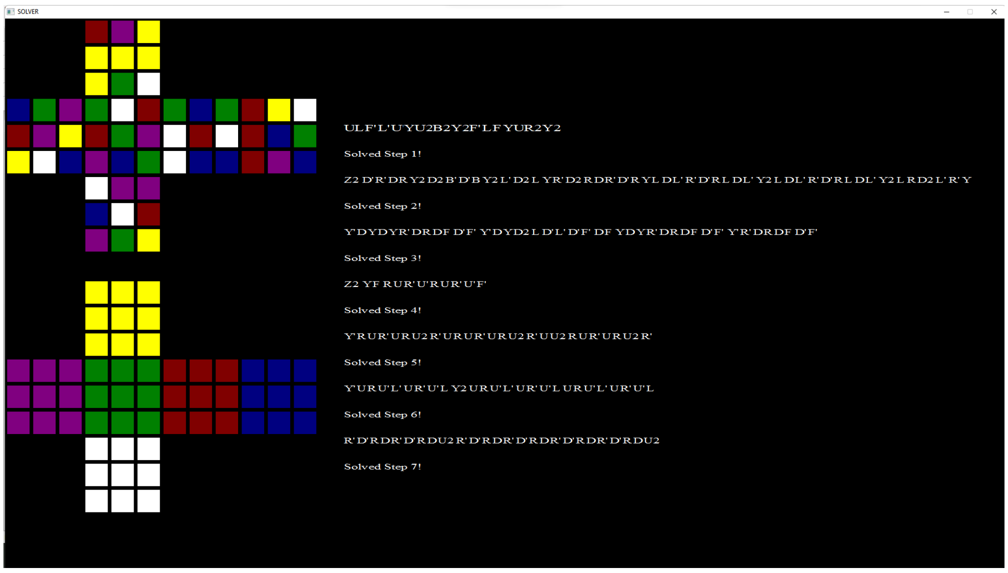


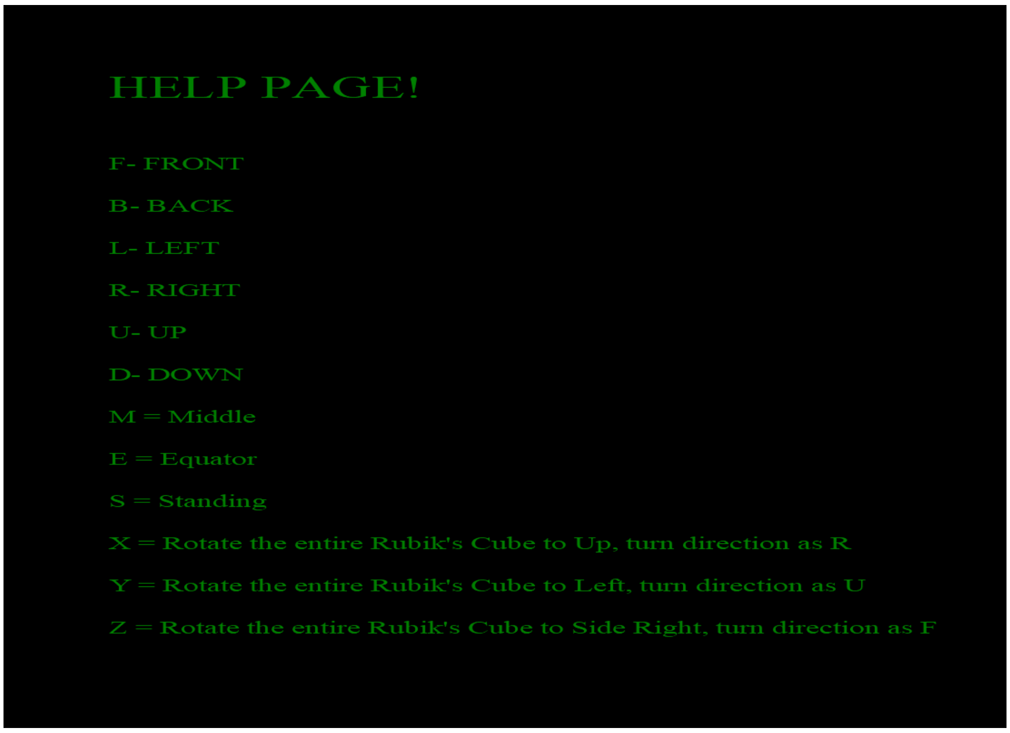
Fig: GUI Menu

The given cube can be solved using the OUTPUT as the form of a solution to the cube:



```
ULF'L'U'YU2B2Y2F'LF YUR2Y2

Solved Step 1!

Z2 D'R'DR Y2 D2 B'D'B Y2 L'D2 L YR'D2 RDR'D'R YL DL'R'D'RL DL' Y2 L DL'R'D'RL DL' Y2 L RD2 L'R' Y

Solved Step 2!

Y'DYDYR'DRDF D'F' Y'DYD2 L D'L'D'F' DF YDYR'DRDF D'F' Y'R'DRDF D'F'

Solved Step 3!

Z2 YF RUR'U'RUR'U'F'

Solved Step 4!

Y'RUR'URU2 R'URUR'URU2 R'UU2 RUR'URU2 R'

Solved Step 5!

Y'URU'L'UR'U'L Y2 URU'L'UR'U'L URU'L'UR'U'L

Solved Step 6!

R'D'RDR'D'RDU2 R'D'RDR'D'RDR'D'RDR'D'RDU2

Solved Step 7!
```

Fig: Steps to reach the solved cube



HELP PAGE!

F- FRONT

B- BACK

L- LEFT

R- RIGHT

U- UP

D- DOWN

M = Middle

E = Equator

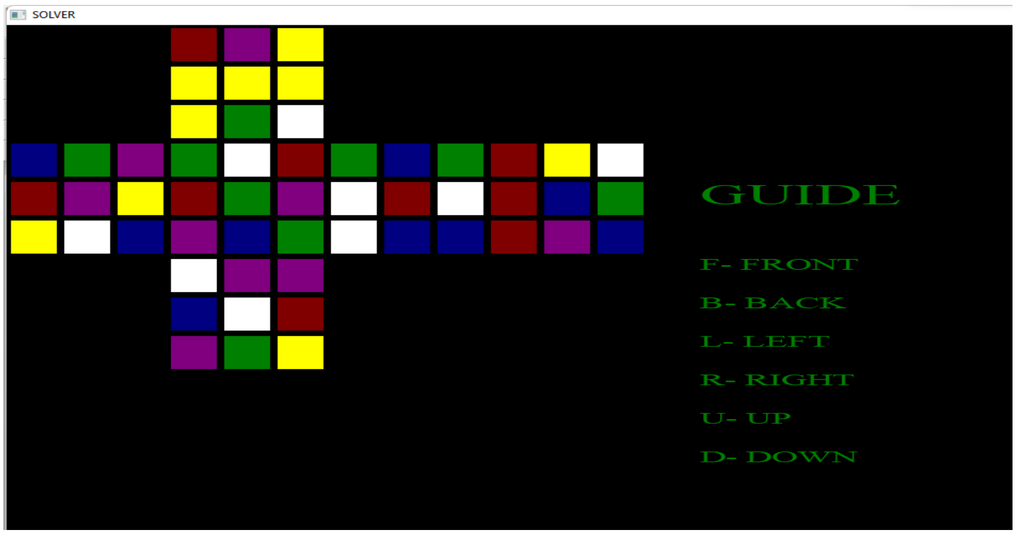
S = Standing

X = Rotate the entire Rubik's Cube to Up, turn direction as R

Y = Rotate the entire Rubik's Cube to Left, turn direction as U

Z = Rotate the entire Rubik's Cube to Side Right, turn direction as F

` Fig: Help Page



SOLVER

GUIDE

F- FRONT

B- BACK

L- LEFT

R- RIGHT

U- UP

D- DOWN

Fig: Users can also play with the Cube dynamically

6. Challenges Faced

** Algorithm Complexity:

Challenge: Developing efficient algorithms for each solving step can be complex, especially as moving beyond the beginner's method.

Overcoming: Research and study established solving algorithms. Break down the problem into smaller components and gradually build more efficient solutions.

Data Structure Selection:

Challenge: Choosing the right data structures to represent and manipulate the Rubik's Cube state can be challenging.

Overcoming: Experimenting with different data structures like arrays, matrices, or linked structures to find the most suitable. Example: I first used a 3D array but then used a one dimensional array for fixed position to maneuver easily.

User Interface Design:

Challenge: Designing an intuitive and user-friendly interface, especially if the project includes a graphical representation of the cube, can be challenging.

Overcoming: Learning GUI and prioritizing simplicity and clarity.

Optimization:

Challenge: Optimizing the solving algorithms to minimize the number of moves may require significant effort.

Overcoming: Employing optimization techniques, considering look-ahead methods, and studying existing optimized algorithms. Giving a balance between simplicity and efficiency.

Edge Cases and Bugs:

Challenge: Handling edge cases and debugging errors, especially in complex algorithms, can be time-consuming.

Overcoming: Implementing thorough testing, including edge cases. Use debugging tools and methods to trace and fix errors systematically.

Learning Curve:

Challenge: Learning and implementing advanced solving methods may have a steep learning curve.

Overcoming: Taking a systematic approach to learning, starting with simpler methods and gradually advancing. Utilizing online resources, tutorials, and forums for guidance.

7. Conclusion

The project was vast. So, naturally, it provided me with various learnings:

Understanding and implementing different solving methods, such as layer-by-layer or advanced algorithms, enhances algorithmic problem-solving skills**.** Manipulating the Rubik's Cube involves managing its state. Learning to use appropriate data structures for efficiency is a valuable skill. The project mostly involves optimizing algorithms to reduce the number of moves required. Learning optimization techniques contributes to algorithm efficiency.

As PixelCube includes a graphical user interface (GUI), skills in UI development and user experience design are acquired. Developing a Rubik's Cube solver requires proficiency in programming languages, strengthening coding skills.

Also, the project can be enhanced in the future in many ways. Incorporating more advanced solving methods like CFOP or Kociemba's Algorithm for faster and more efficient solving, training a machine learning model to learn solving strategies from historical solving data, potentially creating an adaptable solver. Extend the project to an 3D application where users can see virtual cubes and receive real-time solving guidance can also be a notable extension. Implementing the capability to solve multiple interconnected cubes simultaneously, a more complex but intriguing combinatorial challenge.

Introduce features that allow users to compete, share solving strategies, or engage in virtual Rubik's Cube competitions. Consider making the solver accessible to users with disabilities, incorporating voice commands or haptic feedback. Develop interactive tutorials within the application to teach users different solving methods and the underlying algorithms. The journey with PixelCube provides not only a solution to the Rubik's Cube but a foundation for continuous learning and innovation. By extending the project along the suggested lines, it can evolve into a versatile and dynamic tool for both Rubik's Cube enthusiasts and learners interested in algorithmic problem-solving.

References

[1] Ernő Rubik. (1974). "Cubik's cube." U.S. Patent No. 3,655,201.

My Github Repository-
https://github.com/muhaimeen90/Spl-1

Layer by Layer Method-
https://en.wikipedia.org/wiki/Layer_by_Layer