Hemvati Nandan Bahuguna Garhwal University
(A Central University)
Srinagar Garhwal, Uttarakhand
School of Engineering and Technology

Session: 2020 - 2021

A PROJECT REPORT ON

"**CubeX: A Rubik's Cube Solver**"

submitted in partial fulfillment for the award of degree of
Bachelor of Technology
in Computer Science and Engineering
HNBGU, Srinagar Garhwal (Uttarakhand)

Guided By:
Dr. Prem Nath
Associate Professor
Department of Computer Science & Engg.

Submitted By:
Nikhil Singh
Roll No. - 19134501024
Class - B.Tech (CSE)
Semester – Fourth

DECLARATION

I, **Nikhil Singh** having **Roll No. 19134501024**, student of Computer Science and Engineering Department at Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal), Uttarakhand, submits this report entitled **"CubeX: A Rubik's Cube Solver"** to Computer Science and Engineering Department, Hemvati Nandan Bahuguna Garhwal University, for the award of the **Bachelors of Technology (B.Tech) degree in Computer Science and Engineering (CSE)** and declaring that the work done is genuine and produced under the guidance of **Dr. Prem Nath**, Department of Computer Science and Engineering, Hemvati Nandan Bahuguna Garhwal University.

I further declare that the report work in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree in this institute or any other institute/university.

Date: 23/09/2021 **Nikhil Singh**
Place: Srinagar **Roll No. 19134501024**

CERTIFICATE

This is to certify that, this project report titled **“CubeX: A Rubik’s Cube Solver”** submitted by **Nikhil Singh**, bearing **Roll No. 19134501024** is a bonafide record of the work carried out by him in partial fulfilment for the requirement of the award of **Bachelor of Technology (B.Tech) in Computer Science and Engineering (CSE)** degree from Hemvati Nandan Bahuguna Garhwal University (A Central University) at Srinagar (Garhwal), Uttarakhand.

Dr. Prem Nath

Department of Computer Science & Technology
Hemvati Nandan Bahuguna Garhwal University (A Central University)
Srinagar (Garhwal), Uttarakhand

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all people for sprinkling their help and kindness in the completion of this Project. I would like to start this moment by invoking my purest gratitude to **Dr. Prem Nath** { Department of Computer Science and Engineering, Hemvati Nandan Bahuguna Garhwal University (A Central University) }, my project instructor.

The completion of this project would have been impossible without his expertise and invaluable guidance. He helped me in every hour of need while working on this project.

I would like to thank **Prof. M.M.S Rauthan**, **Prof. Y.P Raiwani**, all the lab assistants and other staff of Computer Science and Engineering Department, Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal), Uttarakhand, for their kind support. Last but not the least, I would like to thank my family and friends for their unwavering belief despite ups and downs in this journey.

ABSTRACT

As my minor project, I have created a system that helps users solve a Rubik's cube from any starting condition. The starting configuration of the cube is entered onto a cube net displayed on the computer monitor by using lab kit buttons to select the appropriate color for each cubelet's face.

Once a valid starting configuration is provided, a 3D-visualization of the Rubik's cube helps guide the user towards solving the cube. Each time a designated button on the lab kit is pressed, one section of the displayed Rubik's cube rotates and the colors of the cubelet's faces are updated to reflect the current state of the cube.

The algorithm used to determine the correct sequence of the sequence of moves is a well-known seven step method for solving the cube. If users accurately perform each rotation they see in visualization on their physical cube, they should have a solved Rubik's cube in less than 200 rotations.

Table of Contents

List of Figures / Charts

1. Introduction

1.1. Motivation

The Rubik's cube is a popular puzzle that many people struggle to solve. It is difficult to provide an easy-to-follow algorithm to solve the cube on the sheet of paper, so the solution manual that comes with the kit can often be tough to follow. Algorithms posted on youtube.com (link - https://www.youtube.com/watch?v=qujb-A2cKs4) is slightly more helpful, but if the goal of the user is just to return the cube to its solved state without having to learn a generalized solving method, these videos are unnecessarily complex. The goal of my project is to provide a step-by-step animation on a computer monitor of the rotations necessary to solve a Rubik's cube from any starting configuration that the user inputs.

Figure 1. A 2x2x2 Rubik's cube. It is popularly known as pocket cube.

(Source: appadvice.com)

1.2. Overview

The user must first must enter the starting configuration of the cube, and to aid this process, a cube net as shown in Figure.2 is displayed on a computer monitor. A Rubik's cube is composed of 8 smaller cubelets and the 26-cubelet faces that are visible on the cube are displayed in this cube net. The colors of each of the center cubelet's faces are fixed in the cube net, helping orient the user. It is reasonably likely that the user will make an error when inputting the cube configuration, so our system runs a check on the desired starting configuration to make sure it is solvable before allowing the user to proceed. The user cannot trigger rotations on the 2D-visualization of the cube until the configuration is valid.

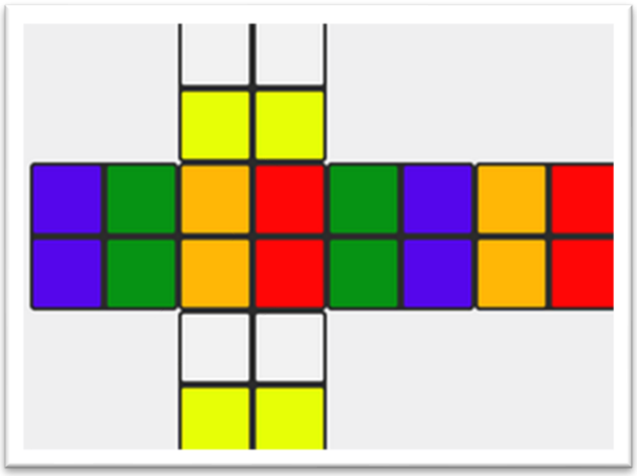


Figure 2. Cube net. The user enters the starting configuration of the Rubik's cube on a cube net. This is a screenshot of a valid starting cube configuration.

(Source: quora.com)

The process of solving a cube with a valid starting configuration is communicated to the user via a 2D-animation displayed on the computer monitor. A screenshot of this visualization is included in Figure.3. This animation consists of a Rubik's cube whose face rotates to illustrate the solution. Following each rotation, which the user triggers by pressing a button on the lab kit, the colors of the visible cubelet's faces update to reflect the current configuration of the cube. The perspective of the cube itself does not change throughout the animation, meaning that the cube as a whole will not move, but rather individual sections of the cube will move. Users should be able to return their cube to a solved state if they are able to perform each of the rotations shown in the visualization while maintaining the same cube orientation throughout.

My system of course includes an implementation of an algorithm that generates the sequence of rotations that will lead to a solved cube. There exist many known algorithms for solving the Rubik's Cube and in general there is a tradeoff between the number of potential moves that must be memorized (or stored in memory in this case) and the number of rotations that are ultimately required to solve the cube. For example, it has been shown that any Rubik's cube configuration can be solved in fourteen (for 2x2x2 Rubik's cube) or fewer moves, but large amount of memory would be required to store the sequence to store the sequence of moves for each starting configuration simply because there are a total of 3674160 starting configurations (in case of 2x2x2 Rubik's cube). The algorithm I choose requires a relatively small amount of memory for move sequences, and fewer than 200 rotations are needed.

1.3. High Level Structure

This system can be easily divided into three main blocks, namely Get Initial State, Solve Cube, and Show Animation blocks as shown Figure.3. The Get Initial State block collects the starting configuration of the Rubik's Cube using the cube net. This information gets sent to the Solve Cube block, which both checks the starting configuration to see if it is valid and determines the sequence of rotations necessary for solving the cube. The Show Animation block communicates these rotations to the user via a 2D-Rubik's cube that is displayed on a computer monitor.

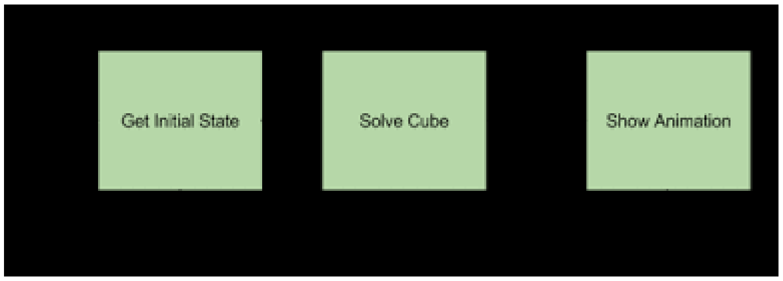


Figure 3. Block diagram. Our implementation is divided into three basic states that gather the initial cube configuration, solve the cube, and display the animation.

(Source: mit.edu)

To manage the communications among these three blocks, protocols are used to represent the current configuration of the Rubik's cube and to describe the rotations of various faces of the cube.

2. Rubik's Cube Notation

Throughout this report, I will be using the following notations (Figure.4) to refer to the sides of the cube.

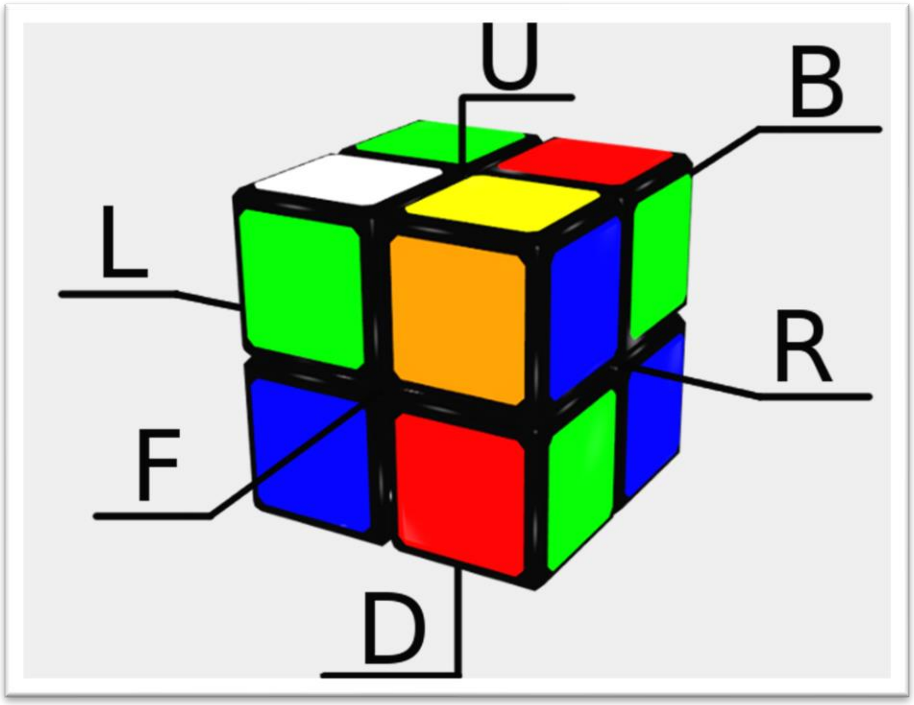


Figure 4. Notations used on Rubik's Cube.

(Source: iberorubik.com)

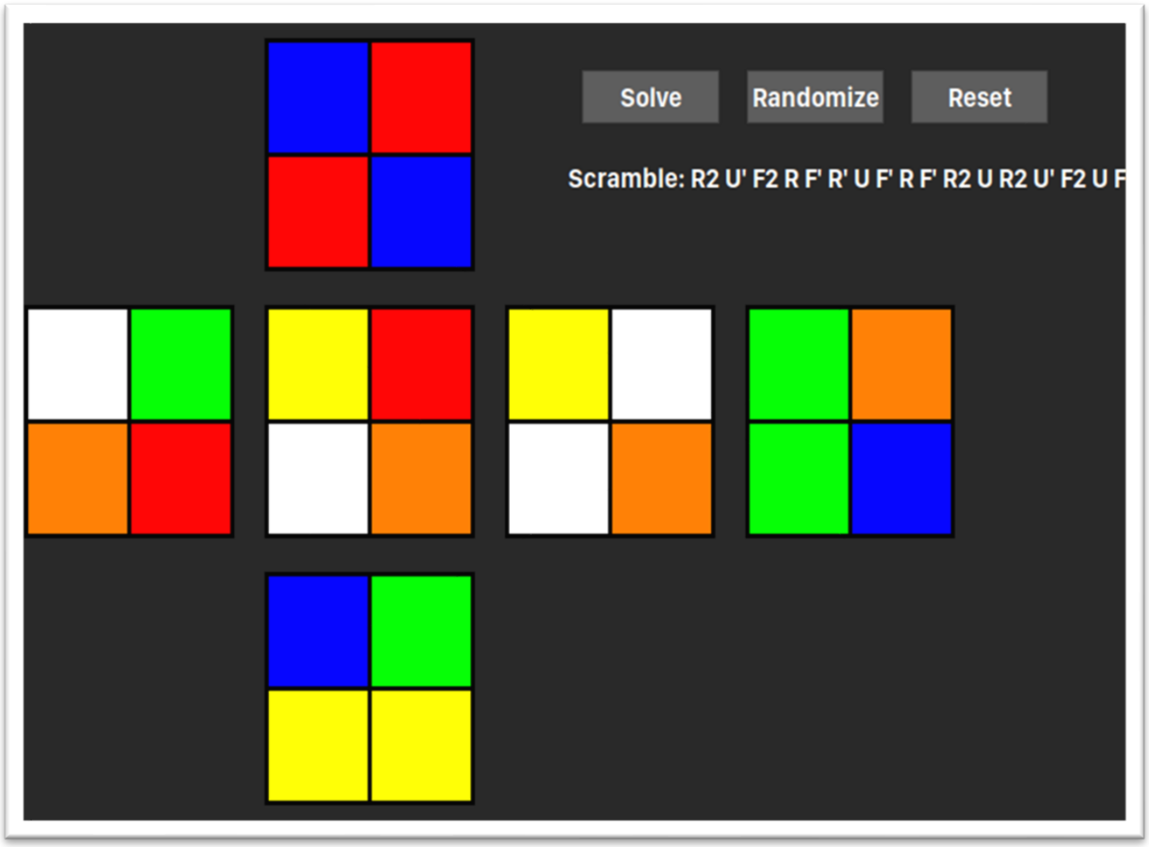
F = Front	**B** = Back
U = Up	**D** = Down
L = Left	**R** = Right

The same notation will be used to refer to face rotations. For instance, F means to rotate the front face 90 degrees, clockwise. A counterclockwise rotation is denoted by (F'). A 180-degree turn is denoted by (F2).

3. Protocols

3.1. Cube Configuration Representation

The protocol for representing the current cube configuration encodes the color of each cubelet face and is depicted in Figure.5. It is used by the Get Initial State block to communicate the starting configuration to the Solve Cube block, and it is used by the Solve Cube block to communicate the current configuration of the cube to the Show Animation block. A cubelet is one of the 8 smaller cubes that together constitute the Rubik's cube. The Rubik's cube has six differently colored stickers, allowing these colors to be represented with three bits: red (001), orange (010), yellow (011), green (100), blue (101), and white (110). The other colors used in the system are grey (000) which is used for blank cubelets, and black (111) which is used for the background and the gaps between cubelets. The ordering that we chose relies on the orientation of each Rubik's cube face as seen in the animation. This orientation is constant, for the cube as a whole never rotates during the animation, but rather movements are restricted to the rotation of certain small sections of the cube. The six orientations of the faces are up, down, front, back, right, and left.



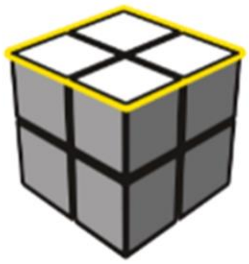
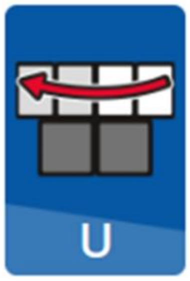
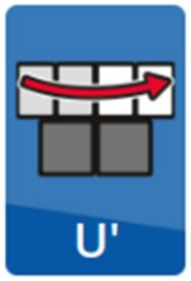
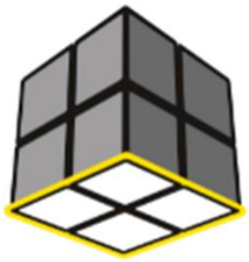
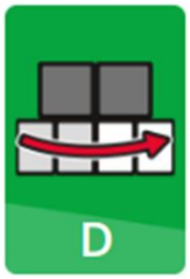
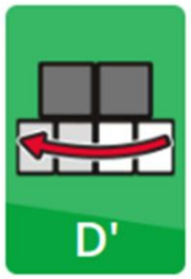
Scramble: R2 U' F2 R F' R' U F' R F' R2 U R2 U' F2 U F

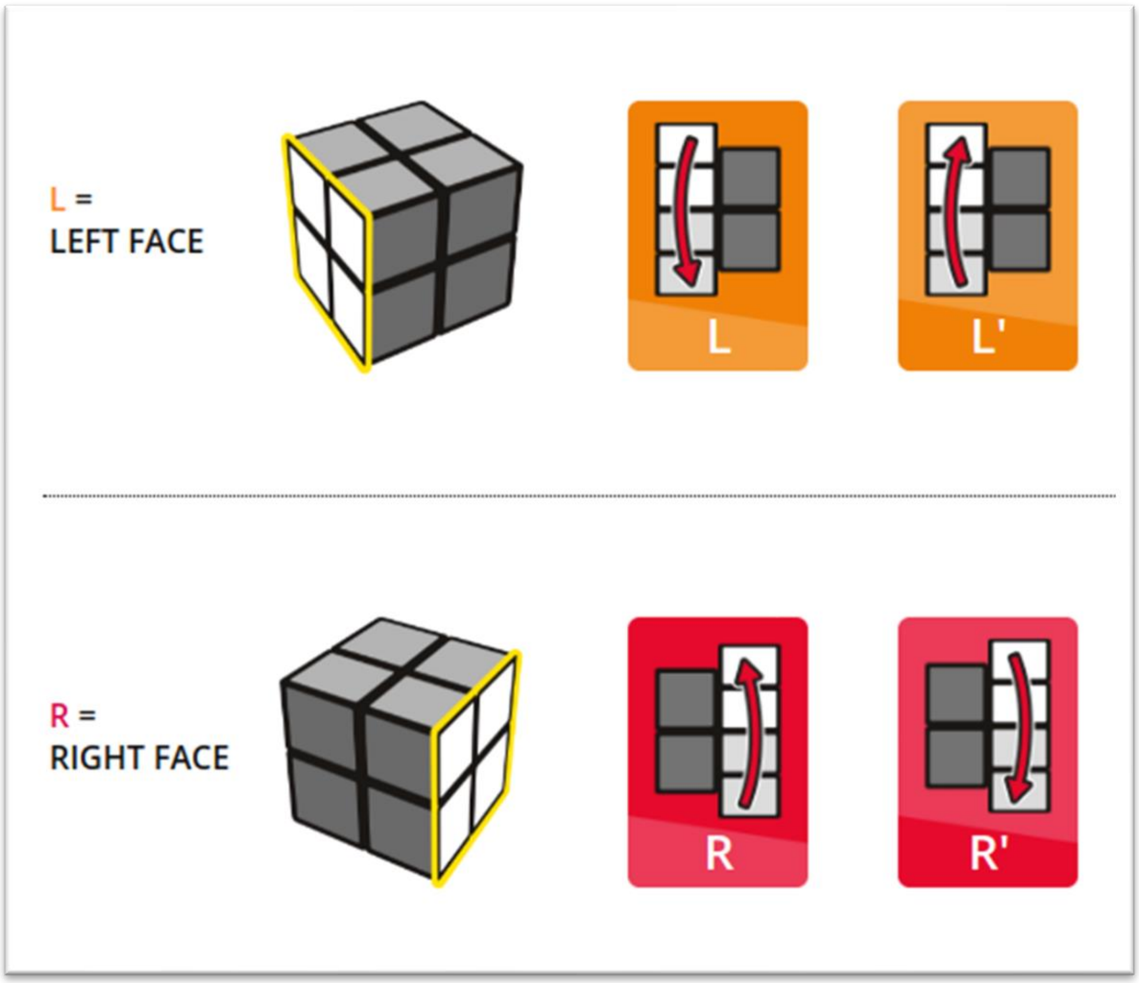
Figure 5. Animated cube net. This is a screenshot from the project with initial configuration.

(Source: from the project)

3.2. Rotation Representation

The second protocol assigns numerical values to the possible rotations that can occur, and the Solve Cube and Show Animation blocks rely on this protocol to communicate effectively. There are 12 distinct possible rotations that are depicted in Figure.6 along with their assigned number, which can be encoded with 5-bits, and their assigned variable name. An example of an assigned variable name is MFR (an acronym for Move Front Face Right), which corresponds to rotating the entire front face clockwise.

FACE KEY		**ALGORITHM KEY**	
Each face is represented by a letter.		Moves used in this guide.	
U = UP FACE			
D = DOWN FACE			


L =
LEFT FACE
L
L'
R =
RIGHT FACE
R
R'

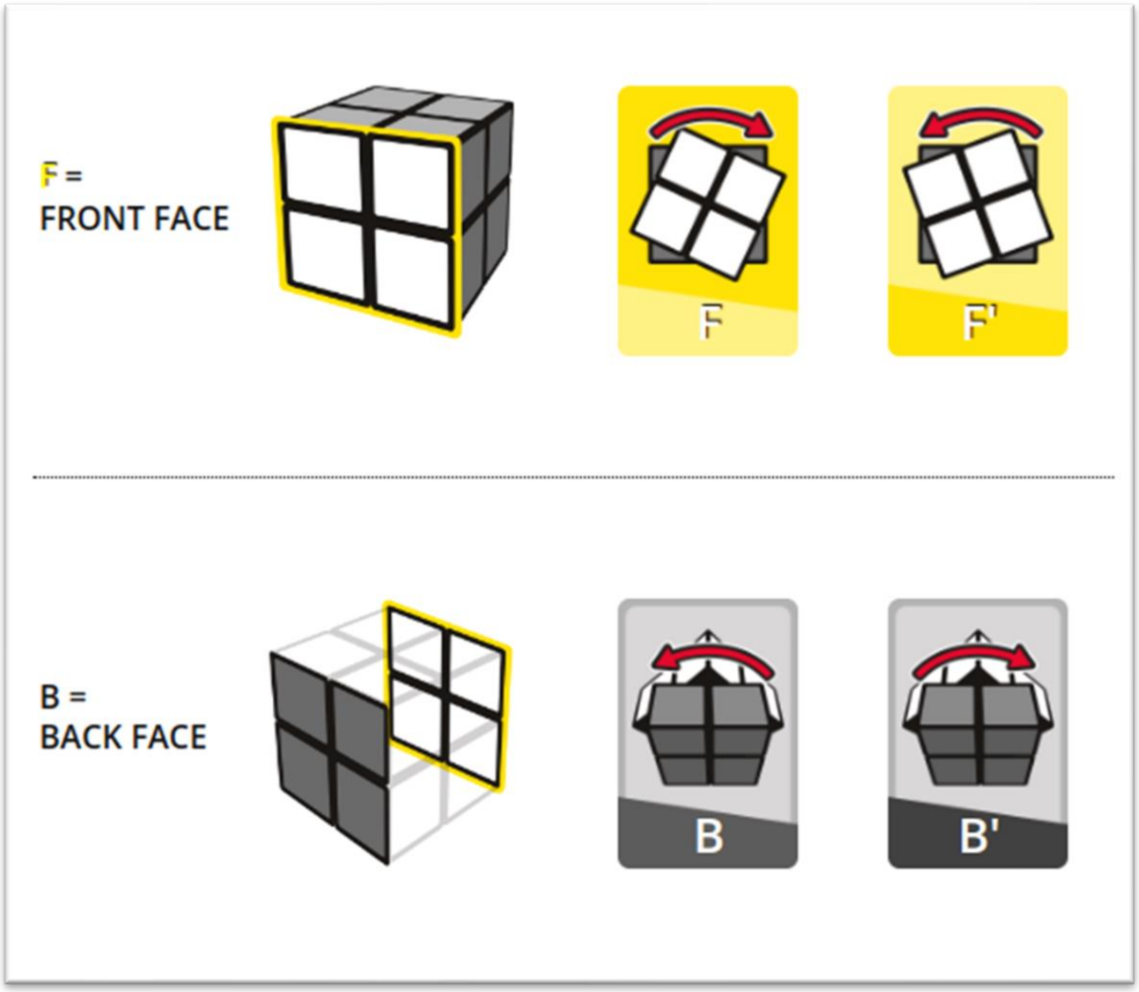


Figure 6. Rubik's cube rotation protocol. There are a total 12 distinct rotations.

(Source: youcandothecube.com)

4. System Requirements

4.1. Hardware Requirements

- **RAM:** 1GB or higher
- **CPU:** Intel Core i3-3210 3.2 GHz / AMD A8-7600 APU 3.1 GHz or equivalent.
- **GPU (Discrete):** Nvidia GeForce 400 Series or AMD Radeon HD 7000 series with OpenGL 4.4.
- **Secondary Storage:** 2 GB or higher

4.2. Software requirements

- **Programming Language:** Oracle Java 8.0
- **Web Browser:** Microsoft Edge, Mozilla Firefox, Google Chrome, Apple Safari, etc.
- **Operating System:**
    - **Microsoft Windows:** Windows 10 (8u51 and above), Windows 8.x, Windows 7 SP1, etc.
    - **Linux:** Ubuntu Linux 12.04 LTS and higher, Arch Linux, etc.
    - **macOs:** Intel-based Mac running Mac OS X 10.8.3+ or higher, Apple M1-chip based macOs, etc.

5. Data Structures and Algorithms

5.1. Implicit Graph Data Structure

In the study of graph algorithms, an implicit graph representation (or more simply implicit graph, Figure.7) is a graph whose vertices or edges are not represented as explicit objects in a computer's memory, but rather are determined algorithmically from some other input, for example a computable function.

Neighborhood Representation using Implicit Graph: Here the notion of implicit graph is used in Bidirectional Breadth First Search, which is described in terms of graph. In this context, an implicit graph may be defined as a set of rules to define all neighbors for any specified vertex. In this case of Rubik’s cube, in searching for a solution each vertex of Rubik’s cube represents one of the possible states of the cube, and each edge represents a move from one state to another. It is straightforward to generate the neighbors of any vertex by trying all possible moves in the puzzle and determining the states reached by each of these moves. However, an implicit representation is necessary, as the state space of Rubik's cube is too large to allow an algorithm to list all of its states. It can be easily visualized from Figure.8.

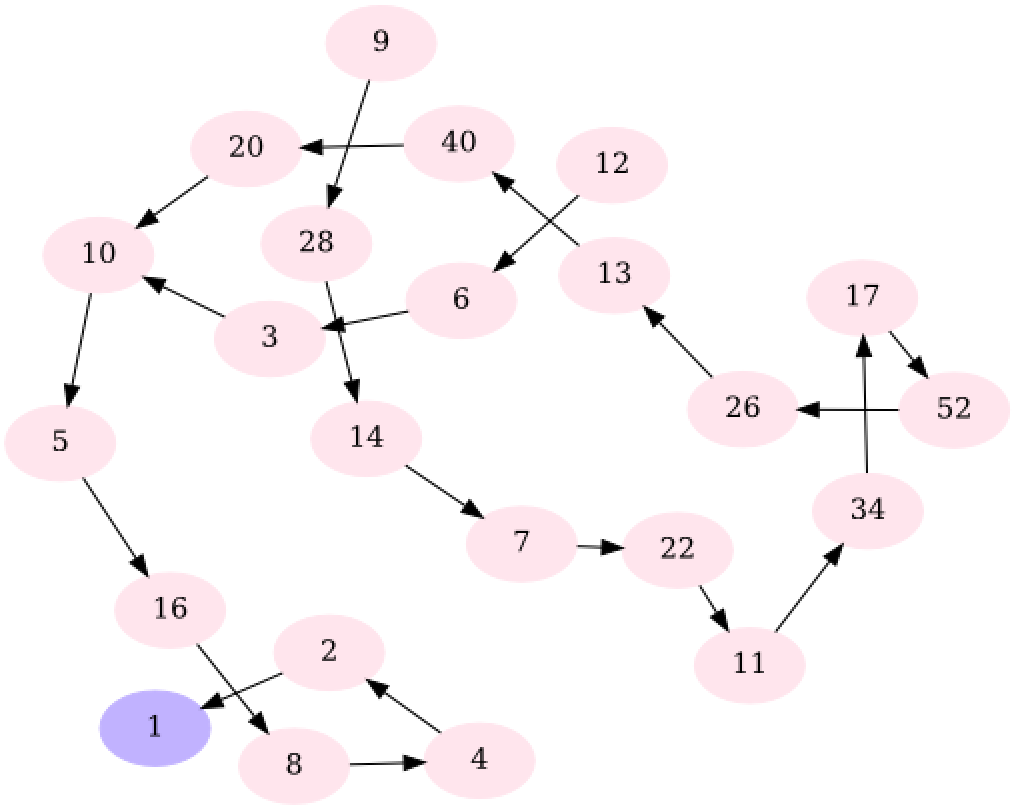


Figure 7. **Implicit graph data structure. Used in solving the Rubik's cube problem.**

(Source: stackoverflow.com)

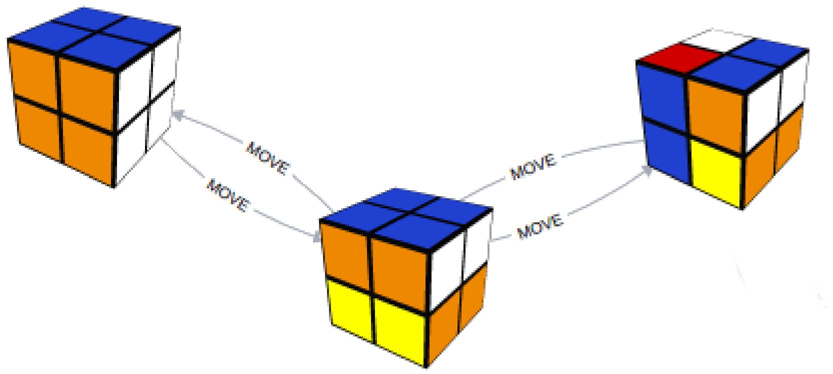


Figure 8. **Real-time usage of implicit graph data structure.**

(Source: medium.com)

5.2. Breadth First Search

Breadth First Search (Figure. 9) helps in finding a path for a Rubik's cube to reach the solved state from a scrambled one. Rubik's cube is among one of the fascinating puzzles and solving them has been a challenge given its vast search space of 3674160 combinations. Breadth First Search algorithm makes use of the fact that the God's number for a 2 × 2 × 2 Rubik's cube is 14, i.e. The fact that any cube scramble within the 3674160 states can be solved within a max of 14 moves.

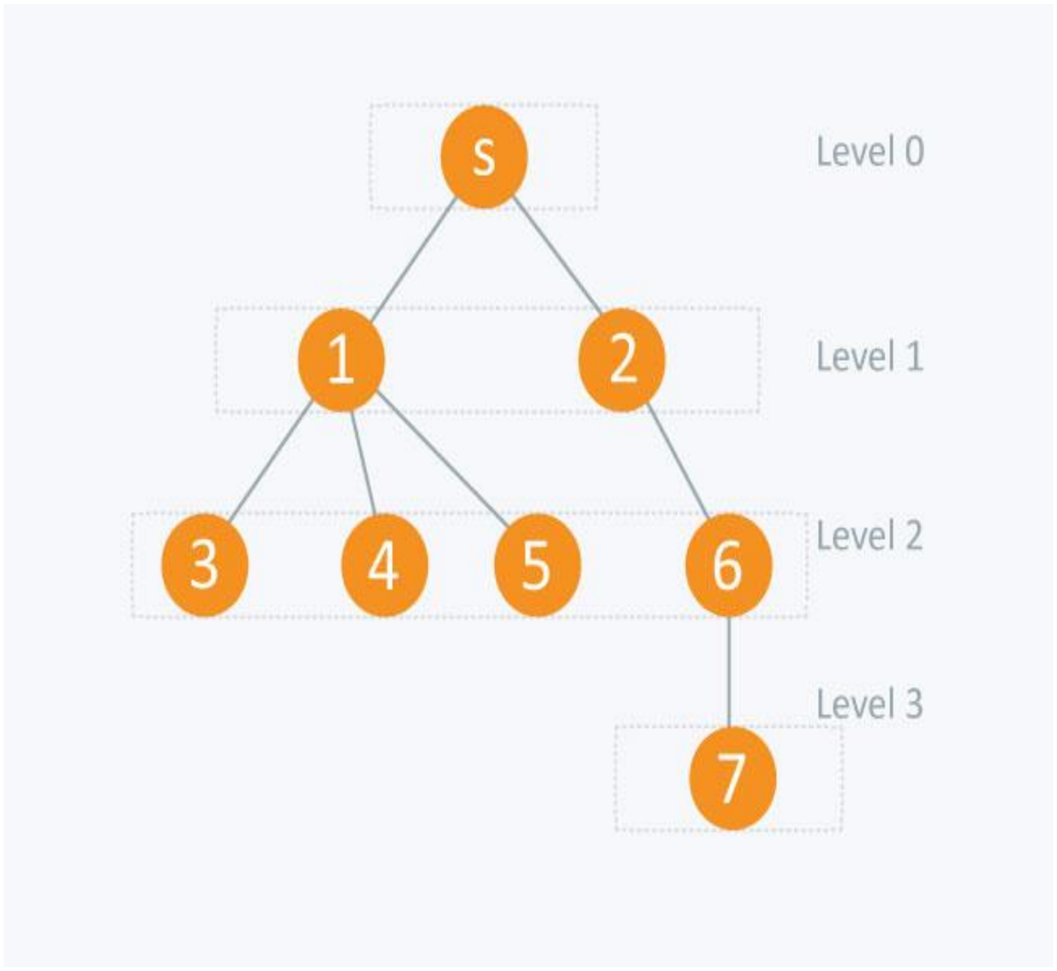


Figure 9. ***Breadth First Search tree. Used in Rubik's cube problem. The distance between the nodes in layer-1 is comparatively lesser than the distance between the nodes in layer-2.***

(Source: hackerearth.com)

6. Developmental Details

6.1. Java Programming Language

This project is entirely written in Java, “a programming language and computing platform first released by Sun Microsystems in 1996” for many reasons. Java.com states that software developers choose Java because it “has been tested, refined, extended, and proven by a dedicated community of Java developers, architects and enthusiasts. Java is designed to enable development of portable, high-performance applications for the widest range of computing platforms possible. By making applications available across heterogeneous environments, businesses can provide more services and boost end-user productivity, communication, and collaboration and dramatically reduce the cost of ownership of both enterprise and consumer applications. Java has become invaluable to developers by enabling them to –

- Write software on one platform and run it on virtually any other platform
- Create programs that can run within a web browser and access available web services
- Develop server-side applications for online forums, stores, polls, HTML forms processing, and more
- Combine applications or services using the Java language to create highly customized applications or services
- Write powerful and efficient applications for mobile phones, remote processors, microcontrollers, wireless modules, sensors, gateways, consumer products, and practically any other electronic device”

Many online sources state that Java is the most popular development language for Android Applications, primarily due to the powerful Java IDE that is provided through the Android Developer Tools. This IDE has advanced features for developing, debugging, and packaging.

Figure 10. Java programming language

(Source: sonarqube.org)

6.2. Java Swing

Swing is a GUI widget toolkit for Java. It is a part of Oracle's Java Foundation Classes (JFC). Basically, Swing is an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panels, scroll panes, trees, tables, and lists.

Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and therefore are platform-independent

6.3. Java Abstract Window Toolkit (AWT)

The **Abstract Window Toolkit** (**AWT**) is Java's original platform-dependent windowing, graphics, and user-interface widget toolkit, preceding Swing. The AWT is part of the Java Foundation Classes (JFC) — the standard API for providing a graphical user interface (GUI) for a Java program. AWT is also the GUI toolkit for several Java ME profiles. For example - Connected Device Configuration profiles require Java runtimes on mobile telephones to support the Abstract Window Toolkit.

When Sun Microsystems first released Java in 1995, AWT widgets provided a thin level of abstraction over the underlying native user-interface.

Features of AWT

- AWT has a set of native user interface components.
- It provides various classes for graphical representation of components like font, shape, color.
- It provides a robust event-handling model.
- It has Layout managers which is helpful for changing window size or screen resolution.
- It provides a large range of libraries that can be used for designing graphics for gaming applications or educational applications.
- It has data transfer classes through which cut and paste operation can be performed using the local clipboard.

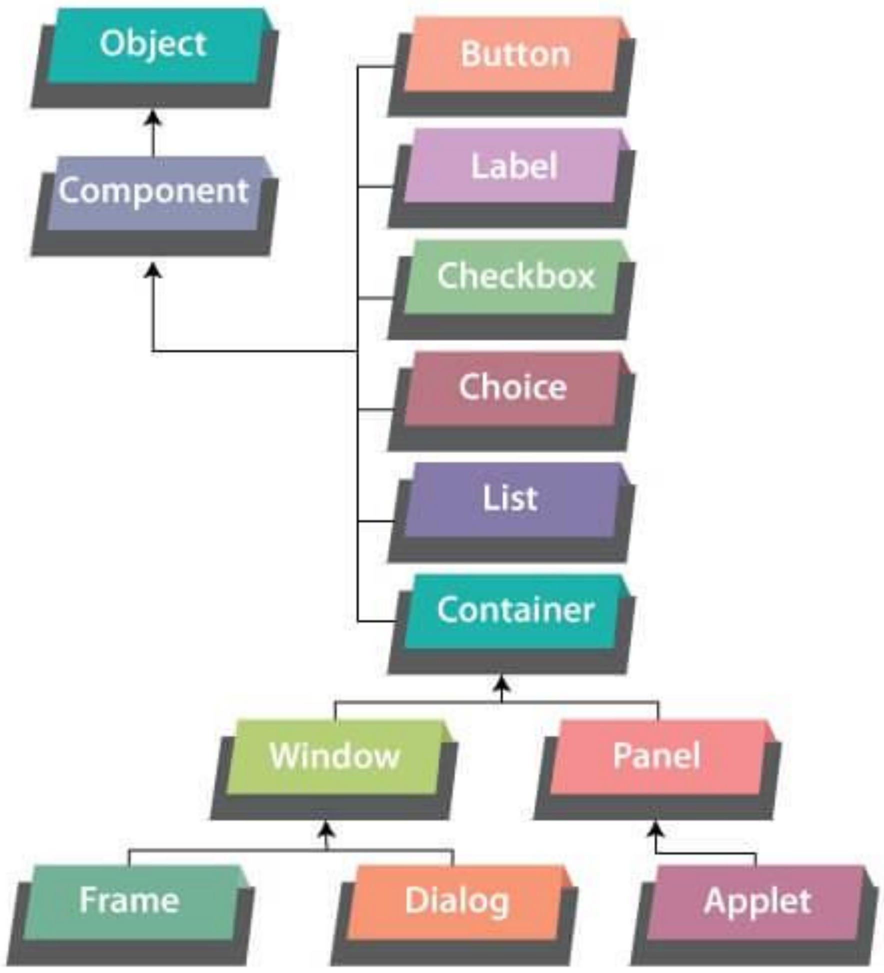


Figure 11. AWT hierarchy. Working of AWT.

(Source: tutorialandexample.com)

7. Source Codes

There are a total four modules of source codes implemented to realize this project. All of the source codes are given below in a modular way:

Java source code for creating cube nets & lab kits inside the project

```
package rubik;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JPanel;

public class RubikNet extends JPanel implements Renderable {

        private static final long serialVersionUID = -7420893625446291398L;
        private static final int CUBLETTE_LENGTH = 75;
        RubikState state;
        List<RubikCublet> cublets = new ArrayList<RubikCublet>();

        public RubikNet(RubikState state) {
                this.state = state;
                this.addMouseListener(mouseInput);
                this.setBackground(new Color(36, 36, 36));
                this.setLayout(null);
                this.repaint();
        }
```

```
@Override
public void render(Graphics2D g) {

        // representation of top of cube
        cublets.add(drawCublette(g, 0, 200, 50));
        cublets.add(drawCublette(g, 1, 275, 50));
        cublets.add(drawCublette(g, 2, 275, 125));
        cublets.add(drawCublette(g, 3, 200, 125));

        // representation of left of cube
        cublets.add(drawCublette(g, 4, 25, 225));
        cublets.add(drawCublette(g, 5, 100, 225));
        cublets.add(drawCublette(g, 6, 100, 300));
        cublets.add(drawCublette(g, 7, 25, 300));

        // representation of front of cube
        cublets.add(drawCublette(g, 8, 200, 225));
        cublets.add(drawCublette(g, 9, 275, 225));
        cublets.add(drawCublette(g, 10, 275, 300));
        cublets.add(drawCublette(g, 11, 200, 300));

        // representation of right of cube
        cublets.add(drawCublette(g, 12, 375, 225));
        cublets.add(drawCublette(g, 13, 450, 225));
        cublets.add(drawCublette(g, 14, 450, 300));
        cublets.add(drawCublette(g, 15, 375, 300));

        // representation of bottom of cube
        cublets.add(drawCublette(g, 16, 200, 400));
        cublets.add(drawCublette(g, 17, 275, 400));
        cublets.add(drawCublette(g, 18, 275, 475));
        cublets.add(drawCublette(g, 19, 200, 475));
        // representation of back of cube
        cublets.add(drawCublette(g, 20, 550, 225));
        cublets.add(drawCublette(g, 21, 625, 225));
        cublets.add(drawCublette(g, 22, 625, 300));
        cublets.add(drawCublette(g, 23, 550, 300));
    }

    private RubikCublet drawCublette(Graphics2D g, int i, int x, int y) {

        int color = -1;
        char c = state.positions[i];

        switch(c) {
            case 'w':
                color = 0;
                g.setColor(Color.WHITE);
                break;
            case 'g':
                color = 1;
                g.setColor(Color.GREEN);
                break;
            case 'o':
                color = 2;
                g.setColor(new Color(255,125,0));
                break;
            case 'r':
                color = 3;
                g.setColor(Color.RED);
                break;
            case 'y':
                color = 4;
                g.setColor(Color.YELLOW);
                break;
                case 'b':
                        color = 5;
                        g.setColor(Color.BLUE);
                        break;
        }

        g.fillRect(x, y, CUBLETTE_LENGTH, CUBLETTE_LENGTH);
        g.setColor(Color.BLACK);
        g.setStroke(new BasicStroke(3));
        g.drawRect(x, y, CUBLETTE_LENGTH, CUBLETTE_LENGTH);
        return new RubikCublet(i, x, y, color);
}

MouseListener mouseInput = new MouseListener() {

   @Override
   public void mouseClicked(MouseEvent e) {
     int x = e.getX(), y = e.getY();
     for (RubikCublet cublet : cublets) {
        if (x >= cublet.netX && x <= cublet.netX + CUBLETTE_LENGTH && y
         >= cublet.netY && y <= cublet.netY + CUBLETTE_LENGTH) {
                state.positions[cublet.stateIndex] = cublet.cycleColor();
                RubikNet.this.repaint();
         }
     }
   }

   @Override
   public void mousePressed(MouseEvent e) {}

   @Override
   public void mouseReleased(MouseEvent e){}

   @Override
   public void mouseEntered(MouseEvent e) {}
```

```
                @Override
                public void mouseExited(MouseEvent e) {}
              };

            @Override
            protected void paintComponent(Graphics g) {
                    super.paintComponent(g);
                    RubikNet.this.render((Graphics2D) g);
            }
    }

    class RubikCublet {

        int stateIndex;
        int netX;
        int netY;
        int color;
        char colors[] = {'w', 'g', 'o', 'r', 'y', 'b'};

        public RubikCublet(int stateIndex, int netX, int netY, int color) {

              this.stateIndex = stateIndex;
              this.netX = netX;
              this.netY = netY;
              this.color = color;
        }

        public char cycleColor() {

            color++;
            color %= 6;
            return colors[color];
        }
    }
```

Java source code for creating states of Rubik’s cube inside project

```
package rubik;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Random;

public class RubikState {

        /*
         * 24 positions for each color
         * w - white
         * y - yellow
         * g - green
         * b - blue
         * o - orange
         *  r - red
         *  each consecutive 4 indices are a face labeled
         */

    char[] positions;
    boolean isNullState;

    public RubikState() {
        // create a fully solved cube
        positions = new char[24];

        for (int i = 0; i < 4; i++)
            positions[i] = 'w';
        for (int i = 4; i < 8; i++)
            positions[i] = 'o';
        for (int i = 8; i < 12; i++)
            positions[i] = 'g';
        for (int i = 12; i < 16; i++)
            positions[i] = 'r';
```

```
		for (int i = 16; i < 20; i++)
			positions[i] = 'y';
		for (int i = 20; i < 24; i++)
			positions[i] = 'b';
		this.isNullState = false;
	}

	public RubikState(char[] positions) {
		this.positions = positions;
		this.isNullState = false;
	}

	public RubikState(boolean nullState) {
		this.isNullState = nullState;
	}

	/*
	* F - front rotated clockwise
	* F' - front counter clockwise
	* R - right clockwise
	* R' - right counter clockwise
	* U - up clockwise
	* U' - up counter clockwise
	*/

	private static int[] F = {0,1,5,6,4,16,17,7,11,8,9,10,3,13,14,2,15,12,18,19, 20,21,22,23};
	private static int[] Fi = permInverse(F);
	private static int[] U = {3,0,1,2,8,9,6,7,12,13,10,11,20,21,14,15,16,17,18,19,4,5,22,23};
	private static int[] Ui = permInverse(U);
	private static int[] R = {0,9,10,3,4,5,6,7,8,17,18,11,15,12,13,14,16,23,20,19,2,21,22,1};
	private static int[] Ri = permInverse(R);
```

```
public static int[] permInverse(int[] p) {

        int n = p.length;
        int[] g = new int[n];

        for (int i = 1; i < n; i++) { g[p[i]] = i; }
        return g;
}

public char[] permApply(int[] perm) {

        char[] newPositions = new char[24];
        for (int i = 0; i < 24; i++) { newPositions[i] = positions[perm[i]]; }
        return newPositions;
}

public HashMap<String, RubikState> getReachableStates() {

        HashMap<String, RubikState> moves = new HashMap<>();

        addBasicMove("F'", F, moves);
        addBasicMove("F", Fi, moves);
        addBasicMove("U'", U, moves);
        addBasicMove("U", Ui, moves);
        addBasicMove("R'", R, moves);
        addBasicMove("R", Ri, moves);
        return moves;
}

private void addBasicMove(String name, int[] perm, HashMap<String, RubikState>
moves) {

        RubikState state = new RubikState(permApply(perm));
        moves.put(name, state);
}
```

```
public void executeMoveSeq(String seq) {

        if (seq == null) { return; }
        String[] moves = seq.toUpperCase().split(" ");

        for (String move : moves) {

                switch(move) {
                case "F":
                        positions = permApply(F);
                        break;
                case "F'":
                        positions = permApply(Fi);
                        break;
                case "F2":
                        positions = permApply(F);
                        positions = permApply(F);
                        break;
                case "U":
                        positions = permApply(U);
                        break;
                case "U'":
                        positions = permApply(Ui);
                        break;
                case "U2":
                        positions = permApply(U);
                        positions = permApply(U);
                        break;
                case "R":
                        positions = permApply(R);
                        break;
                case "R'":
                        positions = permApply(Ri);
                        break;
                case "R2":
                        positions = permApply(R);
                        positions = permApply(R);
                        break;
                }
        }
}

public int getRandomWithExclusion(Random rnd, int start, int end, int… exclude) {

        int random = start + rnd.nextInt(end - start + 1 - exclude.length);

        for (int ex : exclude) {
                if (random < ex) { break; }
                random++;
         }

        return random;
}

public String randomize() {

        String scramble = "";
        Random rand = new Random();
        int prevMoveGroup = -1;

        String[][] moves = {{"F", "F'", "F2"},
                {"U", "U'", "U2"},
                {"R", "R'", "R2"}
        };

        for (int i = 0; i < 17; i++) {

                prevMoveGroup = getRandomWithExclusion(rand, 0, moves.length - 1,
                prevMoveGroup);
                String move = moves[prevMoveGroup]
                [rand.nextInt(moves[0].length)];
                scramble += move + " ";
            }

            scramble = scramble.substring(0, scramble.length() - 2);
            executeMoveSeq(scramble);
            return scramble;
        }

        @Override
        public int hashCode() { return Arrays.toString(this.positions).hashCode(); }

        @Override
        public boolean equals(Object obj) {
            if (!obj.getClass().getSimpleName().equals("RubikState")) {
                return false;
            }
            RubikState state = (RubikState) obj;
            if (state.positions.length != this.positions.length) return false;
            for (int i = 0; i < this.positions.length; i++) {
                if (this.positions[i] != state.positions[i])
                    return false;
            }
            return true;
        }
    }
```

Java Source code for inculcating Graphics2D class to supplement the graphics of this project

```
package rubik;
import java.awt.Graphics2D;

public interface graphics { void render(Graphics2D g); }
```

Java Source for initializing Graphics2D class to supplement project

```
package rubik;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.border.LineBorder;

public class RubikGUI extends JFrame {

        private static final long serialVersionUID = -4226981410944431646L;
        private RubikNet rubiknet = new RubikNet(new RubikState());
        private JButton solve = new JButton("Solve");
        private JButton randomize = new JButton("Randomize");
        private JButton reset = new JButton("Reset");
        private JLabel solution = new JLabel(), time = new JLabel();
        private JLabel scramble = new JLabel();
```

```
public RubikGUI() {

        this.buildFrame();
        this.addComponents();
        this.setContentPane(rubiknet);
        this.setVisible(true);
}

private void buildFrame() {

     this.setTitle("CubeX");
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     this.setSize(950, 700);
     this.setResizable(false);
     this.requestFocus();
}

private void addComponents() {

   solve.setFont(new Font("SF Pro Display", Font.BOLD, 18));
   solve.setBackground(Color.GRAY.darker());
   solve.setForeground(Color.WHITE);
   solve.setBorder(new LineBorder(Color.gray.darker().darker(), 1));
   solve.addMouseListener(solveButtonListener);
   solve.setFocusable(false);
   solve.setSize(100, 35);
   solve.setLocation(430, 70);

   randomize.setFont(new Font("SF Pro Display", Font.BOLD, 18));
   randomize.setBackground(Color.GRAY.darker());
   randomize.setForeground(Color.WHITE);
   randomize.setBorder(new LineBorder(Color.gray.darker().darker(), 1));
   randomize.addMouseListener(randomButtonListener);
   randomize.setFocusable(false);
   randomize.setSize(100, 35);
   randomize.setLocation(550, 70);
```

```
    reset.setFont(new Font("SF Pro Display", Font.BOLD, 18));
    reset.setBackground(Color.GRAY.darker());
    reset.setForeground(Color.WHITE);
    reset.setBorder(new LineBorder(Color.gray.darker().darker(), 1));
    reset.addMouseListener(resetButtonListener);
    reset.setFocusable(false);
    reset.setSize(100, 35);
    reset.setLocation(670, 70);

    scramble.setFont(new Font("SF Pro Display", Font.BOLD, 18));
    scramble.setSize(500, 30);
    scramble.setForeground(Color.WHITE);
    scramble.setLocation(420, 125);
    solution.setFont(new Font("SF Pro Display", Font.BOLD, 18));
    solution.setSize(500, 30);
    solution.setForeground(Color.WHITE);
    solution.setLocation(420, 150);

    time.setFont(new Font("SF Pro Display", Font.BOLD, 18));
    time.setSize(500, 30);
    time.setForeground(Color.WHITE);
    time.setLocation(420, 175);

    rubiknet.add(solve);
    rubiknet.add(randomize);
    rubiknet.add(reset);
    rubiknet.add(scramble);
    rubiknet.add(solution);
    rubiknet.add(time);
}
```

```
MouseListener solveButtonListener = new MouseListener() {

        @Override
        public void mouseClicked(MouseEvent e) {}

        @Override
        public void mousePressed(MouseEvent e) {}

        @Override
        public void mouseReleased(MouseEvent e) {

                long startTime = System.currentTimeMillis();
                solution.setText("Solution: " + Solver.solve(rubiknet.state));
                time.setText("Time Elapsed: " + (System.currentTimeMillis() -
                startTime) + "ms.");
                rubiknet.state.executeMoveSeq(solution.getText());
                rubiknet.repaint();
         }



        @Override
        public void mouseEntered(MouseEvent e) {
                solve.setBackground(new Color(117, 117, 117));
         }

        @Override
        public void mouseExited(MouseEvent e) {
                solve.setBackground(Color.GRAY.darker());
        }
};

MouseListener randomButtonListener = new MouseListener(){

     @Override
     public void mouseClicked(MouseEvent e) {}
```

```
        @Override
        public void mousePressed(MouseEvent e) {}

        @Override
        public void mouseReleased(MouseEvent e) {

            scramble.setText("Scramble: " + rubiknet.state.randomize());
            solution.setText("");
            time.setText("");
            rubiknet.repaint();
        }

        @Override
        public void mouseEntered(MouseEvent e) {
           randomize.setBackground(new Color(117, 117, 117));
        }

        @Override
        public void mouseExited(MouseEvent e) {
           randomize.setBackground(Color.GRAY.darker());
        }
    };

    MouseListener resetButtonListener = new MouseListener() {

        @Override
        public void mouseClicked(MouseEvent e) {}

        @Override
        public void mousePressed(MouseEvent e) {}

        @Override
        public void mouseReleased(MouseEvent e) {
            rubiknet.state = new RubikState();
```

```
                scramble.setText("");
                solution.setText("");
                time.setText("");
                rubiknet.repaint();
            }

            @Override
            public void mouseEntered(MouseEvent e) {
                reset.setBackground(new Color(117, 117, 117));
            }

            @Override
            public void mouseExited(MouseEvent e) {
                 reset.setBackground(Color.GRAY.darker());
            }
        };
}
```

Java Source for implementing driver function (or main function) of this project

```
package rubik;
import java.util.ArrayDeque;
import java.util.HashMap;
import java.util.Map.Entry;

public class Solver {

   public static final int GODS_NUMBER = 14;
   public static void main(String[] args) { new RubikGUI(); }

         /*
         * Implementing a 2-way breadth-first search on the
         * RubikState implicit graph data structure
         * returns the shortest path (solution)
         * (the diameter of a 2x2x2 state graph is 14)
         */

         public static String solve(RubikState state) {

                  HashMap<RubikState, String> forwardParents = new
                  HashMap<RubikState, String>();
                  HashMap<RubikState, String> backwardParents = new
                  HashMap<RubikState, String>();
                  ArrayDeque<RubikState> fqueue = new ArrayDeque<RubikState>();
                  ArrayDeque<RubikState> bqueue = new ArrayDeque<RubikState>();
                  RubikState src = state, end = new RubikState();
                  forwardParents.put(end, null);
                  backwardParents.put(src, null);
                  fqueue.add(end);
                  fqueue.add(new RubikState(true));
                  bqueue.add(src);
```

```
// check if cube already solved
if (end.equals(src))
        return "Already Solved.";

// bfs visits from both ends of graph
int graphRadius = GODS_NUMBER / 2;

for (int i = 0; i <= graphRadius; i++) {

        while (true) {
                end = fqueue.remove();
                if (end.isNullState) {
                        fqueue.add(new RubikState(true));
                        break;
                }

                for (Entry<String, RubikState> move :
                end.getReachableStates().entrySet()) {

                   if (!forwardParents.containsKey(move.getValue())) {
                        forwardParents.put(move.getValue(),
                        move.getKey());
                        fqueue.add(move.getValue());
                   }
                }

                src = bqueue.remove();

                for (Entry<String, RubikState> move:
                src.getReachableStates().entrySet()) {

                   if (!backwardParents.containsKey(move.getValue())) {
                        backwardParents.put(move.getValue(),
                        move.getKey());
                        bqueue.add(move.getValue());
                   }
```

```
                        // same state discovered in both ends
                        if (forwardParents.containsKey(move.getValue())) {
                            String endpath = path(move.getValue(),  forwardParents);
                            String srcpath = path(move.getValue(),backwardParents);
                            srcpath = reverse(srcpath);
                            String solutionPath = srcpath + " " + endpath;
                            return solutionPath.replaceAll("(([RUF])'?) \\1",  "$22");
                        }
                    }
                }
        }

        return "No Solution Possible. Impossible configuration";
    }

    private static String reverse(String path) {

        path += " ";
        String reverse = "";

        for (int i = 0; i < path.length(); i++) {
            if (path.charAt(i) == ' ')
                reverse += "' ";
            else if (path.charAt(i) != '\'')
                reverse += path.charAt(i);
            else {
                reverse += " ";
                i++;
            }
        }

        String ar[] = reverse.split(" ");
        for (int i = 0; i < ar.length/2; i++) {
            String temp = ar[i];
            ar[i] = ar[ar.length-1-i];
```

```
                ar[ar.length-1-i] = temp;
            }
            return String.join(" ", ar);
        }

        private static String path(RubikState state, HashMap<RubikState, String> parents) {

            String path = parents.get(state);
            RubikState next = new RubikState(state.positions);
            next.executeMoveSeq(path);

            while (parents.get(next) != null) {

                path += " " + parents.get(next);
                 next = new RubikState(state.positions);
                next.executeMoveSeq(path);
            }

            return path;
        }
    }
```

8. Conclusion & Further Work

We were able to create a basic system that takes in a starting Rubik's cube configuration, checks it to ensure that it is valid, and then animates rotations on a 2D-visualization of the cube to guide the user towards the solution. Creating this application has proven to be very challenging yet rewarding experience. Through vast amounts of research and programming involved in this project, much knowledge was gained on the topics of Rubik's cube theory, Rubik's cube algorithms, modifying algorithms to fit specific needs, Java programming language, Java 2D API, etc.

Future work on the system would likely consist of improving the quality of user interaction by having a camera capture the initial state, reducing the number of rotations needed to solve the cube, and making the visualization cleaner.

9. References

[1] https://en.wikipedia.org/wiki/Pocket_Cube
[2] https://dspace.mit.edu/handle/1721.1/73771
[3] https://www.youcandothecube.com/solve-it/
[4] https://www.youtube.com/watch?v=Z5urhcA_7nA&t=15s

Blogspot References:

[5] https://programmablebrick.blogspot.com/2017/02/rubiks-cube-tracker-using-opencv.html
[6] https://news.mit.edu/2011/rubiks-cube-0629

Github References:

[7] https://github.com/dwalton76/rubiks-color-resolver

Research Papers References:

[8] Higher Mathematical Concepts Using the Rubik's Cube - by Pawel Nazarewicz

10. Bibliography

To make this project I have taken help from various sources, in particular a research paper written anonymously "The Mathematics of the Rubik's Cube" from Massachusetts Institute of Technology (MIT)s official website. Also, I have used all the sources that I have mentioned above in the references section. Since, the author of this research paper is unknown so I am specifically mentioning it here. I would also like to thank my instructor **Dr. Prem Nath** for all the valuable support and motivation that he puts in behind this project. I completed this project under his guidance and supervision, so I am extremely grateful to Dr. Prem Nath for his able guidance.

Most Used References:

[1] https://github.com/dwalton76/rubiks-color-resolver
[2] https://www.youcandothecube.com/solve-it/
[3] https://programmablebrick.blogspot.com/2017/02/rubiks-cube-tracker-using-opencv.html