



Assignment Report

Jesper Wingren
Emil Ulvagården
1DT905-Object-oriented programming

Rubiks cube simulator with JavaFX

*JavaFX and Java based rubiks cube simulator with
timer*



Abstract

This project delves into the realm of object-oriented programming in Java, specifically focusing on the creation of a 2D graphical user interface (GUI) representation of a Rubik's cube, coupled with a 2D terminal version serving as the backend. The report encompasses various stages of project development, including planning, problem-solving, and learning experiences.

The project commenced with the creation of the cube's backend to track moves systematically. Subsequently, the team integrated the 2D representation using JavaFX, aiming for a well-separated backend and frontend to facilitate testing. The system allows users to perform actions such as scrambling, solving, managing a timer, and rotating cube pieces.

Design decisions involved utilizing letters, ANSI, and Unicode for backend cube representation, while the JavaFX application featured a split-screen layout with move options and an input field.

Tools and techniques encompassed draw.io for UML diagrams, JavaFX library for GUI, Gradle for structure, JUnit and Jacoco for testing and coverage, and SceneBuilder for interface design.

Testing primarily focused on the backend, evaluating rotations visually. The application's final version included 16 distinct moves, a timer, scramble, reset best time, and solve options. However, a limitation arose regarding timer stopping conditions.

In conclusion, the project successfully separated backend and frontend components, and the team expressed satisfaction despite a desire to enhance display features. JavaFX emerged as a significant learning point, emphasizing adaptability in implementing functions.

Keywords: JavaFX, Rubiks cube, java, object-oriented programming

Contents

1	<i>Introduction</i>	<i>4</i>
1.1	Code on GitLab	4
1.2	Link to presentation	4
2	<i>Analysis.....</i>	<i>4</i>
2.1	Text describing the system.....	4
2.2	Use cases	5
2.3	Class diagram	5
3	<i>Design.....</i>	<i>6</i>
3.1	Updated class diagram.....	7
4	<i>Tools and techniques.....</i>	<i>7</i>
5	<i>Implementation</i>	<i>8</i>
6	<i>Test and coverage</i>	<i>9</i>
7	<i>System/Application</i>	<i>10</i>
8	<i>Summary & Conclusion.....</i>	<i>13</i>
8.1	TIL;.....	14
	<i>References.....</i>	<i>15</i>
A	<i>Appendix 1</i>	<i>16</i>

1 Introduction

This project covers some of the part from the course object orientated programming in java. In the report we go through the different stages of a project such as planning and problem solving. Going through the different problems that occurred and how we fixed them as well as what we learned from them. The project consists of a 2D GUI based Rubik's cube with a 2D terminal version as the backend.

1.1 Code on GitLab

Git repo (<https://gitlab.lnu.se/1dt905/student/eu222dq-jw223rn/a02-project>)

1.2 Link to presentation

https://youtu.be/b4_wpOlyGeY

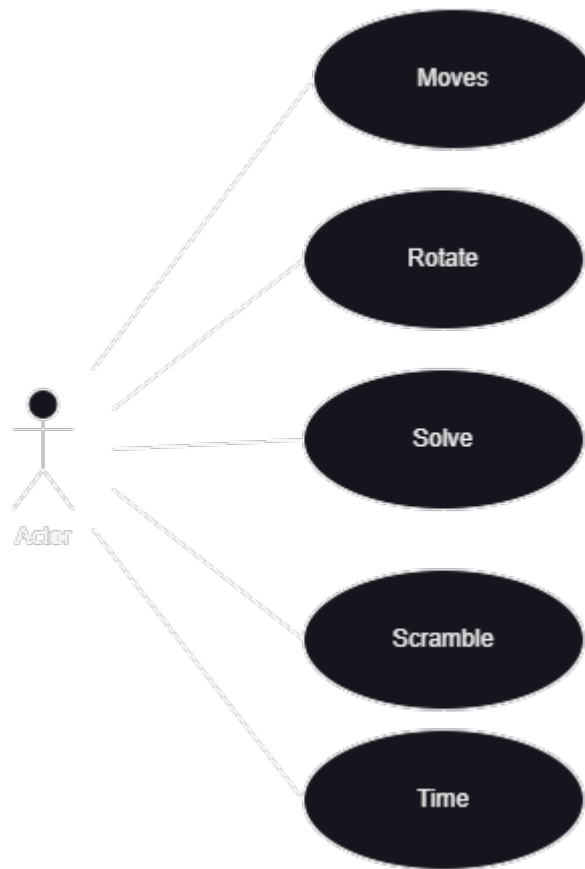
2 Analysis

The project started with us making the backend of the cube as a way of keeping track of what moves were made and in what order. We then started working with the 2D representation of the cube by using JavaFX and connecting the 2D cube to the backend.

2.1 Text describing the system

The system should be a 2D Rubik's cube. The user can do things such as scrambling, solving, starting/stopping a timer and rotating the cube and its pieces. The backend and frontend should be as separated as possible when it comes to coding making the testing easier.

2.2 Use cases



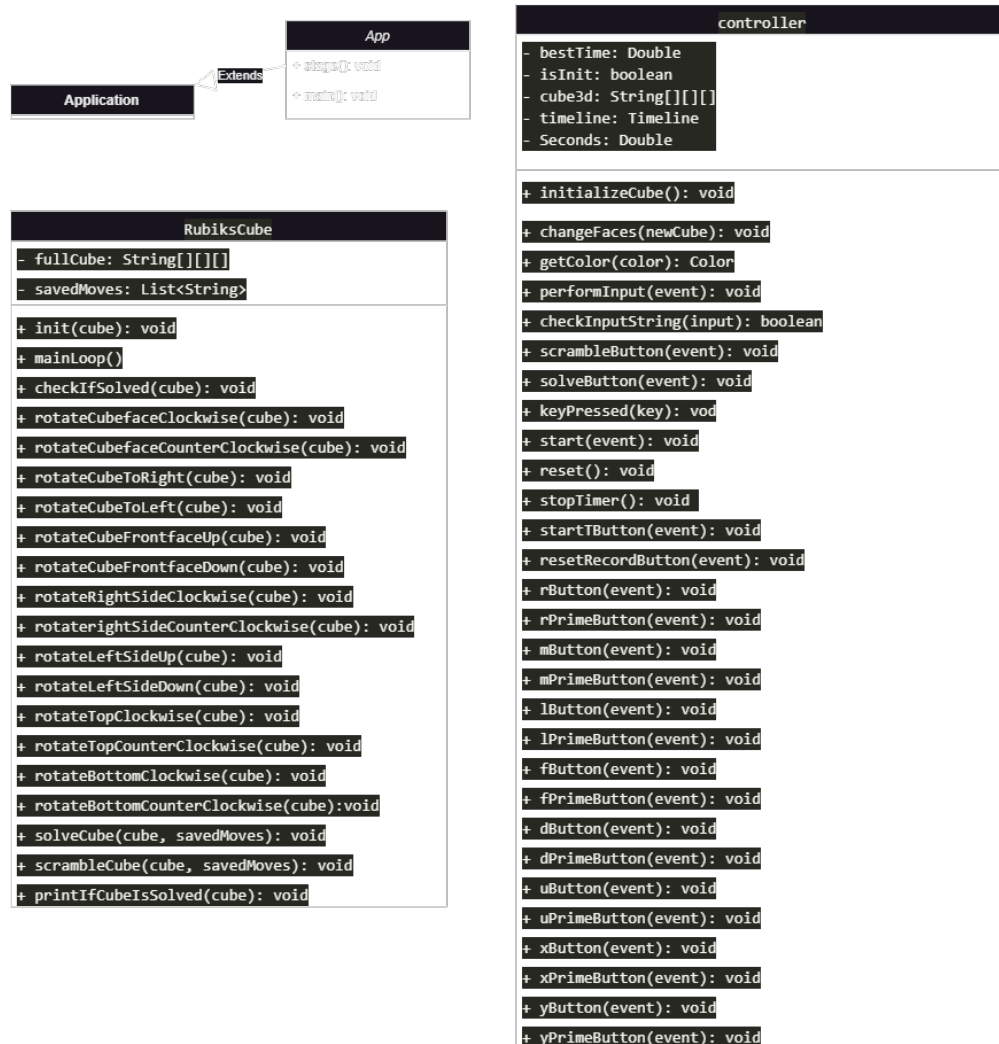
2.3 Class diagram



3 Design

When deciding on how the different cubes should look, we started by doing the backend of the cube using letters to simulate what colors were placed where. We then moved over to using ANSI and Unicode to get a representation of a colored cube in the terminal where orange is changed out for purple to easier see the difference between orange and yellow. When making the javafx application we mad so the cube is on the upper half of the window and on the lower half we have two different tabs one with the different moves the user can make and the other tab having an input field. Both these tabs also consist of buttons for start/stop timer, scramble, reset best time and solve the cube. For making the timer function more useable that also works by pressing space bar which starts or stops the timer, which also shows the latest time and the best time.

3.1 Updated class diagram



4 Tools and techniques

To make the UML diagrams as well as use case diagrams we used draw.io.

We have decided to use JavaFX as our GUI and therefore the JavaFX library is used. The library is used to show the user the cube in 2D and to implement the event that the user chooses to perform. We have also used the java library to implement different lists, arrays and scanners. The cube is structured with Gradle and tested by visual comparison by the developers.

5 Implementation

Gradle was used as the build tool for this project. This gives us an easy way to build a structure for our codebase. Using Junit and jacoco we performed testing on the code. Junit handles the testing code while the jacoco creates a report building on the tests. Using special Javadoc comments, we created a Javadoc which is simple using gradle. For creating the interface with javafx we used scenebuilder which is a separate program for creating a layout. In this we can connect different parts with the code using fx:id and then create a controller class to handle these and methods. The scenebuilder creates an fxml file which is easily implemented into a javfx application. Down below is pictures of the Javadoc and parts of the fxml code, more specifically the buttons for the different moves.

Modifier and Type	Method	Description
static boolean	<code>checkIfSolved(String[][][] cube)</code>	Checks if a Rubik's Cube is solved by comparing the color of each sticker on each face of the cube with the color of the sticker in the top-left corner of that face.
static void	<code>init(String[][][] cube)</code>	Initializes the Rubik's cube with different colored squares.
static String[][][]	<code>mainLoop(String choice)</code>	Performs different operations on a Rubik's Cube based on the user's choice.
static void	<code>makeMove(String[][][] cube, String move, List<String> savedMoves)</code>	Makes a move on the Rubik's Cube by rotating the cube's faces or sides based on the given move.
static void	<code>printIfCubeIsSolved(String[][][] cube)</code>	Checks if the Rubik's Cube is solved and prints a congratulatory message if it is.
static void	<code>rotateBottomClockwise(String[][][] cube)</code>	Rotates the bottom face of a Rubik's Cube clockwise, along with the adjacent faces.
static void	<code>rotateBottomCounterClockwise(String[][][] cube)</code>	Rotates the bottom face of a Rubik's Cube counter-clockwise, along with the adjacent faces.
static void	<code>rotateCubeFrontfaceDown(String[][][] cube)</code>	Rotates the front face of a Rubik's Cube down, along with the adjacent faces, in a clockwise direction.
static void	<code>rotateCubeFrontfaceUp(String[][][] cube)</code>	Rotates the front face of a Rubik's Cube represented by a 3D array of strings in a clockwise direction.
static void	<code>rotateLeftSideDown(String[][][] cube)</code>	Rotates the left side of a Rubik's Cube down by rearranging the colors of the cube's faces.
static void	<code>rotateLeftSideUp(String[][][] cube)</code>	Rotates the left side of a Rubik's Cube upwards, along with the adjacent sides.
static void	<code>rotateRightSideClockwise(String[][][] cube)</code>	Rotates the right side of a Rubik's Cube clockwise.
static void	<code>rotateRightSideCounterClockwise(String[][][] cube)</code>	This method rotates the right side of a Rubik's Cube counterclockwise.
static void	<code>rotateTopClockwise(String[][][] cube)</code>	Rotates the top face of a Rubik's Cube clockwise, along with the adjacent faces.
static void	<code>rotateTopCounterClockwise(String[][][] cube)</code>	Rotates the top face of a Rubik's Cube counter-clockwise, along with the adjacent faces.
static void	<code>scrambleCube(String[][][] cube, List<String> savedMoves)</code>	Randomly scrambles the Rubik's cube by making 80 random moves.
static void	<code>solveCube(List<String> savedMoves, String[][][] cube)</code>	Reverses the effect of the moves stored in the 'savedMoves' list on the Rubik's Cube.

Modifier and Type	Method	Description
static void	<code>changeFaces(String[][][] newCube)</code>	Updates the colors of the rectangles in a Rubik's Cube based on the provided 3D array representation of the cube.
static boolean	<code>checkInputString(String input)</code>	Checks if a given input string is valid according to give instructions.
void	<code>dButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "d" button is clicked.
void	<code>dPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "d'" button is clicked.
void	<code>fButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "f" button is clicked.
void	<code>fPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "f'" button is clicked.
void	<code>initializeCube()</code>	Initializes a 3D array of Rectangle objects and assigns them to the corresponding rectangles in the JavaFX scene.
void	<code>keyPressed(javafx.scene.input.KeyEvent key)</code>	Handles the event when a key is pressed.
void	<code>lButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "l" button is clicked.
void	<code>lPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "l'" button is clicked.
void	<code>mButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "m" button is clicked.
void	<code>mPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "m'" button is clicked.
void	<code>performInput(javafx.event.ActionEvent event)</code>	Takes user input from a text field, checks if it is a valid input, and performs corresponding actions based on the input.
void	<code>rButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "r" button is clicked.
void	<code>reset()</code>	Resets the timer in the Rubik's Cube application.
void	<code>resetRecordButton(javafx.event.ActionEvent event)</code>	Resets the best time record to 0.0 and updates the corresponding label on the user interface.
void	<code>rPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "r'" button is clicked.
void	<code>scrambleButton(javafx.event.ActionEvent event)</code>	Initializes the Rubik's Cube, generates a random scramble sequence of 80 moves, and updates the colors of the cube's faces accordingly.
void	<code>solveButton(javafx.event.ActionEvent event)</code>	Calls the mainLoop("solve") method from the RubiksCube class to solve the cube.
void	<code>solveButton{javafx.event.ActionEvent event}</code>	Calls the mainLoop("solve") method from the RubiksCube class to solve the cube.
void	<code>start(javafx.event.ActionEvent event)</code>	Starts a timer when a button is clicked.
void	<code>startTButton(javafx.event.ActionEvent event)</code>	Starts or stops a timer based on its current state.
void	<code>stopTimer()</code>	Stops the timer, updates the best time if the current time is better, and displays the recent time.
void	<code>uButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "u" button is clicked.
void	<code>uPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "u'" button is clicked.
void	<code>xButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "x" button is clicked.
void	<code>xPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "x'" button is clicked.
void	<code>yButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "y" button is clicked.
void	<code>yPrimeButton(javafx.event.ActionEvent event)</code>	Handles the action event when the "y'" button is clicked.

```

dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#uButton" prefHeight="54.0" prefWidth="96.0" text="u" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#uPrimeButton" prefHeight="54.0" prefWidth="96.0" text="u'" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#dButton" prefHeight="54.0" prefWidth="96.0" text="d" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#dPrimeButton" prefHeight="54.0" prefWidth="96.0" text="d'" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#fButton" prefHeight="54.0" prefWidth="96.0" text="f" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#fPrimeButton" prefHeight="54.0" prefWidth="96.0" text="f'" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#lButton" prefHeight="54.0" prefWidth="96.0" text="l" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#lPrimeButton" prefHeight="54.0" prefWidth="96.0" text="l'" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#mButton" prefHeight="54.0" prefWidth="96.0" text="m" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#mPrimeButton" prefHeight="54.0" prefWidth="96.0" text="m'" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#rButton" prefHeight="54.0" prefWidth="96.0" text="r" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#rPrimeButton" prefHeight="54.0" prefWidth="96.0" text="r'" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#xButton" prefHeight="54.0" prefWidth="96.0" text="x" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#xPrimeButton" prefHeight="54.0" prefWidth="96.0" text="x'" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#yButton" prefHeight="54.0" prefWidth="96.0" text="y" textA
dButton alignment="CENTER" contentDisplay="CENTER" focusTraversable="false" mnemonicParsing="false" onAction="#yPrimeButton" prefHeight="54.0" prefWidth="96.0" text="y'" textA



```

6 Test and coverage















































Testing is only done on the backend. The testing that is done on the backend is very simple since the rotations are tested by a visual inspection. The test on

the code shows that the “init” is correct and when making a rotation the cube is not the same as it was before, and that it is not null. The test has a 99% coverage on instructions and a 96% coverage on branches.

a02.project.com.cube

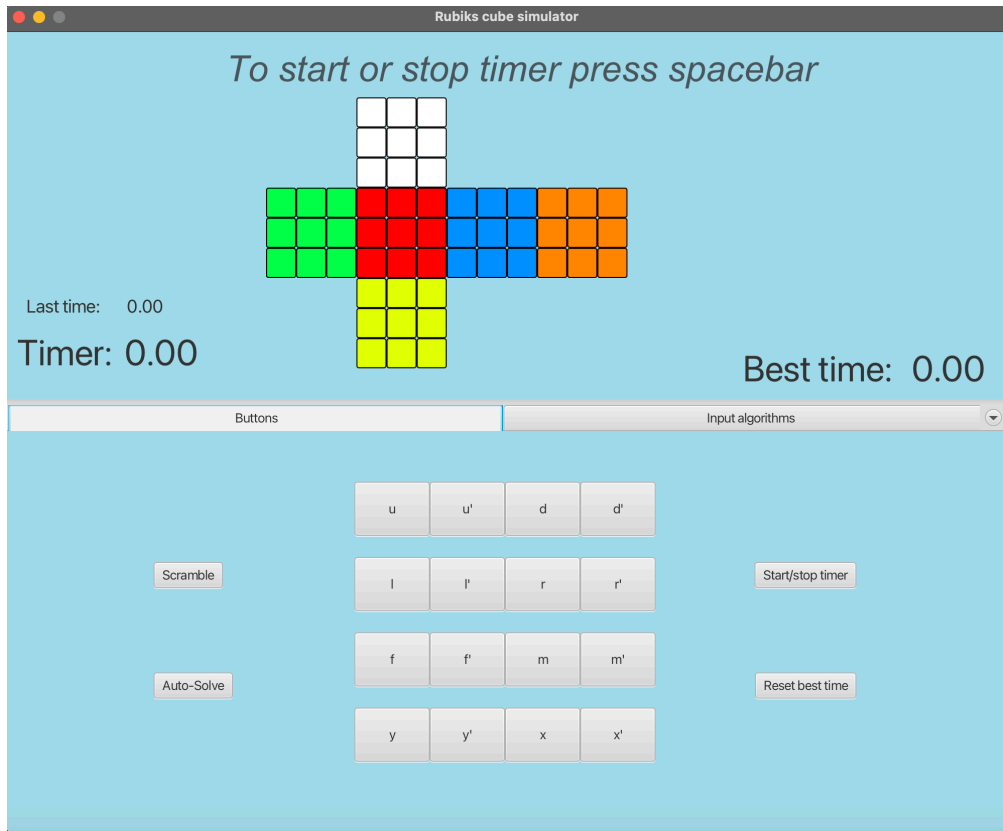
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
RubiksCube		99 %		96 %	7	122	8	664	1	23	0	1
Total	30 of 7 002	99 %	7 of 198	96 %	7	122	8	664	1	23	0	1

RubiksCube

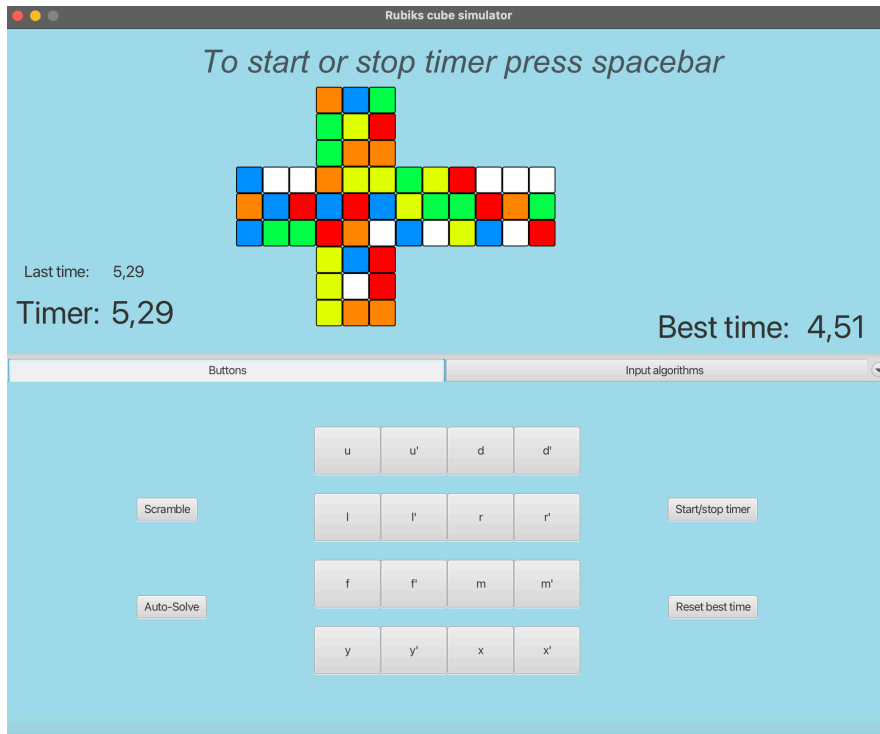
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
mainLoop(String)		90 %		90 %	3	21	5	65	0	1
printIfCubelsSolved(String[][])		57 %		50 %	1	2	1	3	0	1
RubiksCube()		0 %		n/a	1	1	1	1	1	1
checkIfSolved(String[][])		95 %		87 %	1	5	1	7	0	1
rotateCubeToRight(String[][])		100 %		100 %	0	4	0	58	0	1
rotateCubeToLeft(String[][])		100 %		100 %	0	4	0	58	0	1
rotateCubeFrontfaceUp(String[][])		100 %		100 %	0	4	0	58	0	1
rotateCubeFrontfaceDown(String[][])		100 %		100 %	0	4	0	58	0	1
rotateCubeFaceClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
rotateCubeFaceCounterClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
rotateRightSideClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
rotateRightSideCounterClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
rotateLeftSideUp(String[][])		100 %		100 %	0	4	0	26	0	1
rotateLeftSideDown(String[][])		100 %		100 %	0	4	0	26	0	1
rotateTopClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
rotateTopCounterClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
rotateBottomClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
rotateBottomCounterClockwise(String[][])		100 %		100 %	0	4	0	26	0	1
makeMove(String[][], String, List)		100 %		100 %	0	14	0	42	0	1
solveCube(List, String[][])		100 %		96 %	1	16	0	31	0	1
scrambleCube(String[][], List)		100 %		100 %	0	2	0	8	0	1
init(String[][])		100 %		100 %	0	4	0	14	0	1
static {...}		100 %		n/a	0	1	0	1	0	1
Total	30 of 7 002	99 %	7 of 198	96 %	7	122	8	664	1	23

7 System/Application

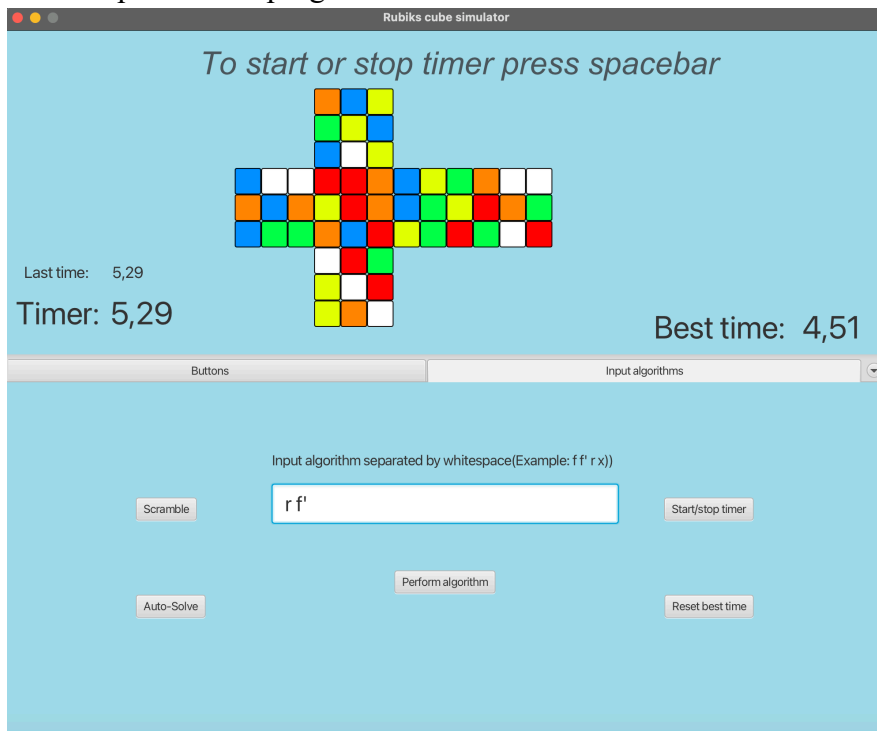
The final application looks like the following picture when starting it with Gradle:

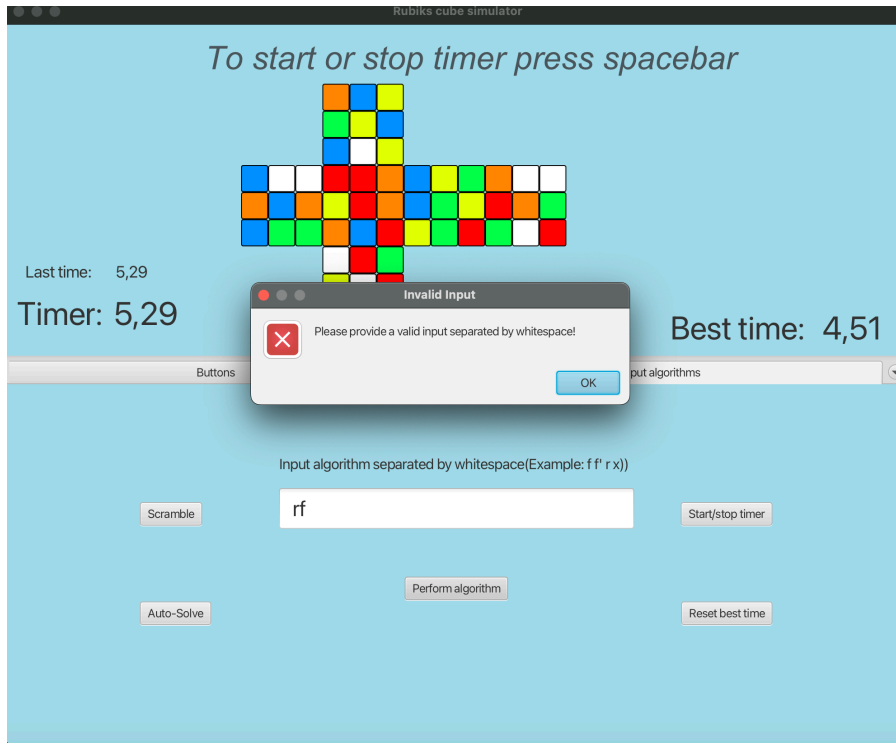


As seen in the picture there is different things the user can do. The user can do 16 different moves which moves the cube as it should. There is also a timer which can be started with either the button or by pressing spacebar. The last time and the best time are also displayed. If the user accidentally sets a new best time, it can be reset. The scramble button does 80 random moves and displays them, if the user wants a solved cube the auto solve button can be pushed which solves the cube. Here is a picture taken after pressing scramble.



The next picture shows the second tab where you can put in an algorithm as an input. The input should be separated by whitespace and if an incorrect move or not separated the program throws an error.





There is something that is a bit of a problem or limitation with the program and that is that you shouldn't be able to stop the timer unless the cube is solved. The problem with that came with the implementation in the backend which checks if the cube is solved for some reason did not cooperate with the javafx code.

8 Summary & Conclusion

Our first plan was to separate the project into two parts, the backend and the frontend. We started focusing on each part separately but keeping contact and making sure it's possible to connect each part smoothly. When doing the connection, it went easy, it almost took no effort which was good. During the programming process we discussed with each other about problems and how we should solve them. We made sure that both of us always were on the same plan and if we wanted to change parts, we notified each other.

Our conclusion is that we are happy with the program although something we would've liked to implement is making the scramble and solve methods display and not only change the cube. The separation of backend and frontend went successfully which was good for writing the tests.

8.1 TIL;

The biggest TIL is javaFX, we learnt a lot of things working on this project making us feel very comfortable when using javaFX.

Another big TIL is the fact that even if you have an idea of how you are to implement different functions it does not always go as planned and that changing the idea when stuck is sometimes the best thing to do.

References

Wikipedia (2023-10-20) ANSI escape code [Online] (Visited 2023-10-23)
Available here: ([ANSI escape code - Wikipedia](#))

Wikipedia (2023-10-15) List of Unicode characters [Online] (Visited 2023-10-23)
Available here: ([List of Unicode characters - Wikipedia](#))

ruwix (Missing date) Rubik's Cube Notation [Online] (Visited 2023-10-23)
Available here: ([Rubik's Cube Notation - What the rotation letters mean: F R' U2 \(ruwix.com\)](#))

[SEP]

A Appendix 1

The abstract is generated by using chat GPT on the teachers instructions.