Assignment-02 Report

Samuel Berg

1DT905

Rubik’s Cube in Java

- Using JavaFX to make it in 3D with

an algorithmic solver

TODO: Fis solver to work after user inputs either variation of (M, E, S); These do not work due to changing the center color of each side which current solving algorithm is dependent on in a way.

Abstract

This report delves into the creation of a Rubik's Cube application in Java, featuring 3D visualization, JavaFX integration, and an algorithmic solver. The subject matter revolves around the development process, with a specific focus on the challenges faced during this project.

The primary motivation behind this endeavor was to create an interactive 3D Rubik's Cube application, allowing users to manipulate the cube, scramble it, and employ a solving algorithm. The methods employed encompass Java programming, JavaFX for 3D rendering, and iterative problem-solving.

The report offers insights into the development stages, from analysis and design to implementation, testing, and quality assurance. Notably, it addresses the limitations encountered, such as the algorithm's inability to solve certain input types and a poorly structured codebase.

In conclusion, the project was a valuable learning experience, emphasizing the need for careful planning, structured coding, and comprehensive testing. This abstract serves as a teaser, providing a glimpse into the project's scope and highlighting the importance of addressing software development challenges.

Keywords: 3D Visualization, Rubik’s Cube, Java, JavaFX, Algorithmic Solver

Contents

1 Introduction

The project in this report covers the things we have gone through in our curriculum in Java. In this report we will go through the various stages of making and planning the project and how I solved problems that occurred. And problems that occurred that I could not solve and how I learned from them. Going over the reason they happened and reasoning for how they could have been solved.

1.1 Link to GitLab and Code

The source code and project files can be found on GitLab at the following link: GitLab Repository.

2 Analysis

I started out thinking about how to build the Rubik's cube in the sense of a cube that would work in 2D and 3D, how I would represent the colors in 2D and how I wanted to structure my code with classes and inheritance.

2.1 Text describing the system

The application should be a Rubik's cube application in either 2D or 3D. I am aiming to make it in 3D. It should have functionalities of making moves by user input and to reset it. It is supposed to have functionalities of scramble, solve solving algorithm, take a string input and it should be able to be rotated in the 3D space.

2.2 Prototype

I started out making a prototype in 2D which I got to work to about 75% of a fully functioning Rubik's cube application which I later scrapped when I realized I had problems converting the application to 3D. On the 2D version of the cube I had to put time in thinking about how to make the matrices for the cube to move as intended as well as making it readable in the console which was where the 2D version was “shown”. When it came to the 3D version of the cube, I had a bit less to worry about the matrices and more to worry about learning and implementing JavaFX properly. The 3D version I started to make here made it to be the definitive version.

2.3 Analysis of similar applications

I analyzed similar applications both by trying them out and looked at git repositories with Rubik's cube applications or similar applications in several languages to get an understanding of how one could or should structure these types of applications. This gave me great insight into what I wanted to do and how I wanted to structure my application, at least the fundamentals of it.

2.4 Use cases

2.5 Class diagram

Analysis UML: UML diagram of the application as a thought before implementation.

3 Design

When planning design wise for the GUI I just had a simple image in my mind of screen of a mobile phone with about 2/5 of the screen being the tabs/options menu where you would press to make moves as well as using the scramble, solve, solving algorithm and user input functionalities. And the remaining 3/5 of the screen being a light blue background with the cubes 3D model being shown in front being able to rotate it fully in the X, Y and Z axes. Below follows a picture of the design.

GUI design: This is the final product design which aligns with the design thought of before production.

When planning the design of the code structure I was thinking of using inheritance for a lot of the classes to relate to the cube to make it more intuitive, maintainable, and readable. Also planned of putting the GUI in its own separate class in the project to make it easy to modify and restructure.

When planning for the solver my first idea was to just make to moves made in reverse but this, I decided not to go with due to it feeling a bit too simple so I decided to make use of a solving algorithm one of my family members knew and noted down how I could use that to solve the cube at the press of a button on the GUI.

3.1 Updated class diagram

Updated UML The updated version of the UML diagram for the completed version of the applicationUpdated UML: UML diagram of the definitive version of the application.

4 Tools and techniques

I used tools such as DrawIO for the making of UML diagrams.

I used the JavaFX library to be able to implement the 3D version of my cube and represent it and to handle events. Techniques used to assist in making it easier for example using Gradle as structuring the code, I also used the method of writing the code in the manner of always being able to test it visually by making the application window as early as possible into production, to then implement function by function. I also used SonarQube for static code analysis throughout the project to try and keep it as readable, maintainable, and functional as possible.

5 Implementation

I started out trying to implement a 2D version which I got a bit stuck on after about four to six hours of work. After that had happened I more or less just scrapped all of the work I did on the 2D version and just decided one afternoon to sit down and just get as far as possible knowing I wanted to do a 3D version towards the end and I managed to make a functioning 3D cube which I afterwards only made minor tweaks and fixes to. The solving algorithm seems to have issues solving manual user input but not the scramble functions cubes for some unknown reason. When I got to the unit testing, I realized that I had baked in the GUI and cubes code so much it made it difficult to actually make any useful/appropriate tests without using GUI testing which at the time I felt like I didn't have the time left over to learn properly enough to use in a good way. I still tried to implement tests to various functions I thought possible but most of them failed due to them being heavily reliant on the GUI due to lack of design pattern thoughts when making the actual code. Also, this code will not be maintained due to those reasons. Aswell though if one spends enough time reading the Javadoc comments and the code you can see it all connecting but difficulties finding the correct way of splitting the code up into smaller bit size pieces which would be necessary for the code to be maintainable. This experience I will learn from and hopefully avoid in the future to do the same mistakes when I just sit down and code for several hours straight to pause and try to adjust the design towards being much more maintainable and readable.

I used Gradle to initialize the project structure,to compile, to try and run tests, to generate javadoc and to read and understand error messages throughout the project's timeline. When it comes to design patterns, I did sadly not use any of them at least not

intentionally due to me having completed nearly the entire application by the time we had a lecture on it there for I had no real option to implement them without going back and redoing most of the application due to how bad the code ended up. External libraries used in the project are JavaFX and Javas own library for example for `SecureRandom`.

My Javadoc comments for functions as the rotations/moves are just copies of each other except for changing the rotation it is preforming in the comment and the same goes for a lot of functions which did the same thing but for example in a different direction. As mentioned above I was sadly not able to do any real testing due to how intertwined my GUI and actual application code ended up which led to me not being able to perform any unit tests at all which has led me to not having any code coverage at all. Which also has resulted in me only having one of the messures graph in SonarQube because what I would have compared it to would have been to the one where the tests had given me sufficient code coverage. Furthermore, when it comes to SonarQube I used it throughout the project to keep track of so-called “bugs,” “code smells” and “vulnerabilities” as they are called in SonarQube which helped me remedy a lot of such things. Under the following Quality assurance header, you will find out more about this.

6 Test and coverage

I tried testing every method throughout my code by plain testing, mocking without a mocking library and overriding. After doing all of that I still seem to be getting stuck at calling a GUI dependent piece of code along the way resulting in a NullPointerException. Possible ways this could be tested properly would be using a mocking library as Mockito or by doing GUI tests. GUI testing would be the smart and supposed way of testing the code but due to it not being part of this course curriculum I realized that I forced myself into a corner of needing to use GUI test to be able to test my code at all.

7 Quality assurance

I have worked with SonarQube every time I got something new to work as intended, I ran the `./gradlew run sonar -(parameters for SonarQube project)`. Now that the project is finished this is how the activity graph ended up.

SonarQube Activity Graph: Picture of the SonarQube activity graph starting on the 11th of October and ending on the 23rd of October.

Now sadly I could not perform any unit tests on my code without learning how GUI tests work and there for there is only when picture of the following measures graph from SonarQube showing the “technical debt” of my code in its current state.

SonarQube measures graph: Graph representing the code coverage over technical debt.

8 System/Application

This is how my final application looks like.

With the lighter green representing the tab you are currently previewing.

In the “Manual input” tab you perform a move according to its notation by pressing on the notation of that move with your mouse pointer and by pressing the “Rs” button you reset the cube. The “Y” and “Y’” you rotate the cube in its Y axis.

In the “Options” tab you have buttons connecting you the functions doing/giving you what they say as for example the “Scramble” button scrambles the cube for you for as many moves as necessary to make it difficult, the “Solve” button solves any version of the cube except for when you have given manual input which moves the cubes center boxes which will make the solving algorithm break, though as long as you don’t switch the positions of the center pieces of each face it works well and when it has solved the cube for you, you can press the "Solving Alg." button and there will be a pop-up window showing you all the moves made to solve the cube from its scrambled position. The “Input” button lets the user input a long string without spaces of moves the user wants it to perform. An example of such a string would be “FBF’B’R’R” which will return it to its solved state if no prior moves have been performed in the application. The arrows that exist on the right side of the options tab are for adjusting the speed of the animation of the moves performed on the cube to preset speeds that avoids any crashes in the application as far as my personal testing goes.

You also as the user have the possibility of with your cursor and spin the cube multidirectional in the X, Y and Z plane.

Limitations the application has is the solving algorithm sadly not solving all types of user input that the user can give, other limitations are that the code base is poorly structured making it exceedingly difficult to maintain and/or read and this possibly has an impact on the performance of the application. One other limitation of the application is that if you press the solve button many times rapidly and possibly others you will generate a “StackBufferOverflow” resulting in an “Exception.”

If I would develop the application further, I would redo the code base to be more structured in the way mentioned in Analysis and Design stages of the project while keeping most of the logic I must use it again. Other improvements would be to fix the solving algorithm, the solve function used to solve the cube no matter the user input which I struggle with currently and to try and remodel the code to minimize the number of exceptions that can occur. One more thing that I might would have liked to add is a solving function that just takes the moves performed and goes through them backwards which would be a fun thing to try and see how much of a time difference there would be in between the two to solve a state of the cube.

9 Summary & Conclusion

This report details the development of a Rubik's Cube application in Java using JavaFX for 3D visualization with an algorithmic solver. The report begins with an abstract, highlighting the project's subject area, motivation, methods used, and a summary of results. The project goes through stages of analysis, design, implementation, testing, and quality assurance. The application aims to create a functional 3D Rubik's Cube with user input, scrambling, solving, and rotation features. The report also discusses the use of tools and techniques, such as Gradle, SonarQube, and external libraries like JavaFX.

In retrospect, this project provided valuable insights into the development process. It is evident that there were both successful and challenging aspects throughout the project. If I were to tackle this project again, I would approach it with a more structured mindset from the start. Planning and design are crucial in avoiding the pitfalls I encountered, especially with regards to the intertwined GUI and application code, which hindered proper testing. Design patterns should be intentionally implemented, to enhance maintainability and readability. The limitations in the solving algorithm and the code's structure would be my top priorities for improvement. The application should be able to handle a wider range of user input types, and the codebase should be restructured to enhance maintainability. Additionally, addressing the exception issues, such as the StackBufferOverflow, is essential for a smoother user experience.

In conclusion, this project has been a valuable learning experience. It highlights the importance of clear planning, structured design, and testing from the preliminary stages of development. It also underlines the potential benefits of incorporating design patterns in non-traditional projects. Moving forward, these lessons will be applied to ensure future projects are more efficient and effective.

9.1 TIL;

I learned that I need to think more in the moment of programming and not just go with what appears in my head or my hands just writes due to it being what got me into the mess of not being able to test my code. I also want to try to implement design patterns more in my own projects though most of them are not Object-Oriented but still it seems as a nice/useful technique to use to one's own benefit.

References

[1] jperm.net. (2019) “3x3 Rubik’s Cube Move Notations.” Available: https://jperm.net/3x3/moves. Accessed: Oct 11th, 2023

A Appendix 1

In the appendix you can put details that do not fit into the main report, but you still want to document them.