

**REDI CUBE (CORNER-TURNING 3D PUZZLE) SIMULATOR AND SOLVER**

By

Lalu Aria Sultan Madani

001201600032

A Thesis

Submitted to the Faculty of Computing

President University

In Partial Fulfilment of the Requirements

For the Degree of Bachelor of Science

In Information Technology

Cikarang, Bekasi, Indonesia

October 2019

Copyright by

Lalu Aria Sultan Madani

2019

**INTERIM REPORT**

**REDI CUBE (CORNER-TURNING 3D PUZZLE) SIMULATOR AND SOLVER**

By

Lalu Aria Sultan Madani

001201600032

Approved:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Rikip Ginanjar M. Sc  
Thesis Advisor

# ABSTRACT

Rubik’s Cube is a widely popular mechanical puzzle that has attracted attention around the world because of its unique characteristics. As a classic brain-training toy well known to the public, Rubik’s Cube was used for scientific research and technology development by many scholars.

However, Rubik’s cube is not very ideal for the people who are new in 3D puzzle because of the complex nature of Rubik’s Cube. Redi Cube is a simpler puzzle to learn and solve for people who are new in 3D puzzle, but Redi Cube are still very new in the market and not readily available thus making Rubik’s cube still the popular choice despite it is more complex to learn.

This research will make an application that will teach the user to play around and solve Redi cube without needing to learn complex algorithm as in Rubik’s cube.

# **TABLE OF CONTENTS**

[ABSTRACT i](#_Toc22859610)

[TABLE OF CONTENTS ii](#_Toc22859611)

[CHAPTER I INTRODUCTION 1](#_Toc22859612)

[1.1 Background 1](#_Toc22859613)

[1.2 Problem Statement 2](#_Toc22859614)

[1.3 Research Objective 2](#_Toc22859615)

[1.4 Scope and Limitation 2](#_Toc22859616)

[1.5 Thesis Methodology 3](#_Toc22859617)

[1.6 Thesis Outline 5](#_Toc22859618)

[CHAPTER II LITERATURE STUDY 7](#_Toc22859619)

[2.1 Representing 3D Object 7](#_Toc22859620)

[2.2 Back Face Culling 9](#_Toc22859621)

[2.3 Related Work 10](#_Toc22859622)

[CHAPTER III SYSTEM ANALYSIS 13](#_Toc22859623)

[3.1 System Overview 13](#_Toc22859624)

[3.2 Functional Analysis 13](#_Toc22859625)

[3.3 Hardware and Software Requirements 14](#_Toc22859626)

[3.4 Use Case Diagram 15](#_Toc22859627)

[3.5 Use Case Narrative 16](#_Toc22859628)

[3.6 Swim Lane Diagram 19](#_Toc22859629)

[CHAPTER IV SYSTEM DESIGN 23](#_Toc22859630)

[4.1 User Interface Design 23](#_Toc22859631)

[4.2 Physical Design 25](#_Toc22859632)

[4.3 Class Diagram 26](#_Toc22859633)

[4.3.1 TPoint Class 26](#_Toc22859634)

[4.3.2 TMatrix Class 26](#_Toc22859635)

[4.3.3 TVector Class 27](#_Toc22859636)

[4.3.4 TSurface Class 27](#_Toc22859637)

[4.3.5 TObject Class 28](#_Toc22859638)

[4.3.6 MainForm Class 28](#_Toc22859639)

[REFERENCES 30](#_Toc22859640)

# INTRODUCTION

This chapter aims to explain the topic of the thesis, its objective, and the method that is used. This chapter also briefly describe the details of the thesis chapters. There are six sections on this chapter, which are: Background, Problem Statement, Thesis Objective, Scope and Limitation, Thesis Methodology, and Thesis Outline.

## Background

Redi Cube is a 3D puzzle invented by Oskar Van Deventer in 2009 and started mass production in 2017 by MoYu corporation. Redi cube is not a WCA (World Cube Association) puzzle thus making Redi cube quite a rare puzzle to find.

To start in playing 3D puzzle, many of people started playing Rubik’s cube because it is readily available but starting with such a complex puzzle for first timer is very challenging and can take a long time especially for first timer. With Redi cube, people can learn how to solve a 3D puzzle intuitively without needing complex algorithms to memorize.

To make the learning process more fun and less frustrating, the application will be made with 3D viewer that can be used to simulate the puzzle as in real life and be able to view the puzzle in any direction, the application will come with automatic shuffle and solver that can also teach the user about the algorithm to solve Redi cube.

## Problem Statement

Specifically, this thesis aims to solve the following problems:

1. How to make a 3D Object Viewer
2. How to make a solver’s algorithm for Redi Cube
3. How to shuffle Redi Cube
4. How to simulate Redi Cube movement

## Research Objective

This research is carried out with the following objectives:

1. To implement an algorithm to solve Redi Cube.
2. To create an application that simulate Redi Cube.
3. To create an application that shows 3D objects.

## Scope and Limitation

This research focuses on:

1. Implementing an algorithm that solve Redi Cube.
2. Creating an application that simulate Redi Cube.
3. Creating an application that shows 3D objects.

The limitation of this research is that the application will only works on Redi cube only, making the application works on other type of puzzle is not possible in this implementation.

## Thesis Methodology

The methodology of this thesis can be break down into four phases, which are as follows:

1. Theoretical Review

This is the part where all the literatures and journals that related to 3D graphics, 3D puzzles, and solving algorithms are reviewed to give a summary or an overview about the topic being studied.

1. Problem Identification

The summary or overview about the 3D graphics, 3D puzzles, and solving algorithms formed from the previous phase are analysed thoroughly to generate a broad range of problems and then narrowing those problems to focus only on the most important for the objectives of this thesis.

1. Proposed Solution

This phase focus only on finding a final solution based on the problems identified from the previous phase by using available literatures.

1. Work Implementation

In this phase, the proposed solution is implemented into an application to test whether the solution gives the desired output or not. The methodology used in the development of the application is Rapid Application Development (RAD) methodology since RAD can adapt to projects with small team or an individual member and aims to use minimal planning in favour of rapid prototyping to develop the product faster with higher quality. RAD consists of four phases for application development, which are as follows:

1. Requirement Planning Phase

The objective of this phase is understanding the business process of the application. Developer should identify what are the required features that need to be exist in the Redi Cube Simulator and Solver

1. User Design Phase

This phase focus on analysing in detail the business activities, data and process of the Redi Cube Simulator and Solver application, develop the application structure in terms of features and functions that will comprise the system, and develop the functional screen layouts and transitions of the important features of the Redi Cube Simulator and Solver application.

1. Rapid Construction Phase

In this phase, the developer will focus on creating functional and integration-tested prototypes of the Redi Cube Simulator and Solver application. The result of this phase is a complete detailed design of the application.

1. Cutover Phase

The final phase tests the features, methods and performances of the application using a set of predefined test cases. Developers should ensure the Redi Cube Simulator and Solver application performs properly.

## Thesis Outline

The thesis consists of seven chapters, which are as follow:

1. CHAPTER I INTRODUCTION

The introduction chapter contains background, problem statement, research objective, scope and limitation, methodology, and outline of this thesis.

1. CHAPTER II: LITERATURE STUDY

This chapter contains explanation about the basic techniques and methods that are used in the 3D viewer, such as 3D object representation and Back face culling.

1. CHAPTER III: SYSTEM ANALYSIS

The system analysis chapter consists of system overview, comparison overview, functional analysis, development processes analysis, hardware and software requirement, use-case diagram (with and without narrative description), and swim lane diagram of the application.

1. CHAPTER IV: SYSTEM DESIGN

The system design describes the definition of the program’s architecture, components, and modules. This chapter consists of User Interface Design, Physical Design, and Class Diagram of the program.

# LITERATURE STUDY

This chapter aims to explain the concepts and techniques that are related to or used in the development of the Redi Cube Simulator and Solver application. The early sections will explain about the basic techniques that are widely used in 3D viewing and will be used extensively in this application.

## Representing 3D Object

A 3D object is an object in 3D space that consists of interconnected polygon called polygon mesh, a polygon can be represented with triangle surfaces because any shape of polygon can be represented by triangles, each triangle consists of 3 points in 3D coordinates called vertices (plural: vertex). In this thesis the application will represent a polygon mesh with list of vertex indices where each polygon mesh has a list of all the vertices in the mesh and a list of all the triangle surface represented each three indices referencing to the index on the list of all vertices, this method of representing a 3D object have a benefit of easier transformation on the object because all the application need to transform is the list of all vertices. Figure 2.2 shows a polygon mesh and each vertex index and surface. Table 2.1 shows the vertex list and Table 2.2 shows the surface list.

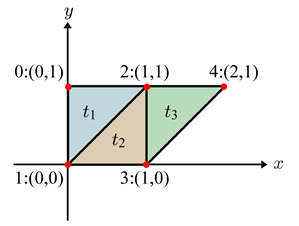


Figure 2.1 Polygon mesh representation in list of Vertex indices

Table 2.1 Vertex list

|  |  |
| --- | --- |
| Index | Coordinates (x, y) |
| 0 | (0, 1) |
| 1 | (0, 0) |
| 2 | (1, 1) |
| 3 | (1, 0) |
| 4 | (2, 1) |

Table 2.2 Surface list

|  |  |
| --- | --- |
| Index | Index of Vertices (p1, p2, p3) |
| 0 | (0, 1, 2) |
| 1 | (1, 3, 2) |
| 2 | (2, 3, 4) |

## Back Face Culling

Back-face culling is a method to remove surfaces that are not visible (not facing the viewer) not only displaying the surfaces that are only visible to the viewer but also improving performance. To determine whether a surface is visible, the angle formed by its normal and the view direction must be more than 90**°**. The normal vector of a surface is obtained from the cross product between two vectors on the surface itself. The first vector is obtained by forming a vector from the one vertex to another vertex of that surface. The second vector is obtained by forming a vector from the first vertex to the remaining vertex. If the result of dot product of its normal and the view direction is less than 0, it is considered as visible surface. Figure 2.2 shows the vector operation of back face culling.

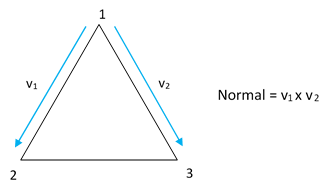


Figure 2.2 Back face culling

## Related Work

Currently there are no simulator for Redi Cube in the internet yet, but there are some works related to 3D puzzle simulator in other puzzle format.

1. Google Rubik’s Doodle [1]

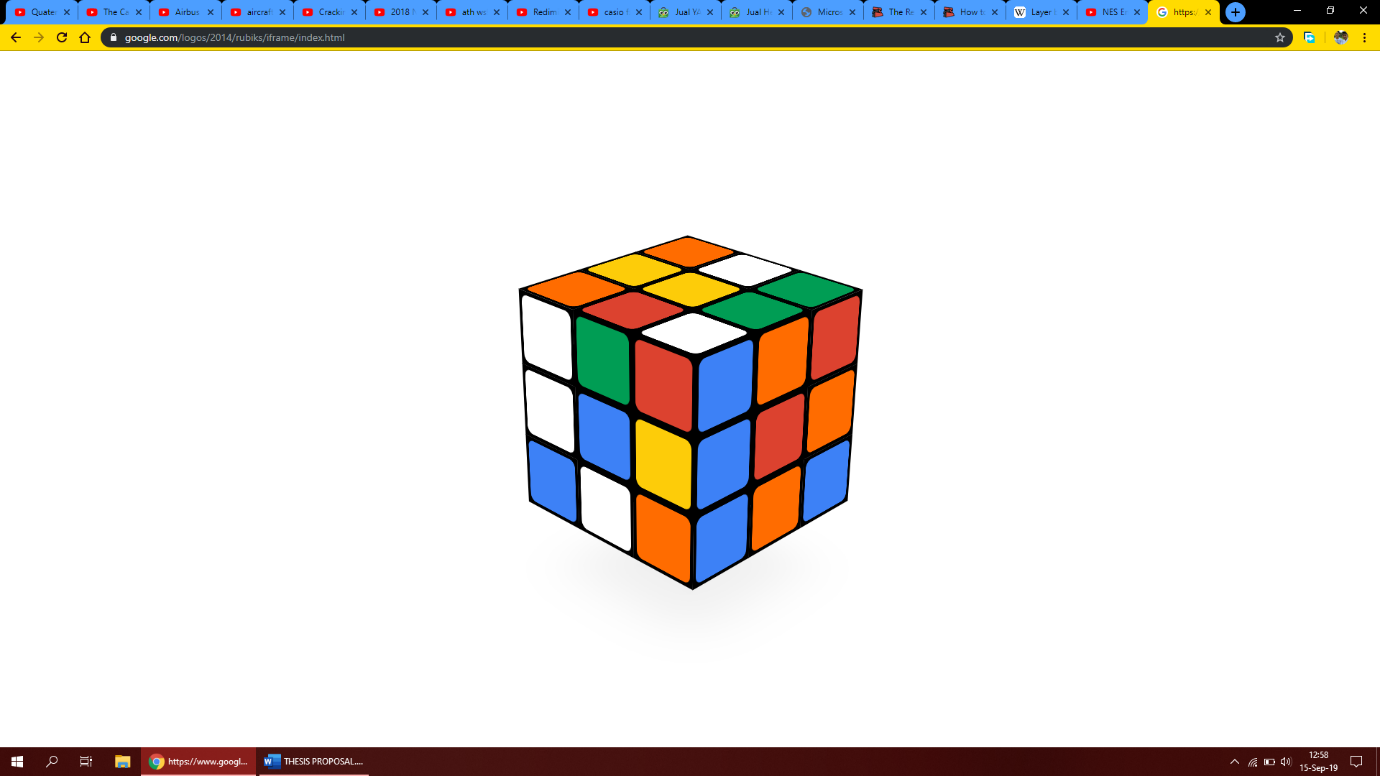


Figure 2.3 Google Rubik’s Doodle

Google made a Rubik’s cube simulator in 19th of May 2014 to commemorate 35th year of the invention, the app featured the visualization of the cube and also a shuffling feature but the app does not have automatic solver and cannot be viewed in any direction

1. Rubiks-cu.be Solver [2]

Rubiks-cu.be Solver is a website that features layer-by-layer solving algorithm for Rubik’s cube, the website will solve a shuffle by the website or inputted by the user. The website lacks the feature of viewing the cube in 3D.

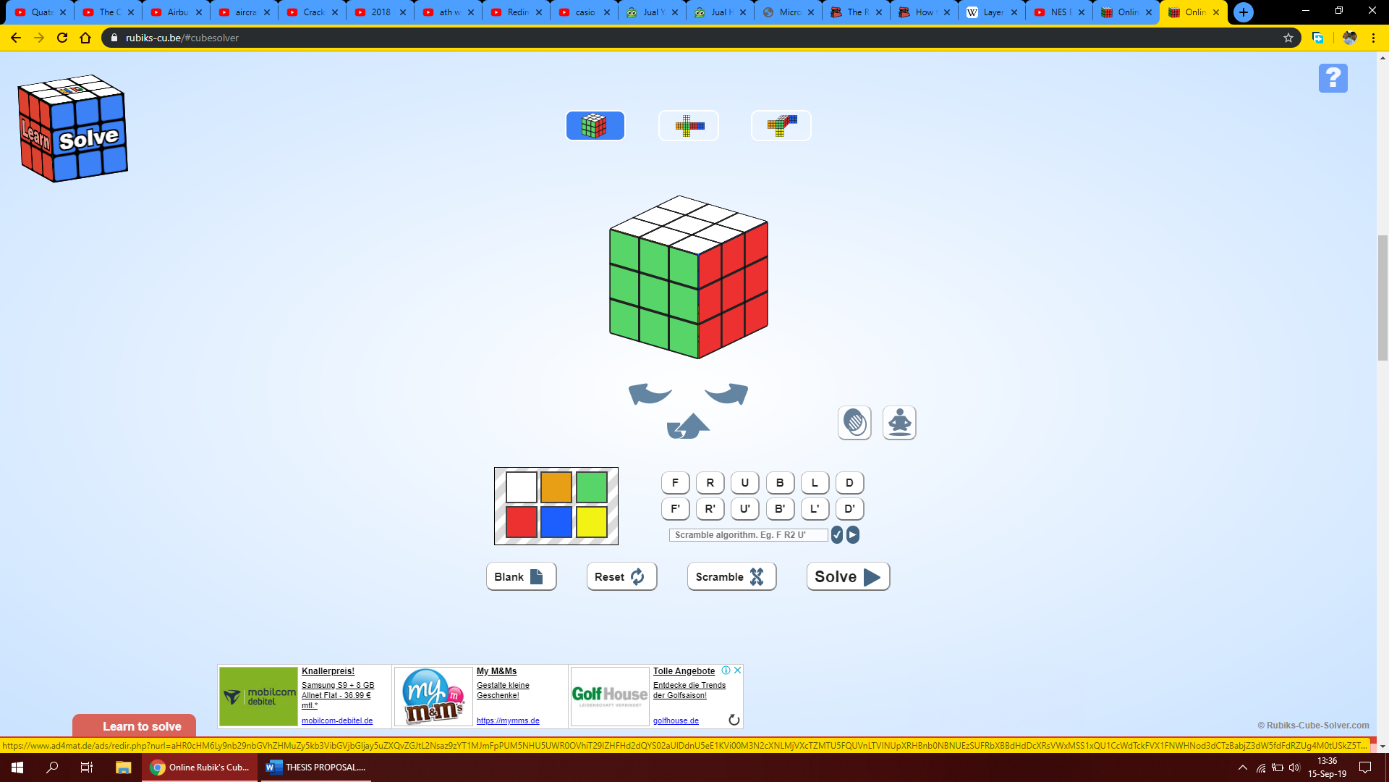


Figure 2.4 Rubiks-cu.be Solver

1. Snix

Snix is a Rubik’s Snake simulator made by Natasha Janice that simulate the movement of Rubik’s snake in 3D, it also has scripted animation to make the Rubik’s snake into many shapes.

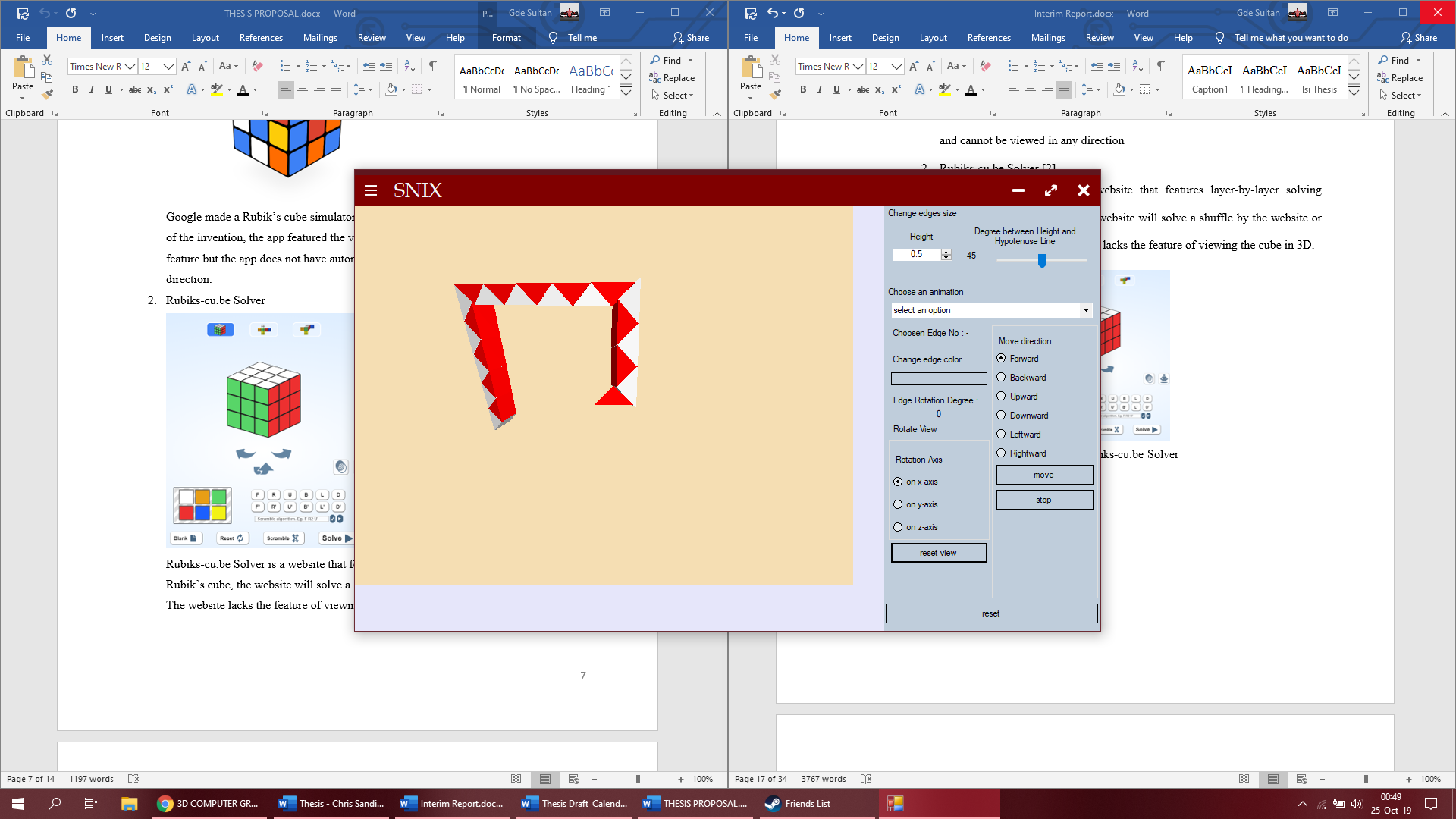


Figure 2.5 Snix

# SYSTEM ANALYSIS

System analysis aims to explain the analysis of the program functions and behaviours based on the prescribed requirements. This chapter described the application requirements and development procedure in order to identify its goals and purpose. It consists of System Overview, Comparison Overview, Functional Analysis, Development Process Analysis, Hardware and Software Requirement, Use Case Diagram, Use Case Narrative, and Swim Lane Diagram.

## System Overview

This program serves as an educational purpose desktop application that its main functionality is to show the process on how to solve a Redi Cube, shuffle Redi Cube and show Redi Cube movement in 3D viewer.

## Functional Analysis

Table 3.1 shows the functional requirements description of the Redi Cube Simulator and Solver.

Table 3.1 Redi Cube Simulator and Solver Application Functionality Table

|  |  |
| --- | --- |
| **No** | **Function Description** |
| 1 | Allow user to play Redi Cube in 3D |
| 2 | Allow user to solve Redi Cube manually and automatically |
| 3 | Allow user to shuffle Redi Cube |
| 4 | Allow user to solve Redi Cube with timer |
| 5 | Allow user to show the movement of a Redi Cube |

## Hardware and Software Requirements

The development of the Redi Cube Simulator and Solver application requires a specific software development application and some specific hardware. There are listed as follows:

* Personal Computer

Personal Computer serves as the workstation of the development process of this application. The development of this application will be on a PC running on Windows 10 operating system.

* Visual Studio

Visual Studio is a complete set of various software development tools and it supports various Integrated Development Environment (IDE) too. Visual C# will be the supporting IDE to develop this application.

* Microsoft Word

Microsoft Word will be used as the documentation tools of the whole process of the application’s development.

## Use Case Diagram

Use-case diagram is a diagram which describes a set of sequential actions (use cases) that the application can or should perform in collaboration with the user. It defines the process flow of the application when it’s running in user’s point of view. The use-case diagram for this application is described in Figure 3.1.

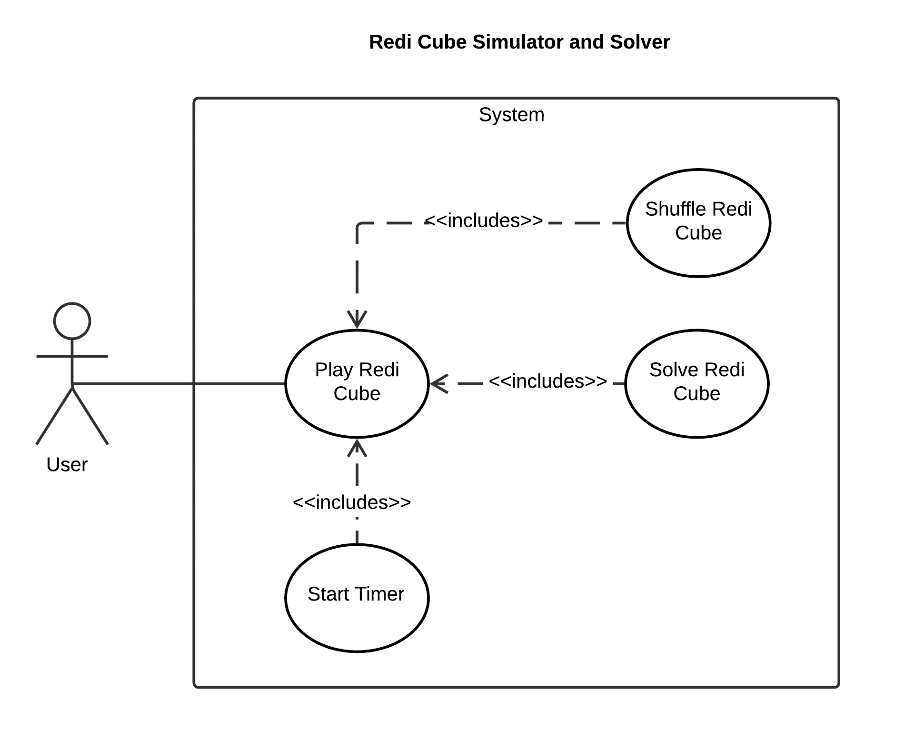


Figure 3.1 Use-case Diagram of Redi Cube Simulator and Solver Application

The actor in this application is user. The user needs to start the application first in order to interact with the application. The program initially starts with displaying Redi Cube and is ready to play with. The user has the option to shuffle, solve, and start the timer by clicking each button respectively.

## Use Case Narrative

Use-case narrative is a textual representation of the course of events encountered when the user is interacting with the application. It describes the detail of use-cases such as what conditions must be met before and after the interaction, what triggers the interaction, the expected results, etc. The Use Case Narratives of this application are described in Table 3.2 to Table 3.5.

Table 3.2 Use Case Narrative for Playing Redi Cube

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Playing Redi Cube | |
| Use Case ID | UC01 | |
| Priority | High | |
| Primary Business Actor | User | |
| Primary System Actor | System | |
| Others Participating Actor | None | |
| Description | This use-case describes the event when user start the application to play Redi Cube | |
| Precondition | None | |
| Trigger | User Start the application | |
| Typical Course Event | **Actor Action** | **System Response** |
|  | Step 1: User start the application | Step 2: Display the Redi Cube |
|  | Step 3: User turn the Redi Cube corner piece | Step 4: The system displays the Redi Cube in its new position based on Step 3 |
| Alternate Course | None | |
| Post Condition | Redi Cube corner piece is successfully turned | |

Table 3.3 Use-case Narrative for Shuffle

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Shuffle Redi Cube | |
| Use Case ID | UC02 | |
| Priority | High | |
| Primary Business Actor | User | |
| Primary System Actor | System | |
| Others Participating Actor | None | |
| Description | This use-case describes the event when user decide to shuffle the Redi Cube | |
| Precondition | Application is started | |
| Trigger | User clicked the “Shuffle” button | |
| Typical Course Event | **Actor Action** | **System Response** |
|  | Step 1: User clicked the “Shuffle” button | Step 2: The system shuffles the Redi Cube |
| Alternate Course | None | |
| Post Condition | The Redi Cube is shuffled | |

Table 3.4 Use-case Narrative for Solve

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Solve Redi Cube | |
| Use Case ID | UC03 | |
| Priority | High | |
| Primary Business Actor | User | |
| Primary System Actor | System | |
| Others Participating Actor | None | |
| Description | This use-case describes the event when user decide to solve the Redi Cube | |
| Precondition | Application is started | |
| Trigger | User clicked the “Solve” button | |
| Typical Course Event | **Actor Action** | **System Response** |
|  | Step 1: User clicked the “Shuffle” button | Step 2: The system shuffles the Redi Cube |
| Alternate Course | None | |
| Post Condition | The Redi Cube is shuffled | |

Table 3.5 Use-case Narrative for Start Timer

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Start Timer | |
| Use Case ID | UC04 | |
| Priority | Medium | |
| Primary Business Actor | User | |
| Primary System Actor | System | |
| Others Participating Actor | None | |
| Description | This use-case describes the event where the user decides to start timer to count the time to solve the Redi Cube | |
| Precondition | The Redi cube is not solved | |
| Trigger | User clicked the “Timer” button | |
| Typical Course Event | **Actor Action** | **System Response** |
|  | Step 1: User clicked the “Timer” button | Step 2: The system shows the time elapse |
|  | Step 3: User play the Redi Cube | Step 4: The system stops the timer when the Redi Cube is solved |
| Alternate Course | None | |
| Post Condition | Timer is started | |

## Swim Lane Diagram

Swim lane diagram defines the process from input to output in detail which sub-processes are responsible when the program runs. Swim lane diagrams of Redi Cube Simulator and Solver application are shown in Figure 3.2 until Figure 3.5.

1. Playing Redi Cube

Figure 3.2 describes the system process when user playing with the Redi Cube, user first must select a corner to turn and then the system will highlighted the corner selected, the user then move turn the corner piece by holding the mouse left button in the play area and drag the corner piece, after the user release the left click button the system will display the corner piece in the new position, it will repeat those steps until the Redi Cube is solved then a Solved message is shown.

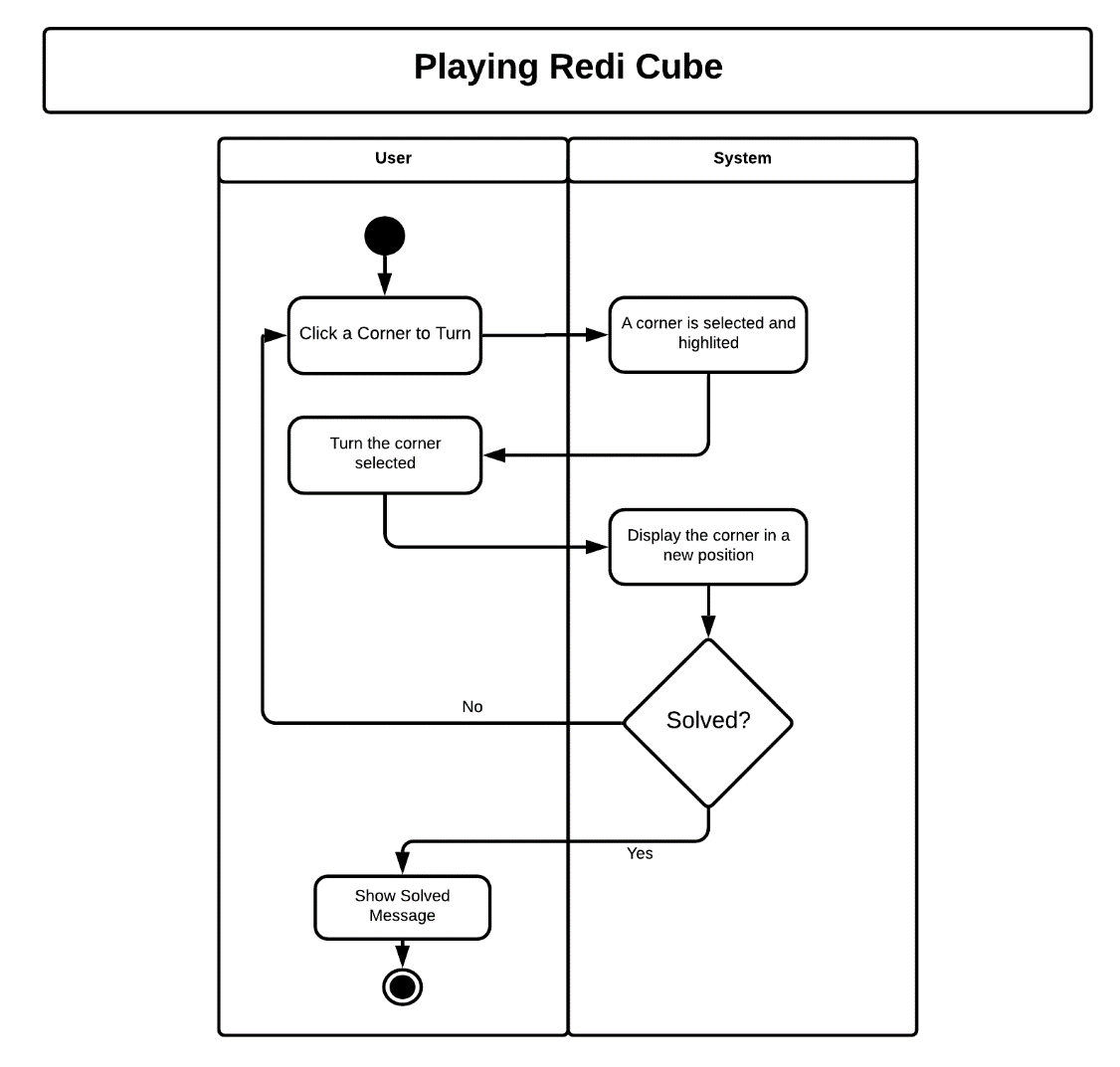


Figure 3.2 Swim-lane Diagram of Playing Redi Cube

1. Shuffle Redi Cube

Figure 3.2 describes the system process when the user chooses to shuffle the Redi Cube, the Redi Cube is shuffled by randomly choosing a corner and turning it in a random direction 30 times, the user can play the Redi Cube after the Redi Cube is being turned randomly 30 times.

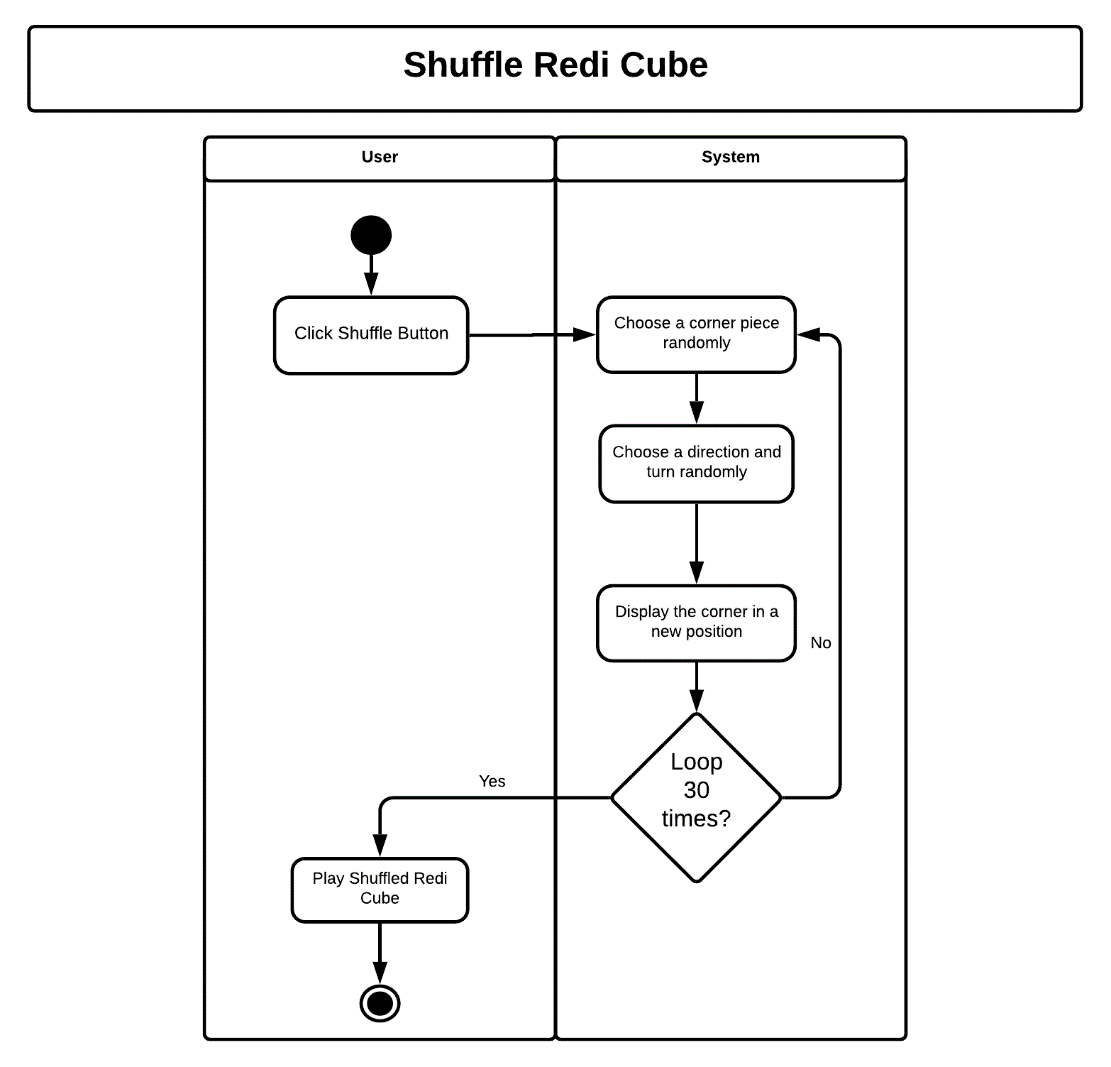


Figure 3.2 Swim-lane Diagram of Shuffle Redi Cube

1. Solve Redi Cube

The user starts this action by clicking on the “Manually Fill Array Content” radio button. On clicking, the content of group box for each sequence will be changed with a rich text box that require the user to fill the elements for each sequence manually. There is some validation that need to be followed in order to generate a valid array.

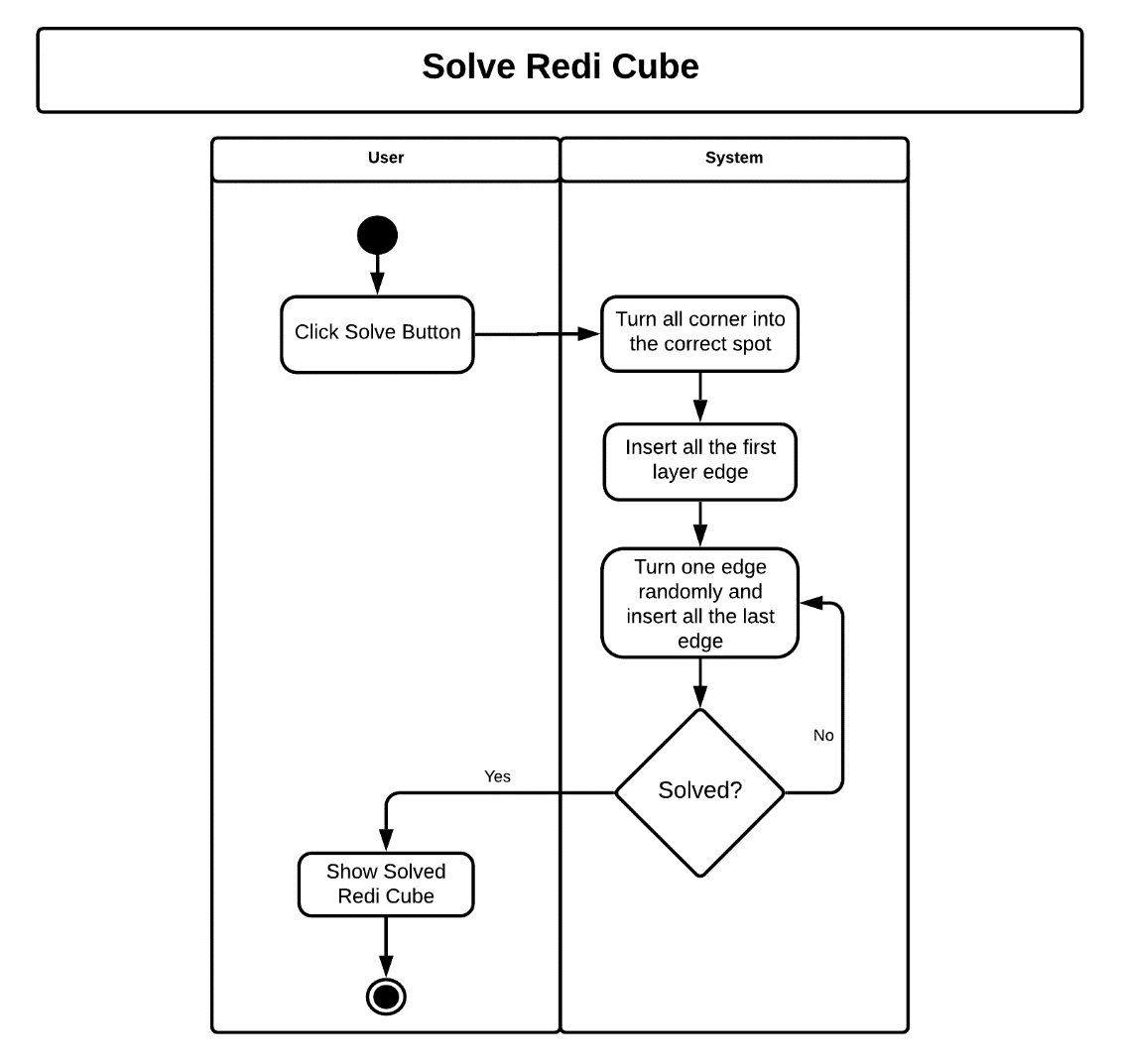


Figure 3.4 Swim-lane Diagram of Solve Redi Cube

# SYSTEM DESIGN

System design will define the architecture, components, modules, and data of a system to satisfy the specified requirements. It is an intermediate stage between system analysis and product development where analysis that have been defined previously are visualized in details here. The system design of Redi Cube Simulator and Solver application consists of three sections: User Interface Design, Physical Design, and Class Diagram.

## User Interface Design

User interface design is one of the most crucial things in developing the applications as it is the component user interact with. A well-designed user interface will determine the user friendliness and ease of use of the application thus makes the user will be more likely to increase the understanding of the features that the application is trying to facilitate.

This application is going to be developed on a desktop-based application using the software development tools defined in previous chapter. As of many with other desktop applications, this application has one main window, which will the first display after the user opening the executable file of the application. The main window will contain several objects that functions independently, such as: Title bar, Labels, Buttons, and etc. Figure 4.1 shows the user interface sketch design of Redi Cube Simulator and Solver application’s main window, and the details are described in Table 4.1.

Table 4.1 . Redi Cube Simulator and Solver Application Description Table

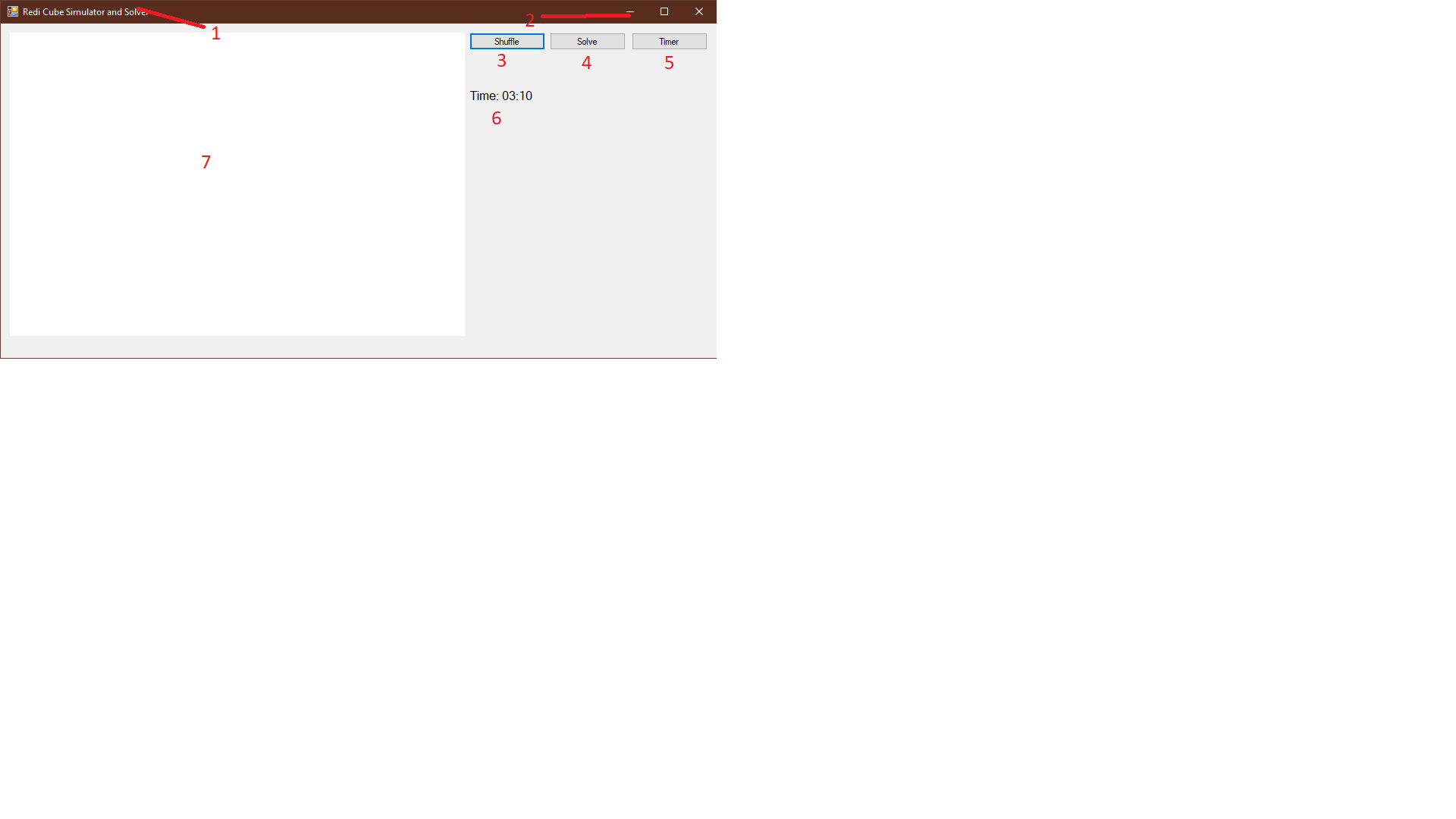


Figure 4.1 User Interface for Redi Cube Simulator and Solver Application

|  |  |
| --- | --- |
| **No** | **Description** |
| 1 | “Redi Cube Simulator and Solver” Title Bar |
| 2 | Tittle Bar Buttons |
| 3 | Shuffle Button |
| 4 | Solve Button |
| 5 | Timer Button |
| 6 | Timer Time Label |
| 7 | Play Area Window |

## Physical Design

Physical design section defines the minimum software and hardware requirements in order to run the Redi Cube Simulator and Solver application. Compliance with the requirements stated will ensure the application runs well. The description of physical design of Redi Cube Simulator and Solver application is shown in Table 4.2 and Table 4.3 for software and hardware specification, respectively.

Table 4.2 Software Tool Requirements for Redi Cube Simulator and Solver Application

|  |  |  |
| --- | --- | --- |
| **No** | **Field** | **Description** |
| 1 | Operating System | Windows 10 |
| 2 | Programming Language | C# |
| 3 | Program Development | Visual Studio Community 2019 |
| 4 | Documentation | Microsoft Word 2016 |

Table 4.3 Hardware Requirements for Redi Cube Simulator and Solver Application

|  |  |  |
| --- | --- | --- |
| **No** | **Field** | **Description** |
| 1 | Processor | Minimum requirement: Intel Core i series |
| 2 | Memory | Minimum requirement: 2GB of RAM |
| 3 | Monitor | Minimum requirement: Resolution 1280x720 |
| 4 | Hard Drive | Minimum requirement: Free space of 100 MB |

## Class Diagram

This section shows the classes of the Redi Cube Simulator and Solver application, the relationships between classes, and the operations and the attributes of the classes. Class diagram for this application is shown in Figure 4.8

### TPoint Class

This class is the data structure for vertex, it holds the 3D homogenous cartesian coordinates of a vertex. It also has three separate method to initialize its value, method to multiply it by 4x4 matrix, and a method to normalize the vertex to the same cartesian coordinate. The preview of this class is shown in Figure 4.2.

|  |
| --- |
| **TPoint** |
| X: double |
| Y: double |
| Z: double |
| w: double |
| MatrixMultiplication(TMatrix m): void |
| Normalize(): void |

Figure 4.2 TPoint Class

### TMatrix Class

This class is a data structure that contains a 4x4 transformation matrix, it has 2-Dimensional 4x4 matrix and a method to Multiply it by another TMatrix. The preview of this class is shown in Figure 4.3.

|  |
| --- |
| **TMatrix** |
| Matrix: double[4,4] |
| Multiply(TMatrix m): void |

Figure 4.3 TMatrix Class

### TVector Class

This class is a data structure that is used to Vector operation in the application, it has 3 attributes to represent direction in x, y, and z direction. It also has a method to normalize the length of the vector to 1, a method to dot product, and a method to cross product. The preview of this class is shown in Figure 4.4.

|  |
| --- |
| **TVector** |
| dX: double |
| dY: double |
| dZ: double |
| Normalize(): void |
| DotProduct(TVector v): double |
| CrossProduct(TVector v): TVector |

Figure 4.4 TVector Class

### TSurface Class

This class is a data structure to represent a surface. It has a list of vertex indices and the colour of the surface. The preview of this class is shown in Figure 4.5.

|  |
| --- |
| **TSurface** |
| Points: List<int> |
| Colour: Color |

Figure 4.5 TSurface Class

### TObject Class

This class is used for representing a 3D Object in a form of a list of vertices, a list surface, a world transformation matrix to transform the object relative the 3D world, and a reference point for the reference for transformation relative to the object. The preview of the class is shown in Figure 4.6.

|  |
| --- |
| **TObject** |
| Vertices: List<TPoint> |
| Surfaces: List<TPoint> |
| WTrs: TMatrix |
| ReferencePoint: TPoint |

Figure 4.6 TObject Class

### MainForm Class

This is the main class that holds and controls all the elements in the form. The class preview is shown in Figure 4.17 and it will only show the methods and attributes related to 3D viewing.

|  |
| --- |
| **MainForm** |
| Objects: List<TObject> |
| Draw(): void |

Figure 4.7 MainForm Class

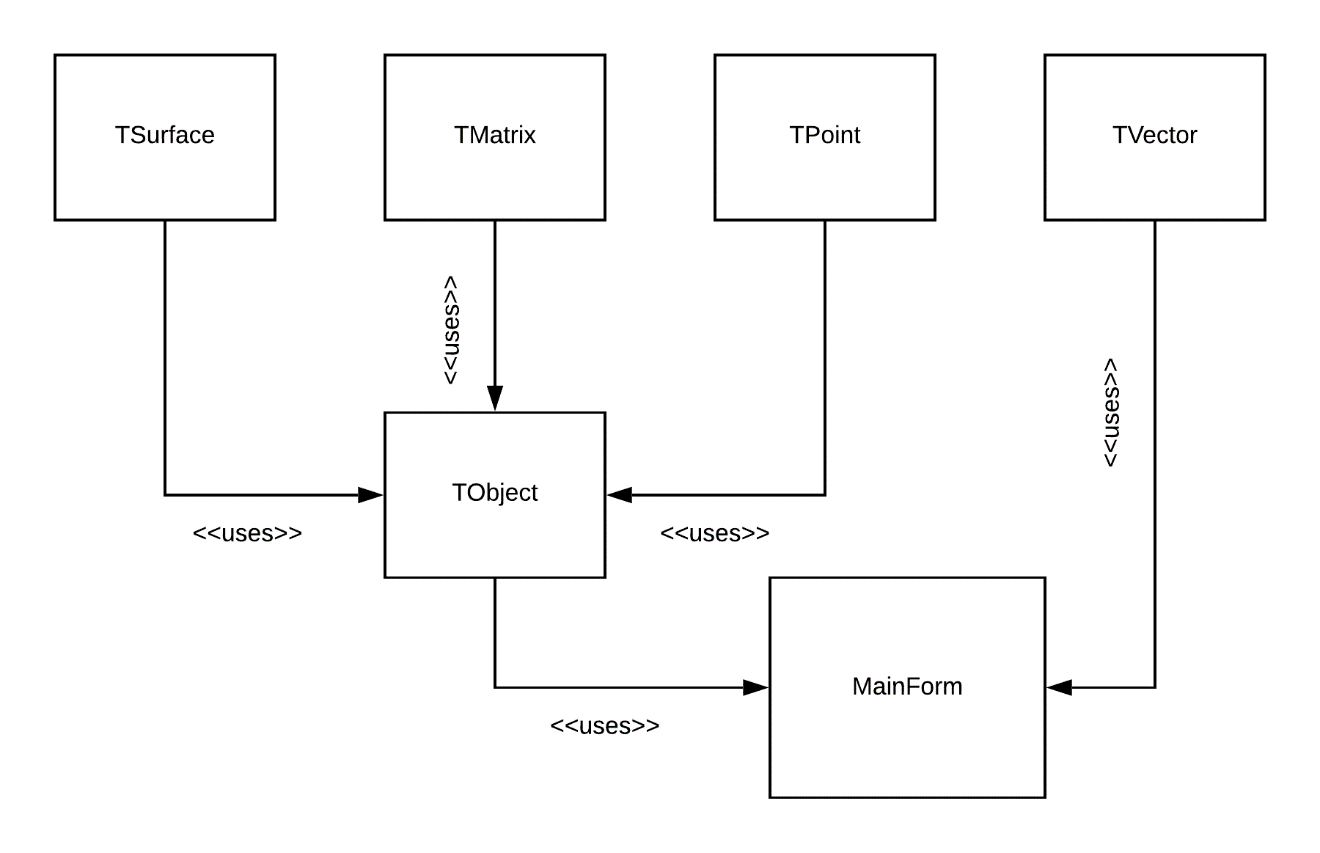


Figure 4.8 Class Diagram of Redi Cube Simulator and Solver

# REFERENCES

1. Google Doodle. (2014, May). *Rubik's Cube*. Retrieved October 23, 2019, from https://www.google.com/doodles/rubiks-cube
2. Rubik’s Cube Solver. (2019, September). *Rubik's Cube*. Retrieved September 11, 2019, from https://rubiks-cu.be/solver/