

# FINAL PROJECT REPORT

## RUBIK'S CUBE FLAT!



*IIT Bombay*

*CS 101: Computer Programming and Utilisation*

*Prof. Deepak B. Phatak*

**Created by:**

*Aamod Kore*

*Harsha Vardhan Kode*

*Charan Teja K*

*Akash Khaire*

*Aditya Krishnamurthy*

*Lokendra Kumar Meena*

*Kamlesh Kumar*

**With the assistance of:**

*Jayanta Borah*

*To  
Prof. D .B. Phatak and  
all our classmates for CS 101,  
the best course of the semester.*

# CONTENTS

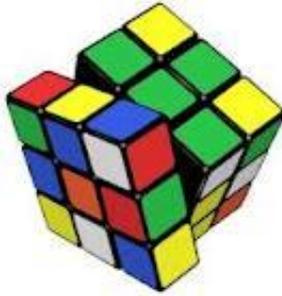
---

Introduction	Page 3
Software Requirement Specification (SRS)	Page 4
Basic Outline of Various Modules	Page 5
So far we have... (Status of Completion)	Page 8
Given more time... (Future Work)	Page 9
References	Page 10
Acknowledgements	Page 11

## INTRODUCTION

---

The Ernő Rubik's Magic Puzzle Cube, most popularly known as the 'Rubik's Cube' after its inventor (the Hungarian sculptor and professor of architecture Ernő Rubik), is one of society's modern-day



mysteries and has frustrated millions of people in the world since its official release in 1980.

The classic 3x3 Rubik's Cube, invented in 1974, is a 3-D mechanical puzzle, in which each of the six faces is covered by nine stickers, among six solid colours (traditionally white, red, blue, orange, green, and yellow). A pivot mechanism enables each face to turn independently, thus mixing up the colours. For the puzzle to be solved, each face must be a solid colour. It won the German Game of the Year Special Award in 1980 for the best puzzle that year. As of today, over 400 million cubes have been sold worldwide.

There are  $8! \times 3^7 \times \frac{12!}{2} \times 2^{11} = 43,252,003,274,489,856,000$  combinations possible for the Rubik's cube (i.e. about 43 quintillion or  $4.3 \times 10^{19}$  combinations). Since its release different algorithms have been developed for solving the Rubik's cube.

# SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)

---

## *Stage-1 Report Modified*

### *ABOUT THIS PROJECT*

---

In our program we open the six faces of the Rubik's cube into a two dimensional surface grid. We also introduced the 3D view using isometric view. Also due to limitation of colours in EzWindows, we changed colour set to red, blue, green, yellow, cyan and magenta. We provide various buttons for the various moves and other functionalities.

Our project is like a game, a virtual Rubik's Cube, in which the player has to solve the Rubik's cube in the least time, with least number of moves. We also have high scores, save and load features.

### *TEAMS AND DISTRIBUTION OF WORK*

---

**TEACHING ASSISTANT:** Jayanta Borah

**TEAM A:** (Internal Coding and Programming)

1. Aamod Kore (*Co-ordinator*) (Roll -110050004)
2. Akash Khaire (Roll -11D260004)
3. Lokendra Kumar (Roll -110040044)
4. Kamlesh Kumar (Roll -115090017)

Team A had to do the coding of the various functions required to modify the cube.

**TEAM B:** (Graphic display and user interface)

1. Harsha Vardhan Kode (*Co-ordinator*) (Roll -110050067)
2. Charan Teja Kattumenu (Roll -110050057)
3. Aditya Krishnamurthy (Roll -110010057)

Team B had to deal with the graphics for developing the user-interface where the player would actually play the game.

# BASIC OUTLINE OF THE VARIOUS MODULES

---

## *1) Graphics and User-Interface:*

---

It contains the program for taking the input from the user; displaying the bitmaps; displaying the 2D and 3D views of the cube depending on the values in the cube array. Basically, this part displays the cube and takes input i.e. creates a graphical interface.

### **LOADING BITMAPS:**

We downloaded bitmaps from google; converted to xpm.  
The loading of the bitmaps is written in the main program (rubix6.0.cpp)

### **TAKING INPUT:**

The input is taken using the mouseclick function.  
It uses the `IsInside` function of the `Bitmap` class and assigns the corresponding values to the input.  
The detailed code is given in `mouseclick.cpp`

### **2D VIEW:**

The 2D view is displayed using `RenderRectangle` function of the 'SimpleWindow' class. Total of  $6 \times 3 \times 3 = 54$  rectangles (squares) are to be drawn. They are drawn using a for loop. First there are six faces, and the top left corner of each face is fixed according to our drawing (for this, an object of `Position` class has been used – 'face\_tl'). Depending on the 'i' value in the array `cube[i][j][k]`, the `face_tl` is fixed.

Now, each face has 9 rectangles (squares). Each square has a topleft position, which can be obtained by adding the value of `face_tl` to `Position(k,j)`.

Now, we have to draw the squares with required colour, which is stored as a number in the array `cube[6][3][3]`, in which each element represents a square. For this, a switch statement is used having `RenderRectangle` commands with different colours for each case and the case is decided using the value of `cube[i][j][k]`.

### **3D VIEW:**

This is somewhat similar to the 2D view program, but here instead of directly using the positions, we defined unit vectors (using `Position` class) which represent x, y and z directions (in the isometric view).

Now, in an isometric view, only 3 faces can be seen at a time, so we fixed the faces, using some int variables – `xy`, `yz` and `zx`; which also represent the planes on which these faces lie. The value of these variables is the face number (0-5).

Depending on which face is on which plane, the unit vectors are chosen using the switch command.

Now, each square on a face is seen as a parallelogram. For this the `RenderPolygon` function is used. We have to pass an array

For the `RenderPolygon`, we need to have an array which contains the coordinates of the vertices of the polygon (`Position` array). I defined an array of `Position` class called 'temp' for this.

Now, how the values in temp are obtained is pure maths, having the unit vectors and row and column (j and k) values, we can derive them.

Here, the j and k values might not directly correspond, since, while folding the cube into 3D, the row and column values actually assigned might get upside down. To correct this, a variable called 'convert' is used and the 3D view is drawn accordingly.

### RATIO AND POSITIONING:

To make the graphics more flexible the scale and position variables are introduced, with which we can change the coordinates of the graphics and also their sizes.

## 2) Internal coding:

---

It contains the program for the moves, save and load, timer, get solution, reset, shuffle and high scores. Basically, this part modifies the values in the cube array and this is the core program.

The colour of the various squares of the Rubik's cube is represented as a 6x3x3 array, cube[6][3][3]. The time taken from the start of the game is stored in the variable int 'time\_elapsed' and no. of moves done in the variable 'int movecount'.

### MOVES:

There are 18 moves in all, in which particular values of the cube array are to be swapped with the other values. For example, turning a face clockwise would mean rotating all the values on that particular face in clockwise.

For all these moves, the code is written in the file – 'moves.cpp'.

### SHUFFLE:

For shuffle, randomly 20 moves are executed. The time(0) function is used to seed the random function(rand()) so that each time a new combination is obtained. Each time a 'SHUFFLE' is carried out a new game begins. (i.e. time\_elapsed and movecount variables are reset to zero)

### RESET:

The 'RESET' function changes all values of the cube to the initial values (initial values are the face numbers.) and the time\_elapsed and movecount are set to zero again.

### TIMER:

The timer is put in the game using the SetTimerCallback function in EzWindows. It calls the timer function for every 1000 milliseconds (= 1 second). In the function, the value of a variable called 'time\_elapsed' is increased by 1. The time is then converted to a string and the string is displayed using RenderText function.

### HIGH SCORES:

The game stores the top 5 highscores (based on the time taken to solve). They are stored in a binary file ("Serocs.bin").

Everytime a player completes a game, the highscores are extracted from the binary file and the time taken by the player is checked for highscores. If the player qualifies for high scores, he is asked for his

name through the Virtual Keyboard, and stores his high score and saves it in the same binary file ("Serocs.bin"). It displays all the top 5 scores in a window after that.

**SAVE & LOAD:**

Saves the game (all variables defining game position & status) in a binary(.bin) file ('EmagDevas.bin') and the 'LOAD' function later loads them back in the same order.

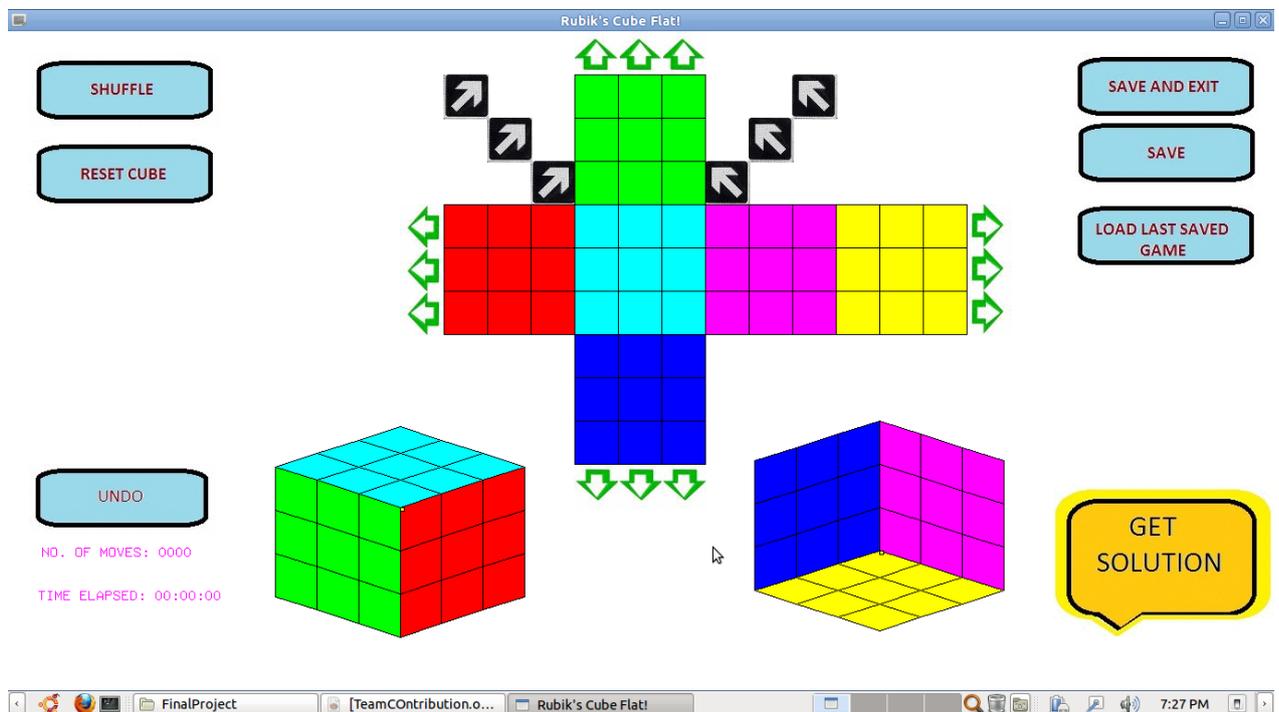
**GET SOLUTION:**

(Not completed yet) Uses a very general algorithm for solving. It outputs the solution in the terminal in term of the moves. Also a penalty of 100 seconds is given to the player for using this facility. After obtaining a solution, it is improved using the improvise() function. This function removes unnecessary moves , for example, if a move is done 4 times subsequently, its as good as no change, and thus those 4 moves are unnecessary!

*On-screen Keyboard*

---

To take input from user, in case he scores a high score, we used an on-screen keyboard. This is not created by any of us. We took it from our friend S S Kausik.



## SO FAR WE HAVE...

---

### *Status of completion*

---

All functions are working correctly except GET SOLUTION and UNDO.

Graphics in 2D and 3D give correct display.

The SOLVER is not completed, it gives the solution for the first layer only (i.e. of about little less than half the first part).

The UNDO function included is not appropriate and not of much help to the user since it undoes only the last move.

### *INDIVIDUAL CONTRIBUTION*

#### *Team A :( Internal Coding and Programming)*

---

##### AAMOD KORE:

- Designing of the moves.
- Programming functions for 'SAVE', 'LOAD'.
- Introduced the variable 'flagwin' in order to denote whether game is in progress.
- Program for highscores saving and display.
- Algorithm and writing the code for the solver ('GET SOLUTION'), which is still in progress (not completely written...).

##### AKASH KHAIRE:

- Writing and debugging the code for the moves.
- Designing function for 'UNDO' using stack library. (NOT included in final program, since it showed error while running.)
- Coding the timer function and the function to display the no. of moves.

##### LOKENDRA KUMAR MEENA:

- Writing function for the moves.
- Code for the 'SHUFFLE'(starts a new game) and 'RESET'(initialising) functions.

##### KAMLESH KUMAR:

- Participated in discussions in meetings, didn't do any programming.

#### *Team B :( Graphic display and user interface)*

---

##### HARSHA VARDHAN KODE:

- Designing of the algorithm for graphics for 2D view of the cube.
- Introduced scale and alignment for graphics output, so that they are flexible.
- Designing, programming and debugging the 3D view output of the cube.
- Downloaded bitmaps and editing and arranging them in display window.
- Integrating coding and graphics sections, compiling and debugging errors in final program.

##### CHARAN TEJA KATTUMENU:

- Designing and programming the 2D view output for the cube.
- Editing bitmaps and creating buttons for display.

##### ADITYA KRISHNAMURTHY:

- Designing and arranging the objects display window.
- Coding for the mouse-click function.

## GIVEN SOME MORE TIME...

---

### *Future Work*

---

- Complete the Rubik's Solver.
- Design better functions for 'UNDO' and 'REDO'.
- Develop better user interface in graphics, by incorporating functions which allow the user to change the view of the cube in 3D and also to directly play from 3D view.
- Include confirmation windows to confirm actions such as 'SHUFFLE', 'RESET', 'LOAD', 'SAVE', 'GET SOLUTION' and 'SAVE AND EXIT'.

## REFERENCES

---

- [http://en.wikipedia.org/wiki/Rubik's\\_Cube](http://en.wikipedia.org/wiki/Rubik's_Cube)
- <http://www.rubiks.com>
- (for algorithm to solve the cube) YouTube - How to solve a Rubik's Cube (Part One)  
<http://www.youtube.com/watch?v=HsQloPyfQzM>

## ACKNOWLEDGEMENTS

---

Our Junior Teaching Assistant, Jayanta Madhab Borah, for his guidance and support.

S S Kausik, for the design and code of the On-Screen Keyboard Input.