Graphical Simulation of the Rubik's Cube Puzzle

Rayfus J. Gary, Graduate Student, Florida Institute of Technology

***Abstract*— This paper describes the capstone project for the Projects in Computer Information Systems Course (CIS 5080). This project requires the student to design and implement an application that will serve as a graphical simulation for the permutation puzzle- the Rubik's cube. The student's knowledge in systems design, data structures and algorithm design, object-oriented programming, and computer graphics are some of the concepts that were tested during the implementation of this project.**

I. Introduction

The "Magic Cube" was designed in Budapest, Hungary by Erno Rubik, a lecturer in the Department of Interior Design at the Academy of Applied Arts and Crafts in Budapest. The first known prototype was revealed in1974 with its patent application. The Cube made its international debut at the Toy Fairs of London, Paris, Nuremberg and New York in January/February, 1980. Rubik's made an immediate impact with his demonstration of the Cube. After negotiations with Ideal Toys the cube was renamed the "Rubik's Cube" and the first cubes were exported from Hungary in 1980. David Singmaster, an English mathematician, was the first to look at the cube from a mathematical prospective by addressing the theoretical problems and ramifications raised by the Cube in his newspaper article in June 1979. This article brought the Cube to the attention of academic circles world wide. This led to an article in Scientific American by Douglas Hotstadter an acknowledged authority in the field of Recreational Mathematics [1].

The purpose of this project is to design and implement a graphical model to represent the 3X3 Rubik's Cube. This model must be able to represent the cube's 43,252,003,274,489,856,000 possible configurations [2]. It will be used to implement an algorithm to solve the Rubik's Cube. This project is implemented using Microsoft Visual Studio C# 2005 Express edition. GDI+, the application programming interface (API) that provides the classes for creating two-dimensional vector graphics, is used to create and manipulate graphics in the final application.

Manuscript received April 18, 2006. This work was supported by the Florida Institute of Technology, Orlando Graduate Center. Major Rayfus J. Gary is with the United States Army School detachment, Orlando, Florida 32835 (e-mail: rayfus.gary@us.army.mil)

II. Detailed Design

A. Use Cases

The 3X3 Rubik's Cube is a permutation puzzle in that it satisfies the following behaviors:

-several finite moves at each stage
-finite sequence of moves that yields a solution
-no chance or random moves
-complete information about each move
-each move depends only on the present position, not the existence or non-existence of a certain previous move (such as chess, where castling is made illegal if the king has been moved previously [2].

The first step in developing a graphical model of the Rubik's Cube is to become familiar with the cube in terms of how the cube works and its behavior. The key to understanding the behavior of the cube is to view the cube as a puzzle where your goal is to manipulate the cube with the purpose of finding a solution. This helps to gain an understanding of how the pieces of the cube work together to form a valid configuration. The cube is a three dimensional object, but in designing programming code to represent this object your are limited to a two dimensional data structure to represent the cube's state. The solution involves applying the object-oriented approach to develop a class that encapsulates the methods and data structures to represent the cube.

The next step involves creating a user interface to graphically display the cube's state and to manipulate the cube's behaviors.

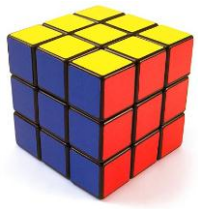

Fig. 1 The 3X3 Rubik's Cube.

By applying Group Theory, an approach used by mathematicians to describe the internal symmetry of objects, one can grasp an understanding of the cube's structure, its moves (behavior) and its permutations (state)[4]. The 3X3 cube had six colored sides or faces, each of which has nine facets for a total of 54 facets. The facets are grouped into 26 smaller cubes. The first group is the twelve side facets, the

next group is the 18 edge facets, and the final group is the six center facets.

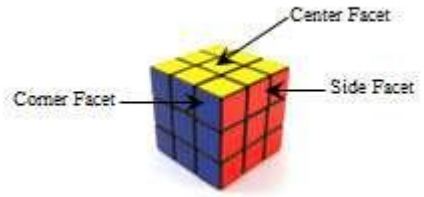

Fig. 2. This figure shows the Cube's three type of facets.

The cubes behavior is displayed in its ability to rotate each face in both directions. The cube can perform six valid moves which essentially correspond to each face turning one quarter clockwise (a quarter counterclockwise turn is 3 quarter clockwise turns). During a move, one important behavior that one must note is that the center facets remain intact. Because the center facets don't move, they become reference points for the Cube's face. Therefore, a data structure designed to represent the cube's state only has to account for 20 facets. A move in essence, repositions eight facets (four edge and four corner).

The data structure designed to represent the cubes state must capture the cubes behavior as a two-dimensional object in the terms of its faces and facets. Each facet has a unique position on the cube and a color. The face is referenced by its center facet: U for the top face, F for the front face, Left for the left face, R for the right face B for the back face, and D for the down face. In the application the letters are used to reference a color (see Fig 3).

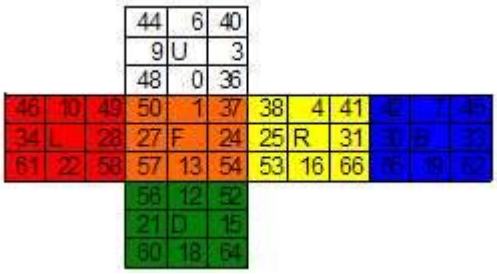

Fig. 3. This shows a two dimensional representation of the cube. Each facet is numbered to represent its unique position on the cube.

A move relocates eight facets on the cube. In order to replicate this, each facet is given a unique letter identifier and is grouped by its type (edge and corner).

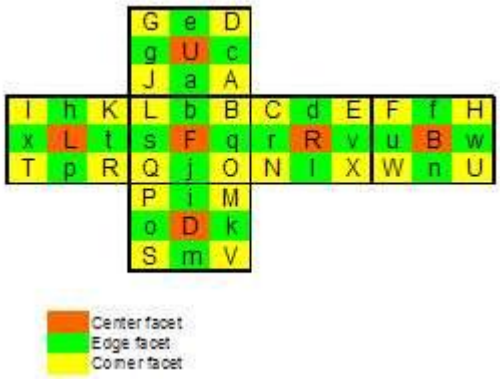

Fig. 4. This represents a solved cube. The facets are grouped into its two types: Edge, and Corner.

In the program a string is used to represent the cube's state (see Fig 4). Each character corresponds to the facet's color, and the facets ordinal position (spaces accounts for a position) on the string represents its unique position on the cube.

```
const string solution =
    "UF UR UB UL DF DR DB DL FR FL BR BL UFR URB UBL ULF DRF DFL DLB DBR";
//   ab cd ef gh ij kl mn op qr st uv wx ABC DEF GHI JKL MNO PQR STU VWX
//   00000000001111111111222222222233333333334444444444555555555566666666
//   01234567890123456789012345678901234567890123456789012345678901234567
```

Fig. 4. This shows the string used representing the cube's state.

Each facet is given a unique identifier, to keep it separate from the facets of the same color. This is used to keep track of the cube's moves. In order to execute a move, the unique facets are grouped by type and the face that has the potential to move the facet. This grouping is captured in a multi-dimensional array of characters.

```
    static string[] permutations =
{
    "cd ef gh ab                        DEF GHI JKL ABC                ", // U
    "           op ij kl mn                          PQR STU VWX MNO", // D
    "ts         rq          ba ji       KLJ         RPQ CAB NOM       ", // F
    "      vu          xw          nm fe     XVW EFD             IGH TUS", // B
    "         wx          st    gh    op          UST HIG     LJK QRP    ", // L
    "   qr          uv        kl    cd    OMN BCA          WXV         FDE", // R
};
```

Fig. 5. This is the multi-dimensional array of characters which groups the characters representing the unique facets by faces.

Each row in the array (Fig. 5) represents the facets on the array which makes up a face. Each column represents a unique position on the face with respect to the cube. This array is used to populate a multi-dimensional array (`moves`) which serves as a template for all of the valid move combinations. This array is then copied back into the string which represents the cube's state.

III. Class Hierarchy

This application implements an object oriented approach to the solution to this project by using two Classes- the `Cube` class and the `CubeDrawer` Class. The `Cube` class encapsulates the data structures and the methods used to duplicate the cube's behavior. The `CubeDrawer` Class encapsulates all the data structures and the methods that are used to manipulate the data structure representation of the

cube into a form to be drawn on the screen. The main form is the graphical interface to the user to trigger events to manipulate the cube.

IV. Short Description of Classes and Membership Methods

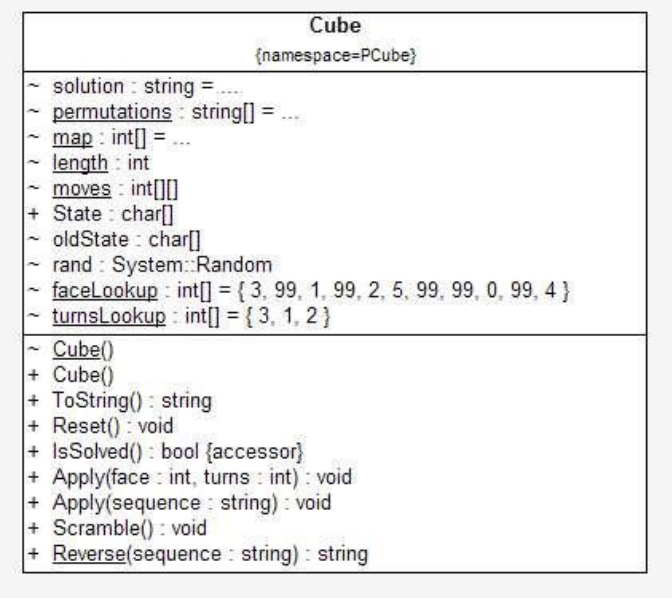

Fig. 5. The Unified Modeling Language (UML) representation of `Cube` Class.

A. *Class `Cube`*

This class defines the data structures used to represent the cube and its methods used to duplicate the cubes various moves.

a) Attributes

The variables in the `Cube` class that deserve mention are `solution`, `permutations`, and `state`. The string constant `solution` represents the cube in a solved state. It is used in the `IsSolved` method as a reference point to ascertain if the cubes current state is in a solved station. The array of strings, `permutations`, represents the cube by breaking down the unique characters of each unique facet into a position on the array which represents its position on the cube. Additionally, this array groups the characters by facet type (side or edge facet). This array is used to initialize the multi-dimensional array, `moves`, which is used in the `Apply` method to apply a valid move sequence to the selected face. Finally, the character array, `state`, represents the state of the current cube. This array is used in the `Cube` class constructor.

b) Method `ToString`

This method is used to convert the array of characters representing the cube to its string representation.

c) Method `Reset`

This method converts the current cube to its solved state,

d) Method `IsSolved`

This method checks the array of characters representing the cube's current state and compares it the array representing the cube in a solved state and returns a Boolean value. If both arrays are equal then a Boolean value of TRUE is returned.

e) Method `Apply`

The Apply method takes two integers, one representing a face and the other representing the number of turns. These two integers are applied to the class' static variable to represent a move.

f) Method `Scramble`

The Scramble method is used to scramble the cube. This method randomly generates two integers, one for face and the other for the number of moves. This method then calls the Apply method and passes in the two integers which execute a move onto the static string representing the cube state. This process is looped 50 times.

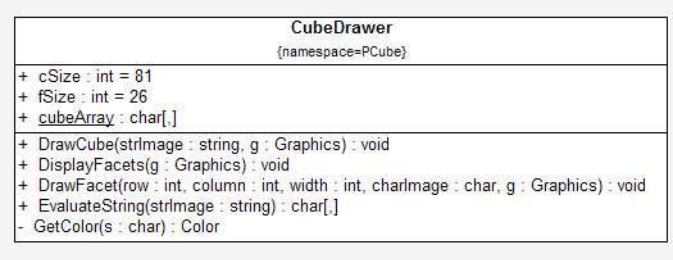

Fig. 6. The UML representation of the `CubeDrawer` Class

B. *Class `CubeDrawer`*

a) Attributes

The variables in the `CubeDrawer` class are used to convert the string representing the cube into the cube's graphical representation. GDI+ uses pixels as a unit of measure for its graphic elements [2]. The `cSize` variable is an integer which represents the size of the cube's face (the cube face is `cSize` X `cSize` in pixels). The `fSize` integer represents the size of each facet. The multidimensional array of characters, `cubeArray` is used to hold the character representing the color of each facet and is broken down by face. This array is used in the `DisplayFacet` method to draw the entire cube to the screen.

b) Method `DrawCube`

This method receives as parameters the string representing the cube and the `Graphics` object derived from the argument `PaintEventArgs` from the calling `OnPaint` method, and initiates the processed to display the cube to the screen. It calls the `EvaluateString` method which analyzes the string and returns the array which represents the cube in a format that can be displayed to the screen, and the `DisplayFacets` method which starts the process to display the cube.

c) Method `DisplayFacets`

This method prepares the screen to display the graphical representation of the cube to the screen. It sets the starting point for each cube, and then analyzes the `CubeArray` variable to determine each facet's color and its position on each face. The method receives the `Graphics` object and passes it to the `DrawFacet` method which draws the cube representing the facet on the screen.

d) Method `DrawFacets`

Method `DrawFacets` receives as parameters the variables (`row and column`) representing the pixel coordinate of the facet's top left corner, the variable which represents the facet's size, the character representing the color of the facet, and the `Graphics` object. The method calls the `GetColor` method and passes the variable representing the facet to get the facets color. It uses these variables to draw the facet on the screen.

e) Method `EvaluateString`

Method `EvaluateString` receives as a parameter (`strImage`) the string representing the cube's state, analyzes it and returns an array (`CubeArray`) that serves as the template for displaying the cube on the screen. The local variable `positionArray,a` 6 X 9 multi-dimensional array, is initialized with the integers representing the ordinal position of each character in the string and groups the integers by faces and its position on the cube. Each row in the array represents a face and each column represents a position on the face. The characters representing the center facets are first loaded into the array. Next a series of selection statements are used to each character in `strImage.` Starting with the first character in the string, the statement evaluates the first character. If the character is a ' ' then the counter is incremented to evaluate the next character. If a character is not a ' ' then the statement searches the `positionArray` for the integer symbolizing the characters position. When the position is found, the charter is copied to the corresponding position on the `CubeArray`.

f) Method `GetColor`

This method receives the variable representing the facet color and returns its corresponding `Color` constant.

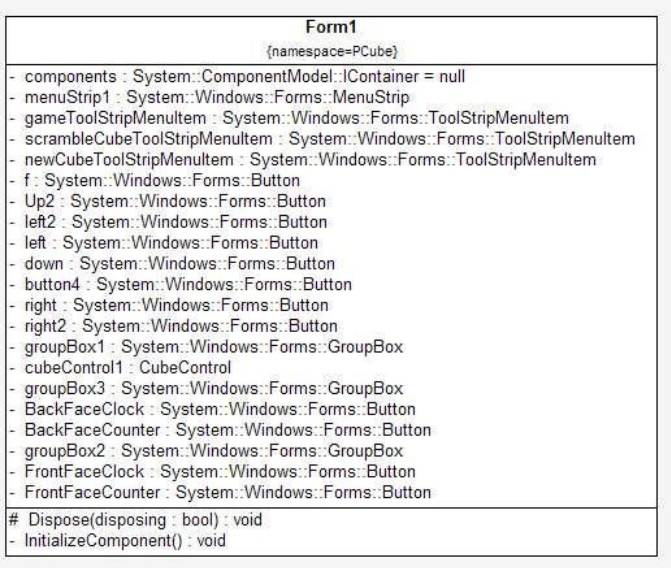

Fig. 7. The UML representation of `Form1`

C. Form1

Fig 7. Illustrates the UML representation of the class representing the GUI for the application. It shows each control and the event handler that it triggers.

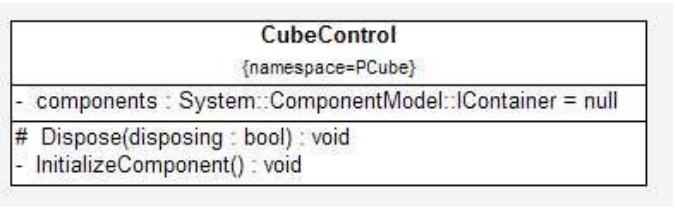

Fig 8. The UML representation of the User Control.

D. User Control `CubeControl`

The User Control Cube Control serves as a drawing surface for the cube. It contains the `OnPaint` method which is used to draw objects on the GUI.

V. FINAL APPLICATION

Microsoft Visual Studio C# 2005 Express Edition is the IDE used to implement this project. The project, PCube, is designed using the Visual Studio's Windows Application Template. This template enhanced the ability to create and manipulate Window's Graphical Users Interfaces (GUIs). Additionally, GDI+, the API that provided the classes for creating graphics, simplified the process of creating and manipulating graphics.

There is one main form used in this application. This form contains all of the controls used to manipulate the cube. A user control is used as a drawing area to hold the graphic representing the cube. The cube is displayed as a two-dimensional object.

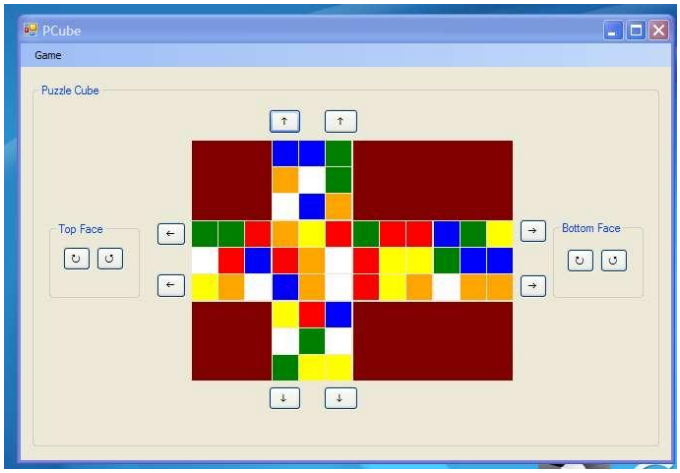

Fig. 9. The PCube application with the cube in a scrambled state.

A. *User Interface*

1) *Menu*

The menustrip (GAME) gives the user two options: (Scramble) which scrambles the cube and (New Cube) which resets the cube to a solved state,

2) *Display Area*

This area displays the cube in a two dimensional format. Using the center facet as a point of reference and holding the cube with the orange side facing the user, orange symbolizes the front face, red symbolizes the left face, white symbolizes the top face, yellow symbolizes the right face, blue symbolizes the back face, and green symbolizes the down face.

3) *User Controls*

The cube is manipulated when a user clicks on the button corresponding to move the affected face. The button click triggers an event handler which applies the move to the data structure, redraws the new cube on the user control, and checks to see if the new configuration solves the puzzle. If it solves the puzzle, a message box is displayed indicating the cube is solved. The buttons immediately surrounding the cube move the face in the direction of the arrow on the button. The buttons used to move the bottom and top faces are contained in group boxes grouped by face.

4) *Messages*

The only message that the program returns is a message box which is display when the cube is solved (See. Fig 8).

B. *Playing the Game*

The purpose of this game is to match the colors of the six sides corresponding to a face on the Cube. At startup, the cube is displayed in a solved state. The user has the option of scrambling the cube by randomly selecting buttons or by selecting scramble from the menu. The user then attempts to solve the cube by applying moves to the cube to match the colors on all six sides. When the cube is solved, the Message box with the message "Congratulations!!! You have solved this puzzle" is displayed (See Fig. 7). The user may select the New Cube option from the menu bar which resets the cube to a solved state.

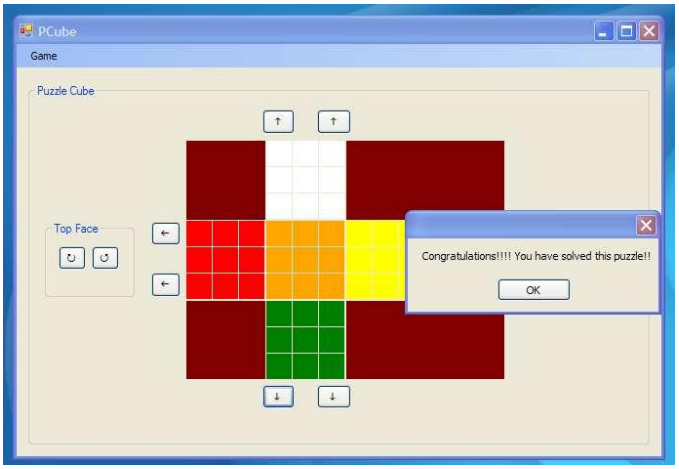

Fig.10. PCube application with a message indicating the user has solved the puzzle.

VI. Conclusion

The final application `PCube` implements the solution to the problem of designing a graphical model to duplicate the behaviors of the Rubik's Cube. With the understanding of how the Cube's state is represented in the project's code, this project can be scaled to implement an algorithm to solve the Rubik's cube and to display a more accurate graphical user interface to represent the cube in 3D.

APPENDIX

Project Source Code

```
-Program.cs
-CubeDrawer.cs
-Cube
-Form1.cs
-CubeControl.cs
```

AKNOWLEDGEMENTS

R. Gary thanks family and a special friend for their inspiration and motivation while completing this project. Further thanks goes to S. Weisfield for his assistance with the coding during the project's implementation phase. Final thanks go to Professor Findling for his sponsorship and guidance.

REFERENCES

[1] [Online] available: http://www.rubiks.com/
[2] D. Joyner. *Adventures in Group Theory: Rubik's Cube, Merlin's Machine and other Mathematical Toys*, Baltimore: Johns Hopkins University Press, ch 3.
[3] H. Deitel and P. Deitel. *Visual C# 2005: How to Program* 2nd Ed., Upper Saddle River, NJ: Pearson Education, Inc., 2006, ch17

Major Rayfus J. Gary (Perry, Georgia). He has a BS degree in Computer Information Systems from Florida Agricultural and Mechanical University (1994) and a MS in Administration with a concentration in Information Resource Management from Central Michigan University (2003).

He is currently assigned to the United States Army Student Detachment where he is pursuing a MS degree in Computer Information System from the Florida Institute of Technology. He has ten years experience as a communications officer and has held various positions at both the tactical and strategic levels.