DEVELOPMENT OF A RUBIK'S CUBE SOLVING APPLICATION FOR ANDROID DEVICES

by

Morgan R. Mirtes

A capstone project submitted in partial fulfillment
of graduating from the Academic Honors Program
at Ashland University
April 2014

Faculty Mentor:	Dr. Paul Cao, Associate Professor of Computer Science
Additional Reader:	Dr. Boris Kerkez, Associate Professor of Computer Science

Acknowledgements

The author would like to thank Dr. Paul Cao for giving constant encouragement and motivation throughout this project, Dr. Boris Kerkez for being a helpful additional mentor, Dr. Gordon Swain for sharing his interest in Rubik's Cubes, and their many friends and family members who assisted in testing the application on a wide variety of Android devices.

Abstract

The Rubik's Cube is a common 3-D combination puzzle known and loved by people of all ages. The mathematics of Rubik's Cubes has been explored in great detail ever since the beginning of their production in the 1980's. The first part of this project consisted of research on the mathematics of the Rubik's Cube and how that is used for various solutions to the cube. Next, focus was put on making an Android application to solve Rubik's Cubes given a random cube or given user input. For this, research was conducted on the use of Java programming language, Android Development libraries, and the Android Development Tool for Eclipse. Overall, this thesis discusses the knowledge gained on Rubik's Cube theory, Rubik's Cube Algorithms, modifying algorithms to fit specific needs, Java programming language, Android development, Open Graphics Library (OpenGL) Application Programming Interface (API), Eclipse Android Development Tools, and testing and debugging with Eclipse as well as programming methodologies for the application based on problems encountered and their relative solutions during development.

Contents

1 Introduction

The topic of this thesis is the mathematics behind Rubik's Cubes and how to create a smartphone application that solves them. This topic is appealing because it incorporates both mathematics and computer science at a practical level. The mathematics of Rubik's Cubes has been explored in great detail ever since the beginning of their production in the 1980's, thus showing its importance in the mathematical world. In addition, software development is important in the computer science field, and recently smartphone applications have developed into a prominent area in computer science due to their high popularity and versatile capabilities. Thus, Rubik's Cube solving Android application development was chosen for this Honors Capstone Project because it is challenging and offers valuable experience that can be used in today's world of mathematics and software development.

This thesis begins with the *Algorithms* section, which discusses the mathematics behind Rubik's Cubes solutions and then moves to discuss the Rubik's Cube solution algorithm that was chosen for this application and why it was chosen.

The *Methodologies* section of the document gives details about the development of the smartphone application. It discusses the algorithm modifications and the creation of the both the two-dimensional and three-dimensional products. Throughout this section, many of the problems encountered while working on these sets of code and the solutions to them are also discussed.

The *Development Details* section walks through the decisions made about what methods should be used to develop the application, including mobile operating system(s), languages(s), libraries(s), development environment(s), testing, and hardware.

The *Conclusion and Future Work* section wraps up the thesis by giving a final outlook on the project. This also includes tasks that could be further completed on the project at the time of the presentation.

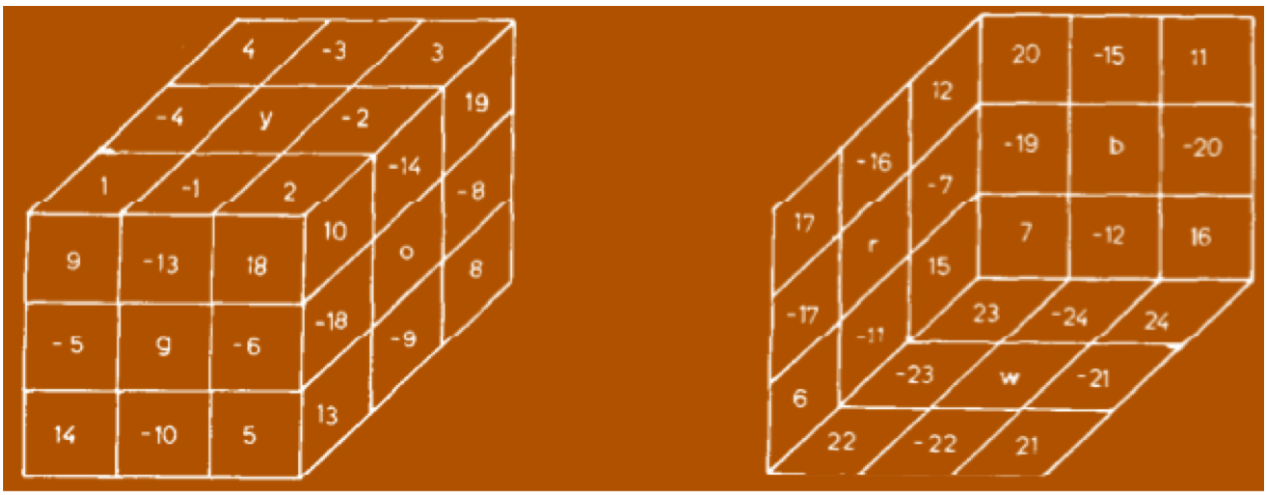

Figure 1: Zassenhaus's numbering of the Rubik's Cube

2 Algorithms

2.1 Introduction to Rubik's Cubes

Rubik's Cubes are combination puzzles that exemplify mathematical group theory. Zassenhaus uses a helpful illustration of a flattened Rubik's Cube with numbered squares in order to better explain the effect of the layer moves [**?**]. In this example, shown in Figure **??**, the center squares are marked with the letter of their face color, corner squares are numbered 1 through 24, and edge squares are numbered -1 through -24. It is also noted that the numbers are laid out in such a way that opposite corner squares add up to 25, opposite edge squares add up to -25, corner squares adjacent to the same point differ by 8 or 16, and edge squares adjacent to the same edge differ by 12.

The six basic turns of the Rubik's Cube are each a rotation of one of the six cube faces. Each of these, as shown on the next page, correspond to different permutations. For example, the permutation for T1 = (1,2,3,4)(9, 10,11,12)(17,18,19,20)(-1,-2,-3,-4)(-13,-14,-15,-16) means that the square labeled as 1 is carried to the square labeled as 2, the square labeled as 2 is carried to the square labeled as 3, the square labeled as 3 is carried to the square labeled as 4, and the square labeled as 4 is carried to the square labeled as 1, and so forth. There are multiple parts, or cycles, of the permutations for each turn because turning one layer of a Rubik's Cube does not only turn the corner squares, but it also turns the edge squares in between those corners. These corner and

edge squares also consist of the top of the face as well as the connecting sides. Thus, any simple turn of the Rubik's Cube will generate a permutation that consists of five cycles.

$$T1 = (1,2,3,4)(9,10,11,12)(17,18,19,20)(-1,-2,-3,-4)(-13,-14,-15,-16)$$

$$T2 = (10,13,8,19)(18,21,16,3)(2,5,24,11)(-8,-14,-18,-9)(-20,-2,-6,-21)$$

$$T3 = (20,11,16,7)(4,19,24,15)(12,3,8,23)(-15,-20,-12,-19)(-3,-8,-24,-7)$$

$$T4 = (9,14,5,18)(17,22,13,2)(1,6,21,10)(-5,-10,-6,-13)(-17,-22,-18,-1)$$

$$T5 = (12,15,6,17)(20,23,14,1)(4,7,22,9)(-7,-11,-17,-16)(-19,-23,-5,-4)$$

$$T6 = (21,22,23,24)(5,6,7,8)(13,14,15,16)(-21,-22,-23,-24)(-9,-10,-11,-12)$$

Zassenhaus then notes that several of these turns applied in a row create an entirely new permutation of the squares on the cube. All of these moves together form the Rubik's Group [12]. By the definition of a group, it follows that the group should also have order, identity, and inverse. Before the explainations, it is important to note that M^2 and M2 are interchangable for all combinations of M. When you repeat a specific move, M, over and over (for instance doing M, then MM=M^2, then M^2M=M^3, and so on) eventually M will "reappear". This happens when you repeat the move M enough times so that it looks like you have only done the move M once. When this happens, we say that $M^{f+1} \approx$ M, where M, M^2, ... , M^f are distinct. Then, f is the order of M, denoted by $f = |M|$. In addition, M^f is the identity permutation: the move which brings every square back to where it started. We then note that:

$$\text{since } f = |M|, M^f = M^{|M|} = 1.$$

It also follows that M^{f-1} is the inverse of M, and

$$M^{|M|-1} = M^{-1} \equiv M^-, \text{thus } MM^- = M^-M = 1.$$

In addition to the Rubik's Group as a whole, the Rubik's Group also has subgroups [12]. In any system of moves defined by S = $\{M^{(1)}, \ldots , M^{(s)}\}$, then <S>is the subgroup of the Rubik's Group generated by S. <S>is closed under composition because if M and M' belong to <S>, then

so does MM'. <S>is also closed under inversion (where each element has an inverse) because if M belongs to <S>, then so does M'. Finally, the number of elements in <S>is called the order of S, and is denoted by |<S>|.

These systems also have equivalence relations and equivalence classes. In any system of moves defined by S = $\{M^{(1)},\ldots,M^{(s)}\}$, there are equivalence relations, such as a $\widetilde{s}$ b, which say that it is possible to carry square a to square b by a finite number of moves that are in S [12]. For Rubik's Cubes, the corresponding equivalence classes are called orbits of S. The most general example of these orbits are for the permutation set {T1,...,T6}, which is the system of all moves that consist of any of the basic turns of the Rubik's Cube that were discussed previously in this section. This particular permutation set has two orbits, one of which is the set {C} that contains the 24 corner squares, the other is the set {E} that contains the 24 edge squares. These orbits mean that by doing any variation of the turns in S, you can take any one corner piece to another corner piece or any one edge piece to another edge piece. In addition, because orbits are equivalence classes, each element is related to itself (where you can "do nothing" to the squares) and there is transitivity, meaning if a $\widetilde{s}$ b and b $\widetilde{s}$ c, then a $\widetilde{s}$ c.

Finally, "the mathematician's way of problem solving" can easily be related to solving Rubik's Cubes. This mathematician's way of problem solving is to first solve a simple problem, then reduce a more complicated problem to that simple problem so that it is easier to solve. This can be related to recursion in computer programming, or in simple math taking 6^5 and reducing it to (((6*6)*6)*6)*6. For a Rubik's Cube, if you know that move M is related to one permutation, the moves X'MX will have the same type of cycle decomposition as M [12]. This is called the conjugate of move M by the move X. A basic example of what this means is if you know that move M takes square 1 to square 2, but you need to move square 5 to square 2, you can do any number of moves that take square 5 to square 1 and consider them as move X, do move M to take square 5 to square 2, and then do X' (the inverse of move X) to get the rest of the cube back to where you started. For most people, this is the easiest way to solve a Rubik's cube without memorizing many complicated algorithms because you only have to memorize a few different moves and you can arrange the cube so that you then use those moves to solve the cube.

2.2 Algorithm for this Project

There are countless Rubik's Cubes Algorithms in existence; some focus on speed and efficiency, others focus on how simple they are for humans to memorize, and others focus on nothing other than solving the cube. Some of the most popular efficient algorithms in existence today are Kociemba's algorithm, Rokicki's algorithm, and "God's algorithm". For this project, the goal was to use an algorithm that would solve a Rubik's Cube efficiently on an Android device. Thus, as suggested by a group of Youngstown State University students doing a hardware-intensive Rubik's Cube application, the Kociemba's algorithm is best fit for Android Development because "it is fast, requires little memory, and typically solves in about 20 moves" [**?**]. Kociemba provides some java code for the two-phase-algorithm on his website, which includes a computer-executable file with an example of how to use the package. He describes the algorithm as such:

> "The 6 different faces of the Cube are called U(p), D(own), R(ight), L(eft), F(ront) and B(ack). While U denotes an Up Face quarter turn of 90 degrees clockwise, U2 denotes a 180 degrees turn and U' denotes a quarter turn of 90 degrees counter-clockwise. A sequence like U D R' D2 of Cube moves is called a maneuver.
>
> If you turn the faces of a solved cube and do not use the moves R, R', L, L', F, F', B and B' you will only generate a subset of all possible cubes. This subset is denoted by G1 = <U,D,R2,L2,F2,B2>. In this subset, the orientations of the corners and edges cannot be changed. That is, the orientation of an edge or corner at a certain location is always the same. And the four edges in the UD-slice (between the U-face and D-face) stay isolated in that slice.
>
> In phase 1, the algorithm looks for maneuvers which will transform a scrambled cube to G1. That is, the orientations of corners and edges have to be constrained and the edges of the UD-slice have to be transferred into that slice. In this abstract space, a move just transforms a triple (x,y,z) into another triple (x',y',z'). All cubes of G1 have the same triple (x0,y0,z0) and this is the goal state of phase 1.
>
> To find this goal state the program uses a search algorithm which is called iterative deepening A* with a lowerbound heuristic function (IDA*). In the case of the Cube, this means that it iterates through all maneuvers of increasing length. The heuristic function h1(x,y,z) estimates for each cube state (x,y,z) the number of moves that are necessary to reach the goal state. It is essential that the function never overestimates this number. In Cube Explorer 2, it gives the exact number of moves which are necessary to reach the goal state in Phase 1. The heuristic allows pruning while generating the maneuvers, which is essential if you do not want to wait a very, very long time before the goal state is reached. The heuristic function h1 is a memory based lookup table and allows pruning up to 12 moves in advance.
>
> In phase 2 the algorithm restores the cube in the subgroup G1, using only moves of this subgroup. It restores the permutation of the 8 corners, the permutation of the 8 edges of the U-face and D-face and the permutation of the 4 UD-slice edges. The heuristic function h2(a,b,c) only estimates the number of moves that are necessary to

reach the goal state, because there are too many different elements in G1.

The algorithm does not stop when a first solution is found but continues to search for shorter solutions by carrying out phase 2 from suboptimal solutions of phase 1. For example, if the first solution has 10 moves in phase 1 followed by 12 moves in phase 2, the second solution could have 11 moves in phase 1 and only 5 moves in phase 2. The length of the phase 1 maneuvers increase and the length of the phase 2 maneuvers decrease. If the phase 2 length reaches zero, the solution is optimal and the algorithm stops.

In the current implementation the Two-Phase-Algorithm does not look for some solutions that are optimal overall, those that must cross into and back out of phase 2. This increases the speed considerably." [**?**]

3 Methodologies

3.1 Algorithm Implementation

In reference to the java-based program provided on his website, Kociemba states, "the tables in this implementation take only about 5MB and are generated within seconds" [**?**]. This is true when using the provided executable file on a computer, but when restricted to cell phone memory and speed, this often took over an hour and even caused the phone to crash during some attempts. When running through the program in debug mode, it was found that the slowest portion of the program occurred because particular slow-loading arrays were being re-loaded over and over again. To solve this, the original program was run once so the arrays could be gathered and dumped into files on the system. Once all of the files were loaded, they were manually copied into the raw resources folder in the application so that they could be read in by the program and populated much faster. In doing this, there were some problems with data types and conversions, so in the end the array elements were written using objectOutputStream's writeByte and writeShort methods. Bytes were read in using objectInputStream's readFully method and shorts were read in using a ByteBuffer to read two bytes in at once and then using big endian order conversion to convert them into shorts. Luckily, pre-existing java methods made this task as simple as knowing that the methods exist and how to use them. Doing this cut the time down to about 15 seconds for the first solve and about 1 second for any subsequent solves since the files are only read on the first solution attempt.

3.2 Two Dimensional Version

The two-dimensional app design was relatively straightforward. Buttons and button click events were utilized to set the colors of the cubes so that the user can either click to specify the color layout of their cube or click a button to set the colors to the layout of a randomly generated cube. Note that the random cube generating method was included in Kociemba's java code. There weren't many arduous problems in working with this approach; it took more time to program all of the repetitive information than anything else. These things included breaking up the solution string into individual moves and changing cube button colors depending on which cube move is called. Button click methods were used to pause the program until the "Next Move" button was pushed. This pausing initially caused problems because there was no simple way for the user to unpause the program. However, this was solved by using a global variable that was incremented each time the button was pushed so the next index of the moves array would be displayed only when that index is increased.

3.3 Three Dimensional Version

The three-dimensional app implements the OpenGL (Open Graphics Library) API (Application Programming Interface) that is used for rendering 2D and 3D vector graphics. The Android Developers website provides tutorials on OpenGL to teach developers the basics of OpenGL and walks through the development of some basic apps. Some of these are apps that display 2D shapes and allow the user to move them around the screen with touchscreen sensors and 3D shapes such as cubes that can rotate automatically. There is an existing Android Kube API that utilizes OpenGL to generate graphics that resemble a Rubik's Cube. This existing API contains the functionality for that cube to start in a solved position and turn random layers jto un-solve itself.

This existing Kube API took a lot of tweaking in order for it to work properly with the Android application. The first step was to change the existing two-dimensional application code to work with the given Kube structure rather than buttons. This proved to be more challenging than expected. It took some time to understand how the Kube structure was laid out and how to manipulate it. After going through the code to get a better understanding, it was established that the Kube is made up of 27 different Cube objects, each of which are made up of six different squares that come

together to form the sides of the Cube. To manipulate the colors of those cubes, the API's Cube class provides a setFaceColor method, which takes parameters for a number (relative to the top, bottom, left, right, front, or back face of the Cube) and a color constant. In order to properly set up the Kube, the middle squares on each face of the Kube must be set to a different color in relation to how they would be laid out on a regular Rubik's Cube.

Currently, the application uses Kociemba's random cube generating method with the Kube API so that when the app starts, it generates a random cube and then finds its solution. To make Kociemba's random cube generating method work with the 3D Kube design, two arrays were created so that the cube number used in Kociemba's algorithm could be associated with the corresponding Cube object number and its proper face to be colored. This ensures that when the cubes are colored in the method, the overall Kube is colored correctly. Initially, the mLayerPermutations array, which says what Cubes are permutated in what order for each turn was hard-coded based on a solved cube. Since the app now started with a random cube rather than a solved cube, it was first thought that the mLayerPermutations array had to be dynamically set depending on how the Kube was laid out from the random cube generator. However, this caused the Cubes to not rotate with the proper layer so they would end up on top of each other or in other wrong positions within the Kube. However, since the program in fact kept the original Cube layout but only changed the colors of the sides, technically the permutations of the cube numbers should have stayed the same. Thus, leaving the default mLayerPermutations array allowed the layers to move as needed to solve the given cube.

The next task was to have specific layers rotate depending on what move was next rather than randomly choosing layers to rotate. This was fixed simply by taking out the random number generator and replacing it with a simple function that takes a string parameter and returns the corresponding layer number. For example, if the current move contained a U, this represents the upper layer, and the function would return a 0 since that is the given number for the upper layer of the Kube. Following that, the program needed to determine which direction the layer was supposed to move for that particular move. To do this, a method called getDirection was implemented to take the current move string as its parameter and return true or false for the direction of movement. To get the true or false value, we went through the default mLayerPermutations array to determine whether the default direction for each layer was clockwise or counter clockwise and then return true

or false based on if the move was, for example, U, U', or U2. Once the direction was found, the program had to check if the move contained a 2, like U2, R2, L2, etc., and if it did, that particular layer rotation was completed twice. In addition, since a rotation direction equal to 'true' is the same as doing a rotation in the 'false' direction three times, the program also had to establish that if the direction was set to 'true', the layers and mPermutaiton tables were updated three times.

Next, buttons needed to be added to the screen in order to control the application because all that the 3D app consisted of at this point was the rotating Kube. This was difficult because the Kube API uses GLSurfaceView and Renderer rather than the Relative Layout that was used for the 2D application. After much trial and error with various approaches, the method that was used was to add the GLSurfaceView and set that as the ContentView for the app, then add a RelativeLayout ConventView that contained the buttons needed. This caused more code to be in the main activity of the program rather than in the provided layout folders, but this worked well for the system because the onClickListeners could be added as the buttons were added.

Finally, the app called for some minor tweaking to make the User Interface more appealing and user-friendly. The first part of this was to stop the layer movement while continuing with spinning the overall Kube. For this, the renderer angle had to be changed even when the layers stopped moving. To make this happen, everything that had to do with layer movement was put inside an if statement and was only ran if the cube had not been solved and the call to the renderer setAngle method was put outside of the if statement so that it was called every time. Next, the reset button was hidden while the application was in the middle of solving a cube, because pushing it mid-solve caused the program to crash. The onClick event for the 'Done' button that shows when the cube is solved was changed so that it would reset the cube. After that, an option was added for the user to either solve the cube step-by-step by pushing the Next Move button, or to solve the cube continuously by pressing one button so the steps run through automatically. Next, some of the layout dimensions were changed to be better displayed given any size of screen. In addition, the button backgrounds were changed to images rather than buttons with text to ensure uniformity across all screen sizes and to make the app look more sleek. Finally, the application's icon was changed to a more relevant image and the application's display name was changed to "Rubik's Cube Solver".

4 Development Details

4.1 Android Devices

"Android is a software bunch comprising not only operating system but also middleware and key applications" [**?**]. "Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model" [**?**]. "Hardwares that support Android are mainly based on ARM (Advanced RISC Machines) architecture platform" [**?**]. "These ARM machines have a 32 bit Reduced Instruction Set Computer (RISC) Load Store Architecture. The relative simplicity of ARM machines for low power applications like mobile, embedded and microcontroller applications and small microprocessors make them a lucrative choice for the manufacturers to bank on. The direct manipulation of memory isn't possible in this architecture and is done through the use of registers. The instruction set offers many conditional and other varieties of operations with the primary focus being on reducing the number of cycles per instruction featuring mostly single cycle operations" [**?**].

In addition, Android is owned and maintained by Google, making it a trusted source for many users. There are also countless android applications available worldwide, which is very likely due to the fact that Android development is entirely open source and is made easy through the android-provided Android SDK (Software Development Kit).

Overall, Android was the chosen platform because of its widespread popularity, free and open source development, and the Android-provided Software Development Kit. The second choice would have been Apple/iOS development, but researchers claimed that Android is in fact more popular than iOS [**?**] and being a student developer, Android's open source development was much more convenient and manageable. In addition, most Android development utilizes Java programming language, which is a very popular programming language and is important to learn and understand in today's world of computing.

4.2 Android Development Tool for Eclipse

The open source Android Development Tools (ADT) plugin for the Eclipse IDE was used for development based off of its high recommendation by Android. Their website states that

"developing in Eclipse with ADT is highly recommended and is the fastest way to get started. With the guided project setup it provides, as well as tools integration, custom XML editors, and debug output pane, ADT gives you an incredible boost in developing Android applications" [**?**]. The Android Developers site also provides documentation on how to download the ADT Plugin with or without having a previous version of Eclipse on your computer, which makes it easy for beginners to start using Eclipse and the Android Development Tools.

4.3 Java Programming Language

This application is written in Java, "a programming language and computing platform first released by Sun Microsystems in 1995" [**?**] for many reasons. Java.com states that software developers choose Java because it "has been tested, refined, extended, and proven by a dedicated community of Java developers, architects and enthusiasts. Java is designed to enable development of portable, high-performance applications for the widest range of computing platforms possible. By making applications available across heterogeneous environments, businesses can provide more services and boost end-user productivity, communication, and collaboration—and dramatically reduce the cost of ownership of both enterprise and consumer applications. Java has become invaluable to developers by enabling them to:

- Write software on one platform and run it on virtually any other platform
- Create programs that can run within a web browser and access available web services
- Develop server-side applications for online forums, stores, polls, HTML forms processing, and more
- Combine applications or services using the Java language to create highly customized applications or services
- Write powerful and efficient applications for mobile phones, remote processors, microcontrollers, wireless modules, sensors, gateways, consumer products, and practically any other electronic device" [**?**]

Many online sources state that Java is the most popular development language for Android Applications, primarily due to the powerful Java IDE that is provided through the Android Developer Tools. This IDE has "advanced features for developing, debugging, and packaging Android apps. Using the IDE, you can develop on any available Android device or create virtual devices that emulate any hardware configuration"[**?**]. Thus, Java was easily proven the most ideal development

language for this project as it is highly supported by Android and therefore is well documented online and has many built-in development and testing features.

4.4 Open Graphics Library

As previously stated, this application also utilizes the Open Graphics Library (OpenGL) for rendering 2D and 3D vector graphics. The OpenGL website claims that OpenGL is "the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms" [**?**]. OpenGL prides itself on being open source and "vendor-neutral", available on all platforms, stable, reliable and portable, evolving, scalable, easy to use, and well documented [**?**]. OpenGL was used for this project because the Android Developers website suggests using OpenGL if you want to have more control of the graphics you use in your application or if you are using 3D graphics. The website states, "the OpenGL ES APIs provided by the Android framework offers a set of tools for displaying high-end, animated graphics that are limited only by your imagination and can also benefit from the acceleration of graphics processing units (GPUs) provided on many Android devices" [**?**]. Therefore with little to no knowledge about graphics, OpenGL was the primary choice because it is well-documented and is easily integrated with Android applications.

4.5 Testing and Hardware

With Android development, testing is made easy on all android-supporting devices. The ADT for Eclipse comes with an Android Emulator that allows you to run and test your Android applications on your computer by setting up virtual devices. These virtual devices can be set up with a wide variety of settings in order to emulate virtually any type of hardware device. The main testing was done on one virtual device, which was modeled off of a 4" WVGA display running Android version 4.2.2 with the ability to use host graphics processing unit (GPU), 512 RAM, 32 VIM Heap, and 200MiB internal storage. I also tested on the following physical devices: a Samsung 4" Galaxy S smartphone running Android version 2.3 with 1GB of available internal storage, a Samsung 4.3" Galaxy S3 smartphone running Android version 4.1.2 with 12GB internal storage, and a Samsung 7" Galaxy Tab 3 tablet running Android version 4.1.2 with 8GB internal storage. Testing on these various pieces of hardware was primarily beneficial in working with graphics and spacing so the

app looks more uniform on all screen sizes.

5 Conclusion and Future Work

Creating this application has proven to be a very challenging yet rewarding experience. Through vast amounts of research and programming involved in this project, much knowledge was gained on the topics of Rubik's Cube theory, Rubik's Cube Algorithms, modifying algorithms to fit specific needs, Java programming language, Android development, OpenGL API, the Eclipse Android Development Tools, and testing and debugging with Eclipse. While the 2D and 3D applications are functional, they do not accomplish all of the goals originally set out for them. The following unfinished tasks are to be worked on in future proceedings with the project.

Currently, the 3D app does not allow the user to input an existing cube like the 2D app does due to problems with changing Cube colors after the Kube has been generated. In both the 2D and 3D apps, the ultimate goal is to also allow the user to input the cube structure by taking pictures of an existing cube with their smartphone/tablet camera, using Android color recognition to analyze the images.

Improvements could also be made to the interface to create a more intuitive and interactive program. Providing instructions on how to use the app, displaying arrows to show the user the exact rotation direction, having a "previous move" option, and creating a pop-up window that displays a message while the program is figuring out the solution would all greatly improve the ease of application use. Giving the users the capability to move the cube around its axes would improve user-interaction. Finally, adding an option to view general Rubik's Cube algorithms would increase the audience and uses of the application greatly.

In an ideal world, if all of these things could be implemented, the next step would be to expand the app to not only solve the classic 3x3x3 Rubik's Cubes, but also add options to solve 2x2x2, 4x4x4, and 5x5x5 Rubik's Cubes.

The existing 2D and 3D apps are now available on all Android devices that support Google's Play Store as "Rubik's Cube Solver (2D)" and "Rubik's Cube Solver", respectively. They can be easily found by searching "mmirtes" in the Play Store, or whereever Android Applications are downloaded from in the future. The final and ongoing goal for the project is to continue with the

development and improvements and to push the updates out through the Play Store so that all who have the apps or download them in the future have access to the most recent versions.

6 Source Code

All java source code for the 3D application is included here. The main activity for the program begins in Kube.java and all other files are called or referenced from that file.

Kube.java

```
1  package com.test.togetherness;
2
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.io.ObjectInputStream;
6  import java.io.StreamCorruptedException;
7  import java.nio.ByteBuffer;
8  import java.nio.ByteOrder;
9  import java.util.ArrayList;
10 import java.util.List;
11 import java.util.Random;
12
13 import com.test.togetherness.KubeRenderer.AnimationCallback;
14 import com.test.togetherness.R;
15 import com.test.togetherness.Search;
16 import com.test.togetherness.CoordCube;
17 import com.test.togetherness.Tools;
18
19 import android.app.Activity;
20 import android.app.AlertDialog;
21 import android.content.Context;
22 import android.content.Intent;
23 import android.graphics.PixelFormat;
24 import android.graphics.drawable.Drawable;
25 import android.opengl.GLSurfaceView;
26 import android.os.Bundle;
27 import android.view.Gravity;
28 import android.view.LayoutInflater;
29 import android.view.View;
30 import android.view.View.OnClickListener;
31 import android.view.ViewGroup;
32 import android.view.Window;
33 import android.widget.Button;
34 import android.widget.LinearLayout;
35 import android.widget.RelativeLayout;
36 import android.widget.RelativeLayout.LayoutParams;
37
38 public class Kube extends Activity implements KubeRenderer.
```

```
    AnimationCallback {
39  private static Drawable[] COLORS = new Drawable[6];
40  public static int[] cubeID =
        {0,1,2,3,4,5,6,7,8,8,5,2,17,14,11,26,23,20,6,7,8,15,16,17,24,25,26,
         24,25,26,21,22,23,18,19,20,0,3,6,9,12,15,18,21,24,2,1,0,11,10,9,
        20,19,18};
41      public static int[] faceID = {Cube.kTop,Cube.kTop,Cube.kTop,Cube.
            kTop,Cube.kTop,Cube.kTop,Cube.kTop,Cube.kTop,Cube.kTop,Cube.
            kRight,Cube.kRight,Cube.kRight,Cube.kRight,Cube.kRight,Cube.
            kRight,Cube.kRight,Cube.kRight,Cube.kRight,Cube.kFront,Cube.
            kFront,Cube.kFront,Cube.kFront,Cube.kFront,Cube.kFront,Cube.
            kFront,Cube.kFront,Cube.kFront,Cube.kBottom,Cube.kBottom,Cube.
            kBottom,Cube.kBottom,Cube.kBottom,Cube.kBottom,Cube.kBottom,Cube
            .kBottom,Cube.kBottom,Cube.kLeft,Cube.kLeft,Cube.kLeft,Cube.
            kLeft,Cube.kLeft,Cube.kLeft,Cube.kLeft,Cube.kLeft,Cube.kLeft,
            Cube.kBack,Cube.kBack,Cube.kBack,Cube.kBack,Cube.kBack,Cube.
            kBack,Cube.kBack,Cube.kBack,Cube.kBack};
42      int one = 0x10000;
43      int half = 0x08000;
44      GLColor red = new GLColor(one, 0, 0);
45      GLColor green = new GLColor(0, one, 0);
46      GLColor blue = new GLColor(0, 0, one);
47      GLColor yellow = new GLColor(one, one, 0);
48      GLColor orange = new GLColor(one, half, 0);
49      GLColor white = new GLColor(one, one, one);
50      GLColor black = new GLColor(0, 0, 0);
51      public static String PACKAGE_NAME;
52  protected static boolean firstSolve=true;
53  public static String currentCube;
54  private int maxDepth = 24, maxTime = 5000;
55  boolean useSeparator = false;
56  boolean showString = true;
57  GLWorld world;
58
59      public GLWorld makeGLWorld()
60      {
61          world = new GLWorld();
62
63          // coordinates for our cubes
64          float c0 = -1.0f;
65          float c1 = -0.38f;
66          float c2 = -0.32f;
67          float c3 = 0.32f;
68          float c4 = 0.38f;
69          float c5 = 1.0f;
70
71          // top back, left to right
72          mCubes[0]  = new Cube(world, c0, c4, c0, c1, c5, c1);
```

```
73          mCubes[1]  = new Cube(world, c2, c4, c0, c3, c5, c1);
74          mCubes[2]  = new Cube(world, c4, c4, c0, c5, c5, c1);
75          // top middle, left to right
76          mCubes[3]  = new Cube(world, c0, c4, c2, c1, c5, c3);
77          mCubes[4]  = new Cube(world, c2, c4, c2, c3, c5, c3);
78          mCubes[5]  = new Cube(world, c4, c4, c2, c5, c5, c3);
79          // top front, left to right
80          mCubes[6]  = new Cube(world, c0, c4, c4, c1, c5, c5);
81          mCubes[7]  = new Cube(world, c2, c4, c4, c3, c5, c5);
82          mCubes[8]  = new Cube(world, c4, c4, c4, c5, c5, c5);
83          // middle back, left to right
84          mCubes[9]  = new Cube(world, c0, c2, c0, c1, c3, c1);
85          mCubes[10] = new Cube(world, c2, c2, c0, c3, c3, c1);
86          mCubes[11] = new Cube(world, c4, c2, c0, c5, c3, c1);
87          // middle middle, left to right
88          mCubes[12] = new Cube(world, c0, c2, c2, c1, c3, c3);
89          mCubes[13] = null;
90          mCubes[14] = new Cube(world, c4, c2, c2, c5, c3, c3);
91          // middle front, left to right
92          mCubes[15] = new Cube(world, c0, c2, c4, c1, c3, c5);
93          mCubes[16] = new Cube(world, c2, c2, c4, c3, c3, c5);
94          mCubes[17] = new Cube(world, c4, c2, c4, c5, c3, c5);
95          // bottom back, left to right
96          mCubes[18] = new Cube(world, c0, c0, c0, c1, c1, c1);
97          mCubes[19] = new Cube(world, c2, c0, c0, c3, c1, c1);
98          mCubes[20] = new Cube(world, c4, c0, c0, c5, c1, c1);
99          // bottom middle, left to right
100         mCubes[21] = new Cube(world, c0, c0, c2, c1, c1, c3);
101         mCubes[22] = new Cube(world, c2, c0, c2, c3, c1, c3);
102         mCubes[23] = new Cube(world, c4, c0, c2, c5, c1, c3);
103         // bottom front, left to right
104         mCubes[24] = new Cube(world, c0, c0, c4, c1, c1, c5);
105         mCubes[25] = new Cube(world, c2, c0, c4, c3, c1, c5);
106         mCubes[26] = new Cube(world, c4, c0, c4, c5, c1, c5);
107
108         // paint the sides
109         int i, j;
110         // set all faces black by default
111         for (i = 0; i < 27; i++) {
112             Cube cube = mCubes[i];
113             if (cube != null) {
114                 for (j = 0; j < 6; j++)
115                     cube.setFaceColor(j, black);
116             }
117         }
118
119
120         //paint middle cubes
```

```
121         mCubes[4].setFaceColor(Cube.kTop, white);
122         mCubes[22].setFaceColor(Cube.kBottom,yellow);
123         mCubes[12].setFaceColor(Cube.kLeft, orange);
124         mCubes[14].setFaceColor(Cube.kRight, red);
125         mCubes[10].setFaceColor(Cube.kBack, blue);
126         mCubes[16].setFaceColor(Cube.kFront, green);
127
128
129         currentCube=genRandom();
130
131         for (i = 0; i < 27; i++)
132             if (mCubes[i] != null)
133                 world.addShape(mCubes[i]);
134
135         // initialize our permutation to solved position
136         mPermutation = new int[27];
137         for (i = 0; i < mPermutation.length; i++){
138             mPermutation[i] = i;
139         }
140
141         // initialize our permutation to given cube
142         for(int k=0;k<27;k++){
143       Cube cubeA = mCubes[k];
144       List<GLColor> cubeColors = new ArrayList<GLColor>();
145       if(k!=13){
146        for(int l=0;l<cubeA.mFaceList.size();l++){
147         GLFace face = cubeA.mFaceList.get(l);
148         cubeColors.add(face.getColor());
149        }
150       }
151      }
152
153
154         createLayers();
155         updateLayers();
156
157         world.generate();
158
159         return world;
160     }
161
162     //get the permutation bassed off of the current cube color
163     public int getPermutation(List<GLColor> cc){
164      if(cc.contains(white)&&cc.contains(blue)&&cc.contains(orange))
            return 0;
165      else if(cc.contains(white)&&cc.contains(blue)&&cc.contains(red))
            return 2;
166      else if(cc.contains(white)&&cc.contains(blue)) return 1;
```

```
167       else if(cc.contains(white)&&cc.contains(green)&&cc.contains(orange
              )) return 6;
168       else if(cc.contains(white)&&cc.contains(green)&&cc.contains(red))
              return 8;
169       else if(cc.contains(white)&&cc.contains(green)) return 7;
170       else if(cc.contains(white)&&cc.contains(orange)) return 3;
171       else if(cc.contains(white)&&cc.contains(red)) return 5;
172       else if(cc.contains(white)) return 4;
173       else if(cc.contains(yellow)&&cc.contains(blue)&&cc.contains(orange
              )) return 18;
174       else if(cc.contains(yellow)&&cc.contains(blue)&&cc.contains(red))
              return 20;
175       else if(cc.contains(yellow)&&cc.contains(blue)) return 19;
176       else if(cc.contains(yellow)&&cc.contains(green)&&cc.contains(
              orange)) return 24;
177       else if(cc.contains(yellow)&&cc.contains(green)&&cc.contains(red))
               return 26;
178       else if(cc.contains(yellow)&&cc.contains(green)) return 25;
179       else if(cc.contains(yellow)&&cc.contains(orange)) return 21;
180       else if(cc.contains(yellow)&&cc.contains(red)) return 23;
181       else if(cc.contains(yellow)) return 22;
182       else if(cc.contains(blue)&&cc.contains(orange)) return 9;
183       else if(cc.contains(blue)&&cc.contains(red)) return 11;
184       else if(cc.contains(blue)) return 10;
185       else if(cc.contains(green)&&cc.contains(orange)) return 15;
186       else if(cc.contains(green)&&cc.contains(red)) return 17;
187       else if(cc.contains(green)) return 16;
188       else if(cc.contains(orange)) return 12;
189       else if(cc.contains(red)) return 14;
190       else return 13;
191     }
192
193
194     /////RANDOM CUBE/////
195     public String genRandom() {
196   // ++++++++++++++++++++++++++++ Call Random function from package
        org.kociemba.twophase //
197   String r = Tools.randomCube();
198   System.out.println(r);
199   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
200   for(int i=0;i<54;i++){
201    switch(r.charAt(i)){
202    case 'U':
203     mCubes[cubeID[i]].setFaceColor(faceID[i], white);
204     break;
205    case 'R':
206     mCubes[cubeID[i]].setFaceColor(faceID[i], red);
207     break;
```

```
208     case 'F':
209      mCubes[cubeID[i]].setFaceColor(faceID[i], green);
210      break;
211     case 'D':
212      mCubes[cubeID[i]].setFaceColor(faceID[i], yellow);
213      break;
214     case 'L':
215      mCubes[cubeID[i]].setFaceColor(faceID[i], orange);
216      break;
217     case 'B':
218      mCubes[cubeID[i]].setFaceColor(faceID[i], blue);
219      break;
220     }
221     }
222   currentCube = r;
223
224   for (int i = 0; i < 27; i++)
225             if (mCubes[i] != null)
226                 world.addShape(mCubes[i]);
227
228         // initialize our permutation to solved position
229         mPermutation = new int[27];
230         for (int i = 0; i < mPermutation.length; i++){
231             mPermutation[i] = i;
232         }
233
234         // initialize our permutation to given cube
235         for(int k=0;k<27;k++){
236       Cube cubeA = mCubes[k];
237       List<GLColor> cubeColors = new ArrayList<GLColor>();
238       if(k!=13){
239        for(int l=0;l<cubeA.mFaceList.size();l++){
240         GLFace face = cubeA.mFaceList.get(l);
241         cubeColors.add(face.getColor());
242        }
243       }
244      }
245
246         createLayers();
247         updateLayers();
248
249         world.generate();
250   return r;
251   }
252
253
254
255     private void createLayers() {
```

```
256             mLayers[kUp] = new Layer(Layer.kAxisY);
257             mLayers[kDown] = new Layer(Layer.kAxisY);
258             mLayers[kLeft] = new Layer(Layer.kAxisX);
259             mLayers[kRight] = new Layer(Layer.kAxisX);
260             mLayers[kFront] = new Layer(Layer.kAxisZ);
261             mLayers[kBack] = new Layer(Layer.kAxisZ);
262             mLayers[kMiddle] = new Layer(Layer.kAxisX);
263             mLayers[kEquator] = new Layer(Layer.kAxisY);
264             mLayers[kSide] = new Layer(Layer.kAxisZ);
265         }
266
267         private void updateLayers() {
268             Layer layer;
269             GLShape[] shapes;
270             int i, j, k;
271
272             // up layer
273             layer = mLayers[kUp];
274             shapes = layer.mShapes;
275             for (i = 0; i < 9; i++)
276                 shapes[i] = mCubes[mPermutation[i]];
277
278             // down layer
279             layer = mLayers[kDown];
280             shapes = layer.mShapes;
281             for (i = 18, k = 0; i < 27; i++)
282                 shapes[k++] = mCubes[mPermutation[i]];
283
284             // left layer
285             layer = mLayers[kLeft];
286             shapes = layer.mShapes;
287             for (i = 0, k = 0; i < 27; i += 9)
288                 for (j = 0; j < 9; j += 3)
289                     shapes[k++] = mCubes[mPermutation[i + j]];
290
291             // right layer
292             layer = mLayers[kRight];
293             shapes = layer.mShapes;
294             for (i = 2, k = 0; i < 27; i += 9)
295                 for (j = 0; j < 9; j += 3)
296                     shapes[k++] = mCubes[mPermutation[i + j]];
297
298             // front layer
299             layer = mLayers[kFront];
300             shapes = layer.mShapes;
301             for (i = 6, k = 0; i < 27; i += 9)
302                 for (j = 0; j < 3; j++)
303                     shapes[k++] = mCubes[mPermutation[i + j]];
```

```
304
305         // back layer
306         layer = mLayers[kBack];
307         shapes = layer.mShapes;
308         for (i = 0, k = 0; i < 27; i += 9)
309             for (j = 0; j < 3; j++)
310                 shapes[k++] = mCubes[mPermutation[i + j]];
311
312         // middle layer
313         layer = mLayers[kMiddle];
314         shapes = layer.mShapes;
315         for (i = 1, k = 0; i < 27; i += 9)
316             for (j = 0; j < 9; j += 3)
317                 shapes[k++] = mCubes[mPermutation[i + j]];
318
319         // equator layer
320         layer = mLayers[kEquator];
321         shapes = layer.mShapes;
322         for (i = 9, k = 0; i < 18; i++)
323             shapes[k++] = mCubes[mPermutation[i]];
324
325         // side layer
326         layer = mLayers[kSide];
327         shapes = layer.mShapes;
328         for (i = 3, k = 0; i < 27; i += 9)
329             for (j = 0; j < 3; j++)
330                 shapes[k++] = mCubes[mPermutation[i + j]];
331     }
332
333     @Override
334     protected void onResume()
335     {
336         super.onResume();
337         mView.onResume();
338     }
339
340     @Override
341     protected void onPause()
342     {
343         super.onPause();
344         mView.onPause();
345     }
346
347     public void animate() {
348         // change our angle of view
349      mRenderer.setAngle(mRenderer.getAngle() + 1.2f);
350
351     if(cubeGenerated){
```

```
352             if(!cubeSolved){
353              if (mCurrentLayer == null) {//must set a new layer
354               if(!continuous){//if they haven't pushed the continuous
                     button, wait for next to be pushed
355                while(!next){
356                }
357                next=false;
358               }
359               int layerID = getLayerIDNumber(moves[moveCount]);
360                  mCurrentLayer = mLayers[layerID];
361                  mCurrentLayerPermutation = mLayerPermutations[layerID];
362                  mCurrentLayer.startAnimation();
363                  direction = getDirection(moves[moveCount]);
364                  if(moves[moveCount].contains("2")) count = 2;
365                  else count = 1;
366                  runOnUiThread(new Runnable() {
367                      @Override
368                      public void run() {
369                       Button currentButton = (Button) findViewById(R.id.
                             currentMove);
370                     if(moveCount<moves.length) currentButton.setText(moves[
                          moveCount]);
371                      }
372                  });
373               moveCount++;
374
375                  mCurrentAngle = 0;
376                   if (direction) {
377                      mAngleIncrement = (float)Math.PI / 50;
378                         mEndAngle = mCurrentAngle + ((float)Math.PI *
                               count) / 2f;
379                     } else {
380                      mAngleIncrement = -(float)Math.PI / 50;
381                         mEndAngle = mCurrentAngle - ((float)Math.PI *
                               count) / 2f;
382                  }
383              }
384
385               mCurrentAngle += mAngleIncrement;
386
387               if ((mAngleIncrement > 0f && mCurrentAngle >= mEndAngle) ||
388                       (mAngleIncrement < 0f && mCurrentAngle <= mEndAngle)
                             ) {
389                   mCurrentLayer.setAngle(mEndAngle);
390                   mCurrentLayer.endAnimation();
391                   mCurrentLayer = null;
392
393                   // adjust mPermutation based on the completed layer
```

```
                rotation
394             int[] newPermutation = new int[27];
395             for (int i = 0; i < 27; i++) {
396                 newPermutation[i] = mPermutation[
                        mCurrentLayerPermutation[i]];
397             }
398             mPermutation = newPermutation;
399
400             if(count==2){//must change the permutation twice
401              newPermutation = new int[27];
402                 for (int i = 0; i < 27; i++) {
403                     newPermutation[i] = mPermutation[
                            mCurrentLayerPermutation[i]];
404                 }
405                 mPermutation = newPermutation;
406             }
407
408             if(direction){//must change the permutation three times
                    (the same as moving it the opposite direction)
409              newPermutation = new int[27];
410                 for (int i = 0; i < 27; i++) {
411                     newPermutation[i] = mPermutation[
                            mCurrentLayerPermutation[i]];
412                 }
413                 mPermutation = newPermutation;
414                 newPermutation = new int[27];
415                 for (int i = 0; i < 27; i++) {
416                     newPermutation[i] = mPermutation[
                            mCurrentLayerPermutation[i]];
417                 }
418                 mPermutation = newPermutation;
419             }
420             updateLayers();
421           if(moveCount >= moves.length)  cubeSolved=true;
422
423
424         } else {
425             mCurrentLayer.setAngle(mCurrentAngle);
426         }
427     }
428         else
429          runOnUiThread(new Runnable() {
430                 @Override
431                 public void run() {
432                  Button currentButton = (Button) findViewById(R.id.
                        currentMove);
433                  currentButton.setText("");
434             currentButton.setBackgroundResource(R.drawable.done);
```

```
435             currentButton.setOnClickListener(new OnClickListener() {
436              @Override
437              public void onClick(View view){
438               Intent i = getBaseContext().getPackageManager().
                     getLaunchIntentForPackage( getBaseContext().
                     getPackageName() );
439               i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
440               startActivity(i);
441              }
442             });
443                     }
444                });
445      }
446     }
447
448     GLSurfaceView mView;
449     GLSurfaceView mView2;
450     public static KubeRenderer mRenderer;
451     Cube[] mCubes = new Cube[27];
452     // a Layer for each possible move
453     Layer[] mLayers = new Layer[9];
454     // permutations corresponding to a pi/2 rotation of each layer
           about its axis
455     static int[][] mLayerPermutations = {
456             // permutation for UP layer
457             { 2, 5, 8, 1, 4, 7, 0, 3, 6, 9, 10, 11, 12, 13, 14, 15, 16,
                    17, 18, 19, 20, 21, 22, 23, 24, 25, 26 },
458             // permutation for DOWN layer
459             { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
                    17, 20, 23, 26, 19, 22, 25, 18, 21, 24 },
460             // permutation for LEFT layer
461             { 6, 1, 2, 15, 4, 5, 24, 7, 8, 3, 10, 11, 12, 13, 14, 21,
                   16, 17, 0, 19, 20, 9, 22, 23, 18, 25, 26 },
462             // permutation for RIGHT layer
463             { 0, 1, 8, 3, 4, 17, 6, 7, 26, 9, 10, 5, 12, 13, 14, 15,
                   16, 23, 18, 19, 2, 21, 22, 11, 24, 25, 20 },
464             // permutation for FRONT layer
465             { 0, 1, 2, 3, 4, 5, 24, 15, 6, 9, 10, 11, 12, 13, 14, 25,
                   16, 7, 18, 19, 20, 21, 22, 23, 26, 17, 8 },
466             // permutation for BACK layer
467             { 18, 9, 0, 3, 4, 5, 6, 7, 8, 19, 10, 1, 12, 13, 14, 15,
                   16, 17, 20, 11, 2, 21, 22, 23, 24, 25, 26 },
468             // permutation for MIDDLE layer
469             { 0, 7, 2, 3, 16, 5, 6, 25, 8, 9, 4, 11, 12, 13, 14, 15,
                   22, 17, 18, 1, 20, 21, 10, 23, 24, 19, 26 },
470             // permutation for EQUATOR layer
471             { 0, 1, 2, 3, 4, 5, 6, 7, 8, 11, 14, 17, 10, 13, 16, 9, 12,
                    15, 18, 19, 20, 21, 22, 23, 24, 25, 26 },
```

```
472             // permutation for SIDE layer
473             { 0, 1, 2, 21, 12, 3, 6, 7, 8, 9, 10, 11, 22, 13, 4, 15,
                    16, 17, 18, 19, 20, 23, 14, 5, 24, 25, 26 }
474     };
475
476
477     //current permutation of (solved) starting position
478     int[] mPermutation;
479
480     //solution string and split into moves
481     String result = "";
482     String[] moves;
483     int moveCount=0;
484     //count for number of spins, either 1 or 2
485 int count;
486 //direction for spinning
487 boolean direction;
488 //if true, layers stop moving
489 boolean cubeSolved=false;
490 //if false, animate does nothing
491 boolean cubeGenerated=false;
492 //pauses between moves
493 boolean next=false;
494 //false if want to wait for next button, true if want continuous solve
495 boolean continuous=false;
496
497
498     // for random cube movements
499     Random mRandom = new Random(System.currentTimeMillis());
500     // currently turning layer
501     Layer mCurrentLayer = null;
502     // current and final angle for current Layer animation
503     float mCurrentAngle, mEndAngle;
504     // amount to increment angle
505     float mAngleIncrement;
506     int[] mCurrentLayerPermutation;
507
508     // names for our 9 layers (based on notation from http://www.
            cubefreak.net/notation.html)
509     static final int kUp = 0;
510     static final int kDown = 1;
511     static final int kLeft = 2;
512     static final int kRight = 3;
513     static final int kFront = 4;
514     static final int kBack = 5;
515     static final int kMiddle = 6;
516     static final int kEquator = 7;
517     static final int kSide = 8;
```

```
518
519     @Override
520     protected void onCreate(Bundle savedInstanceState)
521     {
522         super.onCreate(savedInstanceState);
523      PACKAGE_NAME = getApplicationContext().getPackageName();
524
525      //for no title
526      requestWindowFeature(Window.FEATURE_NO_TITLE);
527
528         mView = new GLSurfaceView(getApplication());
529         mRenderer = new KubeRenderer(makeGLWorld(), this);
530         mView.setEGLConfigChooser(8,8,8,8,16,0);
531         //mView.getHolder().setFormat(PixelFormat.TRANSLUCENT);
532         mView.setRenderer(mRenderer);
533         //mView.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
534         setContentView(mView);
535
536         //the relative layout for the buttons over the cube interface
537         RelativeLayout rel = new RelativeLayout(this);
538         View view;
539   LayoutInflater inflater = (LayoutInflater)  getApplicationContext().
        getSystemService(Context.LAYOUT_INFLATER_SERVICE);
540   view = inflater.inflate(R.layout.activity_kube, null);
541   rel.addView(view);
542   addContentView(rel, new ViewGroup.LayoutParams(ViewGroup.LayoutParams
        .FILL_PARENT, ViewGroup.LayoutParams.FILL_PARENT));
543
544   //create solve button
545   Button b = (Button) rel.findViewById(R.id.SolveCube);
546   b.setBackgroundResource(R.drawable.solve);
547   b.setOnClickListener(new OnClickListener() {
548    @Override
549    public void onClick(View view) {
550     showHideButtons();
551
552      String cubeString = currentCube;
553
554      if (showString) {
555       System.out.println("Cube Definiton String: " + cubeString);
556      }
557
558      if(!cubeString.equals("
           UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB")){
559       if(firstSolve)
560        getTables();
561
562       // ++++++++++++++++++++++ Call Search.solution method from
```

```
        package org.kociemba.twophase ++++++++++++++++++++++++
563      result = Search.solution(cubeString, maxDepth, maxTime,
             useSeparator);
564     }
565     else{
566      result = "Done";
567     }
568
569     // ++++++++++++++++++ Replace the error messages with more
           meaningful ones in your language ++++++++++++++++++++
570     System.out.println(result);
571
572     if (result.contains("Error"))
573      switch (result.charAt(result.length() - 1)) {
574      case '1':
575       result = "There are not exactly nine facelets of each color!";
576       break;
577      case '2':
578       result = "Not all 12 edges exist exactly once!";
579       break;
580      case '3':
581       result = "Flip error: One edge has to be flipped!";
582       break;
583      case '4':
584       result = "Not all 8 corners exist exactly once!";
585       break;
586      case '5':
587       result = "Twist error: One corner has to be twisted!";
588       break;
589      case '6':
590       result = "Parity error: Two corners or two edges have to be
             exchanged!";
591       break;
592      case '7':
593       result = "No solution exists for the given maximum move number!"
             ;
594       break;
595      case '8':
596       result = "Timeout, no solution found within given maximum time!"
             ;
597       break;
598      }
599     else{
600      System.out.println(result);
601      moves = result.split("\\s+");
602      cubeGenerated = true;
603     }
604    }
```

```
605     });
606
607     //create next move button
608     b = (Button) rel.findViewById(R.id.nextMove);
609     b.setBackgroundResource(R.drawable.next_move);
610     b.setOnClickListener(new OnClickListener() {
611      @Override
612      public void onClick(View view) {
613       next=true;
614      }
615     });
616
617     //create continuous button
618     b = (Button) rel.findViewById(R.id.Continuous);
619     b.setBackgroundResource(R.drawable.continuous);
620     b.setOnClickListener(new OnClickListener() {
621      @Override
622      public void onClick(View view) {
623       continuous=true;
624       next=true;
625      }
626     });
627
628     //create reset button
629     b = (Button) rel.findViewById(R.id.Reset);
630     b.setBackgroundResource(R.drawable.reset);
631     b.setOnClickListener(new OnClickListener() {
632      @Override
633      public void onClick(View view){
634       Intent i = getBaseContext().getPackageManager().
            getLaunchIntentForPackage( getBaseContext().getPackageName() );
635       i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
636       startActivity(i);
637      }
638     });
639       }
640
641       public int getLayerIDNumber(String moveString){
642        if(moveString.contains("U")) return kUp;
643        else if (moveString.contains("D")) return kDown;
644        else if (moveString.contains("L")) return kLeft;
645        else if (moveString.contains("R")) return kRight;
646        else if (moveString.contains("F")) return kFront;
647        else if (moveString.contains("B")) return kBack;
648        else return -1;
649       }
650
651       public boolean getDirection(String moveString){
```

```
652       if (moveString.equals("U")) return true;
653       else if (moveString.equals("U'")||moveString.equals("U2")) return
              false;
654       else if (moveString.equals("D")||moveString.equals("D2")) return
              false;
655       else if (moveString.equals("D'")) return true;
656       else if (moveString.equals("L")) return true;
657       else if (moveString.equals("L'")||moveString.equals("L2")) return
              false;
658       else if (moveString.equals("R")||moveString.equals("R2")) return
              false;
659       else if (moveString.equals("R'")) return true;
660       else if (moveString.equals("F")||moveString.equals("F2")) return
              false;
661       else if (moveString.equals("F'")) return true;
662       else if (moveString.equals("B")) return true;
663       else if (moveString.equals("B'")||moveString.equals("B2")) return
              false;
664       else return false;
665     }
666
667
668   public void showHideButtons(){
669    Button showButtons,hideButtons;
670    hideButtons = (Button) findViewById(R.id.SolveCube);
671    hideButtons.setVisibility(View.GONE);
672    hideButtons = (Button) findViewById(R.id.Reset);
673    hideButtons.setVisibility(View.GONE);
674    showButtons = (Button) findViewById(R.id.currentMove);
675    showButtons.setVisibility(View.VISIBLE);
676    showButtons = (Button) findViewById(R.id.nextMove);
677    showButtons.setVisibility(View.VISIBLE);
678    showButtons = (Button) findViewById(R.id.Continuous);
679    showButtons.setVisibility(View.VISIBLE);
680   }
681
682 public void getTables(){
683
684   Button dialogButton = (Button) findViewById(R.id.Solving);
685   dialogButton.setVisibility(View.VISIBLE);
686
687   ObjectInputStream tm = null;
688   InputStream ins1 = getResources().openRawResource(getResources().
          getIdentifier("twistmove","raw", Kube.PACKAGE_NAME));
689   try {
690    tm = new ObjectInputStream(ins1);
691   } catch (StreamCorruptedException e1) {
692    // TODO Auto-generated catch block
```

```
693   e1.printStackTrace();
694  } catch (IOException e1) {
695   // TODO Auto-generated catch block
696   e1.printStackTrace();
697  }
698  ObjectInputStream fm = null;
699  InputStream ins2 = getResources().openRawResource(getResources().
         getIdentifier("flipmove","raw", Kube.PACKAGE_NAME));
700  try {
701   fm = new ObjectInputStream(ins2);
702  } catch (StreamCorruptedException e1) {
703   // TODO Auto-generated catch block
704   e1.printStackTrace();
705  } catch (IOException e1) {
706   // TODO Auto-generated catch block
707   e1.printStackTrace();
708  }
709  ObjectInputStream frbr = null;
710  InputStream ins3 = getResources().openRawResource(getResources().
         getIdentifier("frtobrmove","raw", Kube.PACKAGE_NAME));
711  try {
712   frbr = new ObjectInputStream(ins3);
713  } catch (StreamCorruptedException e1) {
714   // TODO Auto-generated catch block
715   e1.printStackTrace();
716  } catch (IOException e1) {
717   // TODO Auto-generated catch block
718   e1.printStackTrace();
719  }
720  ObjectInputStream merge = null;
721  InputStream ins4 = getResources().openRawResource(getResources().
         getIdentifier("mergeurtoulandubtodf","raw", Kube.PACKAGE_NAME));
722  try {
723   merge = new ObjectInputStream(ins4);
724  } catch (StreamCorruptedException e1) {
725   // TODO Auto-generated catch block
726   e1.printStackTrace();
727  } catch (IOException e1) {
728   // TODO Auto-generated catch block
729   e1.printStackTrace();
730  }
731  ObjectInputStream sfp = null;
732  InputStream ins5 = getResources().openRawResource(getResources().
         getIdentifier("sliceflipprun","raw", Kube.PACKAGE_NAME));
733  try {
734   sfp = new ObjectInputStream(ins5);
735  } catch (StreamCorruptedException e1) {
736   // TODO Auto-generated catch block
```

```
737   e1.printStackTrace();
738  } catch (IOException e1) {
739   // TODO Auto-generated catch block
740   e1.printStackTrace();
741  }
742  ObjectInputStream stp = null;
743  InputStream ins6 = getResources().openRawResource(getResources().
         getIdentifier("slicetwistprun","raw", Kube.PACKAGE_NAME));
744  try {
745   stp = new ObjectInputStream(ins6);
746  } catch (StreamCorruptedException e1) {
747   // TODO Auto-generated catch block
748   e1.printStackTrace();
749  } catch (IOException e1) {
750   // TODO Auto-generated catch block
751   e1.printStackTrace();
752  }
753  ObjectInputStream surfdlf = null;
754  InputStream ins7 = getResources().openRawResource(getResources().
         getIdentifier("sliceurftodlfparityprun","raw", Kube.PACKAGE_NAME));
755  try {
756   surfdlf = new ObjectInputStream(ins7);
757  } catch (StreamCorruptedException e1) {
758   // TODO Auto-generated catch block
759   e1.printStackTrace();
760  } catch (IOException e1) {
761   // TODO Auto-generated catch block
762   e1.printStackTrace();
763  }
764  ObjectInputStream surdf = null;
765  InputStream ins8 = getResources().openRawResource(getResources().
         getIdentifier("sliceurtodfparityprun","raw", Kube.PACKAGE_NAME));
766  try {
767   surdf = new ObjectInputStream(ins8);
768  } catch (StreamCorruptedException e1) {
769   // TODO Auto-generated catch block
770   e1.printStackTrace();
771  } catch (IOException e1) {
772   // TODO Auto-generated catch block
773   e1.printStackTrace();
774  }
775  ObjectInputStream ubdf = null;
776  InputStream ins9 = getResources().openRawResource(getResources().
         getIdentifier("ubtodfmove","raw", Kube.PACKAGE_NAME));
777  try {
778   ubdf = new ObjectInputStream(ins9);
779  } catch (StreamCorruptedException e1) {
780   // TODO Auto-generated catch block
```

```
781   e1.printStackTrace();
782  } catch (IOException e1) {
783   // TODO Auto-generated catch block
784   e1.printStackTrace();
785  }
786  ObjectInputStream urfdlf = null;
787  InputStream ins10 = getResources().openRawResource(getResources().
         getIdentifier("urftodlfmove","raw", Kube.PACKAGE_NAME));
788  try {
789   urfdlf = new ObjectInputStream(ins10);
790  } catch (StreamCorruptedException e1) {
791   // TODO Auto-generated catch block
792   e1.printStackTrace();
793  } catch (IOException e1) {
794   // TODO Auto-generated catch block
795   e1.printStackTrace();
796  }
797  ObjectInputStream urdf = null;
798  InputStream ins11 = getResources().openRawResource(getResources().
         getIdentifier("urtodfmove","raw", Kube.PACKAGE_NAME));
799  try {
800   urdf = new ObjectInputStream(ins11);
801  } catch (StreamCorruptedException e1) {
802   // TODO Auto-generated catch block
803   e1.printStackTrace();
804  } catch (IOException e1) {
805   // TODO Auto-generated catch block
806   e1.printStackTrace();
807  }
808  ObjectInputStream urul = null;
809  InputStream ins12 = getResources().openRawResource(getResources().
         getIdentifier("urtoulmove","raw", Kube.PACKAGE_NAME));
810  try {
811   urul = new ObjectInputStream(ins12);
812  } catch (StreamCorruptedException e1) {
813   // TODO Auto-generated catch block
814   e1.printStackTrace();
815  } catch (IOException e1) {
816   // TODO Auto-generated catch block
817   e1.printStackTrace();
818  }
819
820     try {
821      for(int i=0;i<CoordCube.N_UBtoDF;i++){
822       byte ubdfbyte[] = new byte[CoordCube.N_MOVE*2];
823    ubdf.readFully(ubdfbyte);
824    short[] ubdfshort = new short[ubdfbyte.length/2];
825    // to turn bytes to shorts as either big endian or little endian.
```

```
826     ByteBuffer.wrap(ubdfbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer
            ().get(ubdfshort);
827         CoordCube.UBtoDF_Move[i]=ubdfshort;
828       }
829       for(int i=0;i<CoordCube.N_URFtoDLF;i++){
830         byte urfdlfbyte[] = new byte[CoordCube.N_MOVE*2];
831     urfdlf.readFully(urfdlfbyte);
832     short[] urfdlfshort = new short[urfdlfbyte.length/2];
833     // to turn bytes to shorts as either big endian or little endian.
834     ByteBuffer.wrap(urfdlfbyte).order(ByteOrder.BIG_ENDIAN).
            asShortBuffer().get(urfdlfshort);
835         CoordCube.URFtoDLF_Move[i]=urfdlfshort;
836       }
837       for(int i=0;i<CoordCube.N_URtoDF;i++){
838         byte urdfbyte[] = new byte[CoordCube.N_MOVE*2];
839     urdf.readFully(urdfbyte);
840     short[] urdfshort = new short[urdfbyte.length/2];
841     // to turn bytes to shorts as either big endian or little endian.
842     ByteBuffer.wrap(urdfbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer
            ().get(urdfshort);
843         CoordCube.URtoDF_Move[i]=urdfshort;
844       }
845       for(int i=0;i<CoordCube.N_URtoUL;i++){
846         byte urulbyte[] = new byte[CoordCube.N_MOVE*2];
847     urul.readFully(urulbyte);
848     short[] urulshort = new short[urulbyte.length/2];
849     // to turn bytes to shorts as either big endian or little endian.
850     ByteBuffer.wrap(urulbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer
            ().get(urulshort);
851         CoordCube.URtoUL_Move[i]=urulshort;
852       }
853       for(int i=0;i<CoordCube.N_FRtoBR;i++){
854         byte frbrbyte[] = new byte[CoordCube.N_MOVE*2];
855     frbr.readFully(frbrbyte);
856     short[] frbrshort = new short[frbrbyte.length/2];
857     // to turn bytes to shorts as either big endian or little endian.
858     ByteBuffer.wrap(frbrbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer
            ().get(frbrshort);
859         CoordCube.FRtoBR_Move[i]=frbrshort;
860       }
861       for(int i=0;i<CoordCube.N_TWIST;i++){
862         byte tmbyte[] = new byte[CoordCube.N_MOVE*2];
863     tm.readFully(tmbyte);
864     short[] tmshort = new short[tmbyte.length/2];
865     // to turn bytes to shorts as either big endian or little endian.
866     ByteBuffer.wrap(tmbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().
            get(tmshort);
867         CoordCube.twistMove[i]=tmshort;
```

```
868         }
869         for(int i=0;i<CoordCube.N_FLIP;i++){
870          byte fmbyte[] = new byte[CoordCube.N_MOVE*2];
871       fm.readFully(fmbyte);
872       short[] fmshort = new short[fmbyte.length/2];
873       // to turn bytes to shorts as either big endian or little endian.
874       ByteBuffer.wrap(fmbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().
              get(fmshort);
875          CoordCube.flipMove[i]=fmshort;
876         }
877         for(int i=0;i<336;i++){
878          byte mergebyte[] = new byte[336*2];
879       merge.readFully(mergebyte);
880       short[] mergeshort = new short[mergebyte.length/2];
881       // to turn bytes to shorts as either big endian or little endian.
882       ByteBuffer.wrap(mergebyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer
              ().get(mergeshort);
883          CoordCube.MergeURtoULandUBtoDF[i]=mergeshort;
884         }
885
886     byte stparray[] = new byte[CoordCube.N_SLICE1 * CoordCube.N_TWIST
            /2+1];
887     stp.readFully(stparray);
888         CoordCube.Slice_Twist_Prun=stparray;
889
890         byte sfparray[] = new byte[CoordCube.N_SLICE1 * CoordCube.N_FLIP
              /2];
891         sfp.readFully(sfparray);
892         CoordCube.Slice_Flip_Prun=sfparray;
893
894         byte surfdlfarray[] = new byte[CoordCube.N_SLICE2 * CoordCube.
              N_URFtoDLF * CoordCube.N_PARITY / 2];
895         surfdlf.readFully(surfdlfarray);
896         CoordCube.Slice_URFtoDLF_Parity_Prun=surfdlfarray;
897
898         byte surdfarray[] = new byte[CoordCube.N_SLICE2 * CoordCube.
              N_URtoDF * CoordCube.N_PARITY / 2];
899         surdf.readFully(surdfarray);
900         CoordCube.Slice_URtoDF_Parity_Prun=surdfarray;
901
902         firstSolve=false;
903     } catch (IOException e) {
904      // TODO Auto-generated catch block
905      e.printStackTrace();
906     }
907     try {
908      tm.close();
909      fm.close();
```

```
    frbr.close();
    merge.close();
    sfp.close();
    stp.close();
    surfdlf.close();
    surdf.close();
    ubdf.close();
    urfdlf.close();
    urdf.close();
    urul.close();
   } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
   }
   dialogButton.setVisibility(View.GONE);
   //dialog.dismiss();
}

}
```

Color.java

```
package com.test.togetherness;

//++++++++++++++++++++++++++++++ Names the colors of the cube facelets
    ++++++++++++++++++++++++++++++++++++++++++++++++++
enum Color {
 U, R, F, D, L, B
}
```

CoordCube.java

```
package com.test.togetherness;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

```
14 import java.io.Reader;
15 import java.io.StreamCorruptedException;
16
17 import android.content.*;
18 import android.content.res.Resources;
19 import android.os.Environment;
20
21
22
23 //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
24 // Representation of the cube on the coordinate level
25 class CoordCube {
26
27  static final short N_TWIST = 2187;// 3^7 possible corner orientations
28  static final short N_FLIP = 2048;// 2^11 possible edge flips
29  static final short N_SLICE1 = 495;// 12 choose 4 possible positions of
        FR,FL,BL,BR edges
30  static final short N_SLICE2 = 24;// 4! permutations of FR,FL,BL,BR
       edges in phase2
31  static final short N_PARITY = 2; // 2 possible corner parities
32  static final short N_URFtoDLF = 20160;// 8!/(8-6)! permutation of URF,
       UFL,ULB,UBR,DFR,DLF corners
33  static final short N_FRtoBR = 11880; // 12!/(12-4)! permutation of FR,
       FL,BL,BR edges
34  static final short N_URtoUL = 1320; // 12!/(12-3)! permutation of UR,
       UF,UL edges
35  static final short N_UBtoDF = 1320; // 12!/(12-3)! permutation of UB,
       DR,DF edges
36  static final short N_URtoDF = 20160; // 8!/(8-6)! permutation of UR,UF
       ,UL,UB,DR,DF edges in phase2
37
38  static final int N_URFtoDLB = 40320;// 8! permutations of the corners
39  static final int N_URtoBR = 479001600;// 8! permutations of the
       corners
40
41  static final short N_MOVE = 18;
42  static short[][] twistMove = new short[N_TWIST][N_MOVE];
43  static short[][] flipMove = new short[N_FLIP][N_MOVE];
44  static short[][] FRtoBR_Move = new short[N_FRtoBR][N_MOVE];
45  static short[][] URFtoDLF_Move = new short[N_URFtoDLF][N_MOVE];
46  static short[][] URtoDF_Move = new short[N_URtoDF][N_MOVE];
47  static short[][] URtoUL_Move = new short[N_URtoUL][N_MOVE];
48  static short[][] UBtoDF_Move = new short[N_UBtoDF][N_MOVE];
49  static short[][] MergeURtoULandUBtoDF = new short[336][336];
50  static byte[] Slice_URFtoDLF_Parity_Prun = new byte[N_SLICE2 *
       N_URFtoDLF * N_PARITY / 2];
51  static byte[] Slice_URtoDF_Parity_Prun = new byte[N_SLICE2 * N_URtoDF
       * N_PARITY / 2];
```

```
52  static byte[] Slice_Twist_Prun = new byte[N_SLICE1 * N_TWIST / 2 + 1];
53  static byte[] Slice_Flip_Prun = new byte[N_SLICE1 * N_FLIP / 2];
54
55  // All coordinates are 0 for a solved cube except for UBtoDF, which is
        114
56  short twist;
57  short flip;
58  short parity;
59  short FRtoBR;
60  short URFtoDLF;
61  short URtoUL;
62  short UBtoDF;
63  int URtoDF;
64
65  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
66  // Generate a CoordCube from a CubieCube
67  CoordCube(CubieCube c) {
68   twist = c.getTwist();
69   flip = c.getFlip();
70   parity = c.cornerParity();
71   FRtoBR = c.getFRtoBR();
72   URFtoDLF = c.getURFtoDLF();
73   URtoUL = c.getURtoUL();
74   UBtoDF = c.getUBtoDF();
75   URtoDF = c.getURtoDF();// only needed in phase2
76  }
77
78  // A move on the coordinate level
79  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
80  void move(int m) {
81   twist = twistMove[twist][m];
82   flip = flipMove[flip][m];
83   parity = parityMove[parity][m];
84   FRtoBR = FRtoBR_Move[FRtoBR][m];
85   URFtoDLF = URFtoDLF_Move[URFtoDLF][m];
86   URtoUL = URtoUL_Move[URtoUL][m];
87   UBtoDF = UBtoDF_Move[UBtoDF][m];
88   if (URtoUL < 336 && UBtoDF < 336)// updated only if UR,UF,UL,UB,DR,DF
89    // are not in UD-slice
90    URtoDF = MergeURtoULandUBtoDF[URtoUL][UBtoDF];
91  }
92
93  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
94  // Parity of the corner permutation. This is the same as the parity
        for the edge permutation of a valid cube.
95  // parity has values 0 and 1
96  static short[][] parityMove = { { 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
        1, 0, 1, 1, 0, 1 },
```

```
    { 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0 } };


 //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
 // Set pruning value in table. Two values are stored in one byte.
 static void setPruning(byte[] table, int index, byte value) {
  if ((index & 1) == 0)
   table[index / 2] &= 0xf0 | value;
  else
   table[index / 2] &= 0x0f | (value << 4);
 }

 //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
 // Extract pruning value
 static byte getPruning(byte[] table, int index) {
  if ((index & 1) == 0)
   return (byte) (table[index / 2] & 0x0f);
  else
   return (byte) ((table[index / 2] & 0xf0) >>> 4);
 }

 // **********************************Phase 1 and 2 movetable******//
 //these tables were generated and dropped into the res/raw folder to
     be read in

}
```

Corner.java

```
package com.test.togetherness;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
//The names of the corner positions of the cube. Corner URF e.g., has
    an U(p), a R(ight) and a F(ront) facelet
enum Corner {
 URF, UFL, ULB, UBR, DFR, DLF, DBL, DRB
}
```

Cube.java

```
/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

```
6  * You may obtain a copy of the License at
7  *
8  *      http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16
17 package com.test.togetherness;
18
19
20 public class Cube extends GLShape {
21
22  public Cube(GLWorld world, float left, float bottom, float back, float
        right, float top, float front) {
23   super(world);
24         GLVertex leftBottomBack = addVertex(left, bottom, back);
25        GLVertex rightBottomBack = addVertex(right, bottom, back);
26         GLVertex leftTopBack = addVertex(left, top, back);
27         GLVertex rightTopBack = addVertex(right, top, back);
28         GLVertex leftBottomFront = addVertex(left, bottom, front);
29         GLVertex rightBottomFront = addVertex(right, bottom, front);
30         GLVertex leftTopFront = addVertex(left, top, front);
31         GLVertex rightTopFront = addVertex(right, top, front);
32
33         // vertices are added in a clockwise orientation (when viewed
               from the outside)
34         // bottom
35         addFace(new GLFace(leftBottomBack, leftBottomFront,
               rightBottomFront, rightBottomBack));
36         // front
37         addFace(new GLFace(leftBottomFront, leftTopFront, rightTopFront
               , rightBottomFront));
38         // left
39         addFace(new GLFace(leftBottomBack, leftTopBack, leftTopFront,
               leftBottomFront));
40         // right
41         addFace(new GLFace(rightBottomBack, rightBottomFront,
               rightTopFront, rightTopBack));
42         // back
43         addFace(new GLFace(leftBottomBack, rightBottomBack,
               rightTopBack, leftTopBack));
44         // top
```

```
45          addFace(new GLFace(leftTopBack, rightTopBack, rightTopFront,
                leftTopFront));
46
47  }
48
49      public static final int kBottom = 0;
50      public static final int kFront = 1;
51      public static final int kLeft = 2;
52      public static final int kRight = 3;
53      public static final int kBack = 4;
54      public static final int kTop = 5;
55
56
57  }
```

CubieCube.java

```
1  package com.test.togetherness;
2
3  import static com.test.togetherness.Corner.*;
4  import static com.test.togetherness.Edge.*;
5
6  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
7  //Cube on the cubie level
8  class CubieCube {
9
10  // initialize to Id-Cube
11
12  // corner permutation
13  Corner[] cp = { URF, UFL, ULB, UBR, DFR, DLF, DBL, DRB };
14
15  // corner orientation
16  byte[] co = { 0, 0, 0, 0, 0, 0, 0, 0 };
17
18  // edge permutation
19  Edge[] ep = { UR, UF, UL, UB, DR, DF, DL, DB, FR, FL, BL, BR };
20
21  // edge orientation
22  byte[] eo = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
23
24  // ********************************** Moves on the cubie level
        ***//
25
26  private static Corner[] cpU = { UBR, URF, UFL, ULB, DFR, DLF, DBL, DRB
        };
27  private static byte[] coU = { 0, 0, 0, 0, 0, 0, 0, 0 };
```

```
28  private static Edge[] epU = { UB, UR, UF, UL, DR, DF, DL, DB, FR, FL,
        BL, BR };
29  private static byte[] eoU = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
30
31  private static Corner[] cpR = { DFR, UFL, ULB, URF, DRB, DLF, DBL, UBR
        };
32  private static byte[] coR = { 2, 0, 0, 1, 1, 0, 0, 2 };
33  private static Edge[] epR = { FR, UF, UL, UB, BR, DF, DL, DB, DR, FL,
        BL, UR };
34  private static byte[] eoR = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
35
36  private static Corner[] cpF = { UFL, DLF, ULB, UBR, URF, DFR, DBL, DRB
        };
37  private static byte[] coF = { 1, 2, 0, 0, 2, 1, 0, 0 };
38  private static Edge[] epF = { UR, FL, UL, UB, DR, FR, DL, DB, UF, DF,
        BL, BR };
39  private static byte[] eoF = { 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0 };
40
41  private static Corner[] cpD = { URF, UFL, ULB, UBR, DLF, DBL, DRB, DFR
        };
42  private static byte[] coD = { 0, 0, 0, 0, 0, 0, 0, 0 };
43  private static Edge[] epD = { UR, UF, UL, UB, DF, DL, DB, DR, FR, FL,
        BL, BR };
44  private static byte[] eoD = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
45
46  private static Corner[] cpL = { URF, ULB, DBL, UBR, DFR, UFL, DLF, DRB
        };
47  private static byte[] coL = { 0, 1, 2, 0, 0, 2, 1, 0 };
48  private static Edge[] epL = { UR, UF, BL, UB, DR, DF, FL, DB, FR, UL,
        DL, BR };
49  private static byte[] eoL = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
50
51  private static Corner[] cpB = { URF, UFL, UBR, DRB, DFR, DLF, ULB, DBL
        };
52  private static byte[] coB = { 0, 0, 1, 2, 0, 0, 2, 1 };
53  private static Edge[] epB = { UR, UF, UL, BR, DR, DF, DL, BL, FR, FL,
        UB, DB };
54  private static byte[] eoB = { 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1 };
55
56  // this CubieCube array represents the 6 basic cube moves
57  static CubieCube[] moveCube = new CubieCube[6];
58
59  static {
60   moveCube[0] = new CubieCube();
61   moveCube[0].cp = cpU;
62   moveCube[0].co = coU;
63   moveCube[0].ep = epU;
64   moveCube[0].eo = eoU;
```

```
65
66    moveCube[1] = new CubieCube();
67    moveCube[1].cp = cpR;
68    moveCube[1].co = coR;
69    moveCube[1].ep = epR;
70    moveCube[1].eo = eoR;
71
72    moveCube[2] = new CubieCube();
73    moveCube[2].cp = cpF;
74    moveCube[2].co = coF;
75    moveCube[2].ep = epF;
76    moveCube[2].eo = eoF;
77
78    moveCube[3] = new CubieCube();
79    moveCube[3].cp = cpD;
80    moveCube[3].co = coD;
81    moveCube[3].ep = epD;
82    moveCube[3].eo = eoD;
83
84    moveCube[4] = new CubieCube();
85    moveCube[4].cp = cpL;
86    moveCube[4].co = coL;
87    moveCube[4].ep = epL;
88    moveCube[4].eo = eoL;
89
90    moveCube[5] = new CubieCube();
91    moveCube[5].cp = cpB;
92    moveCube[5].co = coB;
93    moveCube[5].ep = epB;
94    moveCube[5].eo = eoB;
95
96  }
97
98  CubieCube() {
99
100 };
101
102 //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
103 CubieCube(Corner[] cp, byte[] co, Edge[] ep, byte[] eo) {
104   this();
105   for (int i = 0; i < 8; i++) {
106     this.cp[i] = cp[i];
107     this.co[i] = co[i];
108   }
109   for (int i = 0; i < 12; i++) {
110     this.ep[i] = ep[i];
111     this.eo[i] = eo[i];
112   }
```

```
113  }
114
115  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
116  // n choose k
117  static int Cnk(int n, int k) {
118   int i, j, s;
119   if (n < k)
120    return 0;
121   if (k > n / 2)
122    k = n - k;
123   for (s = 1, i = n, j = 1; i != n - k; i--, j++) {
124    s *= i;
125    s /= j;
126   }
127   return s;
128  }
129
130  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
131  static void rotateLeft(Corner[] arr, int l, int r)
132  // Left rotation of all array elements between l and r
133  {
134   Corner temp = arr[l];
135   for (int i = l; i < r; i++)
136    arr[i] = arr[i + 1];
137   arr[r] = temp;
138  }
139
140  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
141  static void rotateRight(Corner[] arr, int l, int r)
142  // Right rotation of all array elements between l and r
143  {
144   Corner temp = arr[r];
145   for (int i = r; i > l; i--)
146    arr[i] = arr[i - 1];
147   arr[l] = temp;
148  }
149
150  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
151  static void rotateLeft(Edge[] arr, int l, int r)
152  // Left rotation of all array elements between l and r
153  {
154   Edge temp = arr[l];
155   for (int i = l; i < r; i++)
156    arr[i] = arr[i + 1];
157   arr[r] = temp;
158  }
159
160  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
```

```
161  static void rotateRight(Edge[] arr, int l, int r)
162  // Right rotation of all array elements between l and r
163  {
164   Edge temp = arr[r];
165   for (int i = r; i > l; i--)
166    arr[i] = arr[i - 1];
167   arr[l] = temp;
168  }
169
170  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
171  // return cube in facelet representation
172  FaceCube toFaceCube() {
173   FaceCube fcRet = new FaceCube();
174   for (Corner c : Corner.values()) {
175    int i = c.ordinal();
176    int j = cp[i].ordinal();// cornercubie with index j is at
177    // cornerposition with index i
178    byte ori = co[i];// Orientation of this cubie
179    for (int n = 0; n < 3; n++)
180     fcRet.f[FaceCube.cornerFacelet[i][(n + ori) % 3].ordinal()] =
           FaceCube.cornerColor[j][n];
181   }
182   for (Edge e : Edge.values()) {
183    int i = e.ordinal();
184    int j = ep[i].ordinal();// edgecubie with index j is at edgeposition
185    // with index i
186    byte ori = eo[i];// Orientation of this cubie
187    for (int n = 0; n < 2; n++)
188     fcRet.f[FaceCube.edgeFacelet[i][(n + ori) % 2].ordinal()] =
           FaceCube.edgeColor[j][n];
189   }
190   return fcRet;
191  }
192
193  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
194  // Multiply this CubieCube with another cubiecube b, restricted to the
        corners.<br>
195  // Because we also describe reflections of the whole cube by
       permutations, we get a complication with the corners. The
196  // orientations of mirrored corners are described by the numbers 3, 4
       and 5. The composition of the orientations
197  // cannot
198  // be computed by addition modulo three in the cyclic group C3 any
       more. Instead the rules below give an addition in
199  // the dihedral group D3 with 6 elements.<br>
200  //
201  // NOTE: Because we do not use symmetry reductions and hence no
       mirrored cubes in this simple implementation of the
```

```
202  // Two-Phase-Algorithm, some code is not necessary here.
203  //
204  void cornerMultiply(CubieCube b) {
205   Corner[] cPerm = new Corner[8];
206   byte[] cOri = new byte[8];
207   for (Corner corn : Corner.values()) {
208    cPerm[corn.ordinal()] = cp[b.cp[corn.ordinal()].ordinal()];
209
210    byte oriA = co[b.cp[corn.ordinal()].ordinal()];
211    byte oriB = b.co[corn.ordinal()];
212    byte ori = 0;
213    ;
214    if (oriA < 3 && oriB < 3) // if both cubes are regular cubes...
215    {
216     ori = (byte) (oriA + oriB); // just do an addition modulo 3 here
217     if (ori >= 3)
218      ori -= 3; // the composition is a regular cube
219
220     // ++++++++++++++++++++not used in this implementation
          ++++++++++++++++++++++++++++++++++
221    } else if (oriA < 3 && oriB >= 3) // if cube b is in a mirrored
222    // state...
223    {
224     ori = (byte) (oriA + oriB);
225     if (ori >= 6)
226      ori -= 3; // the composition is a mirrored cube
227    } else if (oriA >= 3 && oriB < 3) // if cube a is an a mirrored
228    // state...
229    {
230     ori = (byte) (oriA - oriB);
231     if (ori < 3)
232      ori += 3; // the composition is a mirrored cube
233    } else if (oriA >= 3 && oriB >= 3) // if both cubes are in mirrored
234    // states...
235    {
236     ori = (byte) (oriA - oriB);
237     if (ori < 0)
238      ori += 3; // the composition is a regular cube
239     // ++++++++++++++++++++++++++++++++++++++++++++++++++//
240    }
241    cOri[corn.ordinal()] = ori;
242   }
243   for (Corner c : Corner.values()) {
244    cp[c.ordinal()] = cPerm[c.ordinal()];
245    co[c.ordinal()] = cOri[c.ordinal()];
246   }
247  }
248
```

```
249  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
250  // Multiply this CubieCube with another cubiecube b, restricted to the
         edges.
251  void edgeMultiply(CubieCube b) {
252   Edge[] ePerm = new Edge[12];
253   byte[] eOri = new byte[12];
254   for (Edge edge : Edge.values()) {
255    ePerm[edge.ordinal()] = ep[b.ep[edge.ordinal()].ordinal()];
256    eOri[edge.ordinal()] = (byte) ((b.eo[edge.ordinal()] + eo[b.ep[edge.
           ordinal()].ordinal()]) % 2);
257   }
258   for (Edge e : Edge.values()) {
259    ep[e.ordinal()] = ePerm[e.ordinal()];
260    eo[e.ordinal()] = eOri[e.ordinal()];
261   }
262  }
263
264  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
265  // Multiply this CubieCube with another CubieCube b.
266  void multiply(CubieCube b) {
267   cornerMultiply(b);
268   // edgeMultiply(b);
269  }
270
271  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
272  // Compute the inverse CubieCube
273  void invCubieCube(CubieCube c) {
274   for (Edge edge : Edge.values())
275    c.ep[ep[edge.ordinal()].ordinal()] = edge;
276   for (Edge edge : Edge.values())
277    c.eo[edge.ordinal()] = eo[c.ep[edge.ordinal()].ordinal()];
278   for (Corner corn : Corner.values())
279    c.cp[cp[corn.ordinal()].ordinal()] = corn;
280   for (Corner corn : Corner.values()) {
281    byte ori = co[c.cp[corn.ordinal()].ordinal()];
282    if (ori >= 3)// Just for completeness. We do not invert mirrored
283     // cubes in the program.
284     c.co[corn.ordinal()] = ori;
285    else {// the standard case
286     c.co[corn.ordinal()] = (byte) -ori;
287     if (c.co[corn.ordinal()] < 0)
288      c.co[corn.ordinal()] += 3;
289    }
290   }
291  }
292
293  // ********************************************* Get and set
         coordinates **//
```

```
294
295  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
296  // return the twist of the 8 corners. 0 <= twist < 3^7
297  short getTwist() {
298   short ret = 0;
299   for (int i = URF.ordinal(); i < DRB.ordinal(); i++)
300    ret = (short) (3 * ret + co[i]);
301   return ret;
302  }
303
304  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
305  void setTwist(short twist) {
306   int twistParity = 0;
307   for (int i = DRB.ordinal() - 1; i >= URF.ordinal(); i--) {
308    twistParity += co[i] = (byte) (twist % 3);
309    twist /= 3;
310   }
311   co[DRB.ordinal()] = (byte) ((3 - twistParity % 3) % 3);
312  }
313
314  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
315  // return the flip of the 12 edges. 0<= flip < 2^11
316  short getFlip() {
317   short ret = 0;
318   for (int i = UR.ordinal(); i < BR.ordinal(); i++)
319    ret = (short) (2 * ret + eo[i]);
320   return ret;
321  }
322
323  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
324  void setFlip(short flip) {
325   int flipParity = 0;
326   for (int i = BR.ordinal() - 1; i >= UR.ordinal(); i--) {
327    flipParity += eo[i] = (byte) (flip % 2);
328    flip /= 2;
329   }
330   eo[BR.ordinal()] = (byte) ((2 - flipParity % 2) % 2);
331  }
332
333  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
334  // Parity of the corner permutation
335  short cornerParity() {
336   int s = 0;
337   for (int i = DRB.ordinal(); i >= URF.ordinal() + 1; i--)
338    for (int j = i - 1; j >= URF.ordinal(); j--)
339     if (cp[j].ordinal() > cp[i].ordinal())
340      s++;
341   return (short) (s % 2);
```

```
342  }
343
344  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
345  // Parity of the edges permutation. Parity of corners and edges are
        the same if the cube is solvable.
346  short edgeParity() {
347   int s = 0;
348   for (int i = BR.ordinal(); i >= UR.ordinal() + 1; i--)
349    for (int j = i - 1; j >= UR.ordinal(); j--)
350     if (ep[j].ordinal() > ep[i].ordinal())
351      s++;
352   return (short) (s % 2);
353  }
354
355  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
356  // permutation of the UD-slice edges FR,FL,BL and BR
357  short getFRtoBR() {
358   int a = 0, x = 0;
359   Edge[] edge4 = new Edge[4];
360   // compute the index a < (12 choose 4) and the permutation array perm
        .
361   for (int j = BR.ordinal(); j >= UR.ordinal(); j--)
362    if (FR.ordinal() <= ep[j].ordinal() && ep[j].ordinal() <= BR.ordinal
         ()) {
363     a += Cnk(11 - j, x + 1);
364     edge4[3 - x++] = ep[j];
365    }
366
367   int b = 0;
368   for (int j = 3; j > 0; j--)// compute the index b < 4! for the
369   // permutation in perm
370   {
371    int k = 0;
372    while (edge4[j].ordinal() != j + 8) {
373     rotateLeft(edge4, 0, j);
374     k++;
375    }
376    b = (j + 1) * b + k;
377   }
378   return (short) (24 * a + b);
379  }
380
381  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
382  void setFRtoBR(short idx) {
383   int x;
384   Edge[] sliceEdge = { FR, FL, BL, BR };
385   Edge[] otherEdge = { UR, UF, UL, UB, DR, DF, DL, DB };
386   int b = idx % 24; // Permutation
```

```
387   int a = idx / 24; // Combination
388   for (Edge e : Edge.values())
389    ep[e.ordinal()] = DB;// Use UR to invalidate all edges
390
391   for (int j = 1, k; j < 4; j++)// generate permutation from index b
392   {
393    k = b % (j + 1);
394    b /= j + 1;
395    while (k-- > 0)
396     rotateRight(sliceEdge, 0, j);
397   }
398
399   x = 3;// generate combination and set slice edges
400   for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
401    if (a - Cnk(11 - j, x + 1) >= 0) {
402     ep[j] = sliceEdge[3 - x];
403     a -= Cnk(11 - j, x-- + 1);
404    }
405   x = 0; // set the remaining edges UR..DB
406   for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
407    if (ep[j] == DB)
408     ep[j] = otherEdge[x++];
409
410  }
411
412  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
413  // Permutation of all corners except DBL and DRB
414  short getURFtoDLF() {
415   int a = 0, x = 0;
416   Corner[] corner6 = new Corner[6];
417   // compute the index a < (8 choose 6) and the corner permutation.
418   for (int j = URF.ordinal(); j <= DRB.ordinal(); j++)
419    if (cp[j].ordinal() <= DLF.ordinal()) {
420     a += Cnk(j, x + 1);
421     corner6[x++] = cp[j];
422    }
423
424   int b = 0;
425   for (int j = 5; j > 0; j--)// compute the index b < 6! for the
426   // permutation in corner6
427   {
428    int k = 0;
429    while (corner6[j].ordinal() != j) {
430     rotateLeft(corner6, 0, j);
431     k++;
432    }
433    b = (j + 1) * b + k;
434   }
```

```
435     return (short) (720 * a + b);
436   }
437
438   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
439   void setURFtoDLF(short idx) {
440     int x;
441     Corner[] corner6 = { URF, UFL, ULB, UBR, DFR, DLF };
442     Corner[] otherCorner = { DBL, DRB };
443     int b = idx % 720; // Permutation
444     int a = idx / 720; // Combination
445     for (Corner c : Corner.values())
446       cp[c.ordinal()] = DRB;// Use DRB to invalidate all corners
447
448     for (int j = 1, k; j < 6; j++)// generate permutation from index b
449     {
450       k = b % (j + 1);
451       b /= j + 1;
452       while (k-- > 0)
453         rotateRight(corner6, 0, j);
454     }
455     x = 5;// generate combination and set corners
456     for (int j = DRB.ordinal(); j >= 0; j--)
457       if (a - Cnk(j, x + 1) >= 0) {
458         cp[j] = corner6[x];
459         a -= Cnk(j, x-- + 1);
460       }
461     x = 0;
462     for (int j = URF.ordinal(); j <= DRB.ordinal(); j++)
463       if (cp[j] == DRB)
464         cp[j] = otherCorner[x++];
465   }
466
467   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
468   // Permutation of the six edges UR,UF,UL,UB,DR,DF.
469   int getURtoDF() {
470     int a = 0, x = 0;
471     Edge[] edge6 = new Edge[6];
472     // compute the index a < (12 choose 6) and the edge permutation.
473     for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
474       if (ep[j].ordinal() <= DF.ordinal()) {
475         a += Cnk(j, x + 1);
476         edge6[x++] = ep[j];
477       }
478
479     int b = 0;
480     for (int j = 5; j > 0; j--)// compute the index b < 6! for the
481     // permutation in edge6
482     {
```

```
483     int k = 0;
484     while (edge6[j].ordinal() != j) {
485      rotateLeft(edge6, 0, j);
486      k++;
487     }
488     b = (j + 1) * b + k;
489    }
490    return 720 * a + b;
491   }
492
493   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
494   void setURtoDF(int idx) {
495    int x;
496    Edge[] edge6 = { UR, UF, UL, UB, DR, DF };
497    Edge[] otherEdge = { DL, DB, FR, FL, BL, BR };
498    int b = idx % 720; // Permutation
499    int a = idx / 720; // Combination
500    for (Edge e : Edge.values())
501     ep[e.ordinal()] = BR;// Use BR to invalidate all edges
502
503    for (int j = 1, k; j < 6; j++)// generate permutation from index b
504    {
505     k = b % (j + 1);
506     b /= j + 1;
507     while (k-- > 0)
508      rotateRight(edge6, 0, j);
509    }
510    x = 5;// generate combination and set edges
511    for (int j = BR.ordinal(); j >= 0; j--)
512     if (a - Cnk(j, x + 1) >= 0) {
513      ep[j] = edge6[x];
514      a -= Cnk(j, x-- + 1);
515     }
516    x = 0; // set the remaining edges DL..BR
517    for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
518     if (ep[j] == BR)
519      ep[j] = otherEdge[x++];
520   }
521
522   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
523   // Permutation of the six edges UR,UF,UL,UB,DR,DF
524   public static int getURtoDF(short idx1, short idx2) {
525    CubieCube a = new CubieCube();
526    CubieCube b = new CubieCube();
527    a.setURtoUL(idx1);
528    b.setUBtoDF(idx2);
529    for (int i = 0; i < 8; i++) {
530     if (a.ep[i] != BR)
```

```
531     if (b.ep[i] != BR)// collision
532      return -1;
533     else
534      b.ep[i] = a.ep[i];
535   }
536   return b.getURtoDF();
537  }
538
539  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
540  // Permutation of the three edges UR,UF,UL
541  short getURtoUL() {
542   int a = 0, x = 0;
543   Edge[] edge3 = new Edge[3];
544   // compute the index a < (12 choose 3) and the edge permutation.
545   for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
546    if (ep[j].ordinal() <= UL.ordinal()) {
547     a += Cnk(j, x + 1);
548     edge3[x++] = ep[j];
549    }
550
551   int b = 0;
552   for (int j = 2; j > 0; j--)// compute the index b < 3! for the
553   // permutation in edge3
554   {
555    int k = 0;
556    while (edge3[j].ordinal() != j) {
557     rotateLeft(edge3, 0, j);
558     k++;
559    }
560    b = (j + 1) * b + k;
561   }
562   return (short) (6 * a + b);
563  }
564
565  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
566  void setURtoUL(short idx) {
567   int x;
568   Edge[] edge3 = { UR, UF, UL };
569   int b = idx % 6; // Permutation
570   int a = idx / 6; // Combination
571   for (Edge e : Edge.values())
572    ep[e.ordinal()] = BR;// Use BR to invalidate all edges
573
574   for (int j = 1, k; j < 3; j++)// generate permutation from index b
575   {
576    k = b % (j + 1);
577    b /= j + 1;
578    while (k-- > 0)
```

```
579       rotateRight(edge3, 0, j);
580     }
581     x = 2;// generate combination and set edges
582     for (int j = BR.ordinal(); j >= 0; j--)
583       if (a - Cnk(j, x + 1) >= 0) {
584         ep[j] = edge3[x];
585         a -= Cnk(j, x-- + 1);
586       }
587   }
588
589   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
590   // Permutation of the three edges UB,DR,DF
591   short getUBtoDF() {
592     int a = 0, x = 0;
593     Edge[] edge3 = new Edge[3];
594     // compute the index a < (12 choose 3) and the edge permutation.
595     for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
596       if (UB.ordinal() <= ep[j].ordinal() && ep[j].ordinal() <= DF.ordinal
              ()) {
597         a += Cnk(j, x + 1);
598         edge3[x++] = ep[j];
599       }
600
601     int b = 0;
602     for (int j = 2; j > 0; j--)// compute the index b < 3! for the
603     // permutation in edge3
604     {
605       int k = 0;
606       while (edge3[j].ordinal() != UB.ordinal() + j) {
607         rotateLeft(edge3, 0, j);
608         k++;
609       }
610       b = (j + 1) * b + k;
611     }
612     return (short) (6 * a + b);
613   }
614
615   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
616   void setUBtoDF(short idx) {
617     int x;
618     Edge[] edge3 = { UB, DR, DF };
619     int b = idx % 6; // Permutation
620     int a = idx / 6; // Combination
621     for (Edge e : Edge.values())
622       ep[e.ordinal()] = BR;// Use BR to invalidate all edges
623
624     for (int j = 1, k; j < 3; j++)// generate permutation from index b
625     {
```

```
626     k = b % (j + 1);
627     b /= j + 1;
628     while (k--- > 0)
629       rotateRight(edge3, 0, j);
630   }
631   x = 2;// generate combination and set edges
632   for (int j = BR.ordinal(); j >= 0; j--)
633     if (a - Cnk(j, x + 1) >= 0) {
634       ep[j] = edge3[x];
635       a -= Cnk(j, x--- + 1);
636     }
637   }
638
639   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
640   int getURFtoDLB() {
641     Corner[] perm = new Corner[8];
642     int b = 0;
643     for (int i = 0; i < 8; i++)
644       perm[i] = cp[i];
645     for (int j = 7; j > 0; j--)// compute the index b < 8! for the
          permutation in perm
646     {
647       int k = 0;
648       while (perm[j].ordinal() != j) {
649         rotateLeft(perm, 0, j);
650         k++;
651       }
652       b = (j + 1) * b + k;
653     }
654     return b;
655   }
656
657   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
658   void setURFtoDLB(int idx) {
659     Corner[] perm = { URF, UFL, ULB, UBR, DFR, DLF, DBL, DRB };
660     int k;
661     for (int j = 1; j < 8; j++) {
662       k = idx % (j + 1);
663       idx /= j + 1;
664       while (k--- > 0)
665         rotateRight(perm, 0, j);
666     }
667     int x = 7;// set corners
668     for (int j = 7; j >= 0; j--)
669       cp[j] = perm[x--];
670   }
671
672   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
```

```
673  int getURtoBR() {
674   Edge[] perm = new Edge[12];
675   int b = 0;
676   for (int i = 0; i < 12; i++)
677    perm[i] = ep[i];
678   for (int j = 11; j > 0; j--)// compute the index b < 12! for the
          permutation in perm
679   {
680    int k = 0;
681    while (perm[j].ordinal() != j) {
682     rotateLeft(perm, 0, j);
683     k++;
684    }
685    b = (j + 1) * b + k;
686   }
687   return b;
688  }
689
690  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
691  void setURtoBR(int idx) {
692   Edge[] perm = { UR, UF, UL, UB, DR, DF, DL, DB, FR, FL, BL, BR };
693   int k;
694   for (int j = 1; j < 12; j++) {
695    k = idx % (j + 1);
696    idx /= j + 1;
697    while (k-- > 0)
698     rotateRight(perm, 0, j);
699   }
700   int x = 11;// set edges
701   for (int j = 11; j >= 0; j--)
702    ep[j] = perm[x--];
703  }
704
705  //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
706  // Check a cubiecube for solvability. Return the error code.
707  // 0: Cube is solvable
708  // -2: Not all 12 edges exist exactly once
709  // -3: Flip error: One edge has to be flipped
710  // -4: Not all corners exist exactly once
711  // -5: Twist error: One corner has to be twisted
712  // -6: Parity error: Two corners ore two edges have to be exchanged
713  int verify() {
714   int sum = 0;
715   int[] edgeCount = new int[12];
716   for (Edge e : Edge.values())
717    edgeCount[ep[e.ordinal()].ordinal()]++;
718   for (int i = 0; i < 12; i++)
719    if (edgeCount[i] != 1)
```

```
720       return -2;
721
722     for (int i = 0; i < 12; i++)
723      sum += eo[i];
724     if (sum % 2 != 0)
725      return -3;
726
727     int[] cornerCount = new int[8];
728     for (Corner c : Corner.values())
729      cornerCount[cp[c.ordinal()].ordinal()]++;
730     for (int i = 0; i < 8; i++)
731      if (cornerCount[i] != 1)
732       return -4;// missing corners
733
734     sum = 0;
735     for (int i = 0; i < 8; i++)
736      sum += co[i];
737     if (sum % 3 != 0)
738      return -5;// twisted corner
739
740     if ((edgeParity() ^ cornerParity()) != 0)
741      return -6;// parity error
742
743     return 0;// cube ok
744   }
745 }
```

Edge.java

```
1 package com.test.togetherness;
2
3 //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
4 //Then names of the edge positions of the cube. Edge UR e.g., has an U(
     p) and R(ight) facelet.
5 enum Edge {
6  UR, UF, UL, UB, DR, DF, DL, DB, FR, FL, BL, BR
7 }
```

FaceCube.java

```
1 package com.test.togetherness;
2
3 import static com.test.togetherness.Facelet.*;
4 import static com.test.togetherness.Color.*;
5 import static com.test.togetherness.Corner.*;
```

```
6  import static com.test.togetherness.Edge.*;
7
8  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
9  //Cube on the facelet level
10 class FaceCube {
11  public Color[] f = new Color[54];
12
13  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
14  // Map the corner positions to facelet positions. cornerFacelet[URF.
       ordinal()][0] e.g. gives the position of the
15  // facelet in the URF corner position, which defines the orientation.<
       br>
16  // cornerFacelet[URF.ordinal()][1] and cornerFacelet[URF.ordinal()][2]
        give the position of the other two facelets
17  // of the URF corner (clockwise).
18  final static Facelet[][] cornerFacelet = { { U9, R1, F3 }, { U7, F1,
       L3 }, { U1, L1, B3 }, { U3, B1, R3 },
19    { D3, F9, R7 }, { D1, L9, F7 }, { D7, B9, L7 }, { D9, R9, B7 } };
20
21  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
22  // Map the edge positions to facelet positions. edgeFacelet[UR.ordinal
       ()][0] e.g. gives the position of the facelet in
23  // the UR edge position, which defines the orientation.<br>
24  // edgeFacelet[UR.ordinal()][1] gives the position of the other
       facelet
25  final static Facelet[][] edgeFacelet = { { U6, R2 }, { U8, F2 }, { U4,
        L2 }, { U2, B2 }, { D6, R8 }, { D2, F8 },
26    { D4, L8 }, { D8, B8 }, { F6, R4 }, { F4, L6 }, { B6, L4 }, { B4, R6
         } };
27
28  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
29  // Map the corner positions to facelet colors.
30  final static Color[][] cornerColor = { { U, R, F }, { U, F, L }, { U,
       L, B }, { U, B, R }, { D, F, R }, { D, L, F },
31    { D, B, L }, { D, R, B } };
32
33  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
34  // Map the edge positions to facelet colors.
35  final static Color[][] edgeColor = { { U, R }, { U, F }, { U, L }, { U
       , B }, { D, R }, { D, F }, { D, L }, { D, B },
36    { F, R }, { F, L }, { B, L }, { B, R } };
37
38  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
39  FaceCube() {
40   String s = "UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB";
41   for (int i = 0; i < 54; i++)
42    f[i] = Color.valueOf(s.substring(i, i + 1));
43
```

```
44  }
45
46  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
47  // Construct a facelet cube from a string
48  FaceCube(String cubeString) {
49   for (int i = 0; i < cubeString.length(); i++)
50    f[i] = Color.valueOf(cubeString.substring(i, i + 1));
51  }
52
53  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
54  // Gives string representation of a facelet cube
55  String to_String() {
56   String s = "";
57   for (int i = 0; i < 54; i++)
58    s += f[i].toString();
59   return s;
60  }
61
62  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
63  // Gives CubieCube representation of a faceletcube
64  CubieCube toCubieCube() {
65   byte ori;
66   CubieCube ccRet = new CubieCube();
67   for (int i = 0; i < 8; i++)
68    ccRet.cp[i] = URF;// invalidate corners
69   for (int i = 0; i < 12; i++)
70    ccRet.ep[i] = UR;// and edges
71   Color col1, col2;
72   for (Corner i : Corner.values()) {
73    // get the colors of the cubie at corner i, starting with U/D
74    for (ori = 0; ori < 3; ori++)
75     if (f[cornerFacelet[i.ordinal()][ori].ordinal()] == U || f[
        cornerFacelet[i.ordinal()][ori].ordinal()] == D)
76      break;
77    col1 = f[cornerFacelet[i.ordinal()][(ori + 1) % 3].ordinal()];
78    col2 = f[cornerFacelet[i.ordinal()][(ori + 2) % 3].ordinal()];
79
80    for (Corner j : Corner.values()) {
81     if (col1 == cornerColor[j.ordinal()][1] && col2 == cornerColor[j.
        ordinal()][2]) {
82      // in cornerposition i we have cornercubie j
83      ccRet.cp[i.ordinal()] = j;
84      ccRet.co[i.ordinal()] = (byte) (ori % 3);
85      break;
86     }
87    }
88   }
89   for (Edge i : Edge.values())
```

```
90      for (Edge j : Edge.values()) {
91        if (f[edgeFacelet[i.ordinal()][0].ordinal()] == edgeColor[j.ordinal
              ()][0]
92            && f[edgeFacelet[i.ordinal()][1].ordinal()] == edgeColor[j.
                  ordinal()][1]) {
93          ccRet.ep[i.ordinal()] = j;
94          ccRet.eo[i.ordinal()] = 0;
95          break;
96        }
97        if (f[edgeFacelet[i.ordinal()][0].ordinal()] == edgeColor[j.ordinal
              ()][1]
98            && f[edgeFacelet[i.ordinal()][1].ordinal()] == edgeColor[j.
                  ordinal()][0]) {
99          ccRet.ep[i.ordinal()] = j;
100         ccRet.eo[i.ordinal()] = 1;
101         break;
102       }
103     }
104   return ccRet;
105 };
106 }
```

Facelet.java

```
1  package com.test.togetherness;
2
3  /**
4   * <pre>
5   * The names of the facelet positions of the cube
6   *                |************|
7   *                |*U1**U2**U3*|
8   *                |************|
9   *                |*U4**U5**U6*|
10  *                |************|
11  *                |*U7**U8**U9*|
12  *                |************|
13  * ************|************|************|************|
14  * *L1**L2**L3*|*F1**F2**F3*|*R1**R2**F3*|*B1**B2**B3*|
15  * ************|************|************|************|
16  * *L4**L5**L6*|*F4**F5**F6*|*R4**R5**R6*|*B4**B5**B6*|
17  * ************|************|************|************|
18  * *L7**L8**L9*|*F7**F8**F9*|*R7**R8**R9*|*B7**B8**B9*|
19  * ************|************|************|************|
20  *                |************|
21  *                |*D1**D2**D3*|
22  *                |************|
23  *                |*D4**D5**D6*|
```

```
24  *                |***********|
25  *                |*D7**D8**D9*|
26  *                |***********|
27  * </pre>
28  *
29  *A cube definition string "UBL..." means for example: In position U1
        we have the U-color, in position U2 we have the
30  * B-color, in position U3 we have the L color etc. according to the
        order U1, U2, U3, U4, U5, U6, U7, U8, U9, R1, R2,
31  * R3, R4, R5, R6, R7, R8, R9, F1, F2, F3, F4, F5, F6, F7, F8, F9, D1,
        D2, D3, D4, D5, D6, D7, D8, D9, L1, L2, L3, L4,
32  * L5, L6, L7, L8, L9, B1, B2, B3, B4, B5, B6, B7, B8, B9 of the enum
        constants.
33  */
34 public enum Facelet {
35  U1, U2, U3, U4, U5, U6, U7, U8, U9, R1, R2, R3, R4, R5, R6, R7, R8, R9
        , F1, F2, F3, F4, F5, F6, F7, F8, F9, D1, D2, D3, D4, D5, D6, D7,
       D8, D9, L1, L2, L3, L4, L5, L6, L7, L8, L9, B1, B2, B3, B4, B5, B6,
        B7, B8, B9
36 }
```

GLColor.java

```
1  /*
2   * Copyright (C) 2008 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
        implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package com.test.togetherness;
18
19 public class GLColor {
20
21     public final int red;
22     public final int green;
23     public final int blue;
```

```
24      public final int alpha;
25
26      public GLColor(int red, int green, int blue, int alpha) {
27          this.red = red;
28          this.green = green;
29          this.blue = blue;
30          this.alpha = alpha;
31      }
32
33      public GLColor(int red, int green, int blue) {
34          this.red = red;
35          this.green = green;
36          this.blue = blue;
37          this.alpha = 0x10000;
38      }
39
40      @Override
41      public boolean equals(Object other) {
42          if (other instanceof GLColor) {
43              GLColor color = (GLColor)other;
44              return (red == color.red &&
45                      green == color.green &&
46                      blue == color.blue &&
47                      alpha == color.alpha);
48          }
49          return false;
50      }
51  }
```

GLFace.java

```
1  /*
2   * Copyright (C) 2008 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
        implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
```

```
16
17 package com.test.togetherness;
18
19 import android.util.Log;
20
21 import java.nio.ShortBuffer;
22 import java.util.ArrayList;
23
24 public class GLFace {
25
26  public GLFace() {
27
28  }
29
30  // for triangles
31  public GLFace(GLVertex v1, GLVertex v2, GLVertex v3) {
32   addVertex(v1);
33   addVertex(v2);
34   addVertex(v3);
35  }
36  // for quadrilaterals
37  public GLFace(GLVertex v1, GLVertex v2, GLVertex v3, GLVertex v4) {
38   addVertex(v1);
39   addVertex(v2);
40   addVertex(v3);
41   addVertex(v4);
42  }
43
44  public void addVertex(GLVertex v) {
45   mVertexList.add(v);
46  }
47
48  // must be called after all vertices are added
49  public void setColor(GLColor c) {
50
51   int last = mVertexList.size() - 1;
52   if (last < 2) {
53    Log.e("GLFace", "not enough vertices in setColor()");
54   } else {
55    GLVertex vertex = mVertexList.get(last);
56
57    // only need to do this if the color has never been set
58    if (mColor == null) {
59     while (vertex.color != null) {
60      mVertexList.add(0, vertex);
61      mVertexList.remove(last + 1);
62      vertex = mVertexList.get(last);
63     }
```

```
64      }
65
66      vertex.color = c;
67    }
68
69    mColor = c;
70  }
71
72  public GLColor getColor(){
73    return this.mColor;
74  }
75
76  public int getIndexCount() {
77    return (mVertexList.size() - 2) * 3;
78  }
79
80  public void putIndices(ShortBuffer buffer) {
81    int last = mVertexList.size() - 1;
82
83    GLVertex v0 = mVertexList.get(0);
84    GLVertex vn = mVertexList.get(last);
85
86    // push triangles into the buffer
87    for (int i = 1; i < last; i++) {
88      GLVertex v1 = mVertexList.get(i);
89      buffer.put(v0.index);
90      buffer.put(v1.index);
91      buffer.put(vn.index);
92      v0 = v1;
93    }
94  }
95
96  private ArrayList<GLVertex> mVertexList = new ArrayList<GLVertex>();
97  private GLColor mColor;
98 }
```

GLShape.java

```
1 /*
2  * Copyright (C) 2008 The Android Open Source Project
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  *      http://www.apache.org/licenses/LICENSE-2.0
9  *
```

```
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
        implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package com.test.togetherness;
18
19 import java.nio.ShortBuffer;
20 import java.util.ArrayList;
21 import java.util.Iterator;
22
23 public class GLShape {
24
25  public GLShape(GLWorld world) {
26   mWorld = world;
27  }
28
29  public void addFace(GLFace face) {
30   mFaceList.add(face);
31  }
32
33  public void setFaceColor(int face, GLColor color) {
34   mFaceList.get(face).setColor(color);
35  }
36
37  public void putIndices(ShortBuffer buffer) {
38   Iterator<GLFace> iter = mFaceList.iterator();
39   while (iter.hasNext()) {
40    GLFace face = iter.next();
41    face.putIndices(buffer);
42   }
43  }
44
45  public int getIndexCount() {
46   int count = 0;
47   Iterator<GLFace> iter = mFaceList.iterator();
48   while (iter.hasNext()) {
49    GLFace face = iter.next();
50    count += face.getIndexCount();
51   }
52   return count;
53  }
54
55  public GLVertex addVertex(float x, float y, float z) {
56
```

```
57    // look for an existing GLVertex first
58    Iterator<GLVertex> iter = mVertexList.iterator();
59    while (iter.hasNext()) {
60     GLVertex vertex = iter.next();
61     if (vertex.x == x && vertex.y == y && vertex.z == z) {
62      return vertex;
63     }
64    }
65
66    // doesn't exist, so create new vertex
67    GLVertex vertex = mWorld.addVertex(x, y, z);
68    mVertexList.add(vertex);
69    return vertex;
70   }
71
72   public void animateTransform(M4 transform) {
73    mAnimateTransform = transform;
74
75    if (mTransform != null)
76     transform = mTransform.multiply(transform);
77
78    Iterator<GLVertex> iter = mVertexList.iterator();
79    while (iter.hasNext()) {
80     GLVertex vertex = iter.next();
81     mWorld.transformVertex(vertex, transform);
82    }
83   }
84
85   public void startAnimation() {
86   }
87
88   public void endAnimation() {
89    if (mTransform == null) {
90     mTransform = new M4(mAnimateTransform);
91    } else {
92     mTransform = mTransform.multiply(mAnimateTransform);
93    }
94   }
95
96   public M4       mTransform;
97   public M4       mAnimateTransform;
98   protected ArrayList<GLFace>  mFaceList = new ArrayList<GLFace>();
99   protected ArrayList<GLVertex> mVertexList = new ArrayList<GLVertex>();
100  protected ArrayList<Integer> mIndexList = new ArrayList<Integer>(); //
         make more efficient?
101  protected GLWorld mWorld;
102 }
```

```java
/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.test.togetherness;

import java.nio.IntBuffer;

public class GLVertex {

    public float x;
    public float y;
    public float z;
    final short index; // index in vertex table
    GLColor color;

    GLVertex() {
        this.x = 0;
        this.y = 0;
        this.z = 0;
        this.index = -1;
    }

    GLVertex(float x, float y, float z, int index) {
        this.x = x;
        this.y = y;
        this.z = z;
        this.index = (short)index;
    }

    @Override
    public boolean equals(Object other) {
```

```
45          if (other instanceof GLVertex) {
46              GLVertex v = (GLVertex)other;
47              return (x == v.x && y == v.y && z == v.z);
48          }
49          return false;
50      }
51
52      static public int toFixed(float x) {
53          return (int)(x * 65536.0f);
54      }
55
56      public void put(IntBuffer vertexBuffer, IntBuffer colorBuffer) {
57          vertexBuffer.put(toFixed(x));
58          vertexBuffer.put(toFixed(y));
59          vertexBuffer.put(toFixed(z));
60          if (color == null) {
61              colorBuffer.put(0);
62              colorBuffer.put(0);
63              colorBuffer.put(0);
64              colorBuffer.put(0);
65          } else {
66              colorBuffer.put(color.red);
67              colorBuffer.put(color.green);
68              colorBuffer.put(color.blue);
69              colorBuffer.put(color.alpha);
70          }
71      }
72
73      public void update(IntBuffer vertexBuffer, M4 transform) {
74          // skip to location of vertex in mVertex buffer
75          vertexBuffer.position(index * 3);
76
77          if (transform == null) {
78              vertexBuffer.put(toFixed(x));
79              vertexBuffer.put(toFixed(y));
80              vertexBuffer.put(toFixed(z));
81          } else {
82              GLVertex temp = new GLVertex();
83              transform.multiply(this, temp);
84              vertexBuffer.put(toFixed(temp.x));
85              vertexBuffer.put(toFixed(temp.y));
86              vertexBuffer.put(toFixed(temp.z));
87          }
88      }
89  }
```

```
1  /*
2   * Copyright (C) 2008 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
        implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package com.test.togetherness;
18
19 import java.nio.ByteBuffer;
20 import java.nio.ByteOrder;
21 import java.nio.IntBuffer;
22 import java.nio.ShortBuffer;
23 import java.util.Iterator;
24 import java.util.ArrayList;
25
26 import javax.microedition.khronos.opengles.GL10;
27
28 public class GLWorld {
29
30  public void addShape(GLShape shape) {
31   mShapeList.add(shape);
32   mIndexCount += shape.getIndexCount();
33  }
34
35  public void generate() {
36      ByteBuffer bb = ByteBuffer.allocateDirect(mVertexList.size()*4*4);
37      bb.order(ByteOrder.nativeOrder());
38   mColorBuffer = bb.asIntBuffer();
39
40      bb = ByteBuffer.allocateDirect(mVertexList.size()*4*3);
41      bb.order(ByteOrder.nativeOrder());
42      mVertexBuffer = bb.asIntBuffer();
43
44      bb = ByteBuffer.allocateDirect(mIndexCount*2);
```

```
45        bb.order(ByteOrder.nativeOrder());
46        mIndexBuffer = bb.asShortBuffer();
47
48    Iterator<GLVertex> iter2 = mVertexList.iterator();
49    while (iter2.hasNext()) {
50     GLVertex vertex = iter2.next();
51     vertex.put(mVertexBuffer, mColorBuffer);
52    }
53
54    Iterator<GLShape> iter3 = mShapeList.iterator();
55    while (iter3.hasNext()) {
56     GLShape shape = iter3.next();
57     shape.putIndices(mIndexBuffer);
58    }
59   }
60
61   public GLVertex addVertex(float x, float y, float z) {
62    GLVertex vertex = new GLVertex(x, y, z, mVertexList.size());
63    mVertexList.add(vertex);
64    return vertex;
65   }
66
67   public void transformVertex(GLVertex vertex, M4 transform) {
68    vertex.update(mVertexBuffer, transform);
69   }
70
71   int count = 0;
72      public void draw(GL10 gl)
73      {
74    mColorBuffer.position(0);
75    mVertexBuffer.position(0);
76    mIndexBuffer.position(0);
77
78    gl.glFrontFace(GL10.GL_CW);
79          gl.glShadeModel(GL10.GL_FLAT);
80          gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
81          gl.glColorPointer(4, GL10.GL_FIXED, 0, mColorBuffer);
82          gl.glDrawElements(GL10.GL_TRIANGLES, mIndexCount, GL10.
               GL_UNSIGNED_SHORT, mIndexBuffer);
83          count++;
84      }
85
86      static public float toFloat(int x) {
87       return x/65536.0f;
88      }
89
90   private ArrayList<GLShape> mShapeList = new ArrayList<GLShape>();
91   private ArrayList<GLVertex> mVertexList = new ArrayList<GLVertex>();
```

```
92
93  private int mIndexCount = 0;
94
95      private IntBuffer   mVertexBuffer;
96      private IntBuffer   mColorBuffer;
97      private ShortBuffer mIndexBuffer;
98  }
```

KubeRenderer.java

```
1  /*
2   * Copyright (C) 2008 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package com.test.togetherness;
18
19 import javax.microedition.khronos.egl.EGLConfig;
20 import javax.microedition.khronos.opengles.GL10;
21
22 import android.opengl.GLSurfaceView;
23
24 /**
25  * Example of how to use OpenGL|ES in a custom view
26  *
27  */
28 class KubeRenderer implements GLSurfaceView.Renderer {
29  // For controlling cube's z-position, x and y angles and speeds (NEW)
30      float angleX = 0;    // (NEW)
31      float angleY = 0;    // (NEW)
32      float speedX = 0;    // (NEW)
33      float speedY = 0;    // (NEW)
34      float z = -6.0f;     // (NEW)
35
36      public interface AnimationCallback {
```

```
37          void animate();
38      }
39
40      public KubeRenderer(GLWorld world, AnimationCallback callback) {
41          mWorld = world;
42          mCallback = callback;
43      }
44
45
46
47      public void onDrawFrame(GL10 gl) {
48
49           if (mCallback != null) {
50               mCallback.animate();
51           }
52
53          /*
54           * Usually, the first thing one might want to do is to clear
55           * the screen. The most efficient way of doing this is to use
56           * glClear(). However we must make sure to set the scissor
57           * correctly first. The scissor is always specified in window
58           * coordinates:
59           */
60
61          gl.glClearColor(0.5f,0.5f,0.5f,1);
62          gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT)
                ;
63
64          /*
65           * Now we're ready to draw some 3D object
66           */
67
68       gl.glMatrixMode(GL10.GL_MODELVIEW);
69          gl.glLoadIdentity();
70          gl.glTranslatef(0, 0, -3.0f);
71          gl.glScalef(0.5f, 0.5f, 0.5f);
72          gl.glRotatef(mAngle,        0, 1, 0);
73          gl.glRotatef(mAngle*0.25f,  1, 0, 0);
74
75          gl.glColor4f(0.7f, 0.7f, 0.7f, 1.0f);
76          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
77          gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
78          gl.glEnable(GL10.GL_CULL_FACE);
79          gl.glShadeModel(GL10.GL_SMOOTH);
80          gl.glEnable(GL10.GL_DEPTH_TEST);
81
82
83          mWorld.draw(gl);
```

```
84      }
85
86      public void onSurfaceChanged(GL10 gl, int width, int height) {
87          gl.glViewport(0, 0, width, height);
88
89          /*
90           * Set our projection matrix. This doesn't have to be done
91           * each time we draw, but usually a new projection needs to be
                set
92           * when the viewport is resized.
93           */
94
95          float ratio = (float)width / height;
96          gl.glMatrixMode(GL10.GL_PROJECTION);
97          gl.glLoadIdentity();
98          gl.glFrustumf(-ratio, ratio, -1, 1, 2, 12);
99
100         /*
101          * By default, OpenGL enables features that improve quality
102          * but reduce performance. One might want to tweak that
103          * especially on software renderer.
104          */
105         gl.glDisable(GL10.GL_DITHER);
106         gl.glActiveTexture(GL10.GL_TEXTURE0);
107     }
108
109     public void onSurfaceCreated(GL10 gl, EGLConfig config) {
110         // Nothing special, don't have any textures we need to recreate
                .
111     }
112
113     public void setAngle(float angle) {
114         mAngle = angle;
115     }
116
117     public float getAngle() {
118         return mAngle;
119     }
120
121     private GLWorld mWorld;
122     private AnimationCallback mCallback;
123     private float mAngle;
124 }
```

```
1  /*
2   * Copyright (C) 2008 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
        implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package com.test.togetherness;
18
19 public class Layer {
20
21  public Layer(int axis) {
22   // start with identity matrix for transformation
23   mAxis = axis;
24   mTransform.setIdentity();
25  }
26
27  public void startAnimation() {
28   for (int i = 0; i < mShapes.length; i++) {
29    GLShape shape = mShapes[i];
30    if (shape != null) {
31     shape.startAnimation();
32    }
33   }
34  }
35
36  public void endAnimation() {
37   for (int i = 0; i < mShapes.length; i++) {
38    GLShape shape = mShapes[i];
39    if (shape != null) {
40     shape.endAnimation();
41    }
42   }
43  }
44
```

```
45  public void setAngle(float angle) {
46   // normalize the angle
47   float twopi = (float)Math.PI *2f;
48   while (angle >= twopi) angle -= twopi;
49   while (angle < 0f) angle += twopi;
50 //  mAngle = angle;
51
52   float sin = (float)Math.sin(angle);
53   float cos = (float)Math.cos(angle);
54
55   float[][] m = mTransform.m;
56   switch (mAxis) {
57    case kAxisX:
58     m[1][1] = cos;
59     m[1][2] = sin;
60     m[2][1] = -sin;
61     m[2][2] = cos;
62     m[0][0] = 1f;
63     m[0][1] = m[0][2] = m[1][0] = m[2][0] = 0f;
64     break;
65    case kAxisY:
66     m[0][0] = cos;
67     m[0][2] = sin;
68     m[2][0] = -sin;
69     m[2][2] = cos;
70     m[1][1] = 1f;
71     m[0][1] = m[1][0] = m[1][2] = m[2][1] = 0f;
72     break;
73    case kAxisZ:
74     m[0][0] = cos;
75     m[0][1] = sin;
76     m[1][0] = -sin;
77     m[1][1] = cos;
78     m[2][2] = 1f;
79     m[2][0] = m[2][1] = m[0][2] = m[1][2] = 0f;
80     break;
81   }
82
83   for (int i = 0; i < mShapes.length; i++) {
84    GLShape shape = mShapes[i];
85    if (shape != null) {
86     shape.animateTransform(mTransform);
87    }
88   }
89  }
90
91  GLShape[] mShapes = new GLShape[9];
92  M4 mTransform = new M4();
```

```
93 // float mAngle;
94
95  // which axis do we rotate around?
96  // 0 for X, 1 for Y, 2 for Z
97  int mAxis;
98  static public final int kAxisX = 0;
99  static public final int kAxisY = 1;
100 static public final int kAxisZ = 2;
101 }
```

M4.java

```
1  /*
2   * Copyright (C) 2008 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package com.test.togetherness;
18
19 /**
20  *
21  * A 4x4 float matrix
22  *
23  */
24 public class M4 {
25  public float[][] m = new float[4][4];
26
27  public M4() {
28  }
29
30  public M4(M4 other) {
31   for (int i = 0; i < 4; i++) {
32    for (int j = 0; j < 4; j++) {
33     m[i][j] = other.m[i][j];
34    }
```

```
35    }
36   }
37
38   public void multiply(GLVertex src, GLVertex dest) {
39    dest.x = src.x * m[0][0] + src.y * m[1][0] + src.z * m[2][0] + m
         [3][0];
40    dest.y = src.x * m[0][1] + src.y * m[1][1] + src.z * m[2][1] + m
         [3][1];
41    dest.z = src.x * m[0][2] + src.y * m[1][2] + src.z * m[2][2] + m
         [3][2];
42   }
43
44   public M4 multiply(M4 other) {
45    M4 result = new M4();
46    float[][] m1 = m;
47    float[][] m2 = other.m;
48
49    for (int i = 0; i < 4; i++) {
50     for (int j = 0; j < 4; j++) {
51      result.m[i][j] = m1[i][0]*m2[0][j] + m1[i][1]*m2[1][j] + m1[i][2]*
           m2[2][j] + m1[i][3]*m2[3][j];
52     }
53    }
54
55    return result;
56   }
57
58   public void setIdentity() {
59    for (int i = 0; i < 4; i++) {
60     for (int j = 0; j < 4; j++) {
61      m[i][j] = (i == j ? 1f : 0f);
62     }
63    }
64   }
65
66   @Override
67   public String toString() {
68    StringBuilder builder = new StringBuilder("[ ");
69    for (int i = 0; i < 4; i++) {
70     for (int j = 0; j < 4; j++) {
71      builder.append(m[i][j]);
72      builder.append(" ");
73     }
74     if (i < 2)
75      builder.append("\n  ");
76    }
77    builder.append(" ]");
78    return builder.toString();
```

```
 }
}
```

Search.java

```
package com.test.togetherness;
import android.app.Activity;
import android.app.Service;
import android.content.Context;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
/**
 * Class Search implements the Two-Phase-Algorithm.
 */
public class Search {

 static int[] ax = new int[31]; // The axis of the move
 static int[] po = new int[31]; // The power of the move

 static int[] flip = new int[31]; // phase1 coordinates
 static int[] twist = new int[31];
 static int[] slice = new int[31];

 static int[] parity = new int[31]; // phase2 coordinates
 static int[] URFtoDLF = new int[31];
 static int[] FRtoBR = new int[31];
 static int[] URtoUL = new int[31];
 static int[] UBtoDF = new int[31];
 static int[] URtoDF = new int[31];

 static int[] minDistPhase1 = new int[31]; // IDA* distance do goal
    estimations
 static int[] minDistPhase2 = new int[31];

 //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
 // generate the solution string from the array data
 static String solutionToString(int length) {
  String s = "";
  for (int i = 0; i < length; i++) {
   switch (ax[i]) {
   case 0:
    s += "U";
    break;
   case 1:
    s += "R";
    break;
   case 2:
```

```
42      s += "F";
43      break;
44     case 3:
45      s += "D";
46      break;
47     case 4:
48      s += "L";
49      break;
50     case 5:
51      s += "B";
52      break;
53     }
54     switch (po[i]) {
55     case 1:
56      s += " ";
57      break;
58     case 2:
59      s += "2 ";
60      break;
61     case 3:
62      s += "' ";
63      break;
64
65     }
66    }
67    return s;
68   };
69
70   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
71   // generate the solution string from the array data including a
        separator between phase1 and phase2 moves
72   static String solutionToString(int length, int depthPhase1) {
73    String s = "";
74    for (int i = 0; i < length; i++) {
75     switch (ax[i]) {
76     case 0:
77      s += "U";
78      break;
79     case 1:
80      s += "R";
81      break;
82     case 2:
83      s += "F";
84      break;
85     case 3:
86      s += "D";
87      break;
88     case 4:
```

```
89      s += "L";
90      break;
91    case 5:
92      s += "B";
93      break;
94    }
95    switch (po[i]) {
96    case 1:
97      s += "␣";
98      break;
99    case 2:
100     s += "2␣";
101     break;
102   case 3:
103     s += "'␣";
104     break;
105
106   }
107   if (i == depthPhase1 - 1)
108     s += ".␣";
109  }
110  return s;
111 };
112
113 /**
114  * Computes the solver string for a given cube.
115  *
116  * @param facelets
117  *          is the cube definition string, see {@link Facelet} for the
         format.
118  *
119  * @param maxDepth
120  *          defines the maximal allowed maneuver length. For random
       cubes, a maxDepth of 21 usually will return a
121  *          solution in less than 0.5 seconds. With a maxDepth of 20
       it takes a few seconds on average to find a
122  *          solution, but it may take much longer for specific cubes.
123  *
124  *@param timeOut
125  *          defines the maximum computing time of the method in
       seconds. If it does not return with a solution, it returns with
126  *          an error code.
127  *
128  * @param useSeparator
129  *          determines if a " . " separates the phase1 and phase2
       parts of the solver string like in F' R B R L2 F .
130  *          U2 U D for example.<br>
131  * @return The solution string or an error code:<br>
```

```
132   *          Error 1: There is not exactly one facelet of each colour<br
          >
133   *          Error 2: Not all 12 edges exist exactly once<br>
134   *          Error 3: Flip error: One edge has to be flipped<br>
135   *          Error 4: Not all corners exist exactly once<br>
136   *          Error 5: Twist error: One corner has to be twisted<br>
137   *          Error 6: Parity error: Two corners or two edges have to be
          exchanged<br>
138   *          Error 7: No solution exists for the given maxDepth<br>
139   *          Error 8: Timeout, no solution within given time
140   */
141  public static String solution(String facelets, int maxDepth, long
        timeOut, boolean useSeparator) {
142   int s;
143
144   // ++++++++++++++++++++check for wrong input
          ++++++++++++++++++++++++++++
145   int[] count = new int[6];
146   try {
147    for (int i = 0; i < 54; i++)
148     count[Color.valueOf(facelets.substring(i, i + 1)).ordinal()]++;
149   } catch (Exception e) {
150    return "Error 1";
151   }
152   for (int i = 0; i < 6; i++)
153    if (count[i] != 9)
154     return "Error 1";
155
156   FaceCube fc = new FaceCube(facelets);
157   CubieCube cc = fc.toCubieCube();
158   if ((s = cc.verify()) != 0)
159    return "Error " + Math.abs(s);
160
161   // ++++++++++++++++++++++ initialization
          ++++++++++++++++++++++++++++++++
162   CoordCube c = new CoordCube(cc);
163
164   po[0] = 0;
165   ax[0] = 0;
166   flip[0] = c.flip;
167   twist[0] = c.twist;
168   parity[0] = c.parity;
169   slice[0] = c.FRtoBR / 24;
170   URFtoDLF[0] = c.URFtoDLF;
171   FRtoBR[0] = c.FRtoBR;
172   URtoUL[0] = c.URtoUL;
173   UBtoDF[0] = c.UBtoDF;
174
```

```
175   minDistPhase1[1] = 1;// else failure for depth=1, n=0
176   int mv = 0, n = 0;
177   boolean busy = false;
178   int depthPhase1 = 1;
179
180   long tStart = System.currentTimeMillis();
181
182   // ++++++++++++++++++ Main loop
        ++++++++++++++++++++++++++++++++++++++++
183   do {
184    do {
185     if ((depthPhase1 - n > minDistPhase1[n + 1]) && !busy) {
186
187      if (ax[n] == 0 || ax[n] == 3)// Initialize next move
188       ax[++n] = 1;
189      else
190       ax[++n] = 0;
191      po[n] = 1;
192     } else if (++po[n] > 3) {
193      do {// increment axis
194       if (++ax[n] > 5) {
195
196        //if (System.currentTimeMillis() - tStart > timeOut << 10)
197         //return "Error 8";
198
199        if (n == 0) {
200         if (depthPhase1 >= maxDepth)
201          return "Error 7";
202         else {
203          depthPhase1++;
204          ax[n] = 0;
205          po[n] = 1;
206          busy = false;
207          break;
208         }
209        } else {
210         n--;
211         busy = true;
212         break;
213        }
214
215       } else {
216        po[n] = 1;
217        busy = false;
218       }
219      } while (n != 0 && (ax[n - 1] == ax[n] || ax[n - 1] - 3 == ax[n]))
           ;
220     } else
```

```
221       busy = false;
222     } while (busy);
223 
224     // ++++++++++++ compute new coordinates and new minDistPhase1
          +++++++++
225     // if minDistPhase1 =0, the H subgroup is reached
226     mv = 3 * ax[n] + po[n] - 1;
227     flip[n + 1] = CoordCube.flipMove[flip[n]][mv];
228     twist[n + 1] = CoordCube.twistMove[twist[n]][mv];
229     slice[n + 1] = CoordCube.FRtoBR_Move[slice[n] * 24][mv] / 24;
230     minDistPhase1[n + 1] = Math.max(CoordCube.getPruning(CoordCube.
          Slice_Flip_Prun, CoordCube.N_SLICE1 * flip[n + 1]
231       + slice[n + 1]), CoordCube.getPruning(CoordCube.Slice_Twist_Prun,
            CoordCube.N_SLICE1 * twist[n + 1]
232       + slice[n + 1]));
233     // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
234 
235     if (minDistPhase1[n + 1] == 0 && n >= depthPhase1 - 5) {
236       minDistPhase1[n + 1] = 10;// instead of 10 any value >5 is possible
237       if (n == depthPhase1 - 1 && (s = totalDepth(depthPhase1, maxDepth))
            >= 0) {
238         if (s == depthPhase1
239           || (ax[depthPhase1 - 1] != ax[depthPhase1] && ax[depthPhase1 -
              1] != ax[depthPhase1] + 3))
240           return useSeparator ? solutionToString(s, depthPhase1) :
              solutionToString(s);
241       }
242 
243     }
244   } while (true);
245 }
246 
247 //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
248 // Apply phase2 of algorithm and return the combined phase1 and phase2
      depth. In phase2, only the moves
249 // U,D,R2,F2,L2 and B2 are allowed.
250 static int totalDepth(int depthPhase1, int maxDepth) {
251   int mv = 0, d1 = 0, d2 = 0;
252   int maxDepthPhase2 = Math.min(10, maxDepth - depthPhase1);// Allow
        only max 10 moves in phase2
253   for (int i = 0; i < depthPhase1; i++) {
254     mv = 3 * ax[i] + po[i] - 1;
255     URFtoDLF[i + 1] = CoordCube.URFtoDLF_Move[URFtoDLF[i]][mv];
256     FRtoBR[i + 1] = CoordCube.FRtoBR_Move[FRtoBR[i]][mv];
257     parity[i + 1] = CoordCube.parityMove[parity[i]][mv];
258   }
259 
260   if ((d1 = CoordCube.getPruning(CoordCube.Slice_URFtoDLF_Parity_Prun,
```

```
261       (CoordCube.N_SLICE2 * URFtoDLF[depthPhase1] + FRtoBR[depthPhase1])
              * 2 + parity[depthPhase1])) > maxDepthPhase2)
262     return -1;
263
264   for (int i = 0; i < depthPhase1; i++) {
265     mv = 3 * ax[i] + po[i] - 1;
266     URtoUL[i + 1] = CoordCube.URtoUL_Move[URtoUL[i]][mv];
267     UBtoDF[i + 1] = CoordCube.UBtoDF_Move[UBtoDF[i]][mv];
268   }
269   URtoDF[depthPhase1] = CoordCube.MergeURtoULandUBtoDF[URtoUL[
          depthPhase1]][UBtoDF[depthPhase1]];
270
271   if ((d2 = CoordCube.getPruning(CoordCube.Slice_URtoDF_Parity_Prun,
272       (CoordCube.N_SLICE2 * URtoDF[depthPhase1] + FRtoBR[depthPhase1]) *
              2 + parity[depthPhase1])) > maxDepthPhase2)
273     return -1;
274
275   if ((minDistPhase2[depthPhase1] = Math.max(d1, d2)) == 0)// already
          solved
276     return depthPhase1;
277
278   // now set up search
279
280   int depthPhase2 = 1;
281   int n = depthPhase1;
282   boolean busy = false;
283   po[depthPhase1] = 0;
284   ax[depthPhase1] = 0;
285   minDistPhase2[n + 1] = 1;// else failure for depthPhase2=1, n=0
286   // ++++++++++++++++++ end initialization
          ++++++++++++++++++++++++++++++++
287   do {
288     do {
289       if ((depthPhase1 + depthPhase2 - n > minDistPhase2[n + 1]) && !busy
              ) {
290
291         if (ax[n] == 0 || ax[n] == 3)// Initialize next move
292         {
293           ax[++n] = 1;
294           po[n] = 2;
295         } else {
296           ax[++n] = 0;
297           po[n] = 1;
298         }
299       } else if ((ax[n] == 0 || ax[n] == 3) ? (++po[n] > 3) : ((po[n] =
              po[n] + 2) > 3)) {
300         do {// increment axis
301           if (++ax[n] > 5) {
```

```
302         if (n == depthPhase1) {
303          if (depthPhase2 >= maxDepthPhase2)
304           return -1;
305          else {
306           depthPhase2++;
307           ax[n] = 0;
308           po[n] = 1;
309           busy = false;
310           break;
311          }
312         } else {
313          n--;
314          busy = true;
315          break;
316         }
317
318        } else {
319         if (ax[n] == 0 || ax[n] == 3)
320          po[n] = 1;
321         else
322          po[n] = 2;
323         busy = false;
324        }
325       } while (n != depthPhase1 && (ax[n - 1] == ax[n] || ax[n - 1] - 3
            == ax[n]));
326      } else
327       busy = false;
328     } while (busy);
329     // ++++++++++++ compute new coordinates and new minDist ++++++++++
330     mv = 3 * ax[n] + po[n] - 1;
331
332     URFtoDLF[n + 1] = CoordCube.URFtoDLF_Move[URFtoDLF[n]][mv];
333     FRtoBR[n + 1] = CoordCube.FRtoBR_Move[FRtoBR[n]][mv];
334     parity[n + 1] = CoordCube.parityMove[parity[n]][mv];
335     URtoDF[n + 1] = CoordCube.URtoDF_Move[URtoDF[n]][mv];
336
337     minDistPhase2[n + 1] = Math.max(CoordCube.getPruning(CoordCube.
          Slice_URtoDF_Parity_Prun, (CoordCube.N_SLICE2
338       * URtoDF[n + 1] + FRtoBR[n + 1])
339       * 2 + parity[n + 1]), CoordCube.getPruning(CoordCube.
            Slice_URFtoDLF_Parity_Prun, (CoordCube.N_SLICE2
340       * URFtoDLF[n + 1] + FRtoBR[n + 1])
341       * 2 + parity[n + 1]));
342     // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
343
344    } while (minDistPhase2[n + 1] != 0);
345    return depthPhase1 + depthPhase2;
346   }
```

```
347 }
```

Tools.java

```
1  package com.test.togetherness;
2
3  import java.util.Random;
4
5  public class Tools {
6
7   //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
8   // Check if the cube string s represents a solvable cube.
9   // 0: Cube is solvable
10  // -1: There is not exactly one facelet of each colour
11  // -2: Not all 12 edges exist exactly once
12  // -3: Flip error: One edge has to be flipped
13  // -4: Not all corners exist exactly once
14  // -5: Twist error: One corner has to be twisted
15  // -6: Parity error: Two corners or two edges have to be exchanged
16  //
17  /**
18   * Check if the cube definition string s represents a solvable cube.
19   *
20   * @param s is the cube definition string , see {@link Facelet}
21   * @return 0: Cube is solvable<br>
22   *         -1: There is not exactly one facelet of each colour<br>
23   *         -2: Not all 12 edges exist exactly once<br>
24   *         -3: Flip error: One edge has to be flipped<br>
25   *         -4: Not all 8 corners exist exactly once<br>
26   *         -5: Twist error: One corner has to be twisted<br>
27   *         -6: Parity error: Two corners or two edges have to be
         exchanged
28   */
29  public static int verify(String s) {
30   int[] count = new int[6];
31   try {
32    for (int i = 0; i < 54; i++)
33     count[Color.valueOf(s.substring(i, i + 1)).ordinal()]++;
34   } catch (Exception e) {
35    return -1;
36   }
37
38   for (int i = 0; i < 6; i++)
39    if (count[i] != 9)
40     return -1;
41
42   FaceCube fc = new FaceCube(s);
```

```
43    CubieCube cc = fc.toCubieCube();
44
45    return cc.verify();
46   }
47
48   /**
49    * Generates a random cube.
50    * @return A random cube in the string representation. Each cube of
          the cube space has the same probability.
51    */
52   public static String randomCube() {
53    CubieCube cc = new CubieCube();
54    Random gen = new Random();
55    cc.setFlip((short) gen.nextInt(CoordCube.N_FLIP));
56    cc.setTwist((short) gen.nextInt(CoordCube.N_TWIST));
57    do {
58     cc.setURFtoDLB(gen.nextInt(CoordCube.N_URFtoDLB));
59     cc.setURtoBR(gen.nextInt(CoordCube.N_URtoBR));
60    } while ((cc.edgeParity() ^ cc.cornerParity()) != 0);
61    FaceCube fc = cc.toFaceCube();
62    return fc.to_String();
63   }
64  }
```

References

[1] "Android." *EngineersGarage.* Accessed April 16, 2014. http://www.engineersgarage.com/articles/what-is-android-introduction?page=1

[2] "Android, the World's Most Popular Mobile Platform." *Android Developers.* Accessed April 10, 2014. http://developer.android.com/about/index.html

[3] "ADT Plugin." *Android Developers.* Accessed April 13, 2014. http://developer.android.com/tools/sdk/eclipse-adt.html

[4] Bishop, Aaron, Mike Sammartino, and Jarrett Scacchetti. "Rubik's Cube Solving App." Undergraduate thesis, Youngstown State University, 2012.

[5] "Displaying Graphics with OpenGL ES." *Android Developers.* Accessed April 13, 2014. http://developer.android.com/training/graphics/opengl/index.html

[6] "OpenGL Overview." *The Khronos Group.* Accessed April 13, 2014. http://www.opengl.org/about/

[7] Kociemba, Herbert. "The Two-Phase-Algorithm." *Solve Rubik's Cube with Cube Explorer.* Accessed November, 2013. http://kociemba.org/cube.htm

[8] "Learn About Java Technology." *Java.* Accessed April 11, 2014. http://www.java.com/en/about/

[9] McCracken, Harry. "Who's Winning, iOS or Android? All the Numbers, All in One Place: A comprehensive look at the competitive situation, from unit sales to profit margins." *Time.* Last modified April 16, 2013. http://techland.time.com/2013/04/16/ios-vs-android/

[10] Thakur, Anshul. "ARM (Advanced RISC Machines) Processors." *EngineersGarage.* Accessed April 16, 2014. http://www.engineersgarage.com/articles/arm-advanced-risc-machines-processors

[11] "What Is Java and Why Do I Need It?" *Java.* Accessed April 11, 2014. http://www.java.com/en/download/faq/whatis_java.xml

[12] Zassenhaus, Hans. "Rubik's cube: A toy, a galois tool, group theory for everybody." *Physica A* 114, no. 1-3 (1982): 629-637.

Author Biography

Morgan Mirtes grew up in Norwalk, Ohio, graduating from Norwalk St. Paul High School in 2010. At Ashland University, Morgan is majoring in Computer Science and Mathematics with a minor in Music. She is a member of computer science honor society Upsilon Pi Epsilon, mathematics honorary Pi Mu Epsilon, national honorary band fraternity Kappa Kappa Psi, Ashland University Honors Program, Alpha Lambda Delta, and was on the Dean's List all semesters while at Ashland University.

Upon graduation, Morgan plans to work as a programmer at National Interstate Insurance Company in Richfield, Ohio.