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.
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.
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 $T_1 = (1,2,3,4)(9,10,11,12)(17,18,19,20)(-1,-2,-3,-4)(-13,-14,-15,-16)$ 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.

\[
T_1 = (1, 2, 3, 4)(9, 10, 11, 12)(17, 18, 19, 20)(-1, -2, -3, -4)(-13, -14, -15, -16)
\]
\[
T_2 = (10, 13, 8, 19)(18, 21, 16, 3)(2, 5, 24, 11)(-8, -14, -18, -9)(-20, -2, -6, -21)
\]
\[
T_3 = (20, 11, 16, 7)(4, 19, 24, 15)(12, 3, 8, 23)(-15, -20, -12, -19)(-3, -8, -24, -7)
\]
\[
T_4 = (9, 14, 5, 18)(17, 22, 13, 2)(1, 6, 21, 10)(-5, -10, -6, -13)(-17, -22, -18, -1)
\]
\[
T_5 = (12, 15, 6, 17)(20, 23, 14, 1)(4, 7, 22, 9)(-7, -11, -17, -16)(-19, -23, -5, -4)
\]
\[
T_6 = (21, 22, 23, 24)(5, 6, 7, 8)(13, 14, 15, 16)(-21, -22, -23, -24)(-9, -10, -11, -12)
\]

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 explanations, it is important to note that \(M^2\) and \(M^2\) are interchangeable 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 \(M, M^2, \ldots, M^f\) 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 \(S = \{M^{(1)}, \ldots, M^{(6)}\}\), 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 $a \sim b$, 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 $\{T_1, \ldots, T_6\}$, 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 $a \sim b$ and $b \sim c$, then $a \sim c$.

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 permuted 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 ContentView 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 wherever 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

```java
package com.test.togetherness;

import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.StreamCorruptedException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import com.test.togetherness.KubeRenderer.AnimationCallback;
import com.test.togetherness.R;
import com.test.togetherness.Search;
import com.test.togetherness.CoordCube;
import com.test.togetherness.Tools;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.PixelFormat;
import android.graphics.drawable.Drawable;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.Gravity;
import android.viewInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.RelativeLayout.LayoutParams;

public class Kube extends Activity implements KubeRenderer.
```
AnimationCallback {
  private static Drawable[] COLORS = new Drawable[6];
  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, 22, 1, 0, 11, 10, 9,
    20, 19, 18
  };
  public static int[] faceID = {
    Cube.kTop, Cube.kTop, Cube.kTop, 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.kBottom, Cube.kBottom, Cube.kBottom, Cube.kBottom,
    Cube.kBack, Cube.kBack, Cube.kBack, Cube.kBack, Cube.kBack, Cube.kBack
  };
  int one = 0x10000;
  int half = 0x08000;
  GLColor red = new GLColor(one, 0, 0);
  GLColor green = new GLColor(0, one, 0);
  GLColor blue = new GLColor(0, 0, one);
  GLColor yellow = new GLColor(one, one, 0);
  GLColor orange = new GLColor(one, half, 0);
  GLColor white = new GLColor(one, one, one);
  GLColor black = new GLColor(0, 0, 0);

  public GLWorld makeGLWorld() {
    world = new GLWorld();
    // coordinates for our cubes
    float c0 = -1.0f;
    float c1 = -0.38f;
    float c2 = -0.32f;
    float c3 = 0.32f;
    float c4 = 0.38f;
    float c5 = 1.0f;

    // top back, left to right
    mCubes[0] = new Cube(world, c0, c4, c0, c1, c5, c1);
mCubes[1] = new Cube(world, c2, c4, c0, c3, c5, c1);
mCubes[2] = new Cube(world, c4, c4, c0, c5, c5, c1);
// top middle, left to right
mCubes[3] = new Cube(world, c0, c4, c2, c1, c5, c3);
mCubes[4] = new Cube(world, c2, c4, c2, c3, c5, c3);
mCubes[5] = new Cube(world, c4, c4, c2, c5, c5, c3);
// top front, left to right
mCubes[6] = new Cube(world, c0, c4, c4, c1, c5, c5);
mCubes[7] = new Cube(world, c2, c4, c4, c3, c5, c5);
mCubes[8] = new Cube(world, c4, c4, c4, c5, c5, c5);
// middle back, left to right
mCubes[9] = new Cube(world, c0, c2, c0, c1, c3, c1);
mCubes[10] = new Cube(world, c2, c2, c0, c3, c3, c1);
mCubes[11] = new Cube(world, c4, c2, c0, c5, c3, c1);
// middle middle, left to right
mCubes[12] = new Cube(world, c0, c2, c2, c1, c3, c3);
mCubes[13] = null;
mCubes[14] = new Cube(world, c4, c2, c2, c5, c3, c3);
// middle front, left to right
mCubes[15] = new Cube(world, c0, c2, c4, c1, c3, c1);
mCubes[16] = new Cube(world, c2, c2, c4, c3, c3, c1);
mCubes[17] = new Cube(world, c4, c2, c4, c5, c3, c1);
// bottom back, left to right
mCubes[18] = new Cube(world, c0, c0, c0, c1, c1, c1);
mCubes[19] = new Cube(world, c2, c0, c0, c3, c1, c1);
mCubes[20] = new Cube(world, c4, c0, c0, c5, c1, c1);
// bottom middle, left to right
mCubes[21] = new Cube(world, c0, c0, c2, c1, c1, c3);
mCubes[22] = new Cube(world, c2, c0, c2, c3, c1, c3);
mCubes[23] = new Cube(world, c4, c0, c2, c5, c1, c3);
// bottom front, left to right
mCubes[24] = new Cube(world, c0, c0, c4, c1, c1, c5);
mCubes[25] = new Cube(world, c2, c0, c4, c3, c1, c5);
mCubes[26] = new Cube(world, c4, c0, c4, c5, c1, c5);

// paint the sides
int i, j;

// set all faces black by default
for (i = 0; i < 27; i++) {
    Cube cube = mCubes[i];
    if (cube != null) {
        for (j = 0; j < 6; j++)
            cube.setFaceColor(j, black);
    }
}

// paint middle cubes
mcubes[4].setFaceColor(Cube.kTop, white);
mCubes[22].setFaceColor(Cube.kBottom, yellow);
mCubes[12].setFaceColor(Cube.kLeft, orange);
mCubes[14].setFaceColor(Cube.kRight, red);
mCubes[10].setFaceColor(Cube.kBack, blue);
mCubes[16].setFaceColor(Cube.kFront, green);

currentCube = genRandom();

for (i = 0; i < 27; i++)
    if (mCubes[i] != null)
        world.addShape(mCubes[i]);

// initialize our permutation to solved position
mPermutation = new int[27];
for (i = 0; i < mPermutation.length; i++)
    mPermutation[i] = i;

// initialize our permutation to given cube
for (int k = 0; k < 27; k++)
    Cube cubeA = mCubes[k];
    List<GLColor> cubeColors = new ArrayList<GLColor>();
    if (k != 13)
        for (int l = 0; l < cubeA.mFaceList.size(); l++)
            GLFace face = cubeA.mFaceList.get(1);
            cubeColors.add(face.getColor());
    
createLayers();
updateLayers();

world.generate();

return world;

// get the permutation based off of the current cube color
public int getPermutation(List<GLColor> cc){
    if (cc.contains(white) && cc.contains(blue) && cc.contains(orange))
        return 0;
    else if (cc.contains(white) && cc.contains(blue) && cc.contains(red))
        return 2;
    else if (cc.contains(white) && cc.contains(blue))
        return 1;
else if (cc.contains(white) && cc.contains(green) && cc.contains(orange)) return 6;
else if (cc.contains(white) && cc.contains(green) && cc.contains(red)) return 8;
else if (cc.contains(white) && cc.contains(green)) return 7;
else if (cc.contains(white) && cc.contains(orange)) return 3;
else if (cc.contains(white)) return 4;
else if (cc.contains(yellow) && cc.contains(blue) && cc.contains(orange)) return 18;
else if (cc.contains(yellow) && cc.contains(blue) && cc.contains(red)) return 20;
else if (cc.contains(yellow) && cc.contains(blue)) return 19;
else if (cc.contains(yellow) && cc.contains(green) && cc.contains(orange)) return 24;
else if (cc.contains(yellow) && cc.contains(green) && cc.contains(red)) return 26;
else if (cc.contains(yellow) && cc.contains(green)) return 25;
else if (cc.contains(yellow) && cc.contains(orange)) return 21;
else if (cc.contains(blue)) return 22;
else if (cc.contains(blue) && cc.contains(orange)) return 9;
else if (cc.contains(blue) && cc.contains(red)) return 11;
else if (cc.contains(blue)) return 10;
else if (cc.contains(green) && cc.contains(orange)) return 15;
else if (cc.contains(green) && cc.contains(red)) return 17;
else if (cc.contains(green)) return 16;
else if (cc.contains(orange)) return 12;
else if (cc.contains(red)) return 14;
else return 13;
}

// // RANDOM CUBE// //
public String genRandom() {

// Call Random function from package org.kociemba.twophase //
String r = Tools.randomCube();
System.out.println(r);

for (int i = 0; i < 54; i++) {
switch (r.charAt(i)) {
case 'U':
mCubes[cubeID[i]].setFaceColor(faceID[i], white);
break;
case 'R':
mCubes[cubeID[i]].setFaceColor(faceID[i], red);
break;
case 'F':
mCubes[cubeID[i]].setFaceColor(faceID[i], green);
break;

case 'D':
mCubes[cubeID[i]].setFaceColor(faceID[i], yellow);
break;

case 'L':
mCubes[cubeID[i]].setFaceColor(faceID[i], orange);
break;

case 'B':
mCubes[cubeID[i]].setFaceColor(faceID[i], blue);
break;
}
    }
    currentCube = r;

    for (int i = 0; i < 27; i++)
        if (mCubes[i] != null)
            world.addShape(mCubes[i]);

    // initialize our permutation to solved position
    mPermutation = new int[27];
    for (int i = 0; i < mPermutation.length; i++){
        mPermutation[i] = i;
    }

    // initialize our permutation to given cube
    for (int k = 0; k < 27; k++){
        Cube cubeA = mCubes[k];
        List<GLColor> cubeColors = new ArrayList<GLColor>();
        if (k != 13){
            for (int l = 0; l < cubeA.mFaceList.size(); l++){
                GLFace face = cubeA.mFaceList.get(l);
                cubeColors.add(face.getColor());
            }
        }
    }

    createLayers();
    updateLayers();
    world.generate();
    return r;

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
268 private void updateLayers() {
269    Layer layer;
270    GLShape[] shapes;
271    int i, j, k;
272
273    // up layer
274    layer = mLayers[kUp];
275    shapes = layer.mShapes;
276    for (i = 0; i < 9; i++)
277        shapes[i] = mCubes[mPermutation[i]];
278
279    // down layer
280    layer = mLayers[kDown];
281    shapes = layer.mShapes;
282    for (i = 18, k = 0; i < 27; i++)
283        shapes[k++] = mCubes[mPermutation[i]];
284
285    // left layer
286    layer = mLayers[kLeft];
287    shapes = layer.mShapes;
288    for (i = 0, k = 0; i < 27; i += 9)
289        for (j = 0; j < 9; j += 3)
290            shapes[k++] = mCubes[mPermutation[i + j]];
291
292    // right layer
293    layer = mLayers[kRight];
294    shapes = layer.mShapes;
295    for (i = 2, k = 0; i < 27; i += 9)
296        for (j = 0; j < 9; j += 3)
297            shapes[k++] = mCubes[mPermutation[i + j]];
298
299    // front layer
300    layer = mLayers[kFront];
301    shapes = layer.mShapes;
302    for (i = 6, k = 0; i < 27; i += 9)
303        for (j = 0; j < 3; j++)
304            shapes[k++] = mCubes[mPermutation[i + j]];
305
306 }
layer = mLayers[kBack];
shapes = layer.mShapes;
for (i = 0, k = 0; i < 27; i += 9)
    for (j = 0; j < 3; j++)
        shapes[k++] = mCubes[mPermutation[i + j]];

// middle layer
layer = mLayers[kMiddle];
shapes = layer.mShapes;
for (i = 1, k = 0; i < 27; i += 9)
    for (j = 0; j < 9; j += 3)
        shapes[k++] = mCubes[mPermutation[i + j]];

// equator layer
layer = mLayers[kEquator];
shapes = layer.mShapes;
for (i = 9, k = 0; i < 18; i++)
    shapes[k++] = mCubes[mPermutation[i]];

// side layer
layer = mLayers[kSide];
shapes = layer.mShapes;
for (i = 3, k = 0; i < 27; i += 9)
    for (j = 0; j < 3; j++)
        shapes[k++] = mCubes[mPermutation[i + j]];

@Override
protected void onResume() {
    super.onResume();
mView.onResume();
}

@Override
protected void onPause() {
    super.onPause();
mView.onPause();
}

public void animate() {
    // change our angle of view
    mRenderer.setAngle(mRenderer.getAngle() + 1.2f);
    if (cubeGenerated){
if (!cubeSolved) {
    if (mCurrentLayer == null) {// must set a new layer
        if (!continuous)// if they haven’t pushed the continuous
            button, wait for next to be pushed
            while (!next){
                next=false;
            }
    }
    int layerID = getLayerIDNumber(moves[moveCount]);
    mCurrentLayer = mLayers[layerID];
    mCurrentLayerPermutation = mLayerPermutations[layerID];
    mCurrentLayer.startAnimation();
    direction = getDirection(moves[moveCount]);
    if (moves[moveCount].contains("2")) count = 2;
    else count = 1;
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Button currentButton = (Button) findViewById(R.id.
                currentMove);
            if (moveCount < moves.length) currentButton.setText(moves[
                moveCount]);
        }
    });
    moveCount++;
    mCurrentAngle = 0;
    if (direction) {
        mAngleIncrement = (float)Math.PI / 50;
        mEndAngle = mCurrentAngle + ((float)Math.PI *
            count) / 2f;
    } else {
        mAngleIncrement = -(float)Math.PI / 50;
        mEndAngle = mCurrentAngle - ((float)Math.PI *
            count) / 2f;
    }
}

mCurrentAngle += mAngleIncrement;
if ((mAngleIncrement > 0f && mCurrentAngle >= mEndAngle) ||
    (mAngleIncrement < 0f && mCurrentAngle <= mEndAngle)
) {
    mCurrentLayer.setAngle(mEndAngle);
    mCurrentLayer.endAnimation();
    mCurrentLayer = null;

    // adjust mPermutation based on the completed layer
rotation
int[] newPermutation = new int[27];
for (int i = 0; i < 27; i++) {
    newPermutation[i] = mPermutation[mCurrentLayerPermutation[i]];
}
mPermutation = newPermutation;

if (count==2) {//must change the permutation twice
    newPermutation = new int[27];
    for (int i = 0; i < 27; i++) {
        newPermutation[i] = mPermutation[mCurrentLayerPermutation[i]];
    }
    mPermutation = newPermutation;
}

if (direction){//must change the permutation three times (the same as moving it the opposite direction)
    newPermutation = new int[27];
    for (int i = 0; i < 27; i++) {
        newPermutation[i] = mPermutation[mCurrentLayerPermutation[i]];
    }
    mPermutation = newPermutation;
    newPermutation = new int[27];
    for (int i = 0; i < 27; i++) {
        newPermutation[i] = mPermutation[mCurrentLayerPermutation[i]];
    }
    mPermutation = newPermutation;
}
updateLayers();
if (moveCount >= moves.length) cubeSolved=true;

} else {
    mCurrentLayer.setAngle(mCurrentAngle);
}

} else
runOnUiThread(new Runnable() {
    @Override
    public void run() {
        Button currentButton = (Button) findViewById(R.id.currentMove);
        currentButton.setText(""");
        currentButton.setBackgroundResource(R.drawable.done);
    }
});
```java
public void onClick(View view){
    Intent i = getBaseContext().getPackageManager().
getLaunchIntentForPackage(getBaseContext().
getPackageName());
i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(i);
}
}
}
}
}

currentButton.setOnClickListener(new OnClickListener(){
    @Override
    public void onClick(View view){
        GLSurfaceView mView;
        GLSurfaceView mView2;
        public static KubeRenderer mRenderer;
        Cube[] mCubes = new Cube[27];
        // a Layer for each possible move
        Layer[] mLayers = new Layer[9];
        // permutations corresponding to a pi/2 rotation of each layer
        // about its axis
        static int[][] mLayerPermutations = {
            // permutation for UP layer
            { 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 },
            // permutation for DOWN layer
            { 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 },
            // permutation for LEFT layer
            { 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 },
            // permutation for RIGHT layer
            { 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, 26 },
            // permutation for FRONT layer
            { 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 },
            // permutation for BACK layer
            { 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 },
            // permutation for MIDDLE layer
            { 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 },
            // permutation for EQUATOR layer
            { 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 },
        static int[][] mLayerPermutations = {
```
// permutation for SIDE layer
{ 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 }

// current permutation of (solved) starting position
int[] mPermutation;

// solution string and split into moves
String result = "";
String[] moves;
int moveCount = 0;

// count for number of spins, either 1 or 2
int count;

// direction for spinning
boolean direction;

// if true, layers stop moving
boolean cubeSolved = false;

// if false, animate does nothing
boolean cubeGenerated = false;

// pauses between moves
boolean next = false;

// false if want to wait for next button, true if want continuous solve
boolean continuous = false;

// for random cube movements
Random mRandom = new Random(System.currentTimeMillis());

// currently turning layer
Layer mCurrentLayer = null;

// current and final angle for current Layer animation
float mCurrentAngle, mEndAngle;

// amount to increment angle
float mAngleIncrement;

int[] mCurrentLayerPermutation;

// names for our 9 layers (based on notation from http://www.
cubefreak.net/notation.html)
static final int kUp = 0;
static final int kDown = 1;
static final int kLeft = 2;
static final int kRight = 3;
static final int kFront = 4;
static final int kBack = 5;
static final int kMiddle = 6;
static final int kEquator = 7;
static final int kSide = 8;
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    PACKAGE_NAME = getApplicationContext().getPackageName();

    // for no title
    requestWindowFeature(Window.FEATURE_NO_TITLE);

    mView = new GLSurfaceView(getApplication());
    mRenderer = new KubeRenderer(makeGLWorld(), this);
    mView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
    //mView.getHolder().setFormat(PixelFormat.TRANSLUCENT);
    mView.setRenderer(mRenderer);
    //mView.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
    setContentView(mView);

    RelativeLayout rel = new RelativeLayout(this);
    View view;
    LayoutInflater inflater = (LayoutInflater) getApplicationContext().
        getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    view = inflater.inflate(R.layout.activity_kube, null);
    rel.addView(view);
    addContentView(rel, new ViewGroup.LayoutParams(ViewGroup.LayoutParams.FILL_PARENT, ViewGroup.LayoutParams.FILL_PARENT));

    // create solve button
    Button b = (Button) rel.findViewById(R.id.SolveCube);
    b.setBackgroundResource(R.drawable.solve);
    b.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
            showHideButtons();

            String cubeString = currentCube;

            if (showString) {
                System.out.println("Cube_Definition_String:" + cubeString);
            }

            if (!cubeString.equals("UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLBBBBBBBBBB")) {
                if (firstSolve)
                    getTables();

                // +++++++++++++++++++++++++++++ Call Search.solution method from
package org.kociemba.twophase

result = Search.solution(cubeString, maxDepth, maxTime, useSeparator);

} else {
    result = "Done";
}

// Replace the error messages with more meaningful ones in your language
System.out.println(result);

if (result.contains("Error")) {
    switch (result.charAt(result.length() - 1)) {
        case '1':
            result = "There are not exactly nine facelets of each color!";
            break;
        case '2':
            result = "Not all 12 edges exist exactly once!";
            break;
        case '3':
            result = "Flip error: One edge has to be flipped!";
            break;
        case '4':
            result = "Not all 8 corners exist exactly once!";
            break;
        case '5':
            result = "Twist error: One corner has to be twisted!";
            break;
        case '6':
            result = "Parity error: Two corners or two edges have to be exchanged!";
            break;
        case '7':
            result = "No solution exists for the given maximum move number!";
            break;
        case '8':
            result = "Timeout, no solution found within given maximum time!";
            break;
    } else {
        System.out.println(result);
        moves = result.split("\\s+");
        cubeGenerated = true;
    }
}
// create next move button
b = (Button) rel.findViewById(R.id.nextMove);
b.setBackgroundResource(R.drawable.next_move);
b.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        next=true;
    }
});

// create continuous button
b = (Button) rel.findViewById(R.id.Continuous);
b.setBackgroundResource(R.drawable.continuous);
b.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        continuous=true;
        next=true;
    }
});

// create reset button
b = (Button) rel.findViewById(R.id.Reset);
b.setBackgroundResource(R.drawable.reset);
b.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view){
        Intent i = getBaseContext().getPackageManager().getLaunchIntentForPackage(getBaseContext().getPackageName());
        i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(i);
    }
});

public int getLayerIDNumber(String moveString){
    if (moveString.contains("U")) return kUp;
    else if (moveString.contains("D")) return kDown;
    else if (moveString.contains("L")) return kLeft;
    else if (moveString.contains("R")) return kRight;
    else if (moveString.contains("F")) return kFront;
    else if (moveString.contains("B")) return kBack;
    else return -1;
}

public boolean getDirection(String moveString){

if (moveString.equals("U")) return true;
else if (moveString.equals("U'")) || moveString.equals("U2")) return false;
else if (moveString.equals("D")) || moveString.equals("D2")) return false;
else if (moveString.equals("D'")) return true;
else if (moveString.equals("L")) return true;
else if (moveString.equals("L'")) || moveString.equals("L2")) return false;
else if (moveString.equals("R")) || moveString.equals("R2")) return false;
else if (moveString.equals("R'")) return true;
else if (moveString.equals("F")) || moveString.equals("F2")) return false;
else if (moveString.equals("F'")) return true;
else if (moveString.equals("B")) return true;
else if (moveString.equals("B'")) || moveString.equals("B2")) return false;
else return false;
}

public void showHideButtons() {
  Button showButtons, hideButtons;
  hideButtons = (Button) findViewById(R.id.SolveCube);
  hideButtons.setVisibility(View.GONE);
  hideButtons = (Button) findViewById(R.id.Reset);
  hideButtons.setVisibility(View.GONE);
  showButtons = (Button) findViewById(R.id.currentMove);
  showButtons.setVisibility(View.VISIBLE);
  showButtons = (Button) findViewById(R.id.nextMove);
  showButtons.setVisibility(View.VISIBLE);
  showButtons = (Button) findViewById(R.id.Continuous);
  showButtons.setVisibility(View.VISIBLE);
}

public void getTables() {
  Button dialogButton = (Button) findViewById(R.id.Solving);
  dialogButton.setVisibility(View.VISIBLE);

  ObjectInputStream tm = null;
  InputStream ins1 = getResources().openRawResource(getResources().
    getIdentifier("twistmove","raw", Kube.PACKAGE_NAME));
  try {
    tm = new ObjectInputStream(ins1);
  } catch (StreamCorruptedException e1) {
    // 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().
600      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  }
706  // TODO Auto-generated catch block
707  e1.printStackTrace();
708  }
709  ObjectInputStream frbr = null;
710  InputStream ins3 = getResources().openRawResource(getResources().
711      getIdentifier("frtobrmov","raw", Kube.PACKAGE_NAME));
712  try {
713      frbr = new ObjectInputStream(ins3);
714  } catch (StreamCorruptedException e1) {
715      // TODO Auto-generated catch block
716  e1.printStackTrace();
717  }
718  // TODO Auto-generated catch block
719  e1.printStackTrace();
720  }
721  ObjectInputStream merge = null;
722  InputStream ins4 = getResources().openRawResource(getResources().
723      getIdentifier("mergeurtoulandubtolf","raw", Kube.PACKAGE_NAME));
724  try {
725      merge = new ObjectInputStream(ins4);
726  } catch (StreamCorruptedException e1) {
727      // TODO Auto-generated catch block
728  e1.printStackTrace();
729  }
730  // TODO Auto-generated catch block
731  e1.printStackTrace();
732  }
733  ObjectInputStream sfp = null;
734  InputStream ins5 = getResources().openRawResource(getResources().
735      getIdentifier("sliceflipprun","raw", Kube.PACKAGE_NAME));
736  try {
737      sfp = new ObjectInputStream(ins5);
738  } catch (StreamCorruptedException e1) {
739      // TODO Auto-generated catch block
740  e1.printStackTrace();
741  }
el.printStackTrace();

try {
  stp = new ObjectInputStream(ins6);
} catch (StreamCorruptedException e1) {
  // TODO Auto-generated catch block
  e1.printStackTrace();
}

ObjectInputStream srfdf = null;

try {
  srfdf = new ObjectInputStream(ins7);
} catch (StreamCorruptedException e1) {
  // TODO Auto-generated catch block
  e1.printStackTrace();
}

ObjectInputStream ubdf = null;

try {
  ubdf = new ObjectInputStream(ins9);
} catch (StreamCorruptedException e1) {
  // TODO Auto-generated catch block
  e1.printStackTrace();
}
ObjectInputStream urfdlf = null;
InputStream ins10 = getResources().openRawResource(getResources().
    getIdentifier("urftodlfmove","raw", Kube.PACKAGE_NAME));
try {
    urfdlf = new ObjectInputStream(ins10);
} catch (StreamCorruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
// TODO Auto-generated catch block
// TODO Auto-generated catch block
ObjectInputStream urdf = null;
InputStream ins11 = getResources().openRawResource(getResources().
    getIdentifier("urtodfmove","raw", Kube.PACKAGE_NAME));
try {
    urdf = new ObjectInputStream(ins11);
} catch (StreamCorruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
// TODO Auto-generated catch block
// TODO Auto-generated catch block
ObjectInputStream urul = null;
InputStream ins12 = getResources().openRawResource(getResources().
    getIdentifier("urtoulmove","raw", Kube.PACKAGE_NAME));
try {
    urul = new ObjectInputStream(ins12);
} catch (StreamCorruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
// TODO Auto-generated catch block
// TODO Auto-generated catch block
byte ubdfbyte[] = new byte[CoordCube.NMOVE*2];
ubdf.readFully(ubdfbyte);
short[] ubdfsshort = new short[ubdfbyte.length/2];
// to turn bytes to shorts as either big endian or little endian.
ByteBuffer.wrap(ubdfbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(ubdfshort);
CoordCube.UBtoDF_Move[i]=ubdfshort;
for (int i=0;i<CoordCube.N_URFtoDLF;i++){
    byte urfdlfbYTE[] = new byte[CoordCube.N_MOVE*2];
    urfdlf.readFully(urfdlfbYTE);
    short [] urfdlfshort = new short[urfdlfbYTE.length/2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(urfdlfbYTE).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(urfdlfshort);
    CoordCube.UBtoDF_Move[i]=urfdlfshort;
}
for (int i=0;i<CoordCube.N_UFFtoDLF;i++){
    byte urdfbyte[] = new byte[CoordCube.N_MOVE*2];
    urdf.readFully(urdfbyte);
    short [] urdfshort = new short[urdfbyte.length/2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(urdfbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(urdfshort);
    CoordCube.URtoDF_Move[i]=urdfshort;
}
for (int i=0;i<CoordCube.N_URtoDF;i++){
    byte urdfbyte[] = new byte[CoordCube.N_MOVE*2];
    urdf.readFully(urdfbyte);
    short [] urdfshort = new short[urdfbyte.length/2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(urdfbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(urdfshort);
    CoordCube.URtoDF_Move[i]=urdfshort;
}
for (int i=0;i<CoordCube.N_URtoUL;i++){
    byte urulbyte[] = new byte[CoordCube.N_MOVE*2];
    urul.readFully(urulbyte);
    short [] urulshort = new short[urulbyte.length/2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(urulbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(urulshort);
    CoordCube.URtoUL_Move[i]=urulshort;
}
for (int i=0;i<CoordCube.N_FRtoBR;i++){
    byte frbrbyte[] = new byte[CoordCube.N_MOVE*2];
    frbr.readFully(frbrbyte);
    short [] frbrshort = new short[frbrbyte.length/2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(frbrbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(frbrshort);
    CoordCube.FRtoBR_Move[i]=frbrshort;
}
for (int i=0;i<CoordCube.N_TWIST;i++){
    byte tmbyte[] = new byte[CoordCube.N_MOVE*2];
    tm.readFully(tmbyte);
    short [] tmshort = new short[tmbyte.length/2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(tmbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(tmshort);
    CoordCube.twistMove[i]=tmshort;
for (int i = 0; i < CoordCube.N_FLIP; i++) {
    byte fmbyte[] = new byte[CoordCube.N_MOVE * 2];
    fm.readFully(fmbyte);
    short[] fmshort = new short[fmbyte.length / 2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(fmbyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(fmshort);
    CoordCube.flipMove[i] = fmshort;
}
for (int i = 0; i < 336; i++) {
    byte mergebyte[] = new byte[336 * 2];
    merge.readFully(mergebyte);
    short[] mergeshort = new short[mergebyte.length / 2];
    // to turn bytes to shorts as either big endian or little endian.
    ByteBuffer.wrap(mergebyte).order(ByteOrder.BIG_ENDIAN).asShortBuffer().get(mergeshort);
    CoordCube.MergeURtoULandUBtoDF[i] = mergeshort;
}
byte stparray[] = new byte[CoordCube.N_SLICE1 * CoordCube.N_TWIST / 2 + 1];
stp.readFully(stparray);
CoordCube.Slice_Twist_Prune = stparray;
byte sfarray[] = new byte[CoordCube.N_SLICE1 * CoordCube.N_FLIP / 2];
sf.readFully(sfarray);
CoordCube.Slice_Flip_Prune = sfarray;
byte surfdlfarray[] = new byte[CoordCube.N_SLICE2 * CoordCube.N_UFToDLF * CoordCube.N_PARITY / 2];
surfdlf.readFully(surfdlfarray);
CoordCube.Slice_UFToDLF_Parity_Prune = surfdlfarray;
byte surdfarray[] = new byte[CoordCube.N_SLICE2 * CoordCube.N_PARITY / 2];
surf.readFully(surdfarray);
CoordCube.Slice_UFToDF_Parity_Prune = surdfarray;
firstSolve = false;
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
try {
    tm.close();
    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.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
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;

import java.io.Reader;
import java.io.StreamCorruptedException;
import android.content.*;
import android.content.res.Resources;
import android.os.Environment;

class CoordCube {

    static final short N_TWIST = 2187; // 3^7 possible corner orientations
    static final short N_FLIP = 2048; // 2^11 possible edge flips
    static final short N_SLICE1 = 495; // 12 choose 4 possible positions of FR,FL,BL,BR edges
    static final short N_SLICE2 = 24; // 4! permutations of FR,FL,BL,BR edges in phase2
    static final short N_PARITY = 2; // 2 possible corner parities
    static final short N_URFtoDLF = 20160; // 8!(8−6)! permutation of URF, UFL,ULB,UBR,DFR,DLF corners
    static final short N_FRtoBR = 11880; // 12!(12−4)! permutation of FR, FL,BL,BR edges
    static final short N_URtoUL = 1320; // 12!(12−3)! permutation of UR, UF,UL edges
    static final short N_UBtoDF = 1320; // 12!(12−3)! permutation of UB, DR,DF edges
    static final short N_URtoDF = 20160; // 8!(8−6)! permutation of UR,UF,UL,UB,DR,DF edges in phase2

    static final int N_URFtoDLB = 40320; // 8! permutations of the corners
    static final int N_URtoBR = 479001600; // 8! permutations of the corners

    static final short N_MOVE = 18;
    static short [][] twistMove = new short [N_TWIST][N_MOVE];
    static short [][] flipMove = new short [N_FLIP][N_MOVE];
    static short [][] FRtoBR_Move = new short [N_FRtoBR][N_MOVE];
    static short [][] URFtoDLF_Move = new short [N_URFtoDLF][N_MOVE];
    static short [][] URFtoDLF_Move = new short [N_URtoDF][N_MOVE];
    static short [][] URFtoDLF_Move = new short [N_URtoDF][N_MOVE];
    static short [][] URFtoDLF_Move = new short [N_URtoDF][N_MOVE];
    static short [][] MergeURtoULandUBtoDF = new short [336][336];
    static byte [] Slice_URFtoDLF_Parity_Prun = new byte [N_SLICE2 * N_URFtoDLF * N_PARITY / 2];
    static byte [] Slice_URtoDF_Parity_Prun = new byte [N_SLICE2 * N_URtoDF * N_PARITY / 2];
static byte[] Slice_Twist_Prune = new byte[N_SLICE1 * N_TWIST / 2 + 1];
static byte[] Slice_Flip_Prune = new byte[N_SLICE1 * N_FLIP / 2];

// All coordinates are 0 for a solved cube except for UBtoDF, which is
short twist;
short flip;
short parity;
short FRtoBR;
short URFtoDLF;
short URtoUL;
short UptoDF;
int URtoDF;

// Generate a CoordCube from a CubieCube
CoordCube(CubieCube c) {
  twist = c.getTwist();
  flip = c.getFlip();
  parity = c.cornerParity();
  FRtoBR = c.FRtoBR();
  URFtoDLF = c.URFtoDLF();
  URtoUL = c.URtoUL();
  UptoDF = c.UptoDF();
  URtoDF = c.URtoDF(); // only needed in phase2
}

// A move on the coordinate level
// Generate a CoordCube from a CubieCube
void move(int m) {
  twist = twistMove[twist][m];
  flip = flipMove[flip][m];
  parity = parityMove[parity][m];
  FRtoBR = FRtoBR_Move[FRtoBR][m];
  URFtoDLF = URFtoDLF_Move[URFtoDLF][m];
  URtoUL = URtoUL_Move[URtoUL][m];
  UptoDF = UptoDF_Move[UptoDF][m];
  if (URtoUL < 336 && UptoDF < 336) // updated only if UR,UF,UL,UB,DR,DF
    // are not in UD-slice
    URtoDF = MergeURtoULandUptoDF[URtoUL][UptoDF];
}

// Parity of the corner permutation. This is the same as the parity
// for the edge permutation of a valid cube.
// parity has values 0 and 1
static short[][][] parityMove = {{ 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
  1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1 },

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);
}

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(right) and a Front 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.
 */
package com.test.togetherness;

public class Cube extends GLShape {

    public Cube(GLWorld world, float left, float bottom, float back, float right, float top, float front) {
        super(world);
        GLVertex leftBottomBack = addVertex(left, bottom, back);
        GLVertex rightBottomBack = addVertex(right, bottom, back);
        GLVertex leftTopBack = addVertex(left, top, back);
        GLVertex rightTopBack = addVertex(right, top, back);
        GLVertex leftBottomFront = addVertex(left, bottom, front);
        GLVertex rightBottomFront = addVertex(right, bottom, front);
        GLVertex leftTopFront = addVertex(left, top, front);
        GLVertex rightTopFront = addVertex(right, top, front);

        // vertices are added in a clockwise orientation (when viewed from the outside)
        // bottom
        addFace(new GLFace(leftBottomBack, leftBottomFront, rightBottomFront, rightBottomBack));
        // front
        addFace(new GLFace(leftBottomFront, leftTopFront, rightTopFront, rightBottomFront));
        // left
        addFace(new GLFace(leftBottomBack, leftTopBack, leftTopFront, leftBottomFront));
        // right
        addFace(new GLFace(rightBottomBack, rightBottomFront, rightTopFront, rightTopBack));
        // back
        addFace(new GLFace(leftBottomBack, rightBottomBack, rightTopBack, leftTopBack));
        // top
```java
addFace(new GLFace(leftTopBack, rightTopBack, rightTopFront, leftTopFront));

public static final int kBottom = 0;
public static final int kFront = 1;
public static final int kLeft = 2;
public static final int kRight = 3;
public static final int kBack = 4;
public static final int kTop = 5;
```
private static Edge[] epU = {UB, UR, UF, UL, DR, DF, DL, DB, FR, FL, BL, BR};
private static byte[] eoU = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
private static Corner[] cpR = {DFR, UFL, ULB, URF, DRB, DLF, DBL, UBR};
private static byte[] coR = {2, 0, 0, 1, 1, 0, 0, 0, 2};
private static Edge[] epR = {FR, UF, UL, UB, BR, DF, DL, DB, DR, FL, BL, UR};
private static byte[] eoR = {0, 0, 0, 0, 0, 0, 0, 0, 0};
private static Corner[] cpF = {UFL, DLF, ULB, UBR, URF, DFR, DBL, DRB};
private static byte[] coF = {1, 2, 0, 0, 2, 1, 0, 0};
private static Edge[] epF = {UR, FL, UL, UB, DR, FR, DL, DB, UF, DF, BL, BR};
private static byte[] eoF = {0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0};
private static Corner[] cpD = {URF, UFL, ULB, UBR, DLF, DLF, DBL, DFR};
private static byte[] coD = {0, 0, 0, 0, 0, 0, 0, 0, 0};
private static Edge[] epD = {UR, UF, UL, UB, DR, DF, DL, DB, DR, FL, BL, BR};
private static byte[] eoD = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
private static Corner[] cpL = {URF, UFL, ULB, UBR, DFR, UFL, DLF, DBL};
private static byte[] coL = {0, 1, 2, 0, 0, 2, 1, 0};
private static Edge[] epL = {UR, UF, BL, UB, DR, DF, FL, DB, FR, UL, DL, BR};
private static byte[] eoL = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
private static Corner[] cpB = {URF, UFL, UBR, DRB, DFR, DLF, ULB, DBL};
private static byte[] coB = {0, 0, 1, 2, 0, 0, 2, 1};
private static Edge[] epB = {UR, UF, UL, BR, DR, DF, DL, BL, FR, FL, DB, BR};
private static byte[] eoB = {0, 0, 0, 1, 0, 0, 0, 1, 1, 1};

// this CubieCube array represents the 6 basic cube moves
static CubieCube[] moveCube = new CubieCube[6];

static {
moveCube[0] = new CubieCube();
movcCube[0].cp = cpU;
movcCube[0].co = coU;
movcCube[0].ep = epU;
movcCube[0].co = eoU;
moveCube[1] = new CubieCube();
moveCube[1].cp = cpR;
moveCube[1].co = coR;
moveCube[1].ep = epR;
moveCube[1].eo = eoR;

moveCube[2] = new CubieCube();
moveCube[2].cp = cpF;
moveCube[2].co = coF;
moveCube[2].ep = epF;
moveCube[2].eo = eoF;

moveCube[3] = new CubieCube();
moveCube[3].cp = cpD;
moveCube[3].co = coD;
moveCube[3].ep = epD;
moveCube[3].eo = eoD;

moveCube[4] = new CubieCube();
moveCube[4].cp = cpL;
moveCube[4].co = coL;
moveCube[4].ep = epL;
moveCube[4].eo = eoL;

moveCube[5] = new CubieCube();
moveCube[5].cp = cpB;
moveCube[5].co = coB;
moveCube[5].ep = epB;
moveCube[5].eo = eoB;

CubieCube() {
};

CubieCube(Corner[] cp, byte[] co, Edge[] ep, byte[] eo) {
    this();
    for (int i = 0; i < 8; i++) {
        this.cp[i] = cp[i];
        this.co[i] = co[i];
    }
    for (int i = 0; i < 12; i++) {
        this.ep[i] = ep[i];
        this.eo[i] = eo[i];
    }
}
static int Cnk(int n, int k) {
    int i, j, s;
    if (n < k)
        return 0;
    if (k > n / 2)
        k = n - k;
    for (s = 1, i = n, j = 1; i != n - k; i--, j++) {
        s *= i;
        s /= j;
    }
    return s;
}

static void rotateLeft(Corner[] arr, int l, int r)
// Left rotation of all array elements between l and r
{
    Corner temp = arr[l];
    for (int i = l; i < r; i++)
        arr[i] = arr[i + 1];
    arr[r] = temp;
}

static void rotateRight(Corner[] arr, int l, int r)
// Right rotation of all array elements between l and r
{
    Corner temp = arr[r];
    for (int i = r; i > l; i--)
        arr[i] = arr[i - 1];
    arr[l] = temp;
}

static void rotateLeft(Edge[] arr, int l, int r)
// Left rotation of all array elements between l and r
{
    Edge temp = arr[l];
    for (int i = l; i < r; i++)
        arr[i] = arr[i + 1];
    arr[r] = temp;
}
static void rotateRight(Edge[] arr, int l, int r)
// Right rotation of all array elements between l and r
{
    Edge temp = arr[r];
    for (int i = r; i > l; i--)
        arr[i] = arr[i - 1];
    arr[l] = temp;
}

// return cube in facelet representation
FaceCube toFaceCube()
{
    FaceCube fcRet = new FaceCube();
    for (Corner c : Corner.values())
    {
        int i = c.ordinal();
        int j = cp[i].ordinal(); // cornercubie with index j is at
        // cornerposition with index i
        byte ori = co[i]; // Orientation of this cubie
        for (int n = 0; n < 3; n++)
            fcRet.f[FaceCube.cornerFacelet[i][((n + ori) % 3).ordinal()]] =
            FaceCube.cornerColor[j][n];
    }
    for (Edge e : Edge.values())
    {
        int i = e.ordinal();
        int j = ep[i].ordinal(); // edgecubie with index j is at edgeposition
        // with index i
        byte ori = eo[i]; // Orientation of this cubie
        for (int n = 0; n < 2; n++)
            fcRet.f[FaceCube.edgeFacelet[i][((n + ori) % 2).ordinal()]] =
            FaceCube.edgeColor[j][n];
        }
    return fcRet;
}

// Multiply this CubieCube with another cubiecube b, restricted to the corners.<br>
// Because we also describe reflections of the whole cube by
// permutations, we get a complication with the corners. The
// orientations of mirrored corners are described by the numbers 3, 4
// and 5. The composition of the orientations
// cannot
// be computed by addition modulo three in the cyclic group C3 any
// more. Instead the rules below give an addition in
// the dihedral group D3 with 6 elements.<br>
// NOTE: Because we do not use symmetry reductions and hence no
// mirrored cubes in this simple implementation of the
// Two-Phase-Algorithm, some code is not necessary here.

void cornerMultiply(CubieCube b) {
    Corner[] cPerm = new Corner[8];
    byte[] cOri = new byte[8];
    for (Corner corn : Corner.values()) {
        cPerm[corn.ordinal()] = cp[b.cp[corn.ordinal()].ordinal()];
        byte oriA = cOri[b.cp[corn.ordinal()].ordinal()];
        byte oriB = b.co[corn.ordinal()];
        byte ori = 0;
        if (oriA < 3 && oriB < 3) // if both cubes are regular cubes...
            ori = (byte) (oriA + oriB); // just do an addition modulo 3 here
        if (ori >= 3)
            ori -= 3; // the composition is a regular cube
        else if (oriA < 3 && oriB >= 3) // if cube b is in a mirrored state...
            ori = (byte) (oriA + oriB);
        if (ori >= 6)
            ori -= 3; // the composition is a mirrored cube
        else if (oriA >= 3 && oriB < 3) // if cube a is an a mirrored state...
            ori = (byte) (oriA - oriB);
        if (ori < 3)
            ori += 3; // the composition is a mirrored cube
        else if (oriA >= 3 && oriB >= 3) // if both cubes are in mirrored states...
            ori = (byte) (oriA - oriB);
        if (ori < 0)
            ori += 3; // the composition is a regular cube
        cOri[corn.ordinal()] = ori;
    }
    for (Corner c : Corner.values()) {
        cp[c.ordinal()] = cPerm[c.ordinal()];
        co[c.ordinal()] = cOri[c.ordinal()];
    }
}
 void edgeMultiply(CubieCube b) {
    Edge[] ePerm = new Edge[12];
    byte[] eOri = new byte[12];
    for (Edge edge : Edge.values()) {
        ePerm[edge.ordinal()] = ep[b.ep[edge.ordinal()].ordinal()];
        eOri[edge.ordinal()] = (byte)((b.eo[edge.ordinal()] + eo[b.ep[edge.ordinal()].ordinal()]) % 2);
    }
    for (Edge e : Edge.values()) {
        ep[e.ordinal()] = ePerm[e.ordinal()];
        eo[e.ordinal()] = eOri[e.ordinal()];
    }
}

 void multiply(CubieCube b) {
    cornerMultiply(b);
//    edgeMultiply(b);
}

 void invCubieCube(CubieCube c) {
    for (Edge edge : Edge.values())
        c.ep[ep[edge.ordinal()].ordinal()] = edge;
    for (Edge edge : Edge.values())
        c.eo[ep[edge.ordinal()].ordinal()] = eo[c.ep[edge.ordinal()].ordinal()];
    for (Corner corn : Corner.values())
        c.cp[cp[corn.ordinal()].ordinal()] = corn;
    for (Corner corn : Corner.values()) {
        byte ori = co[c.cp[corn.ordinal()].ordinal()];
        if (ori >= 3) // Just for completeness. We do not invert mirrored cubes in the program.
            c.co[corn.ordinal()] = ori;
        else { // the standard case
            c.co[corn.ordinal()] = (byte)-ori;
            if (c.co[corn.ordinal()] < 0)
                c.co[corn.ordinal()] += 3;
        }
    }
}

// ***************************************************************************** Get and set coordinates ***/
short getTwist() {
    short ret = 0;
    for (int i = URF.ordinal(); i < DRB.ordinal(); i++)
        ret = (short) (3 * ret + co[i]);
    return ret;
}

void setTwist(short twist) {
    int twistParity = 0;
    for (int i = DRB.ordinal() - 1; i >= URF.ordinal(); i--)
    {
        twistParity += co[i] = (byte) (twist % 3);
        twist /= 3;
    }
    co[DRB.ordinal()] = (byte) ((3 - twistParity % 3) % 3);
}

short getFlip() {
    short ret = 0;
    for (int i = UR.ordinal(); i < BR.ordinal(); i++)
        ret = (short) (2 * ret + eo[i]);
    return ret;
}

void setFlip(short flip) {
    int flipParity = 0;
    for (int i = BR.ordinal() - 1; i >= UR.ordinal(); i--)
    {
        flipParity += eo[i] = (byte) (flip % 2);
        flip /= 2;
    }
    eo[BR.ordinal()] = (byte) ((2 - flipParity % 2) % 2);
}

short cornerParity() {
    int s = 0;
    for (int i = DRB.ordinal(); i >= URF.ordinal() + 1; i--)
    {
        for (int j = i - 1; j >= URF.ordinal(); j--)
            if (cp[j].ordinal() > cp[i].ordinal())
                s++;
    }
    return (short) (s % 2);
Parity of the edges permutation. Parity of corners and edges are the same if the cube is solvable.

```java
short edgeParity() {
    int s = 0;
    for (int i = BR.ordinal(); i >= UR.ordinal() + 1; i--)
        for (int j = i - 1; j >= UR.ordinal(); j--)
            if (ep[j].ordinal() > ep[i].ordinal())
                s++;
    return (short) (s % 2);
}
```

Permutation of the UD-slice edges FR, FL, BL, and BR

```java
short getFRtoBR() {
    int a = 0, x = 0;
    Edge[] edge4 = new Edge[4];
    // compute the index a < (12 choose 4) and the permutation array perm.
    for (int j = BR.ordinal(); j >= UR.ordinal(); j--)
        if (FR.ordinal() <= ep[j].ordinal() && ep[j].ordinal() <= BR.ordinal())
            a += Cnk(11 - j, x + 1);
    edge4[3 - x++] = ep[j];

    int b = 0;
    for (int j = 3; j > 0; j--)
        // compute the index b < 4! for the permutation in perm
    {
        int k = 0;
        while (edge4[j].ordinal() != j + 8) {
            rotateLeft(edge4, 0, j);
            k++;
        }
        b = (j + 1) * b + k;
    }
    return (short) (24 * a + b);
}
```

Permutation

```java
void setFRtoBR(short idx) {
    int x;
    Edge[] sliceEdge = { FR, FL, BL, BR };  
    Edge[] otherEdge = { UR, UF, UL, UB, DR, DF, DL, DB };  
    int b = idx % 24;  // Permutation
```
```java
int a = idx / 24; // Combination
for (Edge e : Edge.values())
    ep[e.ordinal()] = DB; // Use UR to invalidate all edges
for (int j = 1, k; j < 4; j++) // generate permutation from index b
{
    k = b % (j + 1);
    b /= j + 1;
    while (k-- > 0)
        rotateRight(sliceEdge, 0, j);
}
x = 3; // generate combination and set slice edges
for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
    if (a - Cnk(11 - j, x + 1) >= 0) {
        ep[j] = sliceEdge[3 - x];
        a -= Cnk(11 - j, x--; + 1);
    }
x = 0; // set the remaining edges UR..DB
for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
    if (ep[j] == DB)
        ep[j] = otherEdge[x++];
}

// Permutation of all corners except DBL and DRB
short getURFtoDLF() {
    int a = 0, x = 0;
    Corner[] corner6 = new Corner[6];
    // compute the index a < (8 choose 6) and the corner permutation.
    for (int j = URF.ordinal(); j <= DRB.ordinal(); j++)
        if (cp[j].ordinal() <= DLF.ordinal()) {
            a += Cnk(j, x + 1);
            corner6[x++] = cp[j];
        }
    int b = 0;
    for (int j = 5; j > 0; j--) // compute the index b < 6! for the
    { // permutation in corner6
        int k = 0;
        while (corner6[j].ordinal() != j) {
            rotateLeft(corner6, 0, j);
            k++;
        }
        b = (j + 1) * b + k;
    }
```
return (short) (720 * a + b);
}

void setURFtoDLF(short idx) {
    int x;
    Corner[] corner6 = {URF, UFL, ULB, UBR, DFR, DLF};
    Corner[] otherCorner = {DBL, DRB};
    int b = idx % 720; // Permutation
    int a = idx / 720; // Combination
    for (Corner c : Corner.values())
        cp[c.ordinal()] = DRB; // Use DRB to invalidate all corners
    for (int j = 1, k; j < 6; j++) // generate permutation from index b
    {
        k = b % (j + 1);
        b /= j + 1;
        while (k-- > 0)
        {
            rotateRight(corner6, 0, j);
        }
        x = 5; // generate combination and set corners
        for (int j = DRB.ordinal(); j >= 0; j--)
            if (a - Cnk(j, x + 1) >= 0) {
                cp[j] = corner6[x];
                a -= Cnk(j, x-- + 1);
            }
        x = 0;
    }
}

// Permutation of the six edges UR,UF,UL,UB,DR,DF.
int getURtoDF() {
    int a = 0, x = 0;
    Edge[] edge6 = new Edge[6];
    // compute the index a < (12 choose 6) and the edge permutation.
    for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
        if (cp[j].ordinal() <= DF.ordinal()) {
            a += Cnk(j, x + 1);
            edge6[x++] = ep[j];
        }
    int b = 0;
    for (int j = 5; j > 0; j--)
        if (cp[j].ordinal() <= DR.ordinal()) {
            a += Cnk(j, x + 1);
            edge6[x++] = ep[j];
        }
}

// permutation in edge6

int k = 0;
while (edge6[j].ordinal() != j) {
    rotateLeft(edge6, 0, j);
    k++;
}

b = (j + 1) * b + k;
}
return 720 * a + b;
}

void setURtoDF(int idx) {
    int x;
    Edge[] edge6 = { UR, UF, UL, UB, DR, DF };
    Edge[] otherEdge = { DL, DB, FR, FL, BL, BR };
    int b = idx % 720; // Permutation
    int a = idx / 720; // Combination
    for (Edge e : Edge.values())
        ep[e.ordinal()] = BR; // Use BR to invalidate all edges

    for (int j = 1, k; j < 6; j++) // generate permutation from index b
        {
        k = b % (j + 1);
        b /= j + 1;
        while (k-- > 0)
            rotateRight(edge6, 0, j);

        x = 5; // generate combination and set edges
        for (int j = BR.ordinal(); j >= 0; j--)
            if (a - Cnk(j, x + 1) >= 0)
                {
                ep[j] = edge6[x];
                a -= Cnk(j, x-- + 1);
                }

        x = 0; // set the remaining edges DL..BR
        for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
            if (ep[j] == BR)
                ep[j] = otherEdge[x++];

        for (int j = 0; j < 6; j++)
        {
        k = b % (j + 1);
        b /= j + 1;
        while (k-- > 0)
            rotateRight(edge6, 0, j);

        x = 5; // generate combination and set edges
        for (int j = BR.ordinal(); j >= 0; j--)
            if (a - Cnk(j, x + 1) >= 0)
                {
                ep[j] = edge6[x];
                a -= Cnk(j, x-- + 1);
                }

        x = 0; // set the remaining edges DL..BR
        for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
            if (ep[j] == BR)
                ep[j] = otherEdge[x++];

        // Permutation of the six edges UR,UF,UL,UB,DR,DF
        public static int getURtoDF(short idx1, short idx2) {
            CubieCube a = new CubieCube();
            CubieCube b = new CubieCube();
            a.setURtoUL(idx1);
            b.setUBtoDF(idx2);
            for (int i = 0; i < 8; i++)
                if (a.ep[i] != BR)
if (b.ep[i] != BR) // collision
    return -1;
else
    b.ep[i] = a.ep[i];
}
return b.getURtoDF();

// Permutation of the three edges UR, UF, UL
short getURtoUL() {
    int a = 0, x = 0;
    Edge[] edge3 = new Edge[3];
    // compute the index a < (12 choose 3) and the edge permutation.
    for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
        if (ep[j].ordinal() <= UL.ordinal()) {
            a += Cnk(j, x + 1);
            edge3[x++] = ep[j];
        }
    int b = 0;
    for (int j = 2; j > 0; j--) // compute the index b < 3! for the permutation in edge3
        {
            int k = 0;
            while (edge3[j].ordinal() != j) {
                rotateLeft(edge3, 0, j);
                k++;
            }
            b = (j + 1) * b + k;
        }
    return (short) (6 * a + b);
}

// Permutation of the three edges UR, UF, UL
void setURtoUL(short idx) {
    int x;
    Edge[] edge3 = {UR, UF, UL};
    int b = idx % 6; // Permutation
    int a = idx / 6; // Combination
    for (Edge e : Edge.values())
        ep[e.ordinal()] = BR; // Use BR to invalidate all edges
    for (int j = 1, k; j < 3; j++) // generate permutation from index b
        {
            k = b % (j + 1);
            b /= j + 1;
            while (k-- > 0)
rotateRight(edge3, 0, j);
}

x = 2; // generate combination and set edges
for (int j = BR.ordinal(); j >= 0; j--)
    if (a - Cnk(j, x + 1) >= 0) {
        ep[j] = edge3[x];
        a -= Cnk(j, x-- + 1);
    }
}

// Permutation of the three edges UB, DR, DF
short getUBtoDF() {
    int a = 0, x = 0;
    Edge[] edge3 = new Edge[3];
    // compute the index a < (12 choose 3) and the edge permutation.
    for (int j = UR.ordinal(); j <= BR.ordinal(); j++)
        if (UB.ordinal() <= ep[j].ordinal() && ep[j].ordinal() <= DF.ordinal()) {
            a += Cnk(j, x + 1);
            edge3[x++] = ep[j];
        }
    int b = 0;
    for (int j = 2; j > 0; j--)// compute the index b < 3! for the
        // permutation in edge3
        {
            int k = 0;
            while (edge3[j].ordinal() != UB.ordinal() + j) {
                rotateLeft(edge3, 0, j);
                k++;
            }
            b = (j + 1) * b + k;
        }
    return (short) (6 * a + b);
}

void setUBtoDF(short idx) {
    int x;
    Edge[] edge3 = { UB, DR, DF };
    int b = idx % 6; // Permutation
    int a = idx / 6; // Combination
    for (Edge e : Edge.values())
        ep[e.ordinal()] = BR; // Use BR to invalidate all edges
    for (int j = 1, k; j < 3; j++)// generate permutation from index b
        {
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
646   permutation in perm
647 {
648   int k = 0;
649   while (perm[j].ordinal() != j) {
650     rotateLeft(perm, 0, j);
651     k++;
652   }
653   b = (j + 1) * b + k;
654 }
655 return b;
656 }
657
658 //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
659 void setURFtoDLB(int idx) {
660 Corner[] perm = { URF, UFL, ULB, UBR, DFR, DLF, DBL, DRB };  
661 int k;
662 for (int j = 1; j < 8; j++) {
663   k = idx % (j + 1);
664   idx /= j + 1;
665   while (k-- > 0)
666     rotateRight(perm, 0, j);
667 }
668 int x = 7; // set corners
669 for (int j = 7; j >= 0; j--)
670   cp[j] = perm[x--;];
671 }
672 //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
```java
int getURtoBR() {
    Edge[] perm = new Edge[12];
    int b = 0;
    for (int i = 0; i < 12; i++)
        perm[i] = ep[i];
    for (int j = 11; j > 0; j--)/\ compute the index b < 12! for the
    permutation in perm
        {
            int k = 0;
            while (perm[j].ordinal() != j) {
                rotateLeft(perm, 0, j);
                k++;
            }
            b = (j + 1) * b + k;
        }
    return b;
}

void setURtoBR(int idx) {
    Edge[] perm = {UR, UF, UL, UB, DR, DF, DL, DB, FR, FL, BL, BR};
    int k;
    for (int j = 1; j < 12; j++) {
        k = idx % (j + 1);
        idx /= j + 1;
        while (k-- > 0)
            rotateRight(perm, 0, j);
    }
    int x = 11;  // set edges
    for (int j = 11; j >= 0; j--)
        ep[j] = perm[x--];
}

// Check a cubecube for solvability. Return the error code.
// 0: Cube is solvable
// -2: Not all 12 edges exist exactly once
// -3: Flip error: One edge has to be flipped
// -4: Not all corners exist exactly once
// -5: Twist error: One corner has to be twisted
// -6: Parity error: Two corners ore two edges have to be exchanged
int verify() {
    int sum = 0;
    int[] edgeCount = new int[12];
    for (Edge e : Edge.values())
        edgeCount[ep[e.ordinal()].ordinal()]++;
    for (int i = 0; i < 12; i++)
        if (edgeCount[i] != 1)
```

    return -2;

    for (int i = 0; i < 12; i++)
        sum += eo[i];
    if (sum % 2 != 0)
        return -3;

    int[] cornerCount = new int[8];
    for (Corner c : Corner.values())
        cornerCount[cp[c.ordinal()].ordinal()]++;
    for (int i = 0; i < 8; i++)
        if (cornerCount[i] != 1)
            return -4;// missing corners

    sum = 0;
    for (int i = 0; i < 8; i++)
        sum += co[i];
    if (sum % 3 != 0)
        return -5;// twisted corner

    if (!((edgeParity() ^ cornerParity()) != 0))
        return -6;// parity error

    return 0;// cube ok
}

package com.test.togetherness;

//+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + //
//Then names of the edge positions of the cube. Edge UR e.g., has an U(p) and R(ight) facelet.
enum Edge {
    UR, UF, UL, UB, DR, DF, DL, DB, FR, FL, BL, BR
}

package com.test.togetherness;

import static com.test.togetherness.Facelet.*;
import static com.test.togetherness.Color.*;
import static com.test.togetherness.Corner.*;

57
import static com.test.togetherness.Edge.*;

// Cube on the facelet level
class FaceCube {
    public Color[] f = new Color[54];

    // Map the corner positions to facelet positions. cornerFacelet[URF.ordinal()][0] e.g. gives the position of the facelet in the URF corner position, which defines the orientation.
    final static Facelet[][] cornerFacelet = {{U9, R1, F3}, {U7, F1, L3}, {D3, F9, R7}, {D1, L9, F7}, {D7, B9, L7}, {D9, R9, B7}};

    // Map the edge positions to facelet positions. edgeFacelet[UR.ordinal()][0] e.g. gives the position of the facelet in the UR edge position, which defines the orientation.
    final static Facelet[][] edgeFacelet = {{U6, R2}, {U8, F2}, {U4, L2}, {U2, B2}, {D6, R8}, {D4, L8}, {D8, B8}, {F6, R4}, {F4, L6}, {B6, L4}, {B4, R6}};

    // Map the corner positions to facelet colors.

    // Map the edge positions to facelet colors.

    FaceCube() {
        String s = "UUUUUURRRRRRRRRFFFFFFFDDDDDDDDDDLLLLLLBBBBB";
        for (int i = 0; i < 54; i++)
            f[i] = Color.valueOf(s.substring(i, i + 1));
    }
}
// Construct a facelet cube from a string
FaceCube(String cubeString) {
    for (int i = 0; i < cubeString.length(); i++)
        f[i] = Color.valueOf(cubeString.substring(i, i + 1));
}

// Gives string representation of a facelet cube
String toString() {
    String s = "";
    for (int i = 0; i < 54; i++)
        s += f[i].toString();
    return s;
}

// Gives CubieCube representation of a faceletcube
CubieCube toCubieCube() {
    byte ori;
    CubieCube ccRet = new CubieCube();
    for (int i = 0; i < 8; i++)
        ccRet.cp[i] = URF; // invalidate corners
    for (int i = 0; i < 12; i++)
        ccRet.ep[i] = UR; // and edges
    Color col1, col2;
    for (Corner i : Corner.values()) {
        // get the colors of the cubie at corner i, starting with U/D
        for (ori = 0; ori < 3; ori++)
            if (f[cornerFacelet[i.ordinal()]][ori.ordinal()] == U || f[cornerFacelet[i.ordinal()]][ori.ordinal()] == D)
                break;
        col1 = f[cornerFacelet[i.ordinal()]][(ori + 1) % 3].ordinal();
        col2 = f[cornerFacelet[i.ordinal()]][(ori + 2) % 3].ordinal();
        for (Corner j : Corner.values()) {
            if (col1 == cornerColor[j.ordinal()][1] && col2 == cornerColor[j.ordinal()][2]) {
                // in cornerposition i we have cornercubie j
                ccRet.cp[i.ordinal()] = j;
                ccRet.co[i.ordinal()] = (byte) (ori % 3);
                break;
            }
        }
    }
    for (Edge i : Edge.values())
for (Edge j : Edge.values()) {
    if (f[edgeFacelet[i.ordinal()][0].ordinal()] == edgeColor[j.ordinal()][0] && f[edgeFacelet[i.ordinal()][1].ordinal()] == edgeColor[j.ordinal()][1]) {
        ccRet.ep[i.ordinal()] = j;
        ccRet.ep[i.ordinal()] = 0;
        break;
    }
    if (f[edgeFacelet[i.ordinal()][0].ordinal()] == edgeColor[j.ordinal()][1] && f[edgeFacelet[i.ordinal()][1].ordinal()] == edgeColor[j.ordinal()][0]) {
        ccRet.ep[i.ordinal()] = j;
        ccRet.ep[i.ordinal()] = 1;
        break;
    }
}
return ccRet;
};
A cube definition string "UBL..." means for example: In position U1 we have the U-color, in position U2 we have the 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, R3, R4, R5, R6, R7, R8, R9, F1, F2, F3, F4, F5, F6, F7, F8, F9, D1, D2, D3, D4, D5, D6, D7, D8, L1, L2, L3, L4, L5, L6, L7, L8, L9, B1, B2, B3, B4, B5, B6, B7, B8, B9 of the enum constants.

```
public enum Facelet {
    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, L1, L2, L3, L4,
    L5, L6, L7, L8, L9, B1, B2, B3, B4, B5, B6, B7, B8, B9
}
```

public final int alpha;

public GLColor(int red, int green, int blue, int alpha) {
    this.red = red;
    this.green = green;
    this.blue = blue;
    this.alpha = alpha;
}

public GLColor(int red, int green, int blue) {
    this.red = red;
    this.green = green;
    this.blue = blue;
    this.alpha = 0x10000;
}

@Override
public boolean equals(Object other) {
    if (other instanceof GLColor) {
        GLColor color = (GLColor)other;
        return (red == color.red &&
                green == color.green &&
                blue == color.blue &&
                alpha == color.alpha);
    }
    return false;
}
}
package com.test.togetherness;

import android.util.Log;
import java.nio.ShortBuffer;
import java.util.ArrayList;

public class GLFace {

    public GLFace() {
    }

    public GLFace(GLVertex v1, GLVertex v2, GLVertex v3) {
        addVertex(v1);
        addVertex(v2);
        addVertex(v3);
    }

    public GLFace(GLVertex v1, GLVertex v2, GLVertex v3, GLVertex v4) {
        addVertex(v1);
        addVertex(v2);
        addVertex(v3);
        addVertex(v4);
    }

    public void addVertex(GLVertex v) {
        mVertexList.add(v);
    }

    public void setColor(GLColor c) {
        int last = mVertexList.size() - 1;
        if (last < 2) {
            Log.e("GLFace", "not enough vertices in setColor()");
        } else {
            GLVertex vertex = mVertexList.get(last);

            // only need to do this if the color has never been set
            if (mColor == null) {
                while (vertex.color != null) {
                    mVertexList.add(0, vertex);
                    mVertexList.remove(last + 1);
                    vertex = mVertexList.get(last);
                }
            }
        }
    }
}
GLColor vertex.color = c;

mColor = c;

public GLColor getColor() {
    return this.mColor;
}

public int getIndexCount() {
    return (mVertexList.size() - 2) * 3;
}

public void putIndices(ShortBuffer buffer) {
    int last = mVertexList.size() - 1;
    GLVertex v0 = mVertexList.get(0);
    GLVertex vn = mVertexList.get(last);

    // push triangles into the buffer
    for (int i = 1; i < last; i++) {
        GLVertex v1 = mVertexList.get(i);
        buffer.put(v0.index);
        buffer.put(v1.index);
        buffer.put(vn.index);
        v0 = v1;
    }
}

private ArrayList<GLVertex> mVertexList = new ArrayList<GLVertex>();

private GLColor mColor;

GLShape.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
 */
package com.test.togetherness;

import java.nio.ShortBuffer;
import java.util.ArrayList;
import java.util.Iterator;

public class GLShape {
    public GLShape(GLWorld world) {
        mWorld = world;
    }
    public void addFace(GLFace face) {
        mFaceList.add(face);
    }
    public void setFaceColor(int face, GLColor color) {
        mFaceList.get(face).setColor(color);
    }
    public void putIndices(ShortBuffer buffer) {
        Iterator<GLFace> iter = mFaceList.iterator();
        while (iter.hasNext()) {
            GLFace face = iter.next();
            face.putIndices(buffer);
        }
    }
    public int getIndexCount() {
        int count = 0;
        Iterator<GLFace> iter = mFaceList.iterator();
        while (iter.hasNext()) {
            GLFace face = iter.next();
            count += face.getIndexCount();
        }
        return count;
    }
    public GLVertex addVertex(float x, float y, float z) {

// look for an existing GLVertex first
Iterator<GLVertex> iter = mVertexList.iterator();
while (iter.hasNext()) {
    GLVertex vertex = iter.next();
    if (vertex.x == x && vertex.y == y && vertex.z == z) {
        return vertex;
    }
}

// doesn’t exist, so create new vertex
GLVertex vertex = mWorld.addVertex(x, y, z);
mVertexList.add(vertex);
return vertex;

public void animateTransform(M4 transform) {
    mAnimateTransform = transform;
    if (mTransform != null)
        transform = mTransform.multiply(transform);
    Iterator<GLVertex> iter = mVertexList.iterator();
    while (iter.hasNext()) {
        GLVertex vertex = iter.next();
        mWorld.transformVertex(vertex, transform);
    }
}

public void startAnimation() {
}

public void endAnimation() {
    if (mTransform == null) {
        mTransform = new M4(mAnimateTransform);
    } else {
        mTransform = mTransform.multiply(mAnimateTransform);
    }
}

public M4 mTransform;
public M4 mAnimateTransform;
protected ArrayList<GLFace> mFaceList = new ArrayList<GLFace>();
protected ArrayList<GLVertex> mVertexList = new ArrayList<GLVertex>();
protected ArrayList<Integer> mIndexList = new ArrayList<Integer>(); // make more efficient?
protected GLWorld mWorld;
/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) {
if (other instanceof GLVertex) {
    GLVertex v = (GLVertex) other;
    return (x == v.x && y == v.y && z == v.z);
}
return false;

static public int toFixed(float x) {
    return (int)(x * 65536.0f);
}

public void put(IntBuffer vertexBuffer, IntBuffer colorBuffer) {
    vertexBuffer.put(toFixed(x));
    vertexBuffer.put(toFixed(y));
    vertexBuffer.put(toFixed(z));
    if (color == null) {
        colorBuffer.put(0);
        colorBuffer.put(0);
        colorBuffer.put(0);
    } else {
        colorBuffer.put(color.red);
        colorBuffer.put(color.green);
        colorBuffer.put(color.blue);
        colorBuffer.put(color.alpha);
    }
}

public void update(IntBuffer vertexBuffer, M4 transform) {
// skip to location of vertex in mVertex buffer
    vertexBuffer.position(index * 3);
    if (transform == null) {
        vertexBuffer.put(toFixed(x));
        vertexBuffer.put(toFixed(y));
        vertexBuffer.put(toFixed(z));
    } else {
        GLVertex temp = new GLVertex();
        transform.multiply(this, temp);
        vertexBuffer.put(toFixed(temp.x));
        vertexBuffer.put(toFixed(temp.y));
        vertexBuffer.put(toFixed(temp.z));
    }
}
package com.test.togetherness;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;
import java.nio.ShortBuffer;
import java.util.Iterator;
import java.util.ArrayList;

import javax.microedition.khronos.opengles.GL10;

public class GLWorld {

public void addShape(GLShape shape) {
    mShapeList.add(shape);
    mIndexCount += shape.getIndexCount();
}

public void generate() {
    ByteBuffer bb = ByteBuffer.allocateDirect(mVertexList.size() * 4 * 4);
    bb.order(ByteOrder.nativeOrder());
    mColorBuffer = bb.asIntBuffer();

    bb = ByteBuffer.allocateDirect(mVertexList.size() * 4 * 3);
    bb.order(ByteOrder.nativeOrder());
    mVertexBuffer = bb.asIntBuffer();

    bb = ByteBuffer.allocateDirect(mIndexCount*2);

bb.order(ByteOrder.nativeOrder());
mIndexBuffer = bb.asShortBuffer();

Iterator<GLVertex> iter2 = mVertexList.iterator();
while (iter2.hasNext()) {
   GLVertex vertex = iter2.next();
   vertex.put(mVertexBuffer, mColorBuffer);
}

Iterator<GLShape> iter3 = mShapeList.iterator();
while (iter3.hasNext()) {
   GLShape shape = iter3.next();
   shape.putIndices(mIndexBuffer);
}

public GLVertex addVertex(float x, float y, float z) {
   GLVertex vertex = new GLVertex(x, y, z, mVertexList.size());
   mVertexList.add(vertex);
   return vertex;
}

public void transformVertex(GLVertex vertex, M4 transform) {
   vertex.update(mVertexBuffer, transform);
}

int count = 0;
public void draw(GL10 gl) {
   mColorBuffer.position(0);
   mVertexBuffer.position(0);
   mIndexBuffer.position(0);
   gl.glFrontFace(GL10.GL_CW);
   gl.glShadeModel(GL10.GL_FLAT);
   gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
   gl.glColorPointer(4, GL10.GL_FIXED, 0, mColorBuffer);
   gl.glDrawElements(GL10.GL_TRIANGLES, mIndexCount, GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
   count++;
}

static public float toFloat(int x) {
   return x/65536.0f;
}

private ArrayList<GLShape> mShapeList = new ArrayList<GLShape>();
private ArrayList<GLVertex> mVertexList = new ArrayList<GLVertex>();
private int mIndexCount = 0;

private IntBuffer mVertexBuffer;
private IntBuffer mColorBuffer;
private ShortBuffer mIndexBuffer;

KubeRenderer.java

package com.test.togetherness;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.opengl.GLSurfaceView;

/**
 * Example of how to use OpenGL|ES in a custom view
 *
 */
class KubeRenderer implements GLSurfaceView.Renderer {

    // For controlling cube’s z–position, x and y angles and speeds (NEW)
    float angleX = 0;  // (NEW)
    float angleY = 0;  // (NEW)
    float speedX = 0;  // (NEW)
    float speedY = 0;  // (NEW)
    float z = -6.0f;  // (NEW)

    public interface AnimationCallback {

void animate();

public KubeRenderer(GLWorld world, AnimationCallback callback) {
    mWorld = world;
    mCallback = callback;
}

public void onDrawFrame(GL10 gl) {
    if (mCallback != null) {
        mCallback.animate();
    }

    /* Usually, the first thing one might want to do is to clear
       the screen. The most efficient way of doing this is to use
       glClear(). However we must make sure to set the scissor
       correctly first. The scissor is always specified in window
       coordinates:
       */
    gl.glClearColor(0.5f, 0.5f, 0.5f, 1);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

    /* Now we’re ready to draw some 3D object
    */
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0, 0, -3.0f);
    gl.glScalef(0.5f, 0.5f, 0.5f);
    gl.glRotatef(mAngle, 0, 1, 0);
    gl.glRotatef(mAngle*0.25f, 1, 0, 0);
    gl.glColor4f(0.7f, 0.7f, 0.7f, 1.0f);
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
    gl.glEnable(GL10.GL_CULL_FACE);
    gl.glEnable(GL10.GL_SHADE_MODEL);
    gl.glDepthTest();
    mWorld.draw(gl);
public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);
    /* Set our projection matrix. This doesn't have to be done
     * each time we draw, but usually a new projection needs to be
     * set when the viewport is resized.
     */
    float ratio = (float)width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-ratio, ratio, -1, 1, 2, 12);
    /* By default, OpenGL enables features that improve quality
     * but reduce performance. One might want to tweak that
     * especially on software renderer.
     */
    gl.glDisable(GL10.GL_DITHER);
    gl.glActiveTexture(GL10.GL_TEXTURE0);
}

public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    // Nothing special, don’t have any textures we need to recreate.
}

public void setAngle(float angle) {
    mAngle = angle;
}

public float getAngle() {
    return mAngle;
}

private GLWorld mWorld;
private AnimationCallback mCallback;
private float mAngle;
package com.test.togetherness;

public class Layer {

    public Layer(int axis) {
        // start with identity matrix for transformation
        mAxis = axis;
        mTransform.IDENTITY();
    }

    public void startAnimation() {
        for (int i = 0; i < mShapes.length; i++) {
            GLShape shape = mShapes[i];
            if (shape != null) {
                shape.startAnimation();
            }
        }
    }

    public void endAnimation() {
        for (int i = 0; i < mShapes.length; i++) {
            GLShape shape = mShapes[i];
            if (shape != null) {
                shape.endAnimation();
            }
        }
    }
}
public void setAngle(float angle) {
    // normalize the angle
    float twopi = (float)Math.PI * 2f;
    while (angle >= twopi) angle -= twopi;
    while (angle < 0f) angle += twopi;
    // mAngle = angle;

    float sin = (float)Math.sin(angle);
    float cos = (float)Math.cos(angle);

    float[][] m = mTransform.m;
    switch (mAxis) {
        case kAxisX:
            m[1][1] = cos;
            m[1][2] = sin;
            m[2][1] = -sin;
            m[2][2] = cos;
            m[0][0] = 1f;
            m[0][1] = m[0][2] = m[1][0] = m[2][0] = 0f;
            break;
        case kAxisY:
            m[0][0] = cos;
            m[0][2] = sin;
            m[2][0] = -sin;
            m[2][2] = cos;
            m[1][1] = 1f;
            m[0][1] = m[1][0] = m[1][2] = m[2][1] = 0f;
            break;
        case kAxisZ:
            m[0][0] = cos;
            m[0][1] = sin;
            m[1][0] = -sin;
            m[1][1] = cos;
            m[2][2] = 1f;
            m[2][0] = m[2][1] = m[0][2] = m[1][2] = 0f;
            break;
    }

    for (int i = 0; i < mShapes.length; i++) {
        GLShape shape = mShapes[i];
        if (shape != null) {
            shape.animateTransform(mTransform);
        }
    }

    GLShape[] mShapes = new GLShape[9];
    M4 mTransform = new M4();
// float mAngle;

// which axis do we rotate around?
// 0 for X, 1 for Y, 2 for Z
int mAxis;

static public final int kAxisX = 0;
static public final int kAxisY = 1;
static public final int kAxisZ = 2;

M4.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;

/**
 * A 4x4 float matrix
 *
 */
public class M4 {
    public float[][] m = new float[4][4];

    public M4() {
    }

    public M4(M4 other) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                m[i][j] = other.m[i][j];
            }
        }
    }
public void multiply(GLVertex src, GLVertex dest) {
    dest.x = src.x * m[0][0] + src.y * m[1][0] + src.z * m[2][0] + m[3][0];
    dest.y = src.x * m[0][1] + src.y * m[1][1] + src.z * m[2][1] + m[3][1];
    dest.z = src.x * m[0][2] + src.y * m[1][2] + src.z * m[2][2] + m[3][2];
}

public M4 multiply(M4 other) {
    M4 result = new M4();
    float [][] m1 = m;
    float [][] m2 = other.m;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            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];
        }
    }
    return result;
}

public void setIdentity() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            m[i][j] = (i == j ? 1f : 0f);
        }
    }
}

@Override
public String toString() {
    StringBuilder builder = new StringBuilder("[");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            builder.append(m[i][j]);
            builder.append(" ");
        }
        if (i < 2)
            builder.append("\n");
    }
    builder.append("]");
    return builder.toString();
}
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 to 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:

        }
    }
}
s += "F" ;
break ;
case 3 :
s += "D" ;
break ;
case 4 :
s += "L" ;
break ;
case 5 :
s += "B" ;
break ;
}
switch (po[i]) {
  case 1 :
s += " " ;
  break ;
  case 2 :
s += "2 " ;
  break ;
  case 3 :
s += " ' " ;
  break ;
  }
}
switch (po[i]) {
  case 1 :
s += " " ;
  break ;
  case 2 :
s += " " ;
  break ;
  case 3 :
s += " " ;
  break ;
  }
return s ;
};

// generate the solution string from the array data including a separator between phase1 and phase2 moves
static String solutionToString(int length, int depthPhase1) {
  String s = "" ;
  for (int i = 0 ; i < length ; i++) {
    switch (ax[i]) {
      case 0 :
s += "U" ;
      break ;
      case 1 :
s += "R" ;
      break ;
      case 2 :
s += "F" ;
      break ;
      case 3 :
s += "D" ;
      break ;
  case 4 :
s += "L";
break;
case 5:
s += "B";
break;
}
switch (po[i]) {
case 1:
s += "L";
break;
case 2:
s += "2U";
break;
case 3:
s += "U2";
break;
}
if (i == depthPhase1 - 1)
s += ".U";
}
return s;

/**
 * Computes the solver string for a given cube.
 * @param facelets
 * is the cube definition string, see {@link Facelet} for the format.
 * @param maxDepth
 * defines the maximal allowed maneuver length. For random cubes, a maxDepth of 21 usually will return a
 * solution in less than 0.5 seconds. With a maxDepth of 20
 * it takes a few seconds on average to find a
 * solution, but it may take much longer for specific cubes.
 * @param timeOut
 * defines the maximum computing time of the method in seconds. If it does not return with a solution, it returns with
 * an error code.
 * @param useSeparator
 * determines if a " . " separates the phase1 and phase2 parts of the solver string like in F' R B R L2 F .
 * @return The solution string or an error code:
 */
Error 1: There is not exactly one facelet of each colour
Error 2: Not all 12 edges exist exactly once
Error 3: Flip error: One edge has to be flipped
Error 4: Not all corners exist exactly once
Error 5: Twist error: One corner has to be twisted
Error 6: Parity error: Two corners or two edges have to be exchanged
Error 7: No solution exists for the given maxDepth
Error 8: Timeout, no solution within given time

public static String solution(String facelets, int maxDepth, long timeOut, boolean useSeparator) {
    int s;
    // +++++++++++++++++++check for wrong input +++++++++++++++++++
    int[] count = new int[6];
    try {
        for (int i = 0; i < 54; i++)
            count[Color.valueOf(facelets.substring(i, i + 1)).ordinal()]++;
    } catch (Exception e) {
        return "Error 1";
    }
    for (int i = 0; i < 6; i++)
        if (count[i] != 9)
            return "Error 1";
    FaceCube fc = new FaceCube(facelets);
    CubieCube cc = fc.toCubieCube();
    if ((s = cc.verify()) != 0)
        return "Error" + Math.abs(s);
    // ++++++++++++++++++ initialization ++++++++++++++++++
    CoordCube c = new CoordCube(cc);
    po[0] = 0;
    ax[0] = 0;
    flip[0] = c.flip;
    twist[0] = c.twist;
    parity[0] = c.parity;
    slice[0] = c.FRtoBR / 24;
    URFtoDLF[0] = c.URFtoDLF;
    FRtoBR[0] = c.FRtoBR;
    URtoUL[0] = c.URtoUL;
    UBtoDF[0] = c.UBtoDF;
minDistPhase1[1] = 1; // else failure for depth=1, n=0
int mv = 0, n = 0;
boolean busy = false;
int depthPhase1 = 1;

long tStart = System.currentTimeMillis();

// + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
// Main loop
// + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
do {
  do {
    if ((depthPhase1 - n > minDistPhase1[n + 1]) && !busy) {
      if (ax[n] == 0 || ax[n] == 3) // Initialize next move
        ax[++n] = 1;
      else
        ax[++n] = 0;
      po[n] = 1;
    } else if (++po[n] > 3) {
      do {// increment axis
        if (++ax[n] > 5) {
          // if (System.currentTimeMillis() − tStart > timeOut << 10)
          // return ”Error 8”;

          if (n == 0) {
            if (depthPhase1 >= maxDepth)
              return ”Error 7”;
            else {
              depthPhase1++;
              ax[n] = 0;
              po[n] = 1;
              busy = false;
              break;
            }
          } else {
            n--;
            busy = true;
            break;
          }
        } else {
          po[n] = 1;
          busy = false;
        }
      } while (n != 0 && (ax[n - 1] == ax[n] || ax[n - 1] - 3 == ax[n]))
    }
  } else {
    po[n] = 1;
    busy = false;
  }
} while (n != 0 && (ax[n - 1] == ax[n] || ax[n - 1] - 3 == ax[n]));

} else
```java
busy = false;
}
while (busy);

// ++++++++ compute new coordinates and new minDistPhase1 ++++++++

// if minDistPhase1 =0, the H subgroup is reached
mv = 3 * ax[n] + po[n] - 1;
flip[n + 1] = CoordCube.flipMove[flip[n]][mv];
twist[n + 1] = CoordCube.twistMove[twist[n]][mv];
minDistPhase1[n + 1] = Math.max(CoordCube.getPruning(CoordCube.Slice.Flip.CubePar, CoordCube.N.Slice1 * flip[n + 1]
+ slice[n + 1]), CoordCube.getPruning(CoordCube.Slice.Twist.CubePar, CoordCube.N.Slice1 * twist[n + 1]
+ slice[n + 1]));

// ++++++++++++++++++++

if (minDistPhase1[n + 1] == 0 && n >= depthPhase1 - 5) {
minDistPhase1[n + 1] = 10; // instead of 10 any value >5 is possible
if (n == depthPhase1 - 1 && (s = totalDepth(depthPhase1, maxDepth)) >= 0) {
    if (s == depthPhase1
        || (ax[depthPhase1 - 1] != ax[depthPhase1] && ax[depthPhase1 - 1] != ax[depthPhase1] + 3))
        return useSeparator ? solutionToString(s, depthPhase1) :
        solutionToString(s);
}
}

while (true);

// Apply phase2 of algorithm and return the combined phase1 and phase2 depth. In phase2, only the moves

// U,D,R2,F2,L2 and B2 are allowed.
static int totalDepth(int depthPhase1, int maxDepth) {
    int mv = 0, d1 = 0, d2 = 0;
    int maxDepthPhase2 = Math.min(10, maxDepth - depthPhase1); // Allow only max 10 moves in phase2
    for (int i = 0; i < depthPhase1; i++) {
        mv = 3 * ax[i] + po[i] - 1;
        URFtoDLF[i + 1] = CoordCube.URFtoDLF.Move[URFtoDLF[i]][mv];
        FRtoBR[i + 1] = CoordCube.FRtoBR.Move[FRtoBR[i]][mv];
        parity[i + 1] = CoordCube.parityMove[parity[i]][mv];
    }
    if ((d1 = CoordCube.getPruning(CoordCube.Slice.URFtoDLF.Parity.CubePar,
```
(CoordCube.N_SLICE2 * URFtoDLF[depthPhase1] + FRtoBR[depthPhase1])
    * 2 + parity[depthPhase1]) > maxDepthPhase2)

    return -1;

for (int i = 0; i < depthPhase1; i++) {
    mv = 3 * ax[i] + po[i] - 1;
    URtoUL[i + 1] = CoordCube.URtoUL.Move[URtoUL[i]][mv];
    UBtoDF[i + 1] = CoordCube.UBtoDF.Move[UBtoDF[i]][mv];
}

URtoDF[depthPhase1] = CoordCube.MergeURtoULandUBtoDF[URtoUL[depthPhase1]][UBtoDF[depthPhase1]];

if ((d2 = CoordCube.getPruning(CoordCube.Slice.URtoDF.Parity_Prun,
    (CoordCube.N_SLICE2 * URFtoDF[depthPhase1] + FRtoBR[depthPhase1]) *
    2 + parity[depthPhase1]) > maxDepthPhase2)
    return -1;

if ((minDistPhase2[depthPhase1] = Math.max(d1, d2)) == 0) // already solved
    return depthPhase1;

// now set up search

int depthPhase2 = 1;
int n = depthPhase1;
boolean busy = false;
po[depthPhase1] = 0;
ax[depthPhase1] = 0;
minDistPhase2[n + 1] = 1; // else failure for depthPhase2=1, n=0

// ++++++ end initialization

do {
    do {
        if (((depthPhase1 + depthPhase2 - n > minDistPhase2[n + 1]) && !busy
            )
            {
                if (ax[n] == 0 || ax[n] == 3) // Initialize next move
                    {
                        ax[++n] = 1;
                        po[n] = 2;
                    }
                    else {
                        ax[++n] = 0;
                        po[n] = 1;
                    }
                } else if ((ax[n] == 0 || ax[n] == 3) ? (++po[n] > 3) : ((po[n] =
                        po[n] + 2) > 3)) {
                    do {// increment axis
                        if (++ax[n] > 5) {
                            // do something
                        }
                    }
                }
            }
        }
    }
}
if (n == depthPhase1) {
    if (depthPhase2 >= maxDepthPhase2)
        return -1;
    else {
        depthPhase2++;
        ax[n] = 0;
        po[n] = 1;
        busy = false;
        break;
    }
} else {
    n--;
    busy = true;
    break;
}
else {
    if (ax[n] == 0 || ax[n] == 3)
        po[n] = 1;
    else
        po[n] = 2;
    busy = false;
}
while (n != depthPhase1 && (ax[n - 1] == ax[n] || ax[n - 1] - 3 == ax[n]));
} else
    busy = false;
while (busy);

// + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

mv = 3 * ax[n] + po[n] - 1;

URFtoDLF[n + 1] = CoordCube.URFtoDLF_Move[URFtoDLF[n]][mv];
FRtoBR[n + 1] = CoordCube.FRtoBR_Move[FRtoBR[n]][mv];
parity[n + 1] = CoordCube.parityMove[parity[n]][mv];
URtoDF[n + 1] = CoordCube.URtoDF_Move[URtoDF[n]][mv];

minDistPhase2[n + 1] = Math.max(CoordCube.getPruning(CoordCube.Slice.URtoDF_Parity_Prune, CoordCube.N_SLICE2
    * URtoDF[n + 1] + FRtoBR[n + 1])
    * 2 + parity[n + 1]), CoordCube.getPruning(CoordCube.Slice.URFtoDLF_Parity_Prune, CoordCube.N_SLICE2
    * URFtoDLF[n + 1] + FRtoBR[n + 1])
    * 2 + parity[n + 1]));

// + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

} while (minDistPhase2[n + 1] != 0);
return depthPhase1 + depthPhase2;
package com.test.togetherness;

import java.util.Random;

public class Tools {

  // Check if the cube string s represents a solvable cube.
  // 0: Cube is solvable
  // -1: There is not exactly one facelet of each colour
  // -2: Not all 12 edges exist exactly once
  // -3: Flip error: One edge has to be flipped
  // -4: Not all corners exist exactly once
  // -5: Twist error: One corner has to be twisted
  // -6: Parity error: Two corners or two edges have to be exchanged

  /**
   * Check if the cube definition string s represents a solvable cube.
   * @param s is the cube definition string, see {@link Facelet}
   * @return 0: Cube is solvable
   * -1: There is not exactly one facelet of each colour
   * -2: Not all 12 edges exist exactly once
   * -3: Flip error: One edge has to be flipped
   * -4: Not all 8 corners exist exactly once
   * -5: Twist error: One corner has to be twisted
   * -6: Parity error: Two corners or two edges have to be exchanged
   */
  public static int verify(String s) {
    int[] count = new int[6];
    try {
      for (int i = 0; i < 54; i++)
        count[Color.valueOf(s.substring(i, i + 1)).ordinal()]++;
      catch (Exception e) {
        return -1;
      }

      for (int i = 0; i < 6; i++)
        if (count[i] != 9)
          return -1;

      FaceCube fc = new FaceCube(s);
    }
  }
}
CubieCube cc = fc.toCubieCube();

return cc.verify();
}

/**
 * Generates a random cube.
 * @return A random cube in the string representation. Each cube of
 the cube space has the same probability.
 */

public static String randomCube() {
  CubieCube cc = new CubieCube();
  Random gen = new Random();
  cc.setFlip((short) gen.nextInt(CoordCube.N_FLIP));
  cc.setTwist((short) gen.nextInt(CoordCube.N_TWIST));
  do {
    cc.setURFtoDLB(gen.nextInt(CoordCube.N_URFtoDLB));
    cc.setURtoBR(gen.nextInt(CoordCube.N_URtoBR));
  } while ((cc.edgeParity() ^ cc.cornerParity()) != 0);
  FaceCube fc = cc.toFaceCube();
  return fc.to_String();
}
References


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.