三阶魔方模拟器

一、项目目标

设计一款面向魔方新手和爱好者的三阶魔方模拟器。不仅可以像现实中的三阶魔方一样自由转动，为了方便用户，还实现了以下自动化功能：自动打乱、保存打乱状态、分步自动复原、一步自动复原。魔方新手可以用此软件自由探索魔方还原方法，也可以通过观察自动还原程序进行分阶段的学习；魔方爱好者可以用此软件进行魔方还原练习，自动还原方法也能为他们提供参考。

二、需求与设计

1.对现实三阶魔方的模拟

魔方的 18 种单层 90°转动都以按钮的形式布置在被转动层的旁边，简单易懂。另外布置有四种对魔方整体的 90°转动，这是对观察视角不够自由的补偿：现实中可以通过举起魔方、探头等动作观察到魔方的各个面；程序中虽然同时绘制了两种视角下的魔方，能同时观察到全部六个面，但其空间关系不够明确，于是设计了对魔方整体转动的按钮，方便用户观察。

2.降低用户的学习难度

本程序的自动还原算法所需步数少，公式量适中，适合新手学习。还原顶层块朝向仅需两个步骤，还原顶层块排列仅需两个步骤，相比传统的层先法还原流程更加简短。

设计单步还原按钮，是为了降低学习成本。假设用户对前两层的还原方法不感兴趣，只关心第三层是如何被自动还原的，用户不需要先自行还原前两层，可以用“复原前两层”按钮自动完成。如果用户想要观察某种特定魔方状态的自动还原，也可以在不同复原阶段的基础上很方便地构造出想要的打乱。

为了保证观看体验，本自动还原算法的大部分操作都集中在 R 层、U 层、F 层，并让当前步骤的关键块尽可能多地出现在第一视角中。

在自动还原过程中，魔方每两次转动之间都添加了延时。在此之上，每两个步骤之间会额外延时，以体现分步还原的思想。对魔方整体进行转动之后也会额外添加延时，便于用户理清空间关系。为满足不同用户的需求，窗口菜单中提供设置延时长短的功能，让每个用户都能看清自动还原的过程。

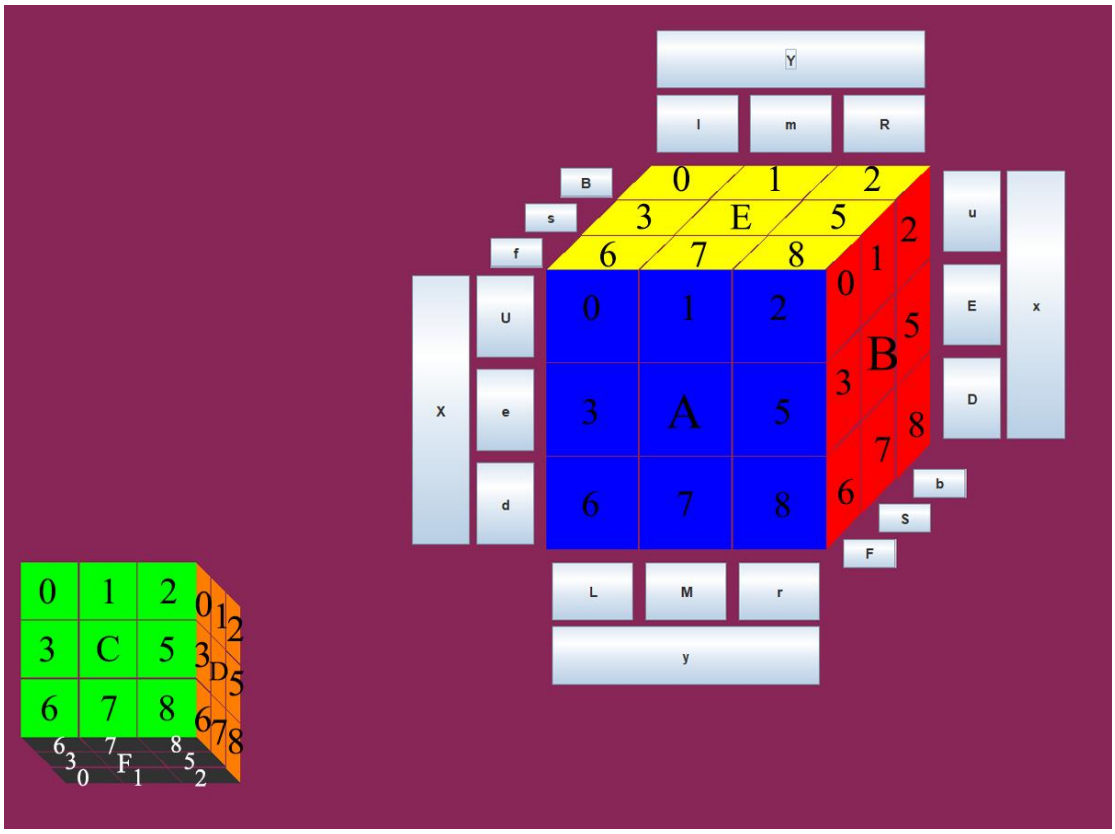
3.便捷性

“新的打乱”按钮会执行一个步数为 30 的打乱公式，并且其中不包含“R r”、“R R R R”之类的无效操作，打乱效果有保证。“回到打乱”按钮将魔方恢复到刚刚打乱完的状态，方便用户对同一打乱进行多次尝试，此功能对刚开始学习还原的新手以及研究最少步数还原的玩家是非常必要的。

三、关键环节的实现

1、魔方基本功能

1.1 术语解释（只适用于本项目）



·魔方颜色特点：蓝绿相对、红橙相对、黄黑相对

·将魔方看作 3*3*3 个小立方体，有 3 个面暴露在外的小立方体是角块，如{A2,B0,E8}，有 2 个面暴露在外的小立方体是棱块，如{A5,B3}，有 1 个面暴露在外的小立方体是中心块，如{A4}

·魔方的“面”和“层”：E 面指 E0~E8 的集合，E 层指 E 面和 A0~A2、B0~B2、C0~C2、D0~D2 的集合，其他面和层以此类推

·转动：大写字母表示顺时针转动 90°，小写字母表示逆时针转动 90°。eg.“U”是“上层”的缩写，表示视角正对上层时，上层顺时针旋转 90°。U、D、R、L、F、B 分别对应上、下、右、左、前、后层。对于中间层转动和魔方整体转动，X，e 方向与 U 相同；Y，m 方向与 R 相同；S 方向与 F 相同

1.2 魔方的程序实现

魔方有六个面，每个面有 9 个色块，使用一个二维数组 int cubeColor[6][9]就可以表示魔方的状态 cubeColor[0],cubeColor[1],cubeColor[2],cubeColor[3],cubeColor[4],cubeColor[5],分别对应 A,B,C,D,E,F 面，需要注意的是，编码的是位置而不是颜色，例如 A4 或 cubeColor[0][4]指的是位于第一视角正面的中心块，而不一定是蓝色中心块。int 值 0,1,2,3,4,5 分别对应蓝，红，绿，橙，黄，黑色。

魔方的转动实质上就是色块的交换，对于转动的程序实现，只需交换

cubeColor 中对应元素的值即可。根据不同操作封装成不同函数。另外，为了提高可读性，实现了一个根据文字公式自动执行多步操作的方法 Moves.perform(String formula)。

2、并发处理

本项目的线程问题主要产生自两个耗时操作：打乱和自动还原。在编程的初期，我发现程序在执行这两个耗时操作的时候图形界面不能正常更新了，经过大量学习后发现了问题：我是通过 Jbutton 中的 actionPerformed 方法进行耗时操作的，而 actionPerformed 方法和组件绘制默认都属于事件派发线程，因此在 actionPerformed 方法中阻塞线程会使图形界面卡死，必须新开一个线程进行耗时操作。

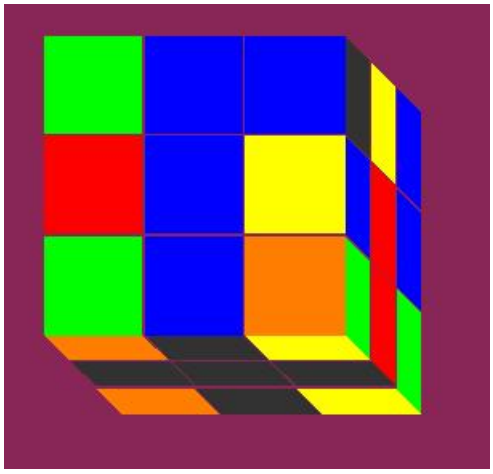
由于自动还原算法并不是每时每刻对魔方状态进行判定，而是判定一次做一个步骤，在耗时操作过程中，用户按下按钮改变魔方状态可能造成还原失败，因此实现了 enableButtons()方法，在执行耗时操作时禁用按钮。

3、自动还原算法

3.0 简介

本算法基于人类魔方玩家的逻辑，不涉及机器学习。为了降低状态匹配难度，本算法中默认黑色中心块在底面，蓝色中心块在正面，每次执行自动还原都会先将魔方的方向校正。还原一共分为五大步骤：还原底层棱块、还原底层角块、还原中间层棱块、还原顶层块朝向、还原顶层块排列。

3.1 还原底层棱块



也叫“还原底层十字”、“Cross”。

在还原的前期，已经还原好的块占比很低，玩家的操作几乎不受限制。因此，将 4 个目标棱块一个接一个地还原到目标位置就可以了，要注意的只有一个，还原顺序靠后的棱块时不能打乱已经还原的底面棱块。本步骤基本流程：

找到黑红棱块，将其黑色面移动到 F3；

找到黑绿棱块，将其黑色面移动到 F7；

找到黑橙棱块，将其黑色面移动到 F5；

找到黑蓝棱块，将其黑色面移动到 F1。

注意到目标黑色面的可能位置共有 24 种，将其移动到 F1、F3、F5、F7 的步骤也各不相同，在此使用一种减少分类数目的方法——借位法。只枚举将目标黑色面移动到 F3 的 24 种情况，如果终点位置是 F1，先做一次 D，将位于 F1 的棱块移动到 F3，然后根据枚举的 24 种情况将目标黑色面移动到 F3，再做 d 让目标黑色面到达 F1,就像先借了 F3 的位置，最后再还回去一样。

用借位法优化后的本步骤流程：

找到黑红棱块，将其黑色面移动到 F3；d；

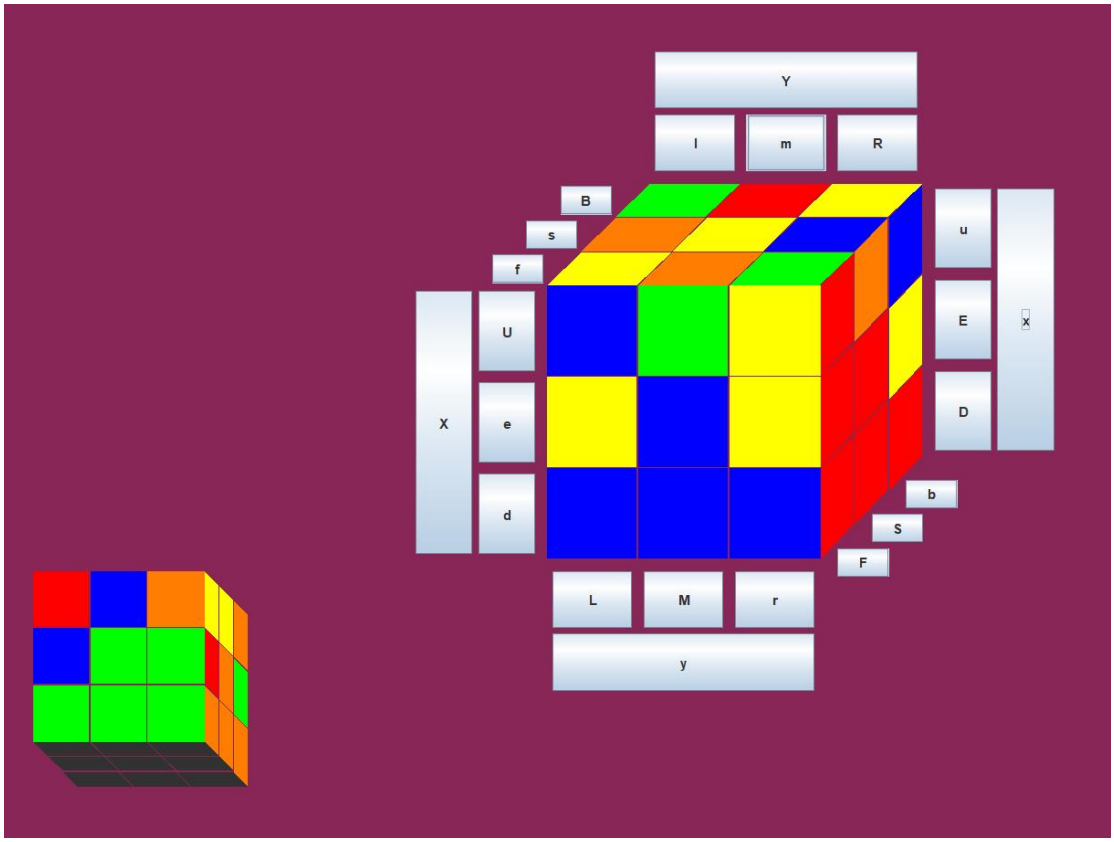
找到黑绿棱块，将其黑色面移动到 F3；d；

找到黑橙棱块，将其黑色面移动到 F3；d；

找到黑蓝棱块，将其黑色面移动到 F3；d。

在程序中，用 findEdge()方法寻找特定颜色的棱块，用 edgeToF3()方法将一个目标面在不影响底层棱块的前提下移动到 F3，提高了代码复用率。

3.2 还原底层角块



本步骤逻辑和 Cross 完全一致，直接给出用借位法优化后的步骤流程：

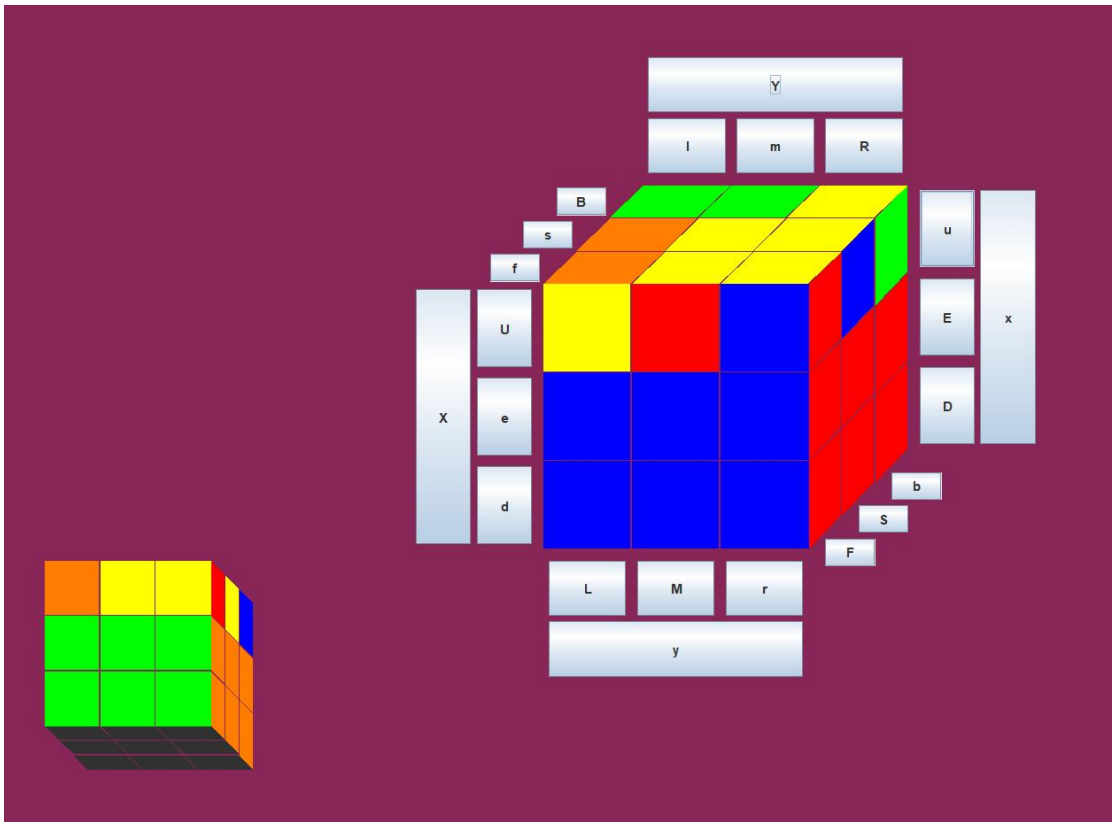
找到黑蓝红角块，将其黑色面移动到 F0；d；

找到黑红绿角块，将其黑色面移动到 F0；d；

找到黑绿橙角块，将其黑色面移动到 F0；d；

找到黑橙蓝角块，将其黑色面移动到 F0；d。

3.3 还原中间层棱块



本步骤逻辑和 Cross 完全一致，直接给出用借位法优化后的步骤流程：

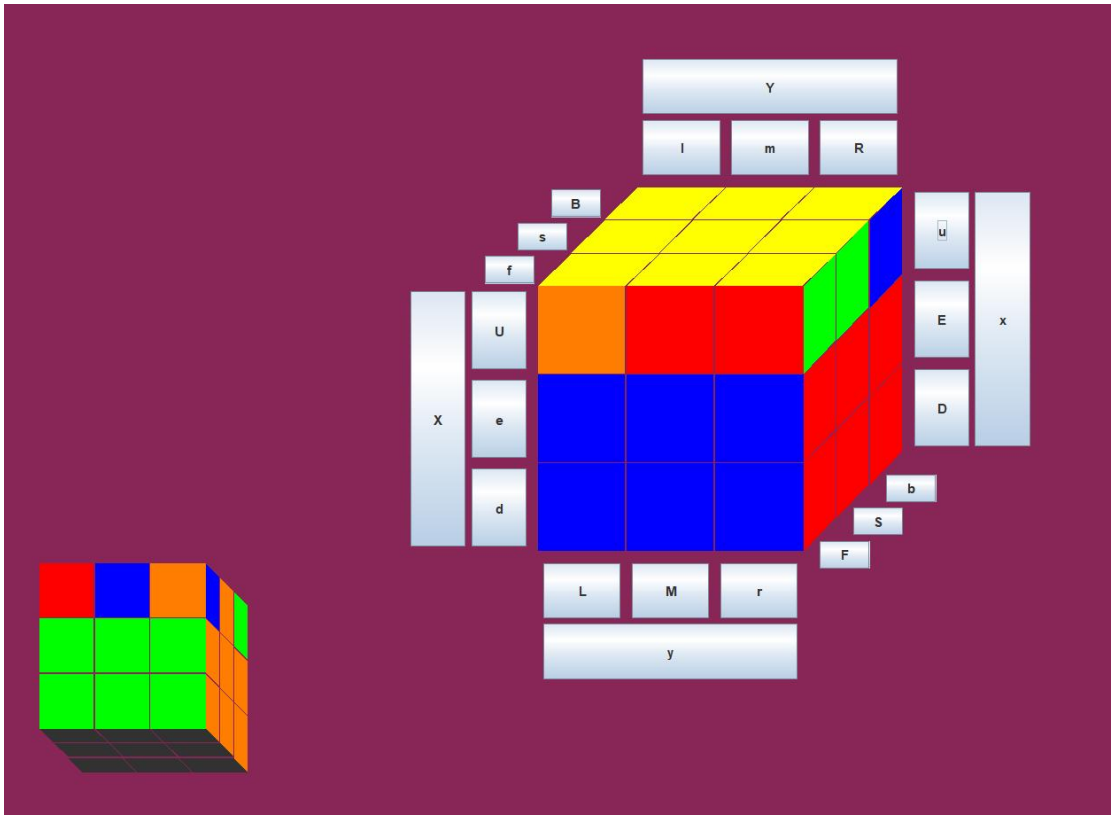
找到蓝红棱块，将其蓝色面移动到 A5；e；

找到红绿棱块，将其红色面移动到 A5；e；

找到绿橙棱块，将其绿色面移动到 A5；e；

找到橙蓝棱块，将其橙色面移动到 A5；e。

3.4 还原顶层块朝向

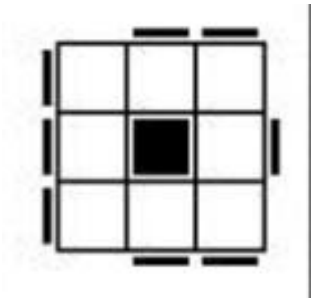


和还原顶面的含义相同，也叫“OLL”、“OrientationofLastLayer”，每个角块拥有 3 种朝向，每个棱块拥有 2 种朝向，对于顶层块来说，黄色朝上就代表朝向正确。

从此步骤开始，一块一块地还原的思路就不适用了，在前两层的还原中，顶层是可以被随意打乱的，顶层有着缓冲区域的作用。而在顶层的复原中，不可能有将一个顶层块移动到目的地而不打乱别的顶层块的方法，只有按某种规律同时移动多个块的方法。因此要同时判定多个色块的位置作为分类讨论的依据。

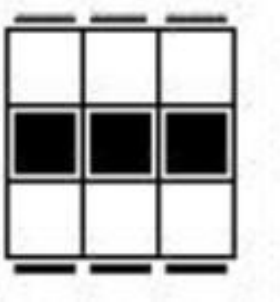
第一步、还原棱块朝向：仅关注顶层棱块朝向，前两层已复原的前提下可分为 4 种状态，其他状态可通过 U 层转动转化为其中之一。

（图例为正对 E 面的俯视图，最下面的边与 A 面邻接）



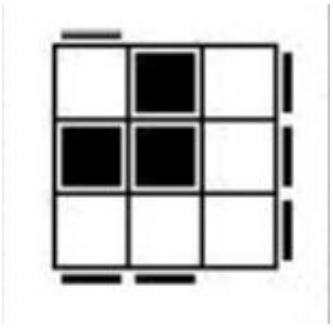
State1:

Solution: F U R u r f U F R U r u f



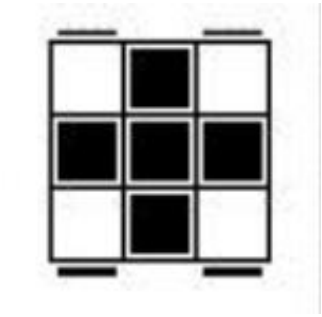
State2:

Solution: F R U r u f



State3:

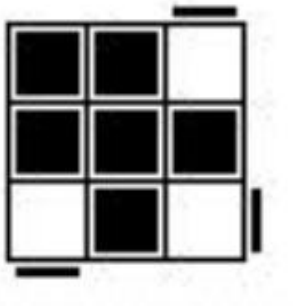
Solution: F U R u r f



State4:

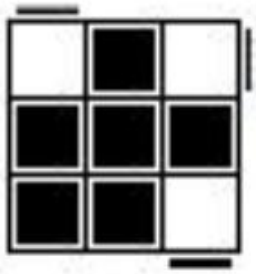
Finished

第二步、还原角块朝向：顶层角块可分为 8 种状态，其他状态可通过转动顶层转化为其中之一



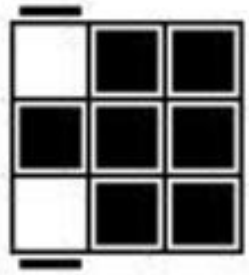
State1:

Solution: r u R u r u u R



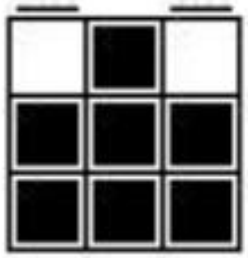
State2:

Solution: R U r U R U U r



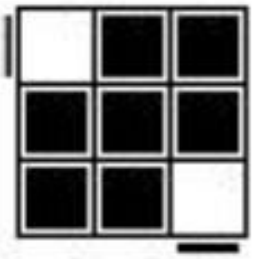
State3:

Solution: m R U r u M r F R f



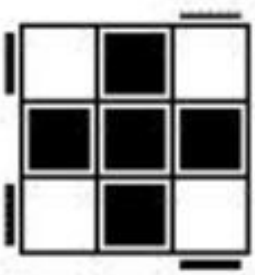
State4:

Solution: r r d R U U r D R U U R



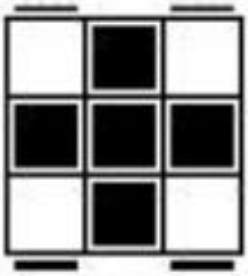
State5:

Solution: f m R U r u M r F R



State6:

Solution: R U U R R u R R u R R u u R



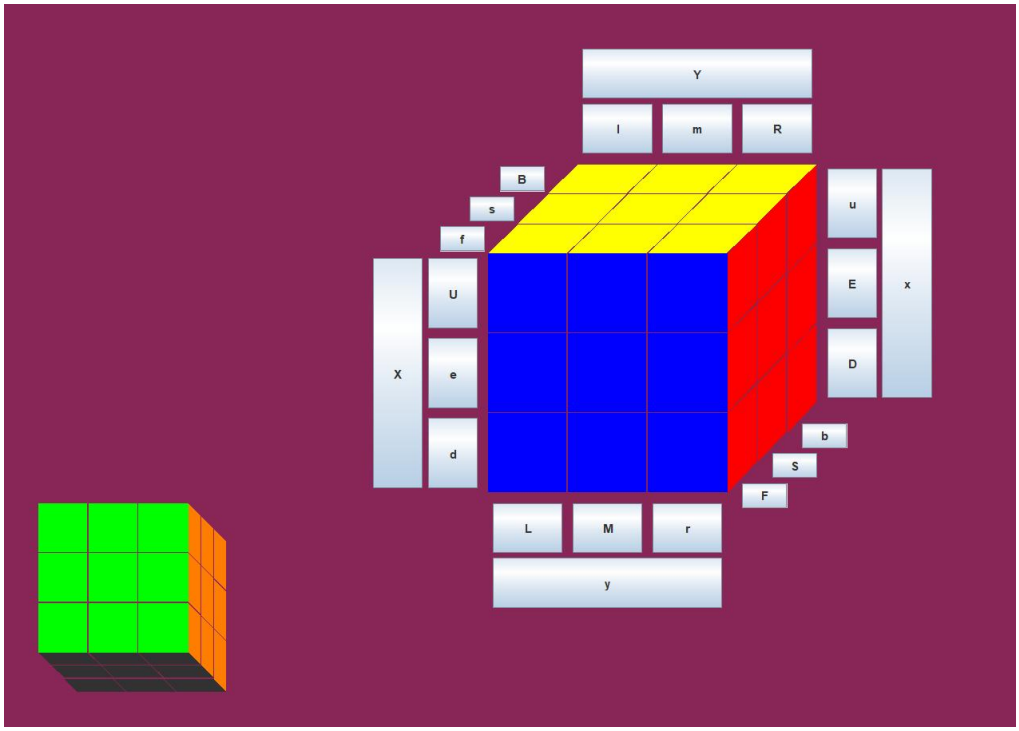
State7:

Solution: R U U r u R U r u R u r

State8:

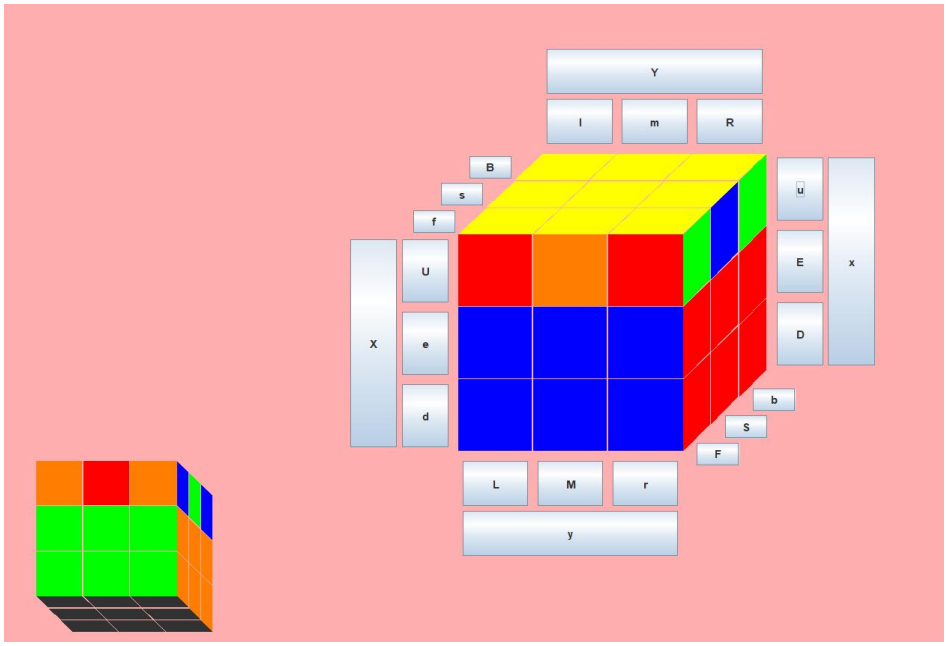
Finished

3.5 还原顶层块排列



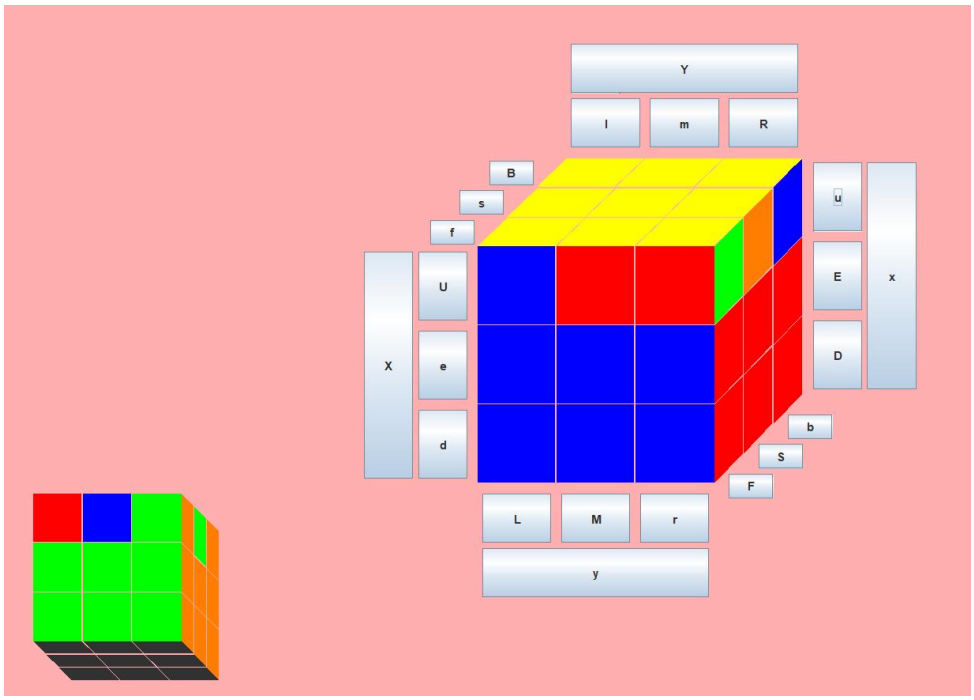
也叫 PLL、PermutationofLastLayer。还原顶层块朝向后，9 个顶层块共有 88 种位置分布，即使将能通过 U 层转动统一的情况归为一类，仍有 22 种相对位置关系，继续分为两步。

第一步、还原顶层四角块相对位置：



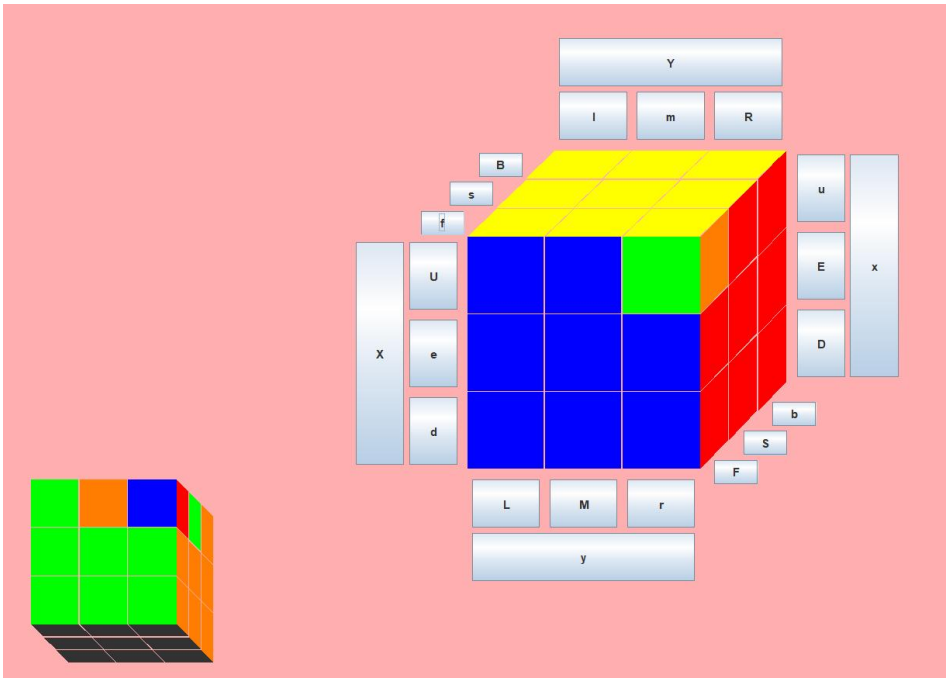
仅关注四个角块的相对位置关系，通过转动 U 层，所有状态可转化为以下 3 个状态之一

State1：相邻角块对调



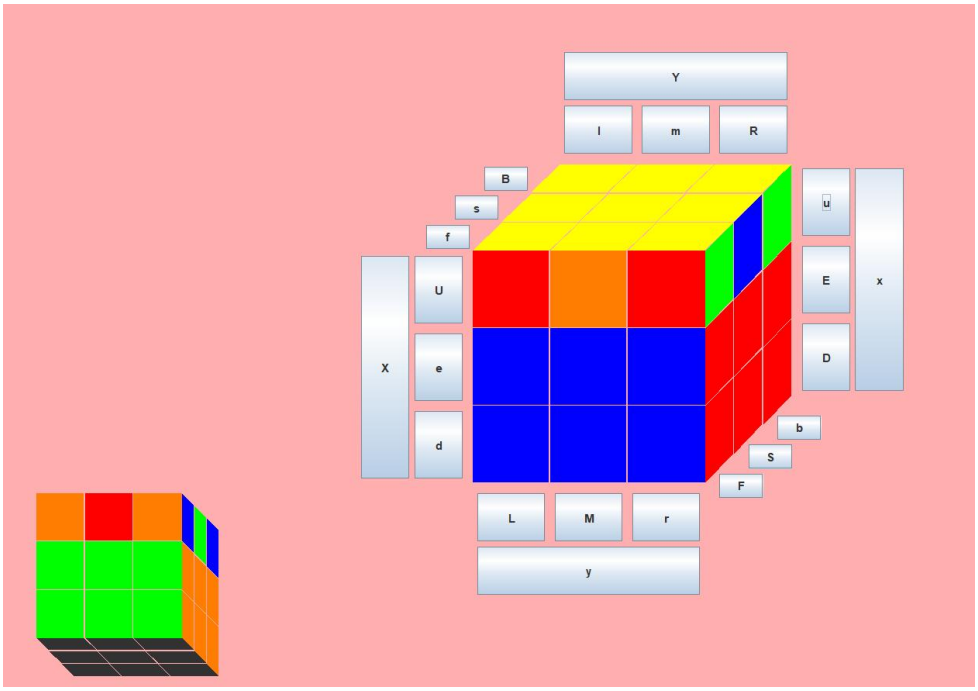
Feature: A、B、C、D 四个面中，有且仅有 D 面的 0 号色块与 2 号色块同色
Solution: R U r u r F R R u r u R U r f

State2：对角角块对调



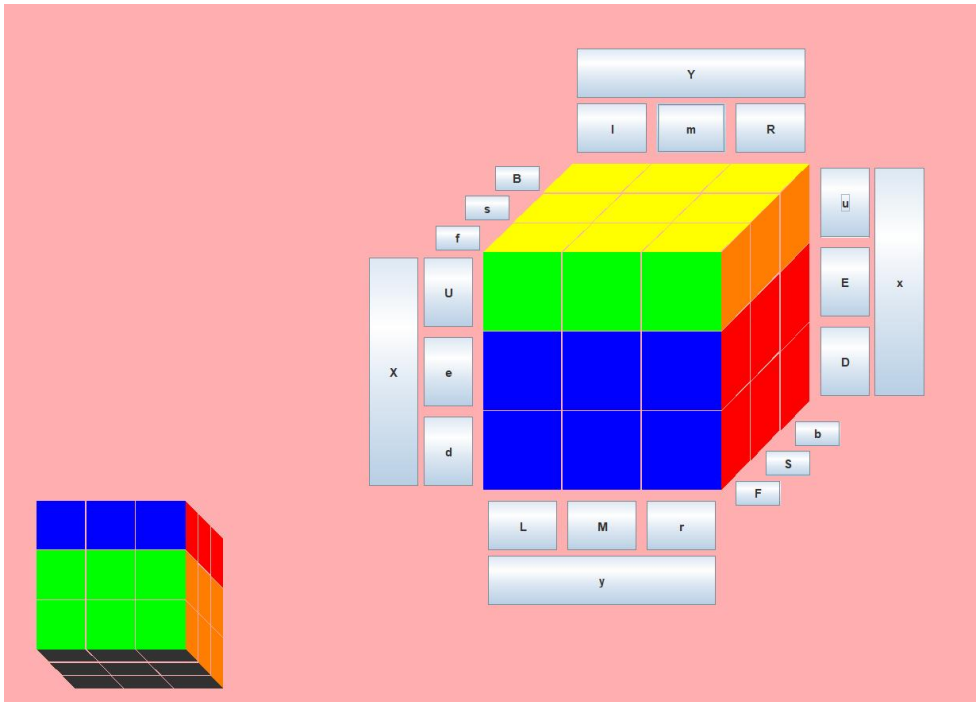
Feature: A、B、C、D 四个面中， 0 号色块与 2 号色块均不同色
Solution: F R u r u R U r f R U r u r F R f

State3: 角块相对位置正确



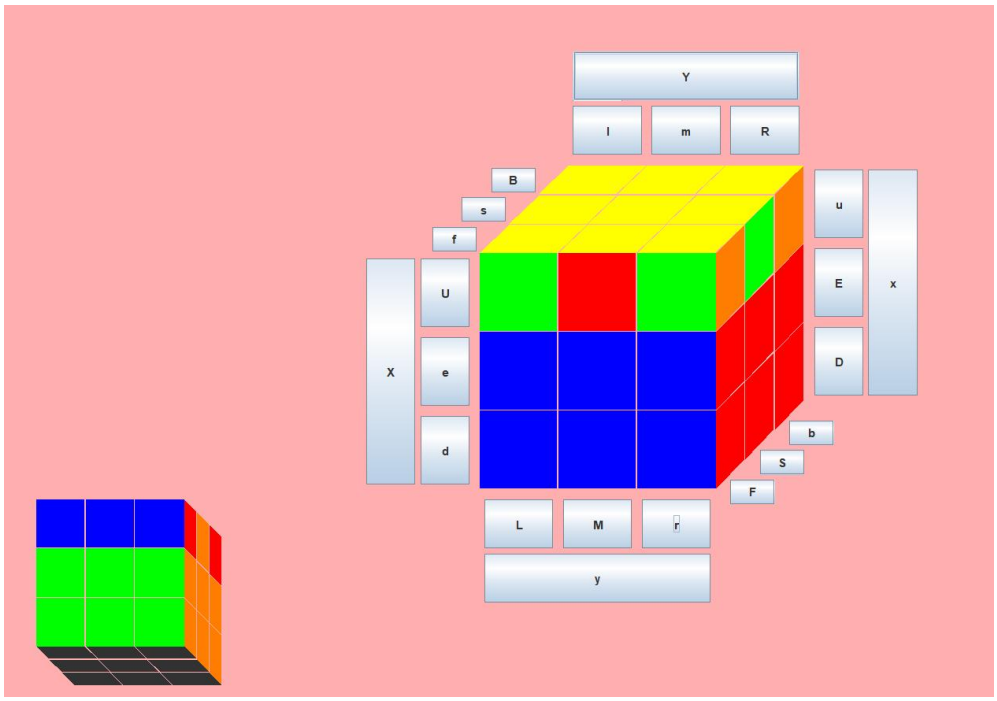
Feature: A、B、C、D 四个面的 0 号色块均与 2 号色块同色
Solution: Finished

第二步、使顶层棱块的位置与顶层角块匹配，完成复原：



通过转动 U 层，所有状态可转化为以下 5 个状态之一

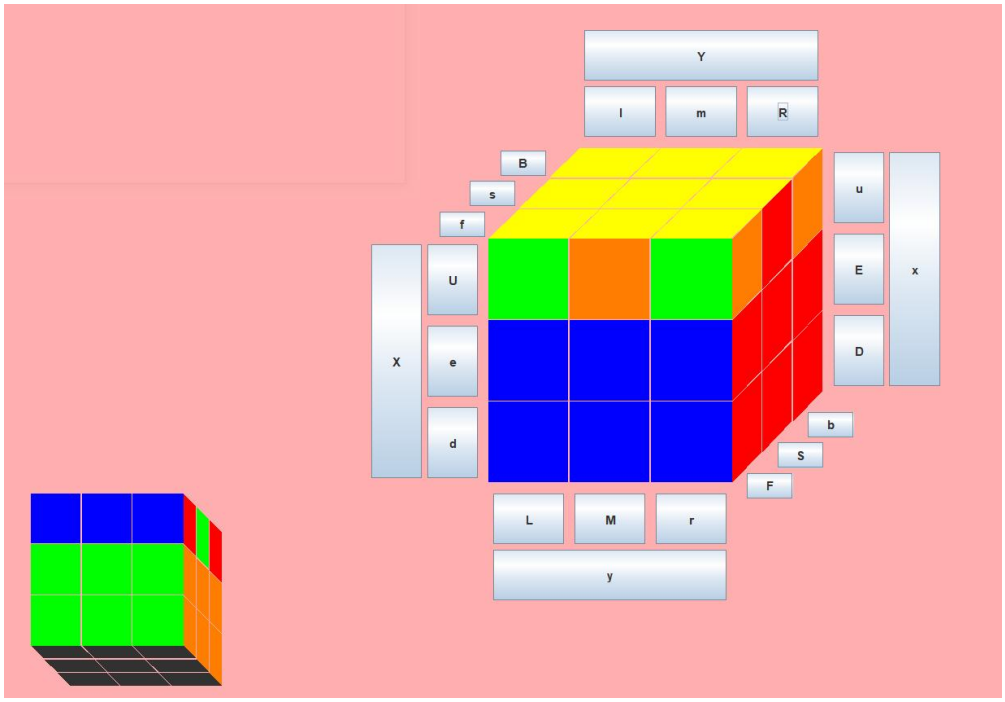
State1：顺三棱换



Feature: A2 与 B1 同色，D2 与 A1 同色

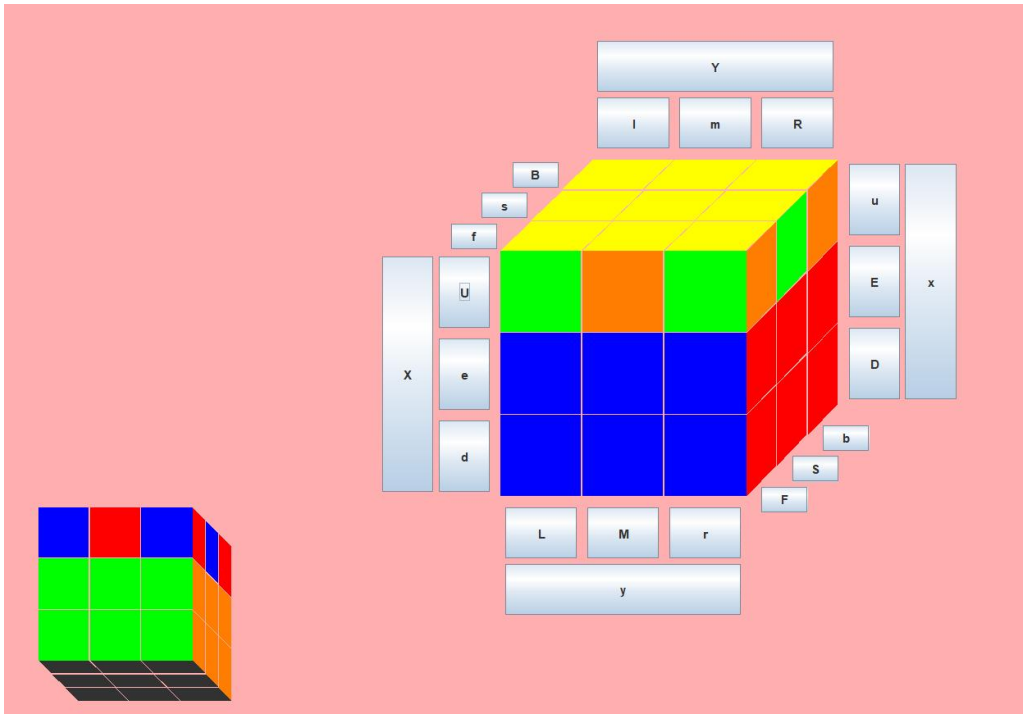
Solution: R R U R U r u r u r U r

State2:逆三棱换



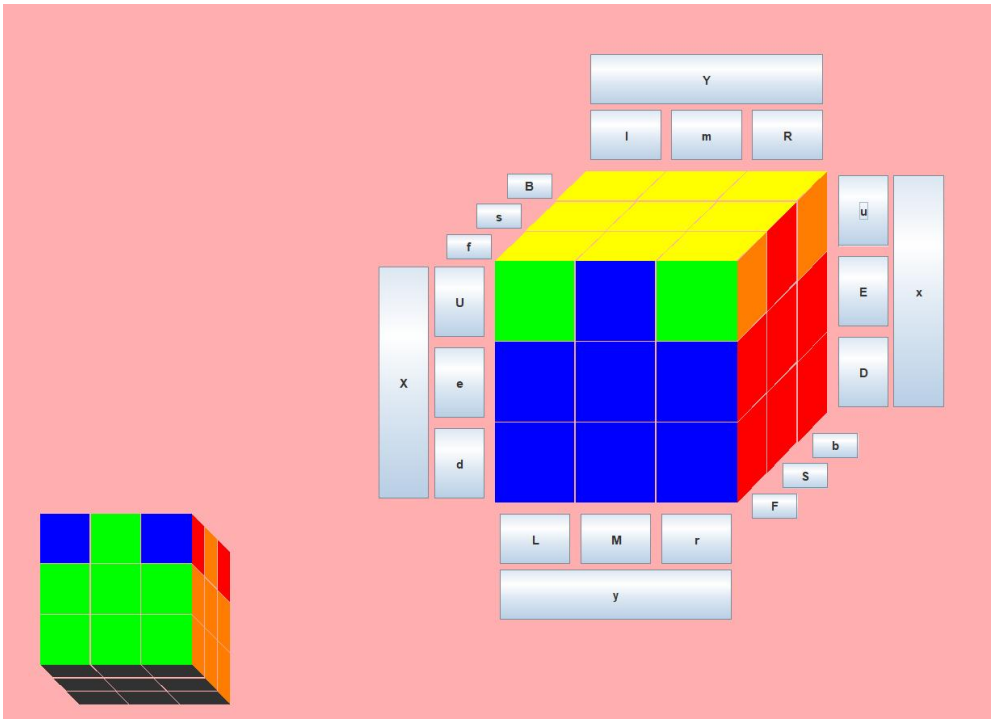
Feature: A1 与 B0 同色，D1 与 A0 同色
Solution: R u R U R U R u r u r r

State3:相邻四棱换



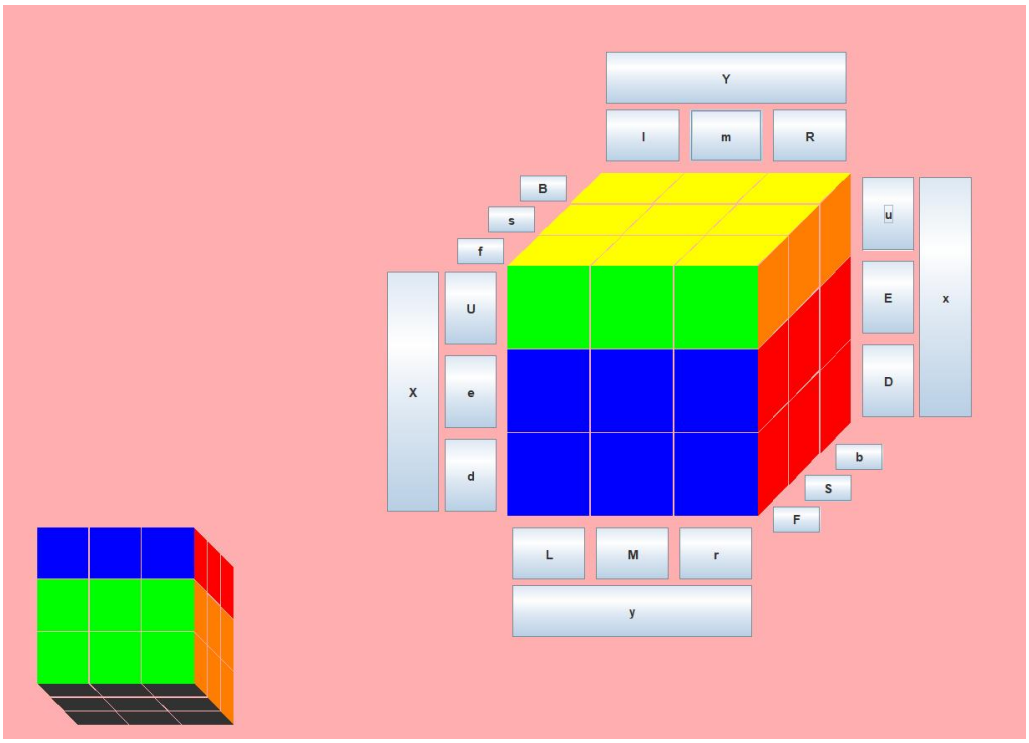
Feature: A2 与 B1 同色，A1 与 B0 同色
Solution: U r u R u R U R u r U R U R R u r U

State4:相对四棱换



Feature: A0 与 C1 同色，A1 与 C0 同色
Solution: m m U m m U U m m U m m

State5:顶层块相对位置正确



Feature: A0 与 A1 同色，B0 与 B1 同色
Solution: Finished

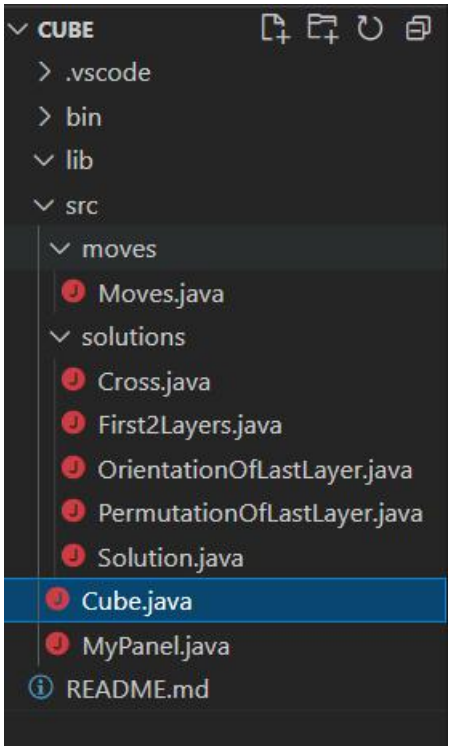
接下来转动 U 层完成还原即可

四、工程基本信息

1.运行环境

JDK：　　OpenJDK 11.0.13
编译器：　　VSCode
VSCode 拓展：　Extension Pack for Java v0.18.7

2.工程文件简介



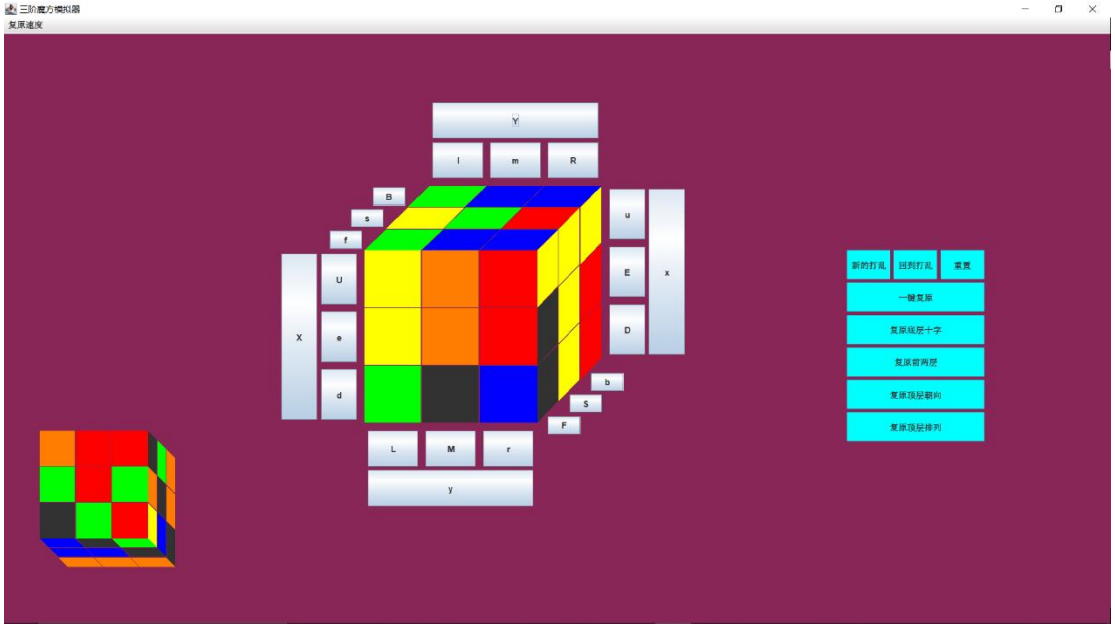
Cube.java:
主类。各个类的实例化、Jframe 的设置、JPanel 的设置、JMenu 的设置、所有按钮的实现、线程冲突的处理。
MyPanel.java:
继承 JPanel，定义图形界面的绘制方法。
Moves.java:
集中定义了所有能改变魔方状态的基本方法，包括打乱、回到打乱、重置、各种单步转动、无延时地执行公式、有延时地执行公式。
Solution.java:
此类为四大复原步骤地父类，包含它们共有的成员变量和方法。
Cross.java:
其 run 方法可复原底层棱块。
First2Layers.java:
其 run 方法可检查前序步骤完成情况，若已完成则复原底层角块和中间层棱块，

否则弹出警告。
OrientationOfLastLayer.java:
其 run 方法可检查前序步骤完成情况，若已完成则复原顶层块朝向，否则弹出警告。
PermutationOfLastLayer.java:
其 run 方法可检查前序步骤完成情况，若已完成则复原顶层块排列，否则弹出警告。

3.使用说明

已验证可行的运行程序的方法：直接运行 cube.jar 或使用 VSCode 打开 cube 文件夹，调试并运行主类 Cube.java。

已验证可行的查看源码的方法：使用 VSCode 打开 cube 文件夹。



程序成功运行的界面