2019.6

魔方求解器分析设计与实现

软件工程小组项目

刘英杰 16337161

王浩翔 16337229

徐靖 16337262

秦李煜 16337199

徐原 16337266

目录

一、 需求分析

1.1 魔方求解器问题陈述

魔方作为一种益智玩具，自诞生之初就吸引了全世界无数人探究它的奥秘。魔方初学者常常苦恼于没有一种简单的工具可以在自身没有学会魔方解法的情况下，把手中的魔方还原，我们的项目正是为他们提供了这样的一个工具。

本软件的目标用户是没有掌握三阶魔方还原方法，希望能用 2-10 分钟还原三阶魔方的人。

用户持有一个处于打乱状态的实体三阶魔方，使用本软件将魔方信息输入计算机中，并获取还原该魔方的具体步骤并且按照该步骤转动手中的魔方，以达到将魔方还原的目的。

用户还可以观看魔方复原的动画。动画中的复原过程是上述系统解出的步骤。

用户获取的还原步骤根据伪 CFOP 方法分为四段，以方便用户在跟随步骤还原魔方的过程中检查操作的正确性。

用户若是想从解魔方的原理入手学习普遍性求解方法，可选择观看魔方还原教程，根据提示进行学习，以达到自学目的。

1.2 魔方求解器用例析取

1.2.1 还原魔方用例图

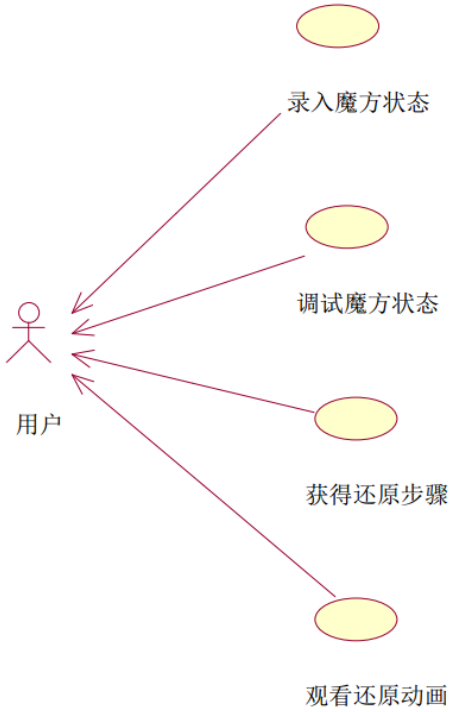


Figure 1 还原魔方用例图

1.2.2 学习魔方用例图

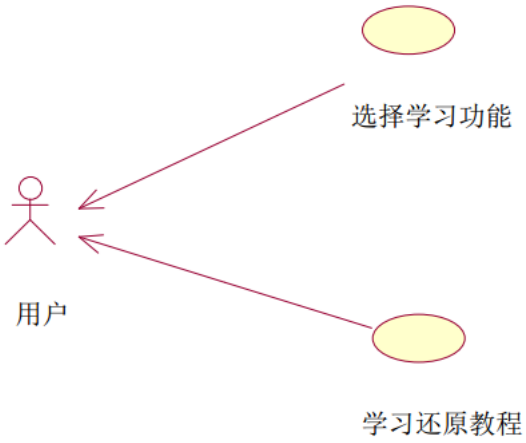


Figure 2 学习魔方教程用例图

1.2.3 其他功能用例图

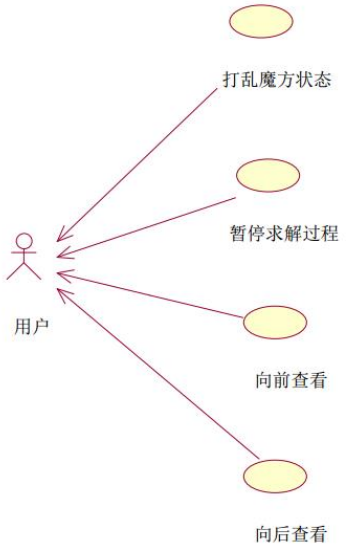
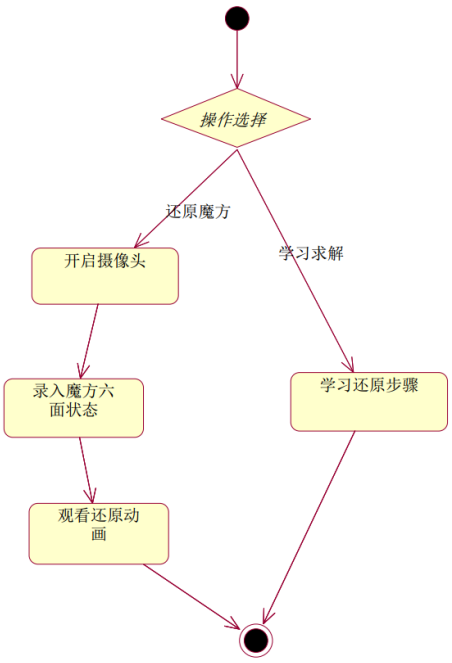


Figure 3 用例 3

1.3 用例规约

整体活动图



1.3.1 拍照输入魔方信息

1）简要说明

本用例说明了用户使用电脑摄像头将实体三阶魔方的信息录入系统的过程。

2）参与者

用户

3）事件流

I.基本事件流

用例开始于用户已打开系统，并点击“开始录入魔方”按钮。

A.系统启动电脑默认摄像头，准备拍摄。

A1.未检测到可用摄像头

B.用户按照屏幕文字提示逐面录入魔方的六个面

B1.录入过程中发生误操作或识别错误

C.六个面全部录入完成后点击完成，魔方信息保存到系统中，并退出拍摄界面。

II.后备事件流

A1.未检测到可用摄像头

系统提示用户需要保证电脑有摄像头，拍摄操作中止。

B1. 录入过程中发生误操作或识别错误

用户可以随时重新录入任意一面以改正错误。

4）特殊需求

无

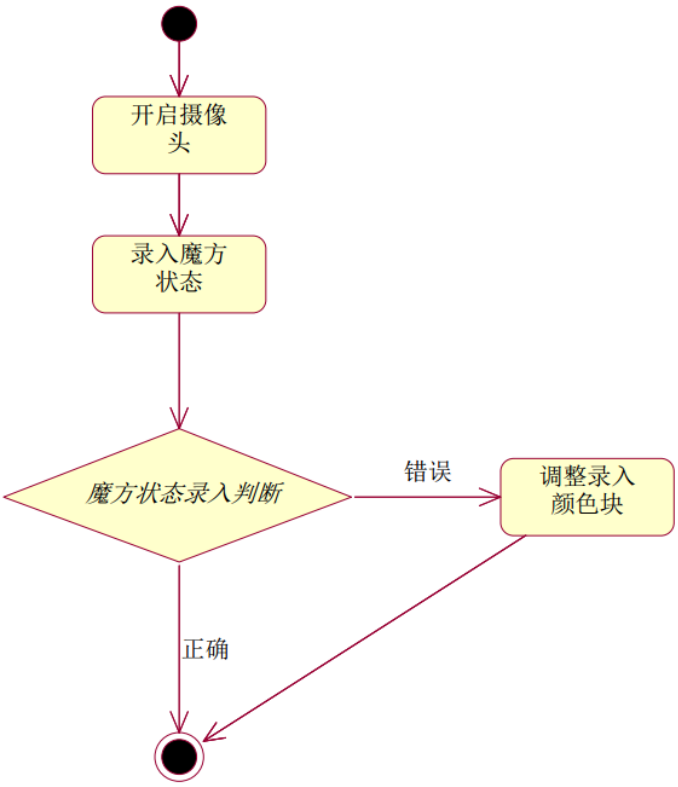
5）前置条件

用户已打开软件

6）后置条件

如果用例成功，那么系统中就存储了用户手中的魔方的信息，否则系统状态不变。

7） 活动图



1.3.2 获得解魔方步骤

1）简要说明

本用例描述了用户获取解魔方步骤的过程

2）参与者

用户

3）事件流

I.基本事件流

用例开始于用户已经录入了魔方信息，并点击“解魔方”按钮

A.系统计算出魔方的还原过程

A1.计算过程中发现魔方是不可解状态

B.系统将魔方还原步骤分四个阶段呈现给用户（使用 RLUDFB 记法）

II.后备事件流

A1.计算过程中发现魔方是不可解状态

系统提示用户魔方不可解，不给出还原步骤。

4）特殊需求

系统要能识别不可解情况

5）前置条件

用户已经用摄像头录入了魔方

6）后置条件

如果用例成功，用户可以看到魔方还原步骤，动画模块也会得到这个步骤，否则系统不变。

1.3.3 观看解魔方动画

1）简要说明

本用例描述了用户观看解魔方动画的过程

2）参与者

用户

3）事件流

I.基本事件流

用例开始于用户已经完成了求解魔方步骤并点击“观看动画”按钮后。

A.系统呈现魔方还原的动画

A1.用户按下暂停按钮

A2.用户按下上一步按钮

A3.用户按下下一步按钮

II.后备事件流

A1.用户按下暂停按钮

暂停动画的播放

A2.用户按下上一步按钮

如果没有暂停则先暂停，然后回到还原过程的上一步状态

A3.用户按下下一步按钮

如果没有暂停则先暂停，然后回到还原过程的下一步状态

4）特殊需求

无

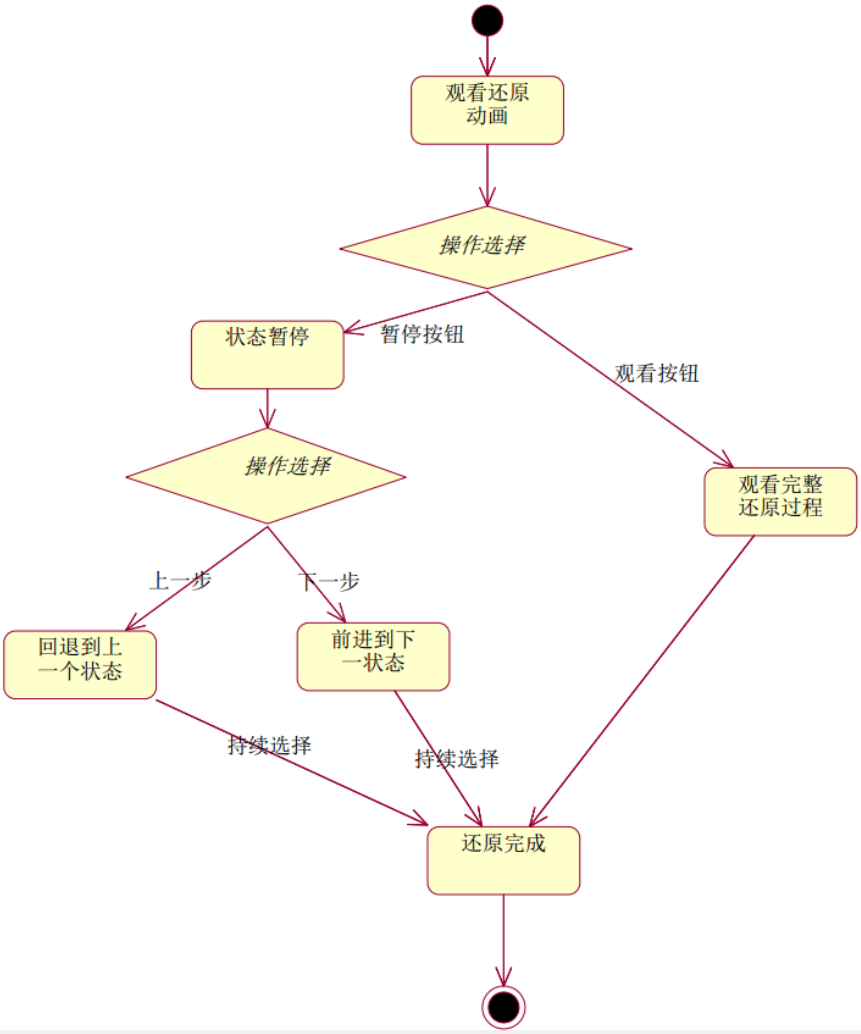
5）前置条件

用户已经完成了求解魔方步骤

6）后置条件

如果用例成功，用户可以看到魔方还原的动画，观看过程中可以控制播放进度。

7） 活动图



1.3.4 学习魔方还原过程

1）简要说明

本用例描述了用户学习求解魔方的过程

2）参与者

用户

3）事件流

I.基本事件流

用例开始于用户选择学习魔方还原原理教程后

A. 系统呈现魔方还原教程图片

B. 用户单击返回键，系统返回至主界面

II.后备事件流

无

4）特殊需求

无

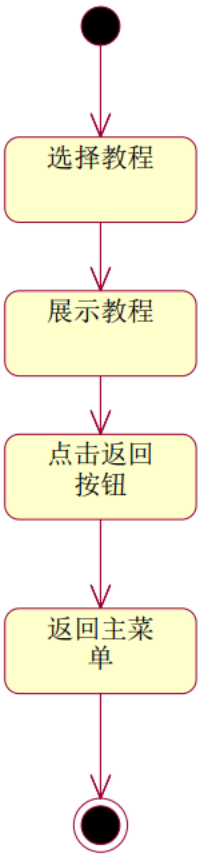
5）前置条件

用户处于主菜单

6）后置条件

用户看完教程后返回主界面

7）活动图



1.4 补充规约

1.4.1 可靠性

在任何情况下点击程序窗口中的任何按钮不会导致程序崩溃，并且误操作都可以补救。

1.4.2 安全性

本程序是单机程序，不涉及个人信息的长期存储。

1.4.3 可用性

本程序目标群体是不具备独立还原魔方的知识的用户，用户按照图形界面的提示即可获取还原魔方的方法，简单易用。

1.5 术语表

名词术语	描述
三阶魔方	（本文中简称为魔方）：一种益智玩具，正方形，每个面有 3*3 排列的九个小块，各面呈现不同颜色，可以通过转动改变每个面小块的颜色组合
魔方状态	魔方六个面当前的颜色组合状态
还原状态	魔方每个面上只有一种颜色的状态
打乱状态	除还原状态以外的魔方状态
可解状态	可以通过转动恢复到还原状态的打乱状态，以及还原状态本身
不可解状态	不能通过转动恢复到还原状态的打乱状态
还原步骤	将魔方从某个打乱状态转成还原状态的一组转动步骤
RLUDFB 记法	魔方某一面正对自己时，描述魔方转动的记号。六个字母分别是“右左上下前后”的英文首字母。通过一个额外的标点符号说明转动顺逆时针方向，例如 R 表示右面转顺时针，R’表示右面转逆时针
CFOP	一种魔方复原方法。分为四个主要步骤：分别是，Cross→First 2 layers→Orientation of last layer→Permutation of last layer(即 Cross→F2L→OLL→PLL)，也就是：底层十字→同时对好前两层→调整好最后一层的朝向→调整好最后一层的顺序（排列）
中心块	魔方九宫格中位于中心的方块
棱块	魔方中有两个可见面的方块
角块	魔方中有三个可见面的方块

二、 技术选型

2.1 Python3

本程序全部代码均使用 Python3 语言，Python 是一种简单易学，功能强大，运行高效的开元脚本语言，通过调用各种 Python 库和自己编写的组件，可以快速

搭建完整的应用程序。

2.2 PyOpenGL

OpenGL 是应用广泛的计算机图形编程规范，PyOpenGL 是 Python 官方提供的在 Python 环境下的 OpenGL 实现。

2.3 PyQt

PyQt 是一个创建 GUI 应用程序的工具包，它是 Python 编程语言和 Qt 库的成功融合。本程序使用了 PyQt 中的多个模块，其中 QtCore 模块包含核心的非 GUI 功能。该模块用于时间、文件和目录、各种数据类型、流、网址、MIME 类型、线程或进程。QtGui 模块包含图形组件和相关的类，例如按钮、窗体、状态栏、工具栏、滚动条、位图、颜色、字体等。QtOpenGL 模块使用 OpenGL 库渲染 3D 和 2D 图形，该模块能够无缝集成 Qt 的 GUI 库和 OpenGL 库。

2.4 Opencv-python

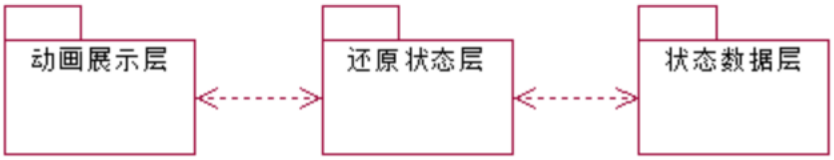
Opencv-python 是 Python 的 opencv 库，含有图像处理和计算机视觉方面的很多通用算法。

三、 架构分析

3.1 逻辑架构

本系统采用 MVC 架构设计，将程序分为 Model，View，Control 三层，使程序不

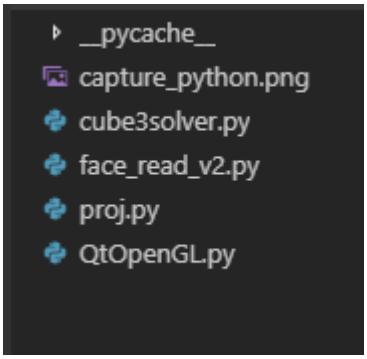
同部分分工明确，降低了系统的耦合性，便于系统的编写和调试。



a. **动画展示层：**提供给用户当前魔方还原的完整可视化过程，即 View 层。
b. **还原状态层：**核心是算法实现/求解公式，相当于处理器，处理当前未到终态的状态，即 Control 层。
c. **状态数据层：**存储整个魔方还原过程的状态数据，相当于 Model 层。

3.2 关键抽象

经分析，本系统有 4 个实体类，分别为魔方类（Cube3），图像识别类（FaceReader），UI 类（UI_MainWindow），改色板类（CheckCube），OpenGL 动画类（GLDemo）。其中，魔方类负责将魔方状态存储在数组中，处理魔方转动时的状态变化，并应用魔方公式给出魔方解法。图像识别类负责将摄像头捕捉到的画面转换为魔方某个面的颜色分布信息。UI 类负责提供功能按钮，并显示动画或图像识别的界面，特别指出，UI 类同时也完全实现了提供魔方教程的功能。改色板类用于手动修改图像识别过程中出现的部分颜色错误问题。OpenGL 动画类负责将魔方类提供的状态矩阵和动作以动画形式呈现出来。对应的文件结构如下图

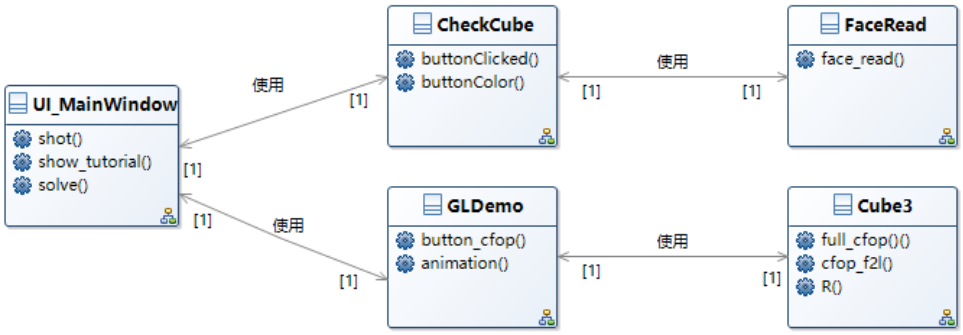


四、 类的设计

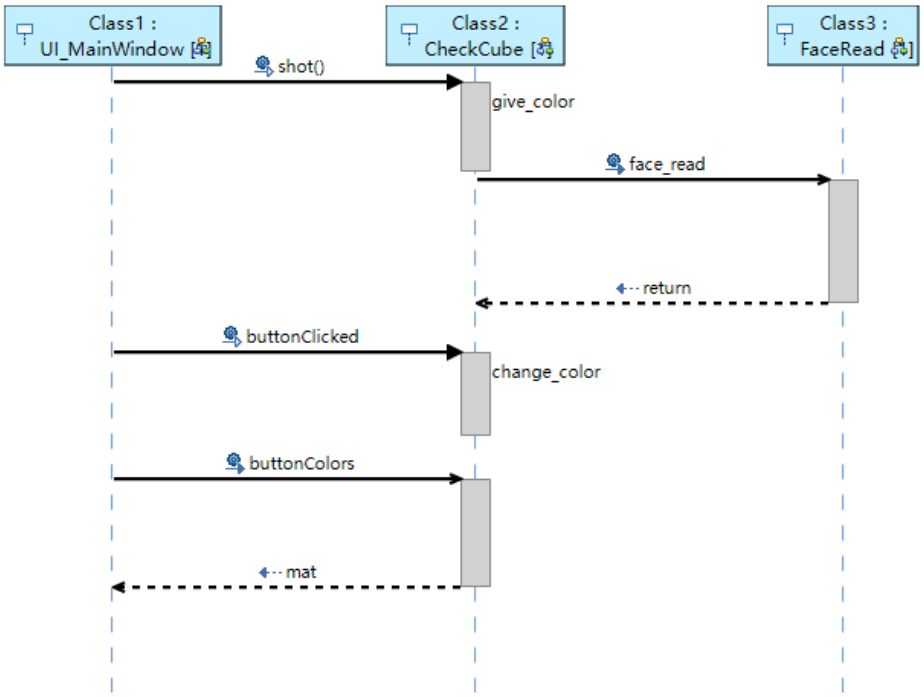
上面已经分析得出本系统的类，这里分析这些类的具体内容。

本系统中用户可见的界面中直接对应图像**识别类**（FaceReader），**UI 类**（UI_MainWindow），**改色板类**（CheckCube），**OpenGL 动画类**（GLDemo）。而魔方类（Cube3）则包含了控制层和数据层的内容。

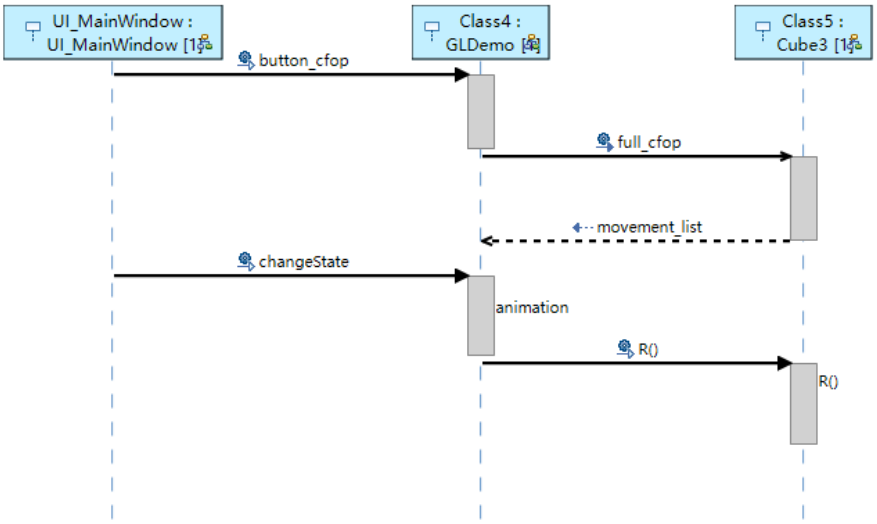
总类图如下：



FaceReader 和 UI 之间，UI 和 CheckCube 之间，UI 和 GLDemo 之间，GLDemo 和 Cube3 之间有通信关系。

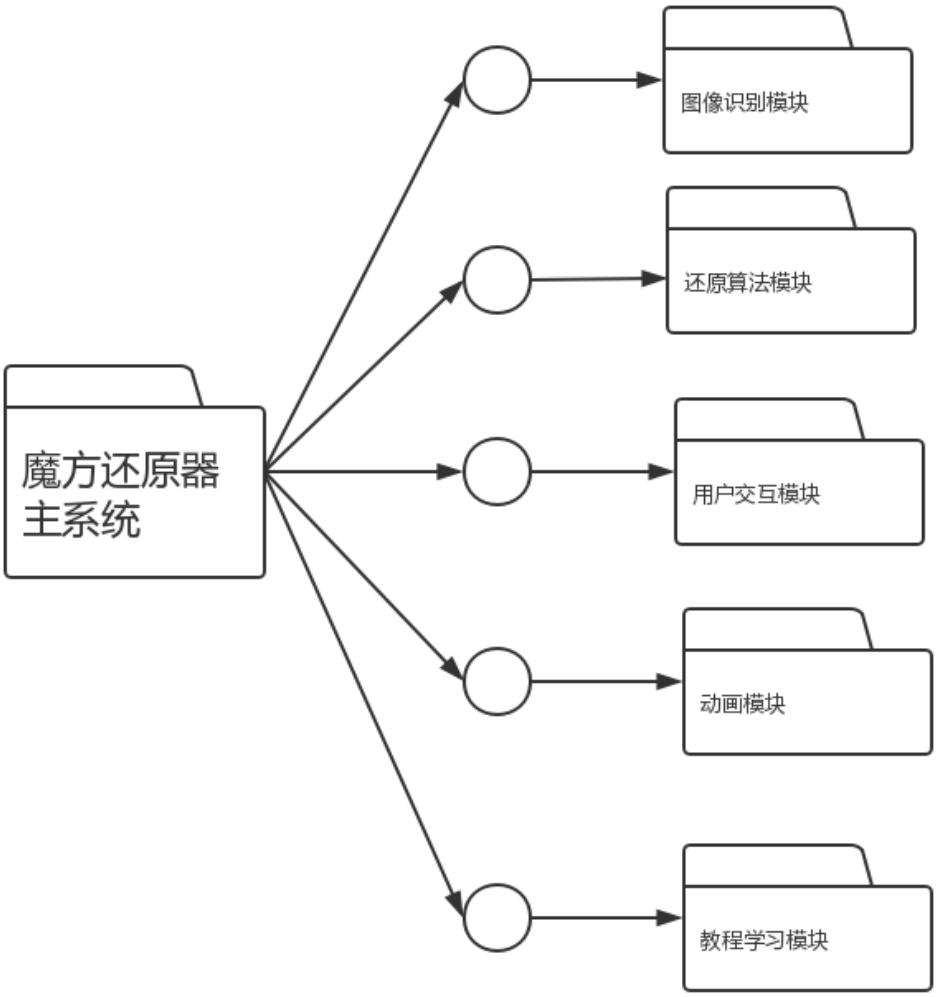


用户在主界面单击“打开相机”按钮进入拍摄界面，将手中的魔方与右侧界面方格对齐后单击“拍摄本面”，系统调用 face_read()函数获取由照片生成的颜色矩阵，并传给 CheckCube 类。然后在左下角使用 CheckCube 类中的按钮方法修改颜色错误的块（如果有），再单击“修改下方颜色后确认”按钮完成本面拍摄，调用 CheckCube 的 buttonColors()函数将 CheckCube 记录的单面颜色矩阵传回 UI 类。用户可以不断重复此过程，直到所有面都至少录入一次（每个面以最后录入的数据为准）。



用户拍摄完六个面后单击“关闭相机”按钮返回主界面，此时 UI 类将记录了六面颜色信息的矩阵传给 GLDemo 类，再由 GLDemo 类传给 Cube3 类完成魔方初始状态的存储。用户单击“解出还原步骤”按钮，则系统调用 Cube3 类的 full_cfop() 函数，返回一个完整的还原动作序列给 GLDemo。若此时系统处于可播放状态，则 GLDemo 类根据其接收到的动作序列播放还原动画。

五、 子系统设计



六、 系统实现

6.1 OpenCV 实现图像提取

a. 对魔方每一面进行色块分割;

b. 对分割出来的色块进行颜色提取，使用 HSV 颜色空间+RGB 颜色空间共

同判断色块的颜色;

c. 对于指定配色方案的魔方识别度非常好，对于其他配色方案的魔方识别度可能有部分下降。

```
def get_color_HSVRGB(h, s, v, r, g, b):
    if (r==0 or b / r > 2) and b > g:
        return color['B']
    elif (r==0 or g / r > 2) and g > b:
        return color['G']
    if (b==0 or r / b > 2) and (g==0 or r / g > 2):
        if h > 15:
            return color['R']
        else:
            return color['O']
    elif h < 20 and s < 150:
        return color['W']
    elif h < 50:
        return color['Y']
    else:
        return color['W']
```

```
def face_read(img):
    '''
        input:
            image read by cv2.imread(pth)
            (the given img must be a square)
        return:
            a list of 9 integers?
                W:0 R:1 G:2 B:3 O:4 Y:5
            representing colors of the 9 stickers on the cube face
    '''
    colors = [0, 0, 0, 0, 0, 0, 0, 0, 0]
    size_len = len(img)
    s_size = int(size_len / 3)
    for i in range(9):
        l = i // 3 + 1
        c = i % 3 + 1
        sticker = img[(l-1)*s_size:l*s_size, (c-1)*s_size:c*s_size]
        colors[i] = sq_color_read(sticker, 0.7)
    name2color = {v: k for k, v in color.items()}
    return colors
```

6.2 算法核心：魔方存储、动作和 CFOP 还原法

a. 魔方使用 3*3*3 矩阵存储，矩阵中每一个数字代表魔方某个色块的颜色编号。

```
def __init__(self):
    """initialize the cube as solved state"""
    self.cube = []
    for i in range(0, 6):
        temp = [[i, i, i] for j in range(3)]
        self.cube.append(temp)
    self.unsolvable = False
```

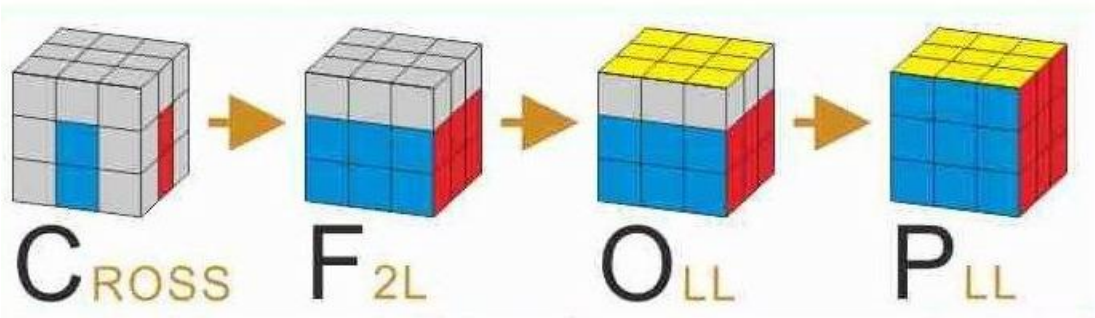
b. 魔方的每一个动作表示为矩阵的一种固定的变换方式。

```python
def R(self, times):
    times = times % 4
    for j in range(times):
        for i in range(3):
            t = self.cube[0][2-i][2]
            self.cube[0][2-i][2] = self.cube[4][2-i][0]
            self.cube[4][2-i][0] = self.cube[5][i][2]
            self.cube[5][i][2] = self.cube[1][i][2]
            self.cube[1][i][2] = t
        self.surface_clockwise(2)

def R_(self, times):
    times = times % 4
    self.R(4-times)

def L(self, times):
    self.rotate_around()
    self.R(1)
    self.rotate_around()
```

c. 把魔方还原分解为 4 个步骤: 底层十字、前两层四个槽位、顶面颜色、顶层位置。



```
def full_cfop(self):
    full_algorithm = []
    if self.cross_done() == False:
        full_algorithm.append(self.cfop_cross())
    if self.f2l_done() == False:
        full_algorithm.append(self.cfop_f2l())
    if self.oll_done() == False:
        full_algorithm.append(self.cfop_oll())
    if self.cfop_done() == False:
        full_algorithm.append(self.cfop_pll())
    return full_algorithm
```

d. 根据魔方的颜色分布模式，选择相应的公式。

6.2 OpenGL 实现魔方动画

a. 针对每个时刻的动画：
   - 首先确定此时魔方的转动状态（正在转哪个面，转了多少度）和颜色分布（6x3x3 的矩阵）；
b. 针对每个方块的动画：
   - 使用 glRotate 和 glTranslate 确定这个方块的位置和旋转角度；
   - 使用类似 glutSolidCube 的方法画出方块；
   - 再次使用用 glTranslate，画出方块上的贴纸。

```
def paintGL(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)

    for block in range(27):
        glPushMatrix()
        glLoadIdentity()
        gluPerspective(90, 1, 0.01, 100)
        gluLookAt(1, -1, 1, 0, 0, 0, 0, 1, 1)
        glRotate(self.view_x, 1, 0, 0)
        glRotate(self.view_y, 0, 1, 0)
        glRotate(self.view_z, 0, 0, 1)
        axis_x = self.currentmove[0]*self.rotate_ack[block][0] + \
            self.currentmove[1]*self.rotate_ack[block][1]
        axis_y = self.currentmove[4]*self.rotate_ack[block][4] + \
            self.currentmove[5]*self.rotate_ack[block][5]
        axis_z = self.currentmove[2]*self.rotate_ack[block][2] + \
            self.currentmove[3] * \
            self.rotate_ack[block][3]+self.currentrotation
        glRotate(self.currentangle*(axis_x+axis_y+axis_z),
                 abs(axis_x), abs(axis_y), abs(axis_z))
        glTranslate(0.3*self.block_pos[block][0], 0.3 *
                    self.block_pos[block][1], 0.3*self.block_pos[block]
        glColor3f(0, 0, 0)
        self.mySolidCube(0.28)
        glTranslate(0.02*self.block_pos[block][0], 0, 0)
        glColor3ubv(self.color_scheme[self.cube3.cube[self.block_color[
                                      [self.block_color[block][0][1]][s
        self.mySolidCube(0.26)
        glTranslate(-0.02*self.block_pos[block]
                    [0], 0.02*self.block_pos[block][1], 0)
        glColor3ubv(self.color_scheme[self.cube3.cube[self.block_color[
                                      [self.block_color[block][1][1]][s
        self.mySolidCube(0.26)
        glTranslate(
            0, -0.02*self.block_pos[block][1], 0.02*self.block_pos[bloc
        glColor3ubv(self.color_scheme[self.cube3.cube[self.block_color[
                                      [self.block_color[block][2][1]][s
        self.mySolidCube(0.26)
```

6.4 PyQt 编写 GUI

a. 使用 Widget 控件;
b. 通过添加 layout 把各个 Widget 放置在窗口中;

```
class Ui_MainWindow(QtWidgets.QWidget):
    text = "start"

    def __init__(self, parent=None):
        super(Ui_MainWindow, self).__init__(parent)
        self.timer_camera = QtCore.QTimer()
        self.timer_shot = QtCore.QTimer()
        self.timer_shot.timeout.connect(self.time_shot)
        self.cap = cv2.VideoCapture()
        self.check_color = CheckCube()
        self.check_color2 = CheckCube()
        self.CAM_NUM = 0
        self.set_ui()
        self.slot_init()
        self.__flag_work = 0
        self.x = 0
        self.shot_count = 0
        self.mat = [[], [], [], [], [], []]
        self.face_stored = 0

    def set_ui(self):
        self.__layout_main = QtWidgets.QHBoxLayout()
        self.__layout_fun_button = QtWidgets.QVBoxLayout()
        self.__layout_data_show = QtWidgets.QVBoxLayout()
        self.__layout_gl = QtWidgets.QVBoxLayout()
        self.__layout_gl_button = QtWidgets.QVBoxLayout()
        self.__layout_gl_rotate = QtWidgets.QVBoxLayout()
        self.__layout_gl_rotate_sub = QtWidgets.QHBoxLayout()
        # 摄像头
        self.button_open_camera = QtWidgets.QPushButton(u'打开相机')
        self.button_close = QtWidgets.QPushButton(u'退出')
        self.button_shot = QtWidgets.QPushButton(u'')
        self.button_open_camera.setMinimumHeight(50)
        self.button_close.setMinimumHeight(50)
        self.button_shot.setMinimumHeight(50)
        self.button_close.move(10, 100)
        # 3D Cube
```

c. Widget 之间可以互相调用方法进行交互（按钮形式）。

```
def shot(self):
    if self.timer_camera.isActive() == False:
        return
    ret, frame = self.cap.read()
    shot_count = self.shot_count
    self.check_color.is_scan = not self.check_color.is_scan
    shot_count += 1
    if (shot_count == 13):
        shot_count = 1
    self.button_shot.setText(u'修改下方颜色后确认')
    if (shot_count % 2 == 0):
        self.button_shot.setText(u'拍摄本面')
        self.mat[self.check_color.buttonColors()[1][1]] = copy.deepcopy
            self.check_color.buttonColors())
        self.button_open_camera.setText(self.faces_remaining())
    frame = frame[100:380, 230:510]
    frame = cv2.flip(frame, 1)
    face_read(frame)
    cv2.imwrite("capture_python.png", frame)
    self.shot_count = shot_count
```

七、 软件测试

7.1 测试目的

测试目的是对魔方求解器软件进行测试分析，考察该软件功能应用范围和缺陷，对各子系统测试的报告。通过测试，确保各子系统的功能、可用性，鲁棒性等符合软件的设计要求，满足用户的使用要求。通过分析测试过程中错误的产生原因和分布特征，可以帮助管理者发现当前软件开发过程的缺陷，以便在系统升级时加以改进。

7.2 测试范围及流程

本报告从方法到整体，分步测试了本项目的绝大部分代码的鲁棒性和正确性，通过手动编写代码和构造输入的方式，测试了魔方求解器各功能的实

现情况，并对其是否满足需求分析中提出的各项要求进行了检测。

测试遵循以下流程：

i. 对部分子系统和方法（主要是对魔方类（Cube3）和图像识别类（FaceReader））进行测试，确保方法实现的正确性和鲁棒性。

ii. 对各个功能单元进行正确性和鲁棒性测试。

iii. 完整运行软件源代码，对功能单元之间接口和相互作用进行测试。

部分关键子系统和功能测试除黑盒测试还有针对性地加上了白盒测试。先根据代码逻辑设计特定输入进行白盒测试，再使用更通常的输入进行黑盒测试。

7.3 测试计划及执行情况

7.3.1 子系统和方法

所在单元	测试函数	预期功能	测试类型	测试输入	结果	改进意见
FaceReader	get_mean_RGB / get_mean_HSV	获得指定图像区块的平均RGB/HSV值	鲁棒性	随机噪点	输出正确，符合需求	无
				只有一个像素点大小的图像		
				大小为0的图像	报错	在调用时规避过小图像
			正确性	渐变色块（正常输入）	输出正确，符合需求	无
	get_color_HSVRGB	根据指定图像区块的平均RGB与HSV值判断区块颜色	根据算法逻辑预测出的可能出问题的输入	偏色光背景（红光）	颜色识别错误	在软件使用说明中加上对应提示如“需将魔方放置于画面正中并占满方框”。后续改进见注
				魔方位置在图像角落		
				画面中有较大异物遮挡		
				图像中魔方较小		
			鲁棒性	画面中有较小异物遮挡	输出正确，符合需求	无
			正确性	白光，魔方置于画面正中并占满屏幕，无遮挡		
	face_read	图像分块，处理得到的颜色结果	图像分块功能	图像边长不为三的整数倍（7）	溢出部分被舍弃	最多涉及两个像素行，影响不大，仍反馈给设计人员，待后续改善
			鲁棒性	长方形照片（正常照片用PS拉伸得到）	错误切分	在调用时事先切分为正方形
			正确性	一张正常魔方照片	输出正确，符合需求	无
	sq_color_read	分块后图像切边	正确性	一张正常魔方照片		
				一张略歪魔方照片		
			鲁棒性	只有一个像素点大小的图像	返回空对象	反馈给设计人员，待后续改善
Cube3	__init__	初始化	正确性	无	功能满足需求	无
	input_scramble,input_raw,print_cube,algo	测试时输出中间结果	仅设计人员调试用，无需测试	无		
	scramble	随机打乱魔方	正确性	无		
	cross_done,f2l_done,oll_done,cfop_done	检验魔方还原阶段	正确性	魔方状态为下一阶段		
				魔方状态为本阶段恰好完成		
				魔方状态为当前阶段未完成		
				魔方状态为上一阶段恰好完成		
				魔方状态为上一阶段未完成		
			鲁棒性	魔方状态为不可能出现的状态	返回false	设计未明确处理方式，和设计人员反映后在调用前添加代码规约输入。
	cfop_cross,cfop_f2l,cfop_oll,cfop_pll,f2l_eR等15个转动方法	实现单阶段还原	正确性	魔方状态为未完成状态	功能满足需求	无
			鲁棒性	魔方状态为当前不可能出现的状态	报错	反馈设计人员进行异常处理。
		实现魔方转动	正确性	随机输入	功能满足需求	无
	surface_clockwise,surface_anticlockwise,	实现单面旋转				
UI_MainWindow	所有	UI				
GLDemo						
CheckCube						

注释 1：考虑添加环节，先让机器识别用户环境下的各面颜色。但会损害可用性。这是需求分析中没有考虑到的问题。

7.3.2 功能单元

功能单元	预期功能	测试类型	测试输入	结果	改进意见
FaceReader	图片识别	正确性	正常魔方图片	正确	无
		鲁棒性	其他图片（如人像）	错误输出，但未崩溃	使用说明中重申正确使用规范
			魔方未对准或过小		
			单色背景光		
Cube3	魔方状态，操作与还原	正确性	魔方转动	正确	无
			请求还原		
			请求打乱		
			状态检测		
		鲁棒性	不存在的魔方状态下的状态检测	认定为完全打乱状态	设计时未考虑识别错误时出现的此类情况。反馈设计人员后代码和设计返工。
			不存在的魔方状态下的请求还原	死循环后跳出	
UI_MainWindow	UI 主界面	功能	随机输入	正确	无
GLDemo	渲染魔方				
CheckCube	手动调整魔方	测试用，无需检测			

7.3.3 整体测试

测试类型	测试输入	结果	改进意见
功能	拍照输入魔方状态	错误率略高	调整 get_color_HSVRGB 中判定算法
	魔方求解	实现目标	无
	分步求解		
	魔方图像转动		
	随机打乱		
鲁棒性	高速连点按钮	卡死	已修复
	按钮不同顺序组合	正常	无
	执行按钮动作时关闭	正常关闭	
	错误的魔方（贴了 10 块白块）	卡死	已修复
	要求求解已复原的魔方	正常，弹出提示告知魔方已还原	无

7.4 测试总结

实现状况整体较好，局部存在问题。设计过程没有预料到的主要是连点动作和魔方当前状态不可能存在的两个 bug（当然，更准确点说也不是 bug，只是输入容

错性能的两个问题)。反馈后得到了比较好的解决。

八、 项目总结

8.1 项目特色陈述

大部分课程项目是基于数据库和前端结合的，以“增删查改”为核心的传统实现，本项目则从实际运用出发，结合组员兴趣爱好（魔方，算法编写，图像识别）编写,是一个对小组合作和个人 code 能力以及写软件文档能力有综合提升的项目。

算法：还原算法是本项目的重点，也是 code 阶段的难点;

图像识别：

图像识别考虑了多情境，多光源下的识别，调试复杂且困难，测试耗时极长，但最终可成功识别色块,。

动画与前端：

前端调用了框架,但动画的立体实现是本组自行设计的,也是 code 的难点之一。

用户交互设计：

考虑到软件的多功能性以及提升用户使用满意度，项目进行了多次更改，增添了可间断还原过程，查看还原上下步骤等功能，改进了前端展示界面。

防错机制:

为防止识别部分出现不可预估的问题，有手动调试输入的功能。

综上，本项目综合了前后端，算法，图像识别等模块，是一个综合的，可使用的，实用性强，且具有良好用户交互性质的软件。提出了对组员写普通“增删查改”功

能的项目之上的能力要求，最终测试通过，实现预期效果，也仍有改进空间。

8,2 项目工作分工

姓名	工作
刘英杰	1. 提出项目想法，设计软件功能； 2. 实现魔方还原算法； 3. 代码融合与发块，文档编写；
徐靖	1. 实现前端界面； 2. 实现需求分析中的软件功能； 3. 图像识别处理；
徐原	1. 实现软件功能，设计软件教程； 2. 编写项目文档，管理开发流程； 3. 软件图像动画显示处理；
秦李煜	1. 实现软件教程需求； 2. 前端调试； 3. 编写文档；
王浩翔	1. 编写测试用例 2. 测试软件功能； 3. 改进需求分析中的软件功能；

8.3 开发流程流程总结图