中国科学院大学

University of Chinese Academy of Sciences

大作业结题报告

课程名称：高级软件工程

报告题目：魔方训练营

学生姓名：段旭 江华禧 恽星彤

班　　级：708

指导教师：罗铁坚

完成日期：2019 年 12 月 11 日

二 0 一九年十二月十一日

目录

1.项目背景

1.1 项目意义

众所周知，魔方又称鲁比克方块（Rubik's Cube），是一个匈牙利建筑学教授鲁比克·埃尔内发明的一种机械益智游戏。它除了能锻炼手眼协调能力、空间解构能力、记忆力，还能促进脑部活跃、预防老年痴呆，甚至可以成为一门非常有用的撩妹技能。

但是，魔方的入门具有一定难度，刚开始学习魔方的过程中，需要记忆大量的公式，同时需要有非常强大的手眼结合能力，而这些难题始终困扰着大部分初学者。随着移动互联网和软件行业日新月异的发展，通过开发一款在线的软件来帮助魔方初学者学会还原魔方，甚至提高高级魔方玩家的还原魔方水平，无疑成为一种非常高效而方便的途径。

1.2 研究现状

目前市场上的相关魔方学习软件良莠不齐，大部分软件仅仅只能自动还原魔方，而不能教会用户如何学会还原魔方，以及满足用户对魔方软件的其他有意思的需求。

例如顶级学术期刊 nature 上发布的一篇关于利用人工智能的方法还原魔方的最新算法的论文《Solving the Rubik's Cube with Deep Reinforcement Learning and Search》中，给出了基于此原理开发的一个魔方还原网站。它基于强化学习+加权路径查找的方法构建一个名为 DeepCubeA 的人工智能模型，可以在不需要任何领域的专业知识或人类游戏指导的情况下，并且只花了人类所需时间的 60%便找到了破解魔方——每一面都显示为单色的最短路径。

但是，这个网页目前只是实现了随机打乱以及以人工智能的方法还原魔方的功能，并且存在以下 4 个缺点：

①无法输入魔方初始状态；

②无魔方转动 3D 动画,导致操作显示很不直观；

③随机状态不全，并不能达到非常实用的效果；

④不能实现所有状态都利用此方法还原。

1.3 项目方向

根据市场上已有的产品存在的缺陷，我们开发了一款名为魔方训练营的软件，它是一个在线的 Web 网页。用户可以通过在线的方式网页上实现学习如何还原魔方、如何提高玩魔方的水准等很多有意思的功能。

2.竞品调研

当今市场已存在一些魔方求解、魔方教学应用，其功能不尽相同。本章对市场上的两款魔方应用进行了调研，总结出这些应用的功能、优点与缺点，从而为本项目的开发提供思路。

2.1 魔方求解器

魔方求解器是一款专注于提供魔方求解服务的 Web 应用，其为用户提供输入魔方状态的接口，并通过后台的算法对魔方进行求解，并将还原步骤与过程展示给用户。该 Web 应用的 Url 为 https://rubiks-cube-solver.com/。该应用的 UI 界面如下图所示：

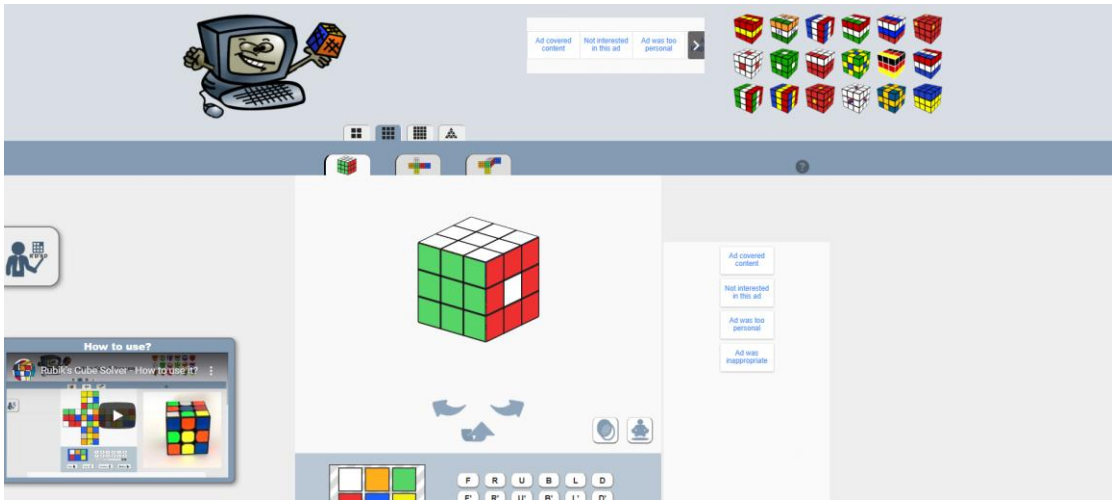

图 1 魔方求解器界面

将充分体验与分析，总结该项目的功能、优点与缺点如下表所示：

表 1 魔方求解器竞品分析

功能	优点	缺点
1）点击或转动色块输入魔方状态； 2）求解魔方； 3）展示魔方求解步骤与过程	1）除三阶魔方外，还提供二阶、四阶和三角魔方的求解； 2）有较详细的使用说明；	1）页面不够美观； 2）通过点击色块依次改变颜色的方式输入状态，繁琐； 3）解法单一；

2.2 DeepCube

DeepCube 求解应用是 DeepCube 论文中为展示 DeepCube 效果所开发的 Web 应用，其

可以提供随机打乱魔方状态的接口，然后通过 DeepCube 算法对魔方进行求解，并展示还原步骤。该 Web 应用的 Url 为 http://deepcube.igb.uci.edu/。该应用的 UI 界面如下图所示：

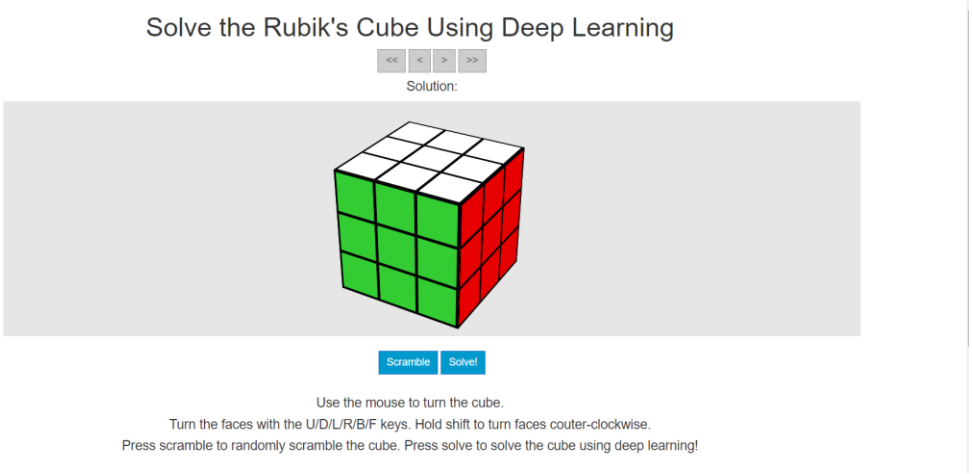

图 2 DeepCube 应用界面

将充分体验与分析，总结该项目的功能、优点与缺点如下表所示：

表 2 DeepCube 竞品分析

功能	优点	缺点
1）随机打乱初始状态； 2）使用 DeepCube 求解魔方； 3）展示魔方求解步骤	1）DeepCube 求解迅速；	1）无法让用户手动输入魔方初始状态； 2）魔方交互性差； 3）解法单一；

3.需求分析

经过大量的调查研究，我们根据用户提供的数据，依据不同的需求将用户分为 3 类。

3.1 第一类目标用户——魔方初学者

第一类目标用户是魔方初学者，在这之前从来没有接触过魔方的还原方法，或者对于如何还原魔方只有概念上的认识，而没有实践经验。对于这类用户，我们使用公式法中最简单的层先法来满足他们的需求，帮助他们实现魔方还原的快速入门。

公式法以一定的规则实现解魔方。其可以告诉初学者，在什么情况下，如何做，能够获得什么样的状态，从而再如何做，最后还原整个魔方，其有利于初学者对解魔方建立最初的认知。常见的公式法包括层先法、角先法、CFOP 公式法等。

层先法将魔方分为三层：底层、中层、顶层，分层复原，原理如下图所示。

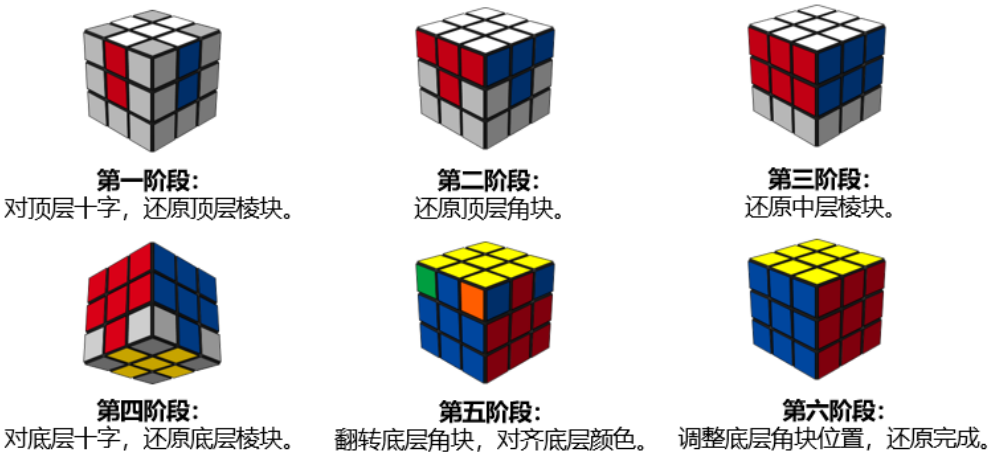

图 3 层先法步骤

3.2 第二类目标用户——想突破自我的魔方高手

第二类目标用户是已经精通普通公式法还原魔方的高手，但是他们不满足于现状，想突破自我，达到利用更短的时间、更快地还原魔方，甚至能参加魔方还

原比赛的水平。对于这类用户，我们使用两种方法实现他们的需求，可供他们自己参考选择。一种是公式法中的速拧公式法 CFOP 法，另一种是 nature 论文《Solving the Rubik's Cube with Deep Reinforcement Learning and Search》给出网站中的人工智能还原魔方模型 DeepCubeA 法。

3.2.1 CFOP 速拧公式法

CFOP 的意思是我们要分四步还原魔方，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）。（引用自百度百科）

原理如下图所示

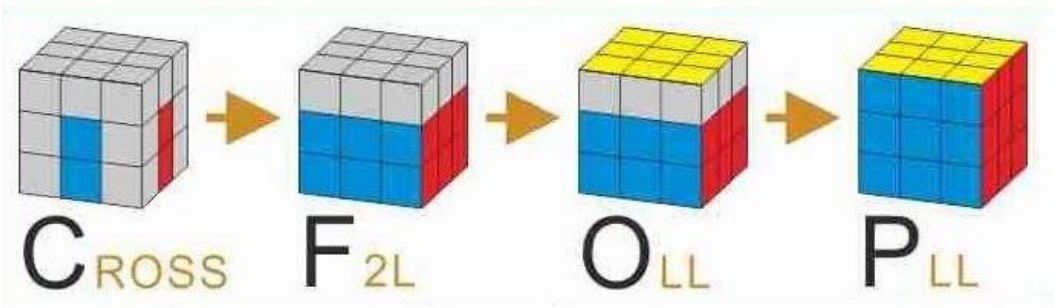

图 4 CFOP 步骤

3.2.2 DeepCubeA 模型

DeepCubeA 模型通过强化学习和加权路径查找的方法，查找到达目标状态的最短路径，以此来找到还原魔方的最快解法。大概有以下 3 个特性：

①使用近似值迭代训练深度神经网络，来近似一个输出达到目标成本的；

②训练从目标状态开始并随机反函数向进行移动而获得的状态，训练后，将学习的成本函数用作启发式函数；

③使用加权 A*搜索。

DeepCubeA 找到对称状态的对称解决方案的示例如下图所示

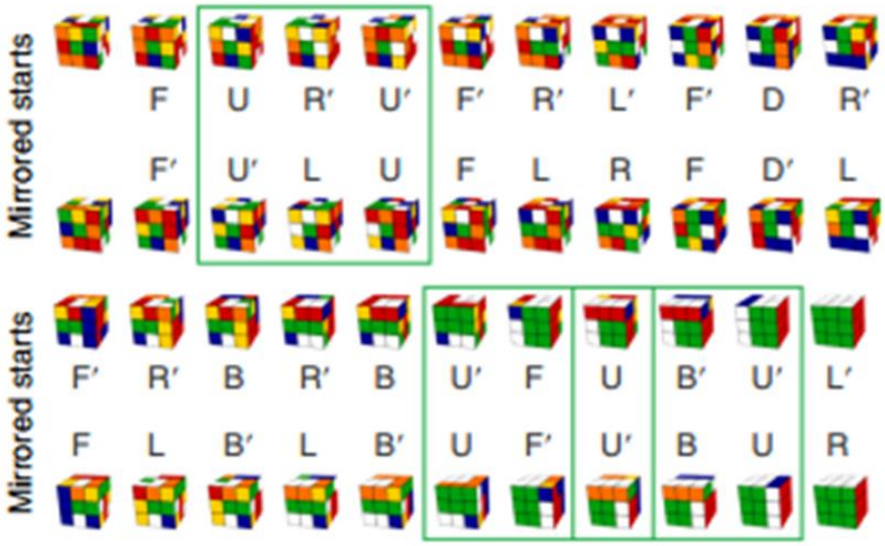

图 5 DeepCubeA 状态示意图

效果：100%地解决了所有测试项目，在 60.3％的时间内找到到达目标状态的最短路径。该算法也适用于其他组合游戏，如滑动拼图、熄灯和推箱子游戏。

3.3 第三类目标用户——需要在线玩魔方的用户

第三类目标用户是手上没有实体魔方，但是想玩魔方的用户。这类用户或者没有途径购买魔方，或者因为特殊情况不便使用实体魔方，但是迫切需要一个玩魔方的途径。对于这类用户，我们设计了一个交互的 3D 魔方组件，帮助他们实现没有魔方也通过网页在线玩魔方的效果。

这种魔方组件必须满足 3 个条件：

（1）交互性：该魔方组件必须可以像实物一样按用户所想方向旋转。

（2）合法性：该魔方组件的每个色块必须符合一定的约束条件，使魔方的每个可能出现的状态都是合法的。

（3）可输入性：该魔方组件能在不用旋转的情况下，直接输入初始魔方状态。

3.4 具体功能需求

3.4.1 随机打乱+解魔方+步骤显示

在 nature 论文《Solving the Rubik's Cube with Deep Reinforcement Learning and Search》中实现了 3 个功能，分别是随机打乱魔方顺序、利用 DeepCubeA 模型进行魔方还原以及魔方还原过程中的步骤显示。所以我们把这 3 个功能作为魔方训练营项目要实现的第一个需求。

3.4.2 初始状态输入

如果用户需要通过魔方训练营学习如何还原魔方，那么就必须允许用户能够将魔方的初始状态输入到网页上，从而为实体魔方提供一个可供参考的模型。即，这个需求是实现用户学习还原魔方，乃至水平进一步提升的基础。

3.4.3 魔方合法性验证

在用户对魔方状态进行输入时，难保不会出现输错的情况。一旦出现错误的魔方状态，就会导致整个魔方求解过程的失败。此时，我们设计约束条件，判断输入魔方状态的合法性就显得尤为重要。

3.4.4 魔方 3D 转动效果

在用户使用魔方训练营进行学习时，如果网页上仅仅展示的是一个二维的画面，首先魔方状态无法正常输入，其次没有 3D 效果，整个操作会显得非常不直观，用户就很难通过网页上魔方组件显示的步骤进行魔方的还原学习。所以我们添加了通过拖拽、点击等方式实现魔方转动的 3D 视觉效果的功能需求。

3.4.5 四种解魔方的方法

根据初学者和魔方高手这两类不同的用户的需求，我们提供了 4 种不同的方法解决魔方的求解还原问题。初学者适合通过公式法中最简单的层先法来进行魔方还原的入门学习，魔方高手可以在魔方速拧公式 CFOP 法、Kociemba 法和 DeepCubeA 模型法 3 种解法中找到适合自己的快速提高魔方水准的方法。

3.4.6 单步执行及回退功能

用户需要通过这个软件进行学习如何还原魔方，则魔方的整个还原过程就显得尤为重要，魔方的还原速度也不宜太快。所以需要达到的需求就是，允许用户逐步运行解法，方便用户对还原步骤进行回退，从而提供用户对每步的思考时间，真正达到学习魔方还原的效果。

4.系统设计与实现

项目架构整体分为上层应用层和基础服务层两部分，上层应用层主要用于实现魔方训练营的基础功能，基础服务层主要为上层应用层提供底层服务和开源架构。总体架构如下图所示：

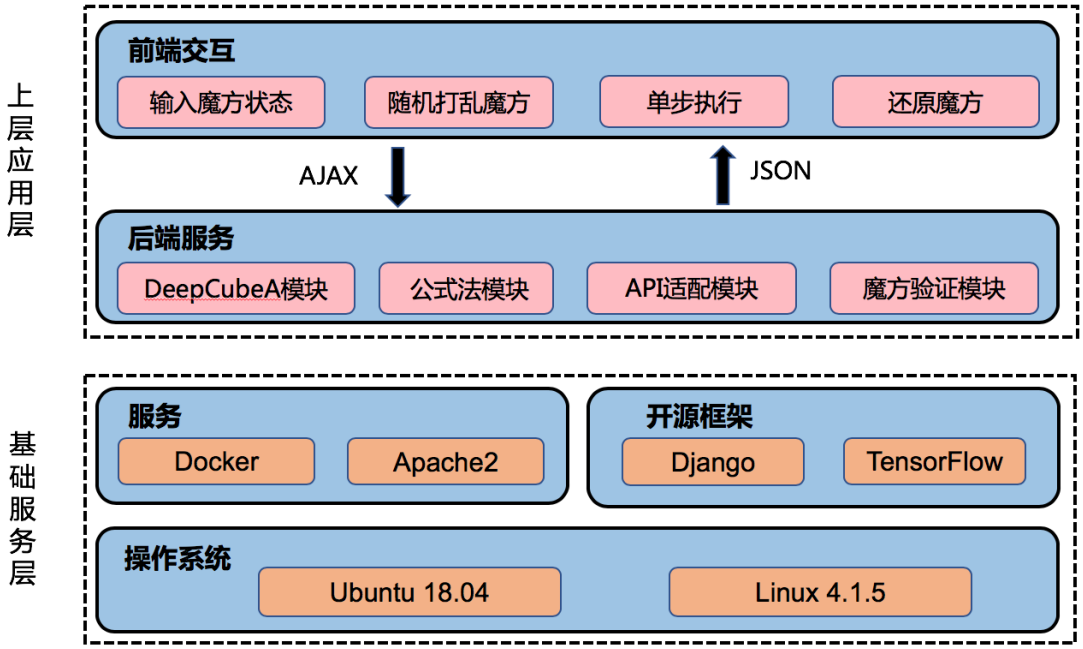

图 6 系统架构图

4.1 基础服务层

基础服务层主要分为服务、开源框架和操作系统三个部分。

4.1.1 服务部分

服务部分主要由开源应用容器引擎 Docker 和 Web 服务器 Apache2 构成。

Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 linux 服务器，也可以实现虚拟化。使用 Docker 发布服务可以不用担心服务器的运行环境，所有的服务器都自动分配 docker，自动部署，自动安装，自动运行，资源利用更加出色，管理更加便利。

Apache HTTP Server 是一个多模块化的服务器，经过多次修改，成为目前世界使用排名第一的 Web 服务器软件。可以运行在几乎所有广泛使用的计算机

平台上。Apache 服务器的特点是使用简单，速度快，性能稳定，可以作为负载均衡及代理服务器来使用。

4.1.2 开源框架部分

开源框架部分主要使用了 Django 和 TensorFlow 两个开源框架。Django 是一个基于 Python 语言的 Web 框架，它是一套用于帮助开发交互式网站的工具，能够响应网页请求，还能让开发者更轻松地读写数据库、管理用户。我们用 Django 来开发用于展示魔方训练营主要功能的 Web 网页。

TensorFlow 官网上称 TensorFlow 是一个用于机器智能的开源软件库，是谷歌研发的第二代人工智能学习系统，可被用于语音识别或图像识别等多项机器学习和深度学习领域。我们用 TensorFlow 来实现 DeepCubeA 模型还原魔方，实现第二类用户的需求。

4.1.3 操作系统部分

我们的后端服务器搭建等工作是在 Linux 操作系统发行版 Ubuntu 18.04 下完成的。

4.2 上层应用层

上层应用部分采用前后端分离的设计，前端负责与用户的交互，后端负责具体的服务。前端有：输入魔方状态模块，随机打乱魔方模块，单步执行模块和还原魔方模块。后端有魔方验证模块，API 适配模块，DeepCubeA 模块、公式法模块和 K 算法模块。

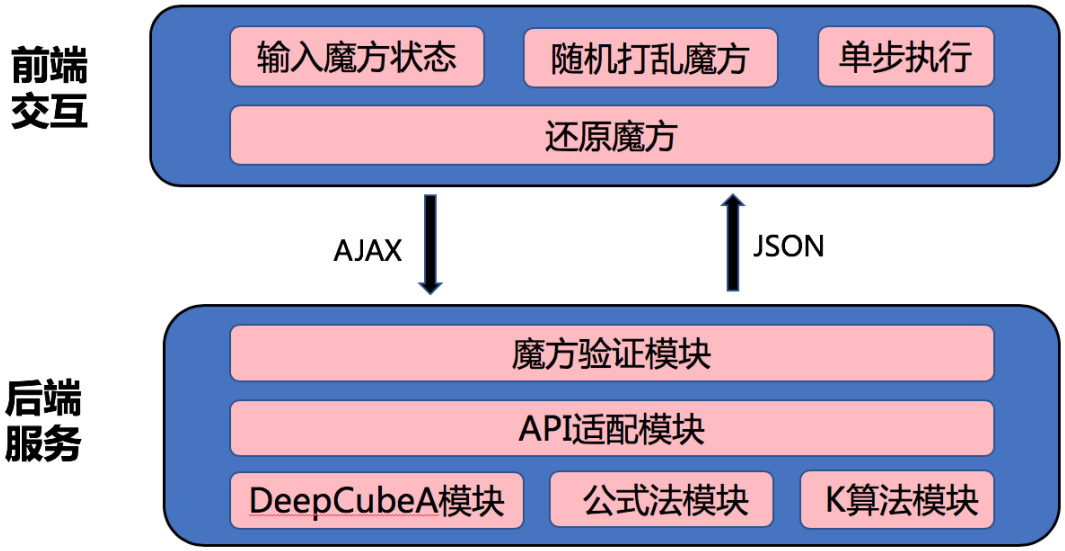

图 7 项目架构图

4.2.1 前端页面

①前端设计

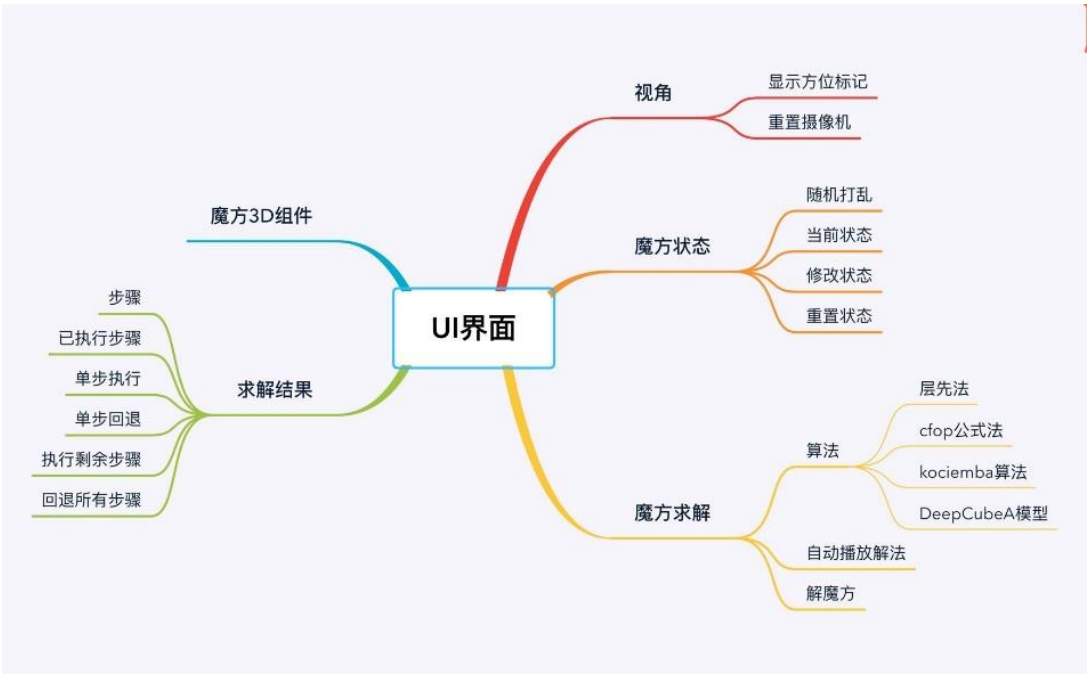

图 8 UI 设计导图

最终 UI 在网页上呈现的是一个魔方组件以及实现相应操作的工具栏。

整个工具栏中包括视角、魔方状态、魔方求解以及求解结果这四部分。

②视角栏

视角栏主要包括显示方位标记和重置摄像机部分，用于调整用户视角。

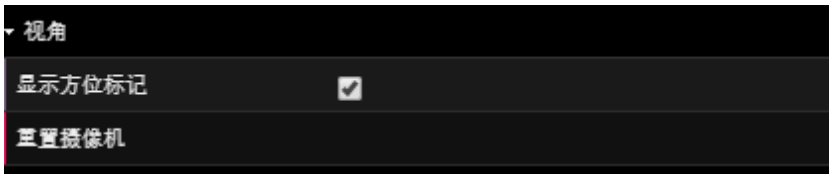

图 9 视角栏

③魔方状态栏

魔方状态栏主要用于对当前魔方状态进行显示及修改重置。

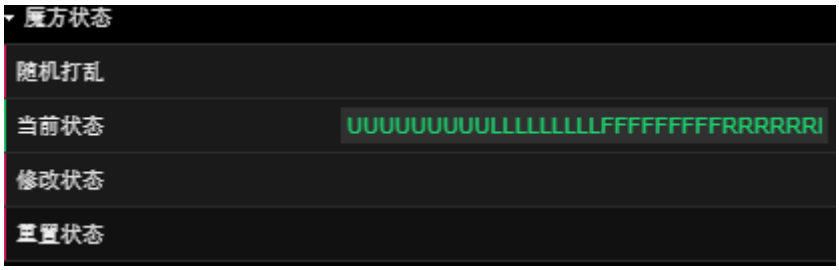

图 10 魔方状态栏

④魔方求解栏

魔方求解栏主要包括四类魔方求解算法、解法播放功能以及解魔方按钮。

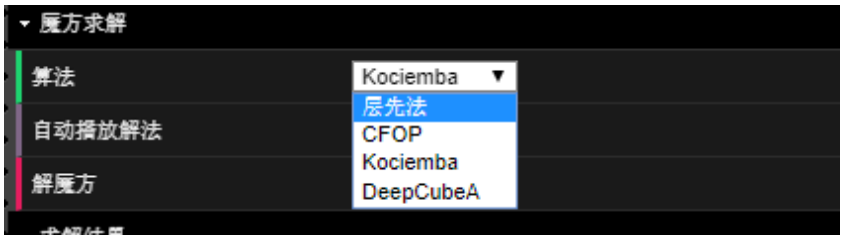

图 11 魔方求解栏

⑤求解结果栏

求解结果栏主要用于对魔方求解步骤进行显示，以及对魔方进行单步求解等功能。

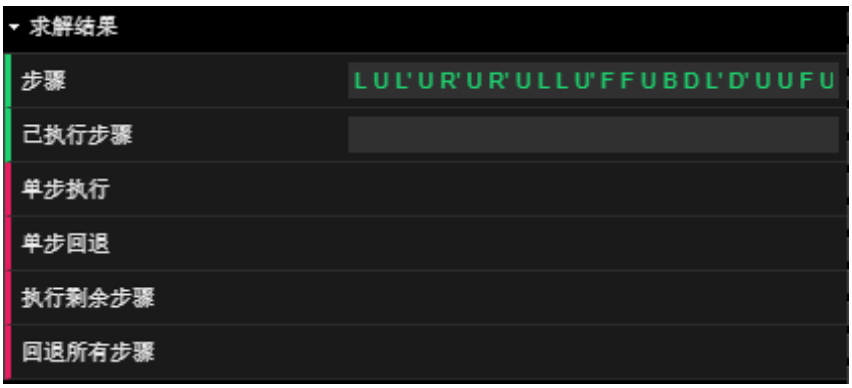

图 12 求解结果栏

⑥魔方组件

魔方组件展示的是一个 3D 的魔方，我们可以对它进行左键单击，输入手上的实

体魔方的状态，也可以按 U、B、L、F、D、R 键进行魔方转动，达到在线转魔方的效果。

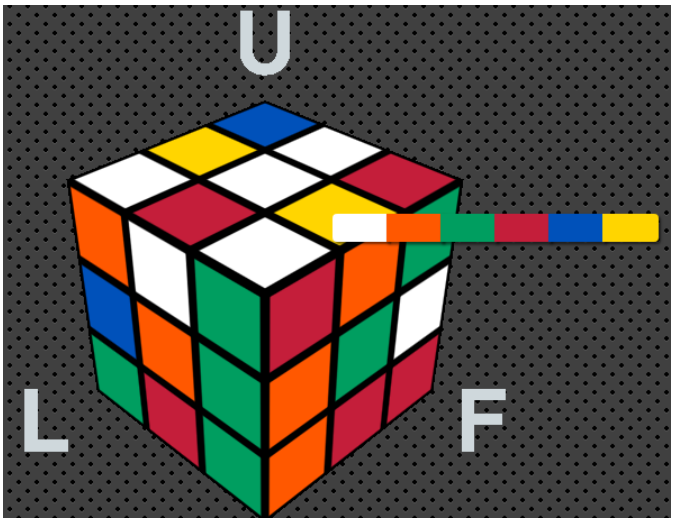

图 13 魔方组件

⑦实际 UI 界面显示

实际 UI 界面如下图所示

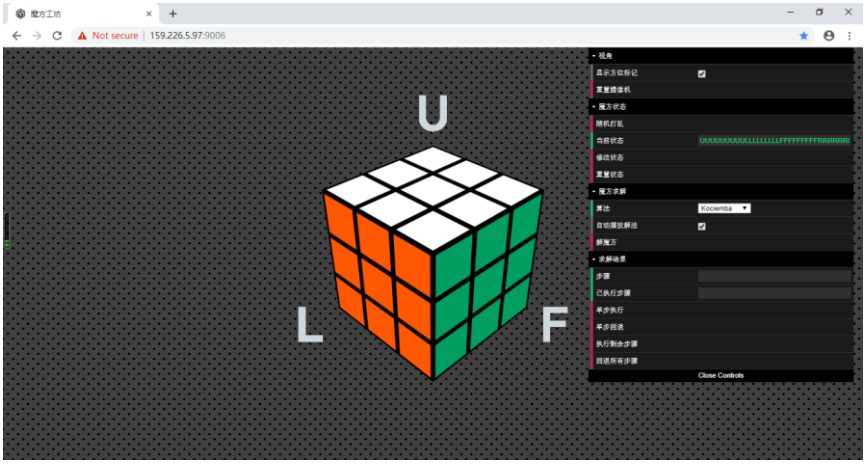

图 14 UI 界面效果图

4.2.2 接口设计

前后端通过一个接口进行交互，接口包括请求信息和返回信息两部分。请求信息部分设计如下图所示：

URL	请求方式	请求参数	数据类型	是否必须	说明
/solve	POST	state_str	string，长度为54	是	表示当前魔方各面的状态
		method_type	int	是	解法类型，0为使用层先法，1为使用CFOP算法；2为使用K算法

图 15 请求参数

返回信息部分设计如下图所示：

返回参数	数据类型	说明
code	int	状态代码
message	string	状态代码对应的文字描述信息
moves	list，元素类型为string	解魔方步骤

图 16 响应参数

4.2.3 魔方验证模块

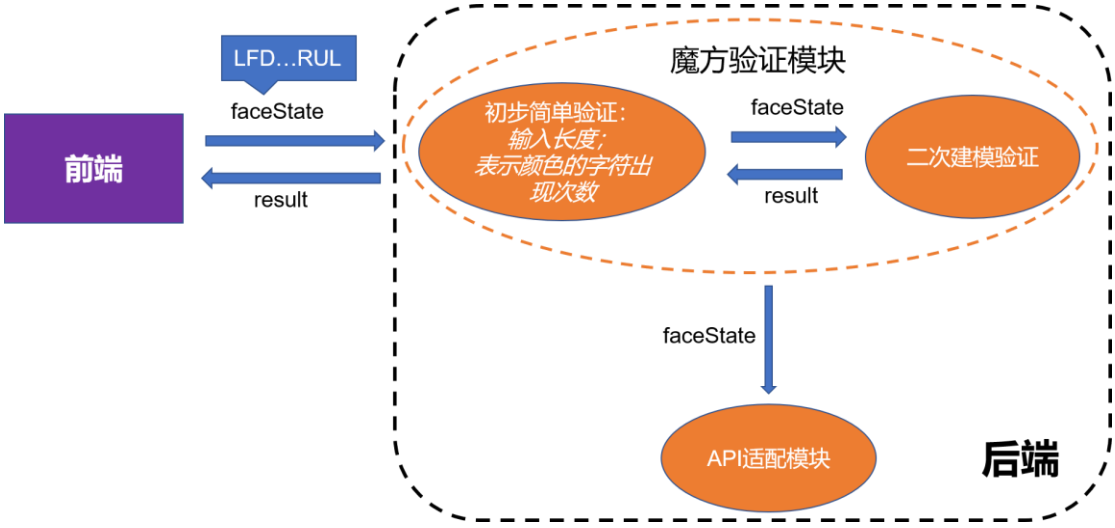

图 17 魔方验证模块数据流图

魔方验证模块用于验证魔方输入状态的合法性。

具体运行如下：

①前端发送魔方状态 faceState（形如：'ULFFFL...'）到魔方验证模块；

②对输入的魔方状态做一个初步验证，主要是对输入长度和表示颜色的的字符出现次数做一次初步验证；

③继续对 faceState 做二次建模验证，此次验证用于保证输入的魔方状态不会出现不符合实体魔方构造规则的情况；

④验证完成后将 faceState 发送至 API 适配模块，选择合适的算法对魔方求解后，发送至前端。

4.2.4 API 适配模块

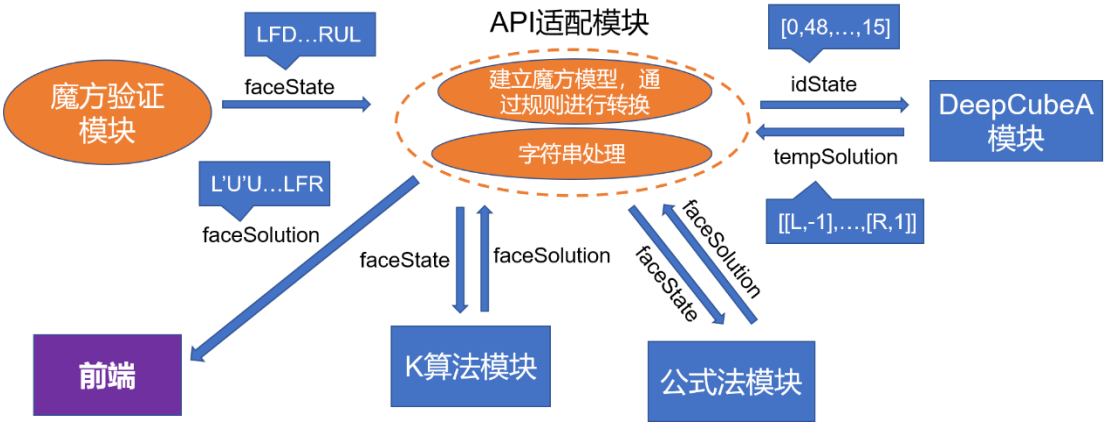

图 18 API 适配模块数据流图

API 适配模块主要用于建立魔方模型并转换，同时对字符串进行处理。

具体运行如下：

①魔方验证模块将验证通过的 faceState 发送至 API 适配模块；

②对传过来的状态进行建模、规则转换，变成 DeepCubeA 的输入格式，魔方状态 idState(形如：[23,5,8,43,...])；

③DeepCubeA 返回 tempSolution（形如：[['R', -1], ['D', -1], ['U', 1], ['R', 1], ['B', 1], ['L', -1], ['F', 1]']），API 适配模块对其进行字符串处理，得到 faceSolution。

④若不用 DeepCubeA 模块，则将 faceState 传入 K 算法模块，这里 K 算法模块是一个引入的库，它可以作为一个魔方求解模块进行使用，也可以作为一个中间模块给公式法模块提供服务，将数据转换为公式法模块需要的格式；

⑤选择对应的算法进行魔方求解后，得到的 faceSolution 传回 API 适配模块，再传回前端。

4.2.5 DeepCubeA 模块

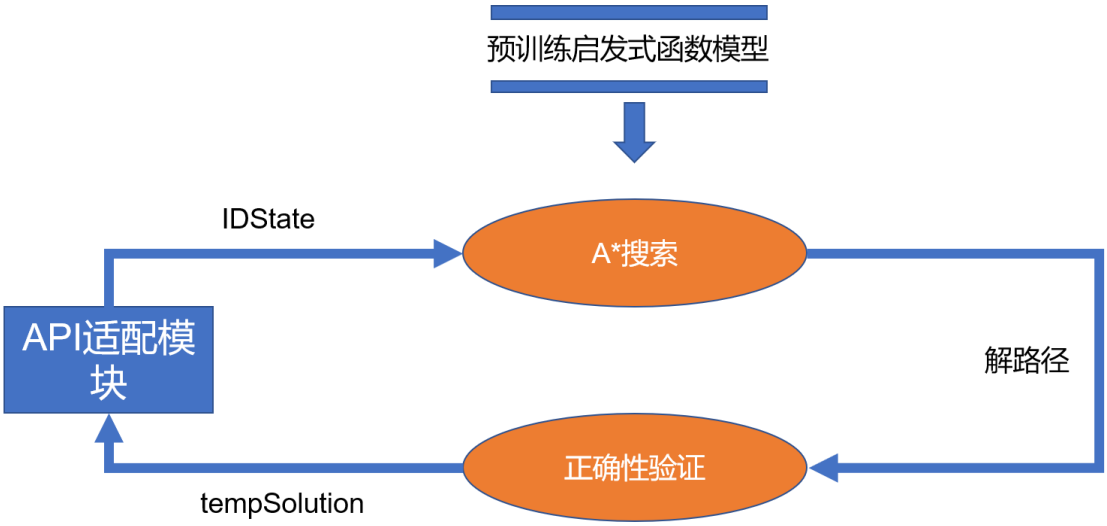

图 19 DeepCubaA 模块数据流图

DeepCubeA 模块用 DeepCubeA 模型，即强化学习+加权路径查找的方法进行魔方求解。

具体运行如下：

①首先加载训练后的启发式搜索函数模型；

②对 API 模块输入的 idState 进行 A*搜索；

③对解路径进行正确性验证，将解法 tempSolution 发回到 API 适配模块。

4.2.6 公式法模块

4.2.6.1 层先法

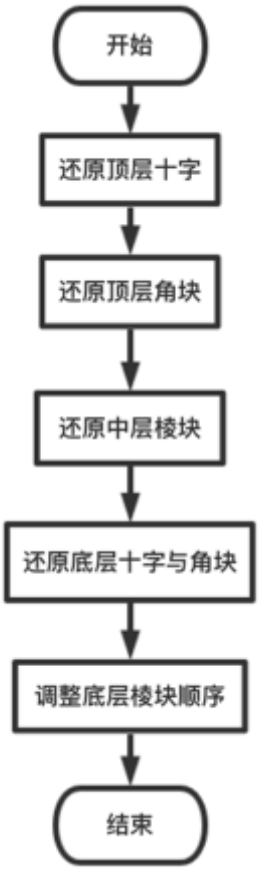

图 20 层先法模块流程图

层先法模块的功能主要是应用公式法中的层先法进行魔方求解，具体步骤如上流程图。

4.2.6.2 CFOP 法

图 21 CFOP 模块流程图

CFOP 法模块主要功能是用魔方速拧公式 CFOP 公式法进行魔方求解。具体步骤如上流程图。

4.2.7 Kociemba 算法模块

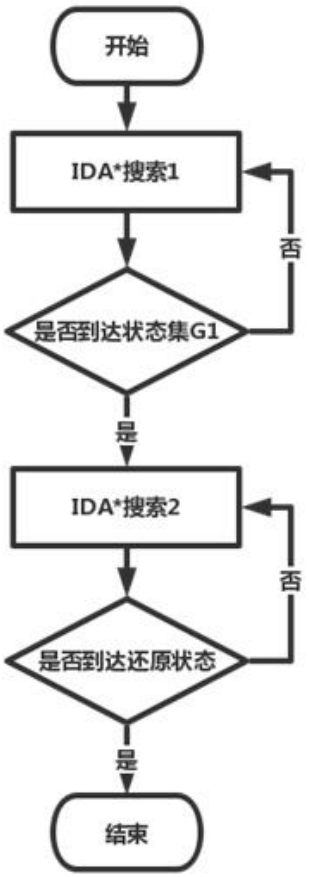

图 22 Kociemba 模块流程图

Kociemba 算法，又称为二阶段算法，是一个使用较短时间和较少次数还原魔方的算法。其原理是根据魔方的色块性质将魔方的状态集划分为两个子集 G1 和 G2。

状态集 G1 是为魔方的原始状态经过若干步规定以内的旋转所组成的状态集。该状态集中状态数量较小。

状态集 G2 表示以 G1 的所有状态为原始状态，经过若干次任意旋转后的状态集。

Kociemba 算法首先使用 IDA*搜索转换到状态集 G1，然后再使用另一个 IDA*搜索转换为原始状态。

具体实现步骤如上流程图。

5.预期目标及测试

魔方训练营项目着重实现魔方组件可视化以及对一般状态的魔方实现六面还原的相关功能，并根据项目申报书上的预期成果进行了相应的实现。具体完成情况如下。

5.1 功能测试

5.1.1 用户视角可视化调控

完成状态：已完成

需求描述：

用户可以自由切换方位标记显示，也可以在状态栏内调控还原初始状态时的用户视角。

测试用例及预期结果：

1.给显示方位标记栏打勾，魔方六面出现 U、B、L、F、D、R 六个字母；

2.给显示方位标记栏去勾，魔方六面 U、B、L、F、D、R 六个字母消失；

3.单击重置摄像头按钮，魔方视角回复到初始状态。

5.1.2 魔方状态调控

完成状态：已完成

需求描述：

用户可以随机打乱当前魔方状态，可以直接修改魔方状态，也可以重置魔方状态。

测试用例及预期结果：

1.单击随机打乱按钮，魔方随机旋转打乱状态；

2.在当前状态栏随机输入一个合法状态，并点击修改状态，3D 魔方组件自动显示对应状态模型；

3.单击重置状态，魔方回复到六面还原状态。

5.1.3 魔方求解调控

完成状态：已完成

需求描述：

用户可以根据自己的不同需求，选择不同的算法来求解魔方。

测试用例及预期结果：

1.算法栏选择层先法，进入层先法解魔方状态；

2.算法栏选择 CFOP 公式法，进入 CFOP 公式法解魔方状态；

3.算法栏选择 Kociemba 算法，进入 Kociemba 算法解魔方状态；

4.算法栏选择 DeepCubeA，进入 DeepCubeA 模型法解魔方状态；

5.给自动播放栏打钩，解魔方时自动播放求解魔方过程，直至魔方六面还原；

6.给自动播放栏去钩，解魔方时不播放求解过程，不显示魔方六面还原状态，只在求解结果中显示求解步骤；

7.点击解魔方按钮，判断魔方状态合法性，若合法，按照算法栏的指定解法解魔方。

5.1.4 单步执行调控

完成状态：已完成

需求描述：

用户可以对魔方的求解还原过程进行单步执行和单步回退等操作。

测试用例及预期结果：

1.给自动播放栏去勾，点击解魔方，同时点击单步执行，求解结果中显示魔方整体求解还原步骤，同时魔方执行一个求解步骤；

2.给自动播放栏去勾，点击解魔方，单步执行后，点击单步回退，魔方回退到一个执行步骤前的状态；

3.给自动播放栏去勾，点击解魔方，单步执行后，点击执行剩余步骤，魔方按照指定算法执行剩余步骤，直至解得魔方的六面还原状态；

4.点击回退所有状态，魔方回退到最初始的打乱状态。

5.1.5 魔方组件调控

完成状态：已完成

需求描述：

用户可以对魔方组件进行旋转拖动，也可以手动输入魔方状态。

测试用例及预期结果：

1.鼠标左键点击魔方色块，对应色块改成相应颜色；

2.鼠标左键长按选中魔方，拖动魔方，产生魔方按指定拖动方向旋转的 3D 效果。

3.键盘上按下 U、B、L、F、D、R，魔方对应平面逆时针旋转。

5.2 代码覆盖率测试

代码覆盖率（Code Coverage）是反映测试用例对被测软件覆盖程度的重要指标，也是衡量测试工作进展情况的重要指标。本项目利用 coverage 库对所有测试用例进行代码覆盖率测试，结果如下图所示：

Coverage report: 84%

Module ↓	statements	missing	excluded	coverage
RubikCubeWebApp/__init__.py	0	0	0	100%
RubikCubeWebApp/calculate_states/__init__.py	0	0	0	100%
RubikCubeWebApp/calculate_states/cube_string.py	16	0	0	100%
RubikCubeWebApp/calculate_states/enums.py	25	0	0	100%
RubikCubeWebApp/calculate_states/main.py	42	2	0	95%
RubikCubeWebApp/calculate_states/model/__init__.py	0	0	0	100%
RubikCubeWebApp/calculate_states/model/block.py	8	0	0	100%
RubikCubeWebApp/calculate_states/model/cube.py	91	4	0	96%
RubikCubeWebApp/calculate_states/rules.py	3	0	0	100%
RubikCubeWebApp/calculate_states/tests/__init__.py	0	0	0	100%
RubikCubeWebApp/calculate_states/tests/test_cases.py	3	0	0	100%
RubikCubeWebApp/calculate_states/tests/test_cube_string.py	8	0	0	100%
RubikCubeWebApp/calculate_states/tests/test_main.py	13	0	0	100%
RubikCubeWebApp/solver/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/cfop/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/cfop/cfop_solver.py	34	1	0	97%
RubikCubeWebApp/solver/cfop/cross.py	94	1	0	99%
RubikCubeWebApp/solver/cfop/f2l.py	157	28	0	82%
RubikCubeWebApp/solver/cfop/oll.py	36	5	0	86%
RubikCubeWebApp/solver/cfop/pll.py	45	9	0	80%
RubikCubeWebApp/solver/cfop/util.py	27	1	0	96%
RubikCubeWebApp/solver/deepcubea/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/deepcubea/environments/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/deepcubea/ml_utils/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/deepcubea/ml_utils/tensorflow_utils/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/deepcubea/scripts/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/deepcubea/solvers/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/deepcubea/solvers/cube3/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/kociemba/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/kociemba/kociemba_solver.py	10	0	0	100%
RubikCubeWebApp/solver/model/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/model/cube.py	322	54	0	83%
RubikCubeWebApp/solver/model/formula.py	305	94	0	69%
RubikCubeWebApp/solver/model/frozen_dict.py	8	3	0	62%
RubikCubeWebApp/solver/tests/__init__.py	0	0	0	100%
RubikCubeWebApp/solver/tests/test_cfop.py	8	0	0	100%
RubikCubeWebApp/solver/tests/test_kociemba.py	8	0	0	100%
RubikCubeWebApp/solver/tests/test_performance.py	0	0	0	100%
run_tests.py	6	0	0	100%
Total	**1269**	**202**	**0**	**84%**

coverage.py v4.5.4, created at 2019-12-05 10:32

图 23 代码覆盖率测试结果

如上图所示，使用 Coverage 库进行测试的结果为代码覆盖率为 84%。对于未覆盖的代码进行分析发现，其都是一些定义但未调用的方法或者是抛出异常的语句。

6.遇到的问题及解决方案

6.1 DeepCubeA 的输入含义

通过分析网站的前端代码得知，DeepCubeA 的输入是一个长度为 9x6 的一维数组，表示每一个色块所在的位置。

6.2 魔方状态转换

原本的魔方是从一个已知状态经过若干操作变成另一个状态，现在如何获得输入魔方的状态（数组表示）。通过观察得到的规则如下图所示。

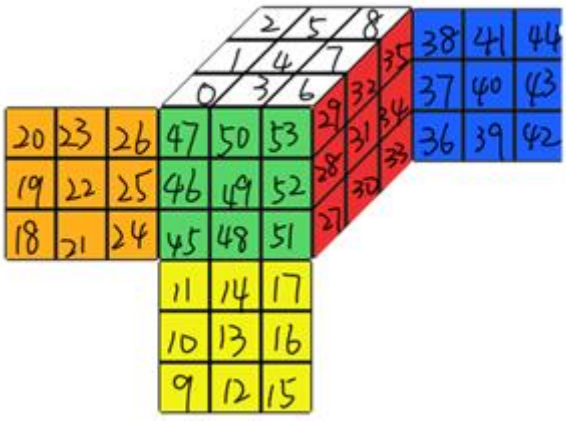

图 24 状态序号编码规则示意图

6.3 如何使用 Git 进行协作开发？

①提前确定好.gitignore 文件内容(.idea/)；

②多个分支：master，dev，功能分支；

③前后端分离，不对同一个文件进行修改；

④规范 commit,push 和 merge。

6.4 DeepCubeA 求解时间长

6.4.1 问题发现——DeepCubeA 求解时间长

在测试时，本组随机构造 1000 条魔方初始状态作为测试用例，对层先法、CFOP 法和 DeepCubeA 三种解法的平均求解时间和平均解路径长度进行实

验，结果如下图所示：

	平均求解时间	平均解路径长度
层先法	0.825 (s)	138.6
CFOP	27.1 (s)	71.49
DeepCubeA	22.6 (s)	25.07

图 25 层先法、CFOP 和 DeepCubeA 性能实验

可以发现，DeepCubeA 虽然可以找出路径很短的解法，但是由于限于硬件环境，其求解时间略长，不利于用户体验。

6.4.2 问题解决——加入 Kociemba 算法

	平均求解时间	平均解路径长度
层先法	0.825 (s)	138.6
CFOP	27.1 (s)	71.49
DeepCubeA	22.6 (s)	25.07
Kociemba	0.0275 (s)	30.44

图 26 层先法、CFOP、DeepCubeA、Kociemba 性能实验

可发现，Kociemba 算法平均求解时间大幅下降，虽然解路径长度有所上升，但在可接受的范围之内。

硬件环境：128G RAM；2T SSD；Xeon E7-4809 v4 2.10GHz CPUs；

注：实验数据受多因素影响，例如算法原理、硬件条件、算法具体实现等。

7.项目部署

在完成项目开发之后，其被部署在私有云上。考虑到项目移植的难度，项目本身被封装在 Docker 中。在 Docker 中，选择 Apache2 作为 HTTP 服务器，选择 wsgi 连接 Web 应用和 HTTP 服务器。在项目部署时，启动 Docker 容器，将 Web 应用端口通过 Docker 端口映射功能映射到主环境的 9006 端口提供服务。

7.1 项目进度

项目计划及分工表如下图所示

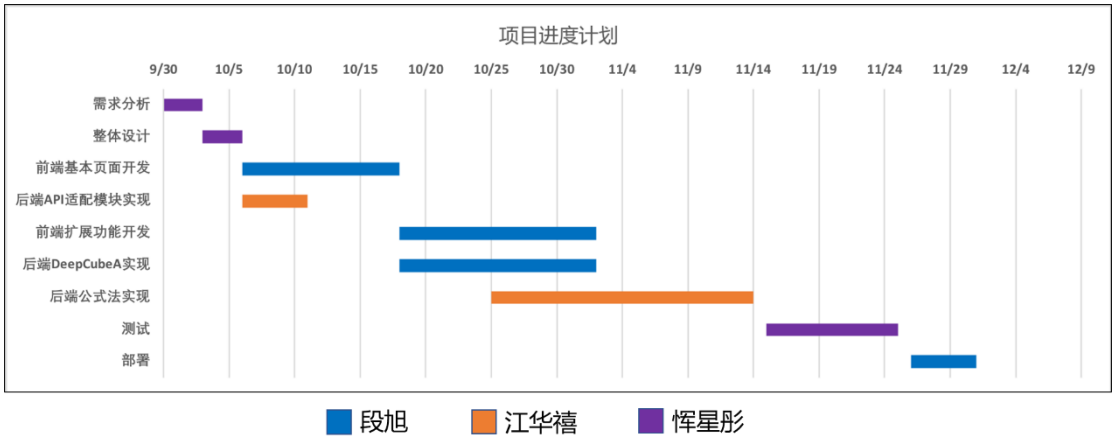

图 27 项目进度甘特图

7.2 项目发布

HTTP 服务器：Apache2
操作系统：Ubuntu 18.04
Web 框架：Django 1.11
软件环境：TensorFlow 1.8.0，Python 2.7，Docker CE 18.06
硬件环境：128G RAM，2T SSD，Xeon E7-4809 v4 2.10GHz CPUs
网站网址：http://159.226.5.97:9006/
代码管理：https://github.com/DuanXu-97/RubikCube-Web