Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина:** Алгоритмы и структуры данных

**Тема:** Решатель кубика рубика

Выполнил

студент гр. 3530901/90003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ребик Р.В.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ахин М.Х.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург   
2020

**Оглавление**

Оглавление

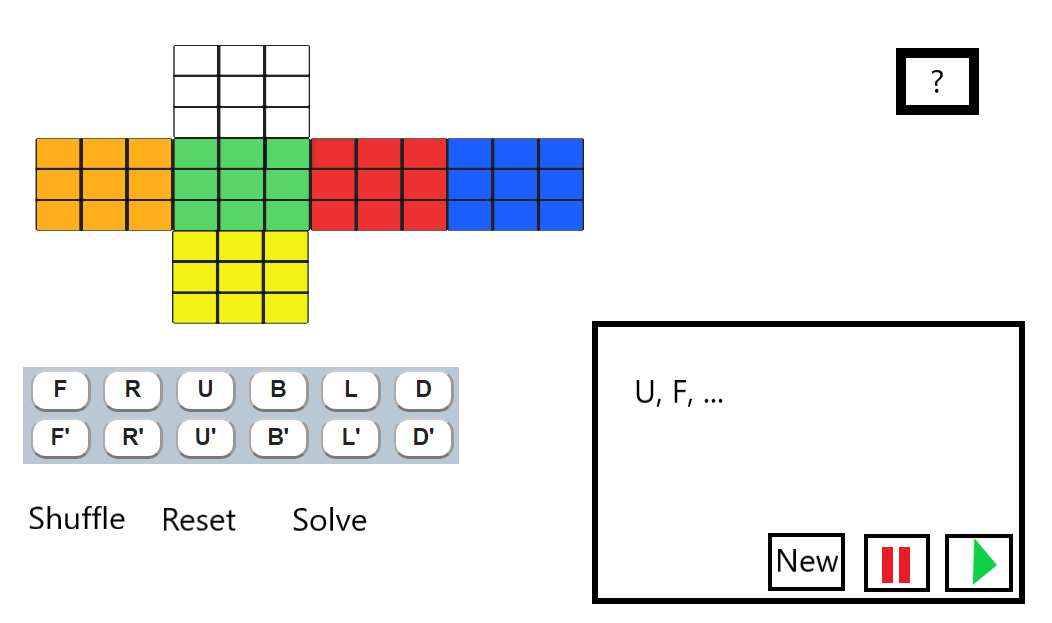
[**1. Техническое задание** 3](#_Toc57231376)

[**2. Метод решения** 4](#_Toc57231377)

[**3. Описание классов и методов** 5](#_Toc57231378)

[**4. Работа программы** 12](#_Toc57231379)

1. **Техническое задание**



Программа состоит из двух окон. На первом окне есть:

1. Динамичная развертка кубика.
2. Поле взаимодействия с кубиком. Есть команды F, R, U и т.д. При нажатии на любую команду развертка кубика изменяется в соответствии с командой. Для подробного описания команд смотреть п.3.
3. Справка (правый верхний угол). Открывает новое окно с текстовым описанием команд и возможностей программы.
4. Кнопка Shuffle. Перемешивает кубик случайным образом.
5. Кнопка Reset. Возвращает кубик к собранному виду.
6. Кнопка Solve. Запуск решения кубика, которое представляет из себя последовательность команд, которые нужно выполнить чтобы привести кубик к исходному виду.
7. Окно с решением. Изначально неактивно. Состоит из текстовой последовательности команд решения, кнопок «запуск», «стоп» и «New». Первая – запускает решение: каждую секунду выполняется одна команда решения, при этом развертка кубика изменяется. Кнопка «стоп» останавливает анимацию. Кнопка «New» делает это окно снова неактивным и возвращает нас к ручному (или с помощью Shuffle) перемешиванию кубика.



Второе окно – обычная справка.

Гитхаб: https://github.com/Relfick/RubiksCubeSolver

1. **Метод решения**

В проекте использована концепция MVC (Model-View-Controller) для отделения бизнес-логики от визуализации. Код разделен на три пакета классов: model, controller, fxmls.

В основе решателя лежит алгоритм сборки кубика рубика для новичков, который заключается в последовательных сборке белого креста, первого слоя, второго слоя, желтого креста, ориентирования углов и их развороте.

Важно различать понятия **Куб** и **Кубик**. Весь **куб** представляется как набор из 27 маленьких **кубиков** (по 9 кубиков на слое). Кубики могут быть центральными (имеют 1 цвет), граневыми (2 цвета) и угловыми (3 цвета). Каждая сторона имеет свое название по направлению: up, right, back, left, up и down. Каждый цвет каждого кубика тоже имеет свое направление, идентичное стороне, на которой он лежит.

Вращение каждой из 6 граней кубика обозначено отдельной буквой:

F (Front) - передняя

B (Back) - задняя

R (Right) - правая

L (Left) - левая

U (Up) - верхняя

D (Down) - нижняя

Каждое из этих действий вращает указанную грань по часовой стрелке. Если у буквы есть апостроф ( например, U' ), это значит, что вращение происходит против часовой стрелки. Также используется обозначение y, при котором весь кубик поворачивается так, что правая грань становится передней, передняя левой, левая задней, а задняя правой. Верхняя и нижняя местами не меняются. y' - то же, но в противоположную сторону

**3. Описание классов и методов**

**Класс CubieColor:**

Представление цвета конкретного кубика на конкретной грани

**Методы класса CubieColor:**

getColor():

Возвращает цвет

setColor(char ncolor):

Устанавливает цвет

getDir():

Возвращает направление данного цвета

setDir(char ndir):

устанавливает направление данного цвета

**Класс Cubie:**

Представление конкретного маленького кубика

**Методы класса Cubie:**

getDirOfColor(char color):

Возвращает направление цвета color

getColorOfDir(char dir):

Возвращает цвет с данного направления dir

getColors():

Возвращает массив цветов этого кубика

setColors(CubieColor[] newColors):

Устанавливает цвета этому кубику.

setColorOfDir(char dir, char ncolor):

Устанавливает конкретный цвет ncolor на конкретном направлении dir

isCornerCubie():

Показывает, является ли кубик угловым

isEdgeCubie():

Показывает, является ли кубик граневым

verticalFace():

Для каждой грани не из второго слоя возвращает слой, в котором находится данный кубик

isWhiteCorner():

Показывает, есть ли на этом кубике белый цвет

**Класс Cube:**

Представление целого куба как набор 26 кубиков (центральный не учитывается, так как бесполезный).

**Методы класса Cube:**

turn(String turn):

Совершает один поворот turn

rotateSide(Cubie[][] orig, boolean clockwise, char[] preChange, char[] postChange):

Поворачивает сторону по часовой стрелке или против

performMoves(String moves):

Выполняет последовательность поворотов

performReverseMoves(String moves):

Выполняет последовательность поворотов, инвертируя каждый

optimizeMoves(String moves):

Сокращает последовательность поворотов, удаляя бессмысленные команды или заменяя их более короткими

randScramble():

Генерация рандомных поворотов куба, чтобы задать случайное решаемое состояние

scramble():

Выполняет повороты, возвращенные randScramble()

makeSunFlower():

Сборка белого креста с желтым центром

makeWhiteCross():

Сборка белого креста

prepareSlot(int x, int y, int z, char color):

Утилита для MSF (makeSunFlower), позволяет не нарушать уже поставленные белые грани

numWhiteEdgesOriented():

Утилита для MSF, возвращает кол-во белых граней в верхнем слое

finishWhiteLayer():

Завершает сборку нижнего слоя

whiteCornerInU():

Проверяет, есть ли белые углы в верхнем слое

insertWhiteCorners():

Вставляет белые углы на нижний слой

uAndyTurns(StringBuilder moves, int numUTurns, int yRotations):

Утилита для insertWhiteCorners() и insertEdgesInU(), добавляет в moves (последовательность команд) нужное число U и y поворотов

insertMisorientedCorners():

Исправляет положение белых углов, которые уже в нижнем слое.

whiteCornerPrepared():

Утилита для insertWhiteCorners(), проверяет, в правильном ли положении находится нужный угол.

cornerInserted(int x, int y, int z)

Проверяет, вставлен ли белый угол

insertAllEdges():

Сборка второго слоя

nonYellowEdgesInU():

Утилита для insertAllEdges(), проверяет, есть ли не желтые грани в верхнем слое

insertEdgesInU():

Вставляет не желтые грани из верхнего слоя во второй в нужные слоты

insertMisorientedEdges():

Правильно ориентирует неправильно ориентированные грани второго слоя

numYellowEdgesOriented():

Утилита для yellowEdgeOrientation() и makeYellowCross(), возвращает количество правильно ориентированных желтых граней во втором слое

numYellowCornersOriented():

Утилита для orientLastLayer(), возвращает количество желтых правильно ориентированных углов в верхнем слое

yellowEdgeOrientation():

Утилита для makeYellowCross(). Возвращает форму правильно ориентированных желтых граней в верхнем слое (точка, угол, прямая или крест)

makeYellowCross():

Сборка желтого креста

orientLastLayer():

Ориентирует углы в верхнем слое

permuteLastLayer():

Финальная часть сборки

checkIfFailed(String currCommand):

Проверяет, не является ли команда “failed”. Если на каком-то шаге алгоритму не удается решить установленную задачу, вместо строки команд он возвращает строку “failed”

solve():

Запускает функции для решения кубика рубика

solved():

Проверяет, собран ли кубик

setTheState(char[][][] colors):

Устанавливает куб в заданное состояние

reset()

Возвращает куб в исходное состояние

setCubieColor(int x, int y, int z, char dir, char ncolor):

устанавливает цвет конкретного кубика

getTheState():

Возвращает состояние куба

**Class PrimaryWindowController**

Контроллер главного окна. Связывает модель куба с его представлением.

**Методы класса PrimaryWindowController**

initialize():

Инициализация главного окна

handleFrontClockwise():

Производит поворот F

handleFrontCounterclockwise():

Производит поворот F’

handleRightClockwise():

Производит поворот R

handleRightCounterclockwise():

Производит поворот R’

handleBackClockwise():

Производит поворот B

handleBackCounterclockwise():

Производит поворот B’

handleLeftClockwise():

Производит поворот L

handleLeftCounterclockwise():

Производит поворот L’

handleUpClockwise():

Производит поворот U

handleUpCounterclockwise():

Ловит нажатие кнопки U’

handleDownClockwise():

Производит поворот D

handleDownCounterclockwise():

Производит поворот D’

handleShuffle():

Задает кубу случайное состояние

handleReset():

Обновляет куб

handleSolve():

Решает куб

handleNext():

Отрисовать следующую команду решения

handlePrevious():

Отрисовать предыдущую команду решения

handleNew():

Обновляет куб

handleHelp():

Открывает окно справки

setMainApp(Main MainApp):

Связывает контроллер с классом Main

turn(String turn):

Выполняет поворот turn и отрисовывает куб

executeTurns():

Отрисовывает куб

arrayOfNeededColors():

Возвращает цвета куба для отрисовки в executeTurns()

**Class HelpWindowController**

Контроллер окна справки. Методов в нем нет, так как все, что делает это окно – представляет текстовую информацию

**Class Utility**

Класс с вспомогательными методами

**Методы класса Utility**

getDirOfSide(int sideIndex):

Возвращает буквенное обозначение стороны

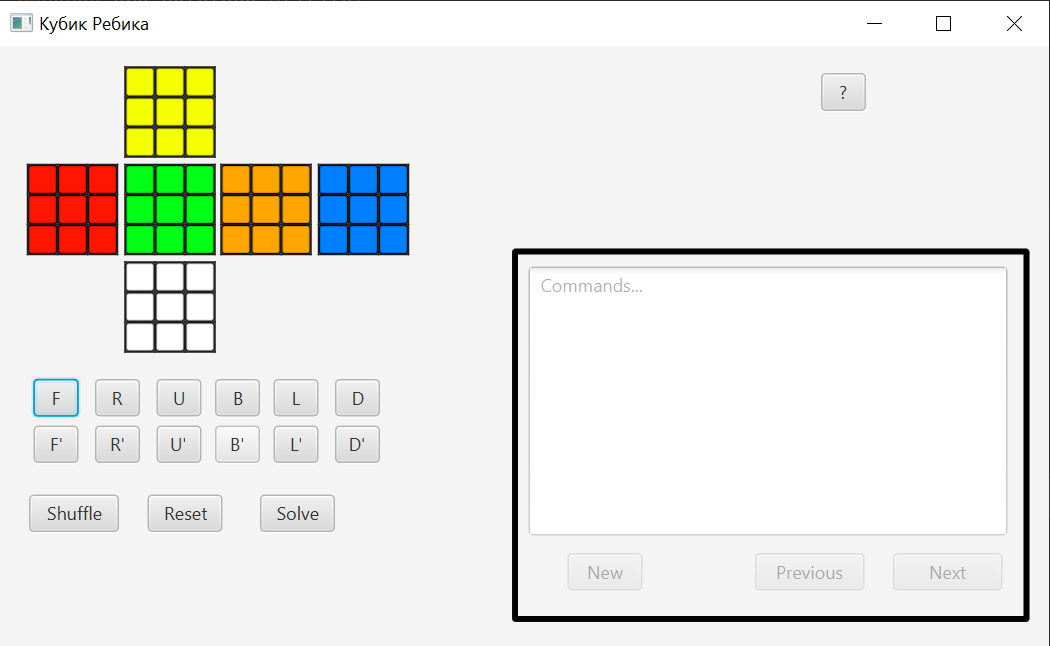
getColorOfInt(int col):

Возвращает буквенное обозначение цвета

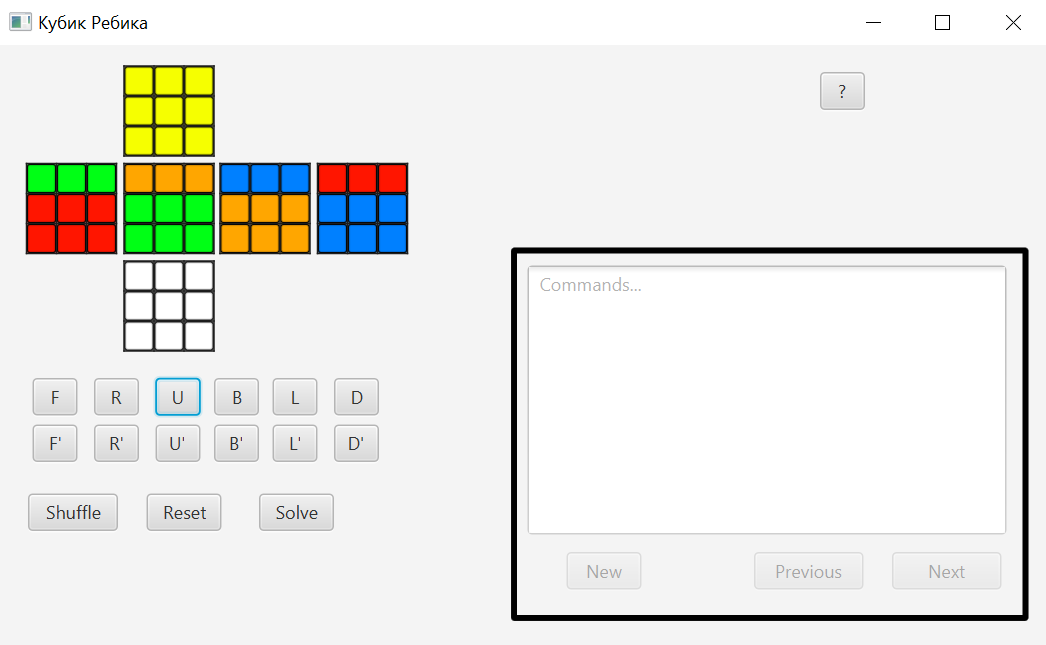
getColorOfChar(char col):

Возвращает числовое обозначение цвета

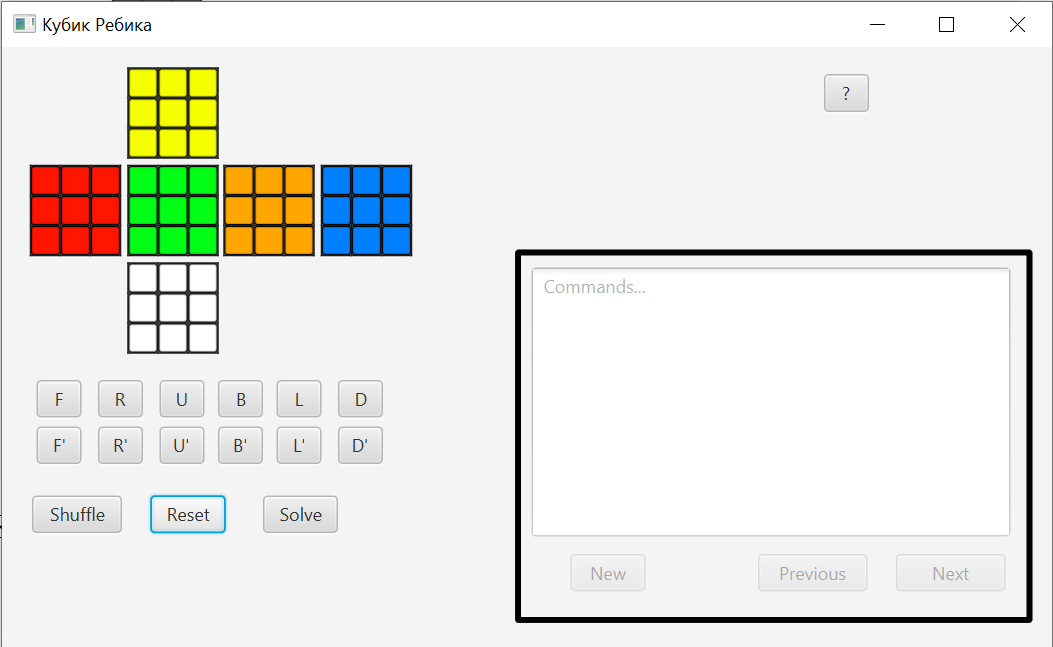
**4. Работа программы**



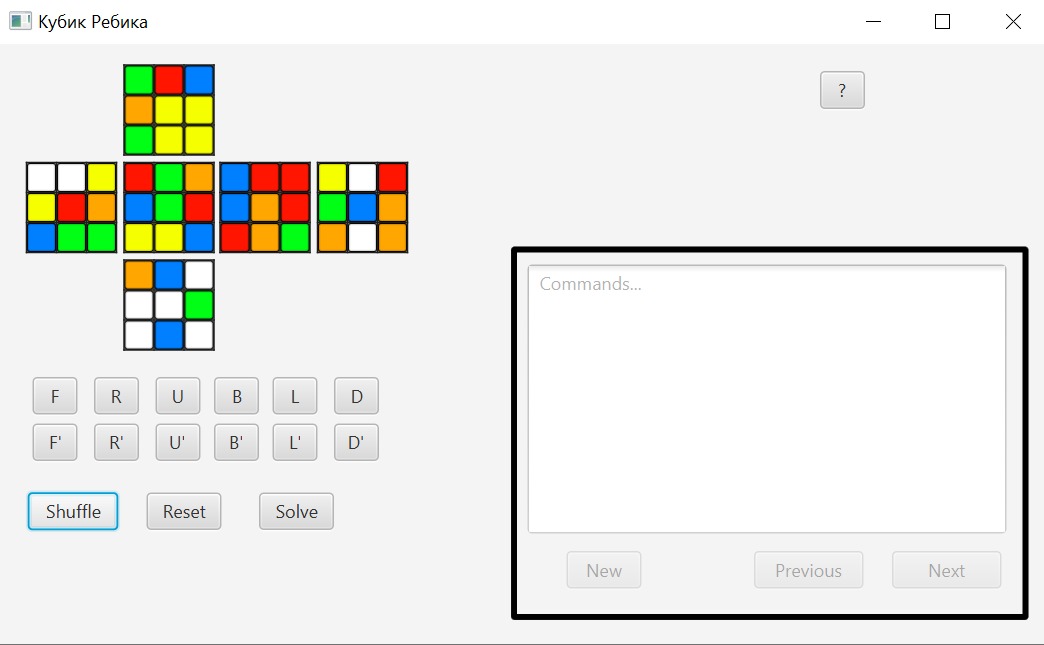
Окно поделено на 2 области: левую и правую. В левой есть развертка кубика рубика и кнопки-команды поворотов. В правой – текстовое поле, в котором будут команды. Кнопки Previous и Next отвечают за исполнение следующей и предыдущей команд решения соответственно. Пока активно одно окно, второе неактивно. Нажатие на кнопку New очищает поле команд, сбрасывает кубик и меняет активную часть поля.

Нажав, например, на U, мы увидим, как на кубике произошел поворот верхней грани по часовой стрелке. 

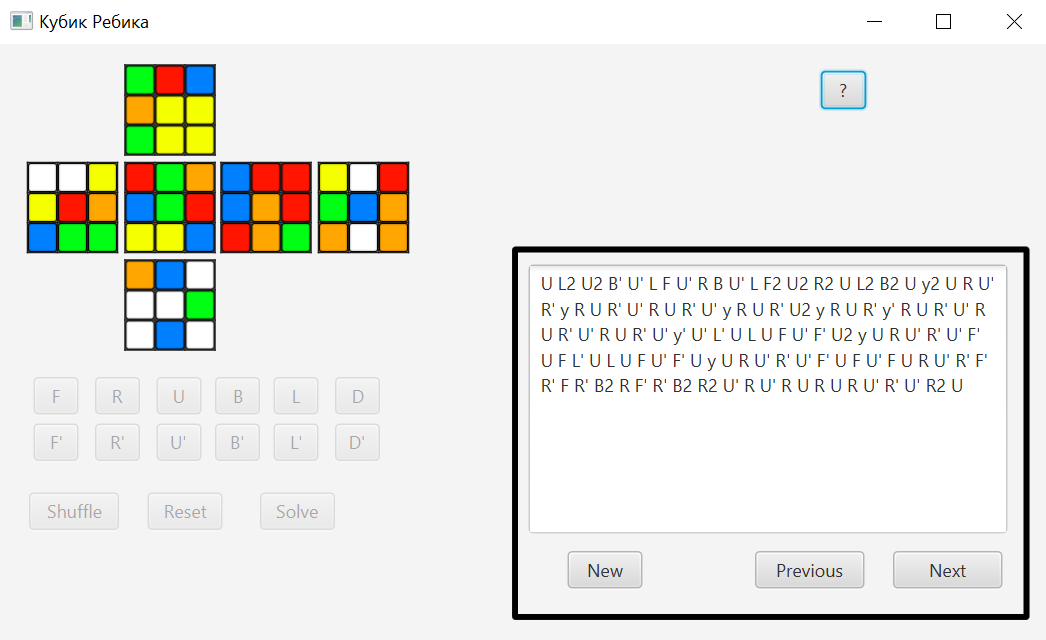
Так же менять цвета конкретных кубиков можно просто нажимая на них. Цвета меняются цикличестки в заданном порядке: зеленый-оранжевый-синий-красный-желтый-белый.

Кнопка reset сбрасывает состояние кубика в стандартное. 

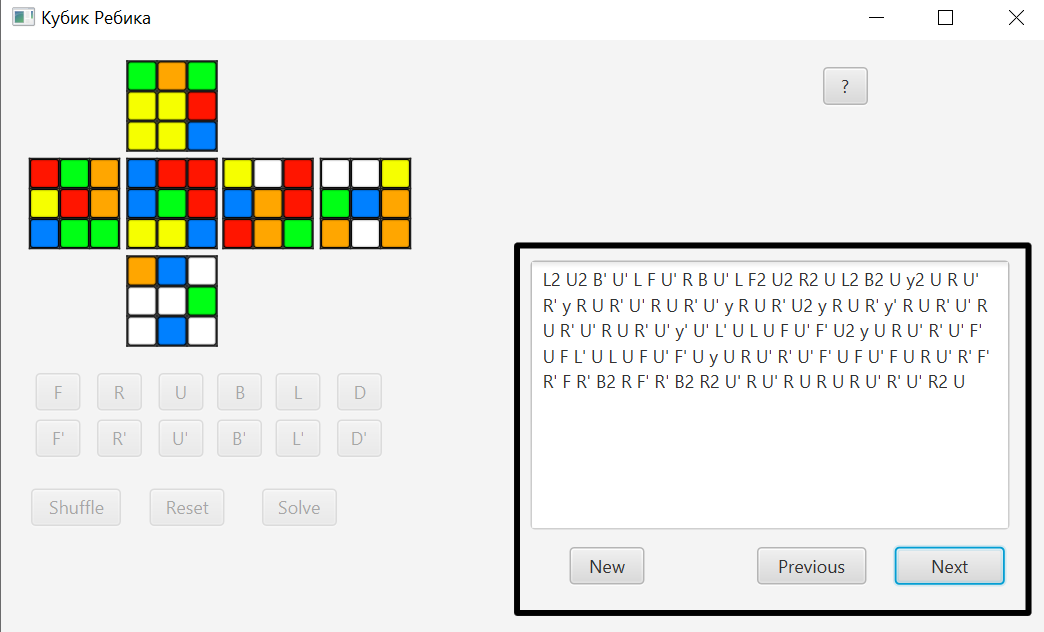
Кнопка Shuffle перемешивает кубик.



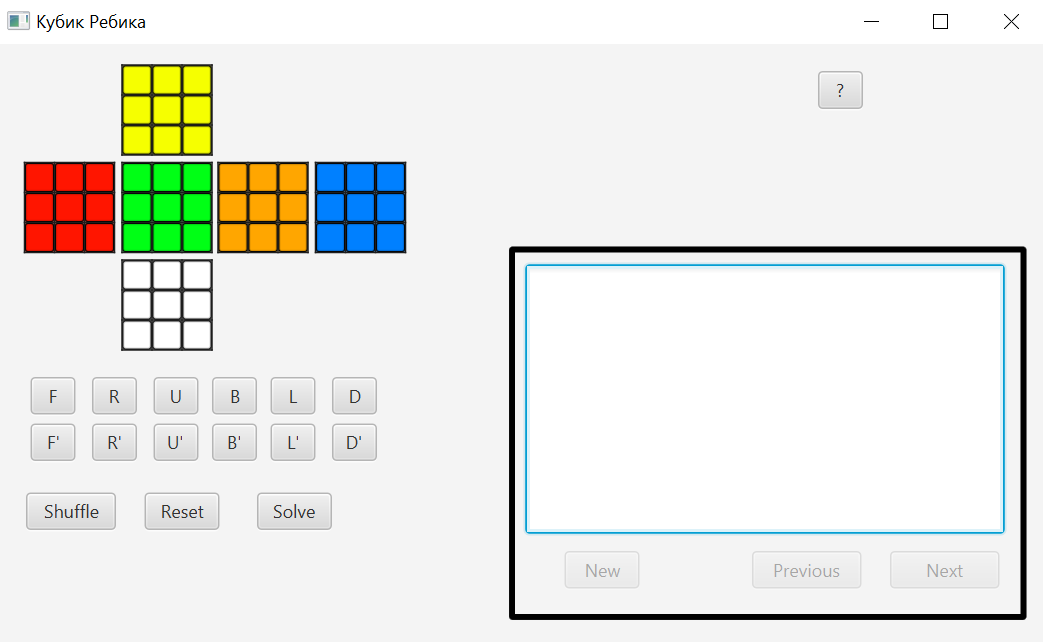
Нажатие на Solve запустит алгоритм решения команд. Последовательность для решения отобразится в правом поле, а левое станет неактивным



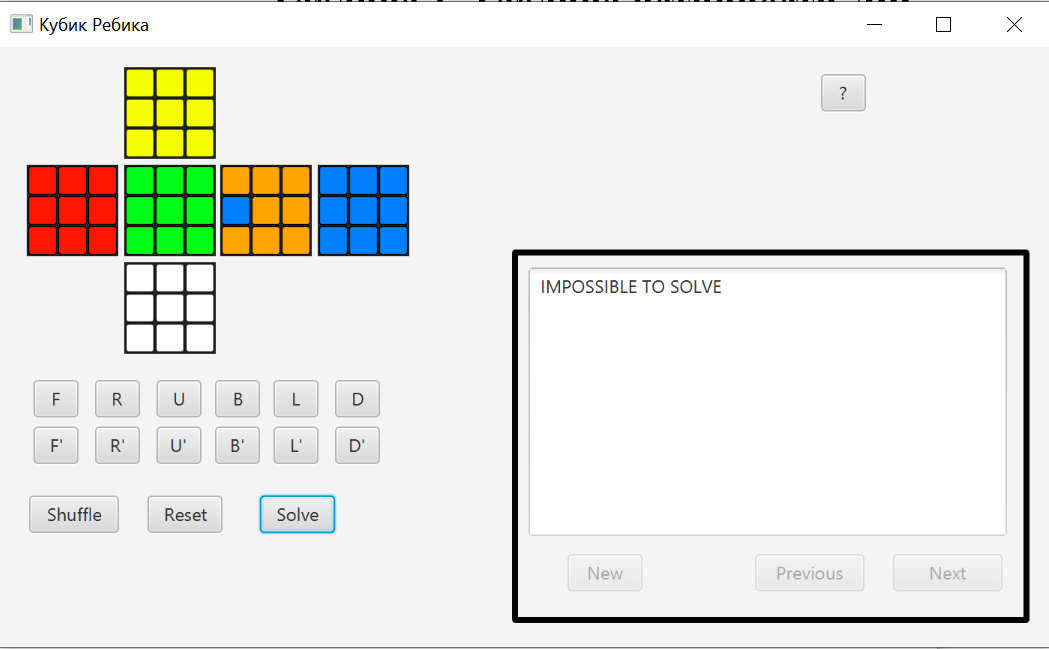
Теперь при нажатии на кнопку Next выполнится команда U, кубик отрисуется заново, а команда удалится из списка команд.



Previous работает аналогично, но в обратную сторону.

Кнопка New сбрасывает кубик:

Если попробуем задать нерешаемое состояние (как, например, на скриншоте) и нажмем на Solve, в поле команд появится надпись “Impossible to solve”.



При нажатии на “?” открывается окно справки, блокирующее основное окно

