Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Информационных технологий и программирования

Программирование на C++

Работа: Лабораторная работа №8. Кубика Рубика.

Выполнил: Гаджиев Саид M3115

Санкт-Петербург

2023

Лабораторная работа №8. Кубика Рубика.

Спроектировать и реализовать программу, имитирующую сборку Кубика Рубика 3x3.

К программе предъявляются следующие функциональные требования:

- Сохранение и чтение состояния кубика рубика из файла
- Проверка корректности текущего состояния (инвариант состояний кубика)
- Вывод в консоль текущего состояния
- Вращение граней кубика рубика с помощью вводимых команд
- Генерация случайного состояния Кубика Рубика, корректного с точки зрения инварианта состояний
- Нахождения “решения” для текущего состояния в виде последовательности поворотов граней

Нефункциональные требования:

- Программа должны быть спроектирована, с использованием ОПП
- Логические сущности должны быть выделены в отдельный классы

Критерии оценки:

- Логично выстроенная архитектура приложения
- Применение возможностей языка программирования C++ включая стандартную библиотеку

Дополнительно (за дополнительные баллы):

Реализовать графический интерфейс приложения, с использование OpenGL Utility Toolkit

Решение:

File main.cpp

```
#include "Cube/RubikCube.h"
#include "Cube/VisualFunctions.h"
#include "Other/GLUTMenu.h"
#include <glut.h>

extern RubikCube Cube;

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH); //
Устанавливаем параметры дисплея
    glutInitWindowSize(640, 640); // Устанавливаем размер окна
    glutInitWindowPosition(1, 1); // Устанавливаем позицию окна на экране
    glutCreateWindow("CUBE"); // Создаем окно с заголовком "CUBE"
    init(); // Вызываем функцию инициализации
    glutDisplayFunc(display); // Устанавливаем функцию для отображения
графики
    glutReshapeFunc(reshape); // Устанавливаем функцию для изменения размеров
окна
    glutTimerFunc(5, timer, 0); // Устанавливаем функцию-таймер с интервалом
в 5 миллисекунд
    glutSpecialFunc(specialKeys); // Устанавливаем функцию для обработки
специальных клавиш
    createGLUTMenus(); // Создаем меню
    glutMainLoop(); // Запуск главного цикла обработки запросов
    return 0;
}
```

File VisualFunctions.h:

```
#ifndef visualFunctions_h
#define visualFunctions_h
#pragma once

#include "RubikCube.h"

extern RubikCube Cube;

void display() {
    glPushMatrix(); // сохраняем текущую матрицу видовой модели
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // очищаем буферы
цвета и глубины
    glColor3f(1, 0, 0); // устанавливаем цвет
    glTranslatef(0, 0, translateZ); // транслируем по осям X, Y и Z
    glRotatef(xRot, 1, 0, 0); // вращаем по оси X
    glRotatef(yRot, 0, 1, 0); // вращаем по оси Y
    glTranslatef(CUBE_SIZE / -2, CUBE_SIZE / -2, CUBE_SIZE / -2); // смещаем
куб в центр координатной оси
    Cube.draw(); // отображаем куб
    glPopMatrix(); // восстанавливаем матрицу видовой модели
    glutSwapBuffers(); // смена переднего и заднего буферов
}

void init() {
    // Установка цвета фона
    glClearColor(0.36, 0.18, 0.38, 0.4);
    // Установка освещения
    float matSpecular[] = {0.3, 0.3, 0.3, 0}; // установка характеристик
материала
    float diffuseLight[] = {0.2, 0.2, 0.2, 1}; // установка диффузного света
    float ambientLight[] = {0.9, 0.9, 0.9, 1}; // установка окружающего света
    glShadeModel(GL_SMOOTH); // установка режима заливки
    glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecular); // установка цвета
отражаемого света
    glMateriali(GL_FRONT, GL_SHININESS, 128); // установка коэффициента
отражения света
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight); // установка диффузного
света
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight); // установка окружающего
света
    glEnable(GL_LIGHT0); // включение источника света
    glEnable(GL_COLOR_MATERIAL); // включение материала для цвета
    glColorMaterial(GL_FRONT, GL_AMBIENT); // выбор цвета материала
    glEnable(GL_DEPTH_TEST); // включение буфера глубины
    glEnable(GL_LIGHTING); // включение освещения

    // Инициализация куба
    Cube.setVisualCube(CUBE_SIZE, colors);
}

void specialKeys(int key, int, int) {
    // Обработка клавиш со стрелками для вращения объекта в разных
направлениях
    if (key == GLUT_KEY_DOWN) {
        xRot -= 10;
        if (xRot >= 360)
            xRot -= 360;
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_UP) {
        xRot += 10;
```

```
        if (xRot < 0)
            xRot += 360;
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_RIGHT) {
        yRot -= 10;
        if (yRot >= 360)
            yRot -= 360;
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_LEFT) {
        yRot += 10;
        if (yRot < 0)
            yRot += 360;
        glutPostRedisplay();
    }

    // Обработка клавиш 'F1'-'F6' для вращения грани кубика
    if (key == GLUT_KEY_F1) {
        Cube.RotateUpPlane('+');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F7) {
        Cube.RotateUpPlane('-');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F2) {
        Cube.RotateLeftPlane('+');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F8) {
        Cube.RotateLeftPlane('-');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F3) {
        Cube.RotateFrontPlane('+');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F9) {
        Cube.RotateFrontPlane('-');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F4) {
        Cube.RotateRightPlane('+');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F10) {
        Cube.RotateRightPlane('-');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F5) {
        Cube.RotateBackPlane('+');
        glutPostRedisplay();
    }
```

```
    if (key == GLUT_KEY_F11) {
        Cube.RotateBackPlane('-');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_F6) {
        Cube.RotateDownPlane('+');
        glutPostRedisplay();
    }

    if (key == GLUT_KEY_HOME) { //Если биндить F12 он почему-то вызывает
ошибку ;(
        Cube.RotateDownPlane('-');
        glutPostRedisplay();
    }
}

void timer(int) {
    glutTimerFunc(5, timer, 0);
    // Визуальное вращение выбранной грани кубика при помощи мини-машинки,
если текущая грань не равна -1
    if (Cube.currentPlane != -1) {
        Cube.visualRotateMiniMachineGun(Cube.currentPlane,
ROTATE_START_VALUE, -1);
    }
    display();
}

void processMenu(int action);

void reshape(int width, int height) {
    // Изменение размеров окна
    glViewport(0, 0, width, height);

    // Устанавливаем матрицу проекции
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Вычисляем соотношение сторон окна
    GLfloat fAspect = (GLfloat)width / (GLfloat)height;

    // Устанавливаем параметры перспективной проекции
    gluPerspective(60, fAspect, 1, 1000);

    // Устанавливаем матрицу моделирования-вида
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

#endif /* visualFunctions_h */
```

File VisualCube.h

```
#ifndef visualCube_h
#define visualCube_h

#include "CubeSettings.h"

// Цвета
unsigned int colors[6] = { green, blue, yellow, white, orange, red };

class VisualCube {
```

```
private:
    // Этот массив используется для передачи компонентов цвета
    // в функцию glColor3ubv, которая устанавливает текущий цвет
    // рисования для отображения грани куба определенным цветом. (RGB)
    unsigned char RGBColor[3];

public:
    int color[6]; // Массив цветов граний куба
    GLfloat visualSize; // Размер отображаемого куба

    // Функция для поворота цветов куба по часовой стрелке на 90 градусов
    void rotateORb() {
        int temp = color[5];
        color[5] = color[3];
        color[3] = color[4];
        color[4] = color[2];
        color[2] = temp;
    }

    // Функция для поворота цветов куба по часовой стрелке на 180 градусов
    void rotateRR() {
        int temp = color[5];
        color[5] = color[3];
        color[3] = color[4];
        color[4] = color[2];
        color[2] = temp;
    }

    // Функция для поворота цветов куба по часовой стрелке на 270 градусов
    void rotateGR() {
        int temp = color[2];
        color[2] = color[1];
        color[1] = color[3];
        color[3] = color[0];
        color[0] = temp;
    }

    // Функция для поворота цветов куба против часовой стрелки на 90 градусов
    void rotateGD() {
        int temp = color[0];
        color[0] = color[4];
        color[4] = color[1];
        color[1] = color[5];
        color[5] = temp;
    }

    // Функция для поворота цветов куба против часовой стрелки на 180
градусов
    void rotateGU() {
        int temp = color[0];
        color[0] = color[4];
        color[4] = color[1];
        color[1] = color[5];
        color[5] = temp;
    }

    // Установка цвета для определенной стороны куба
    void setColor(int i, int color) { this->color[i] = color; }

    // Преобразование цвета из формата 0xFF0000 в формат RGB - разделение
цвета[i] на 3 компонента (R, G, B)
    unsigned char* getRGB(int i) {
        RGBColor[0] = color[i] >> 16;
        RGBColor[1] = color[i] >> 8;
```

```
        RGBColor[2] = color[i];
        return RGBColor;
    }

    /*** Отображение куба ***/
    // Установка цветов и векторов нормали
    void draw() {
        glPushMatrix();  // Сохраняем текущую матрицу просмотра в стеке
        glBegin(GL_QUADS);  // Начинаем определение вершин для рисования
четырехугольников

        // Верх
        glColor3ubv(getRGB(0));  // Устанавливаем цвет вершин в формате RGB
        glNormal3f(0, 0, 1);  // Устанавливаем нормаль (вектор
перпендикулярен плоскости)
        glVertex3f(visualSize, visualSize, visualSize);  // Указываем
координаты вершины (x, y, z)
        glVertex3f(0, visualSize, visualSize);  // Указываем координаты
вершины (x, y, z)
        glVertex3f(0, 0, visualSize);  // Указываем координаты вершины (x, y,
z)
        glVertex3f(visualSize, 0, visualSize);  // Указываем координаты
вершины (x, y, z)

        // Низ
        glColor3ubv(getRGB(1));
        glNormal3f(0, 0, -1);
        glVertex3f(visualSize, 0, 0);
        glVertex3f(0, 0, 0);
        glVertex3f(0, visualSize, 0);
        glVertex3f(visualSize, visualSize, 0);

        // Перед
        glColor3ubv(getRGB(2));
        glNormal3f(0, -1, 0);
        glVertex3f(visualSize, 0, visualSize);
        glVertex3f(0, 0, visualSize);
        glVertex3f(0, 0, 0);
        glVertex3f(visualSize, 0, 0);

        // Зад
        glColor3ubv(getRGB(3));
        glNormal3f(0, 1, 0);
        glVertex3f(visualSize, visualSize, 0);
        glVertex3f(0, visualSize, 0);
        glVertex3f(0, visualSize, visualSize);
        glVertex3f(visualSize, visualSize, visualSize);

        // Лево
        glColor3ubv(getRGB(4));
        glNormal3f(-1, 0, 0);
        glVertex3f(0, visualSize, visualSize);
        glVertex3f(0, visualSize, 0);
        glVertex3f(0, 0, 0);
        glVertex3f(0, 0, visualSize);

        // Право
        glColor3ubv(getRGB(5));
        glNormal3f(1, 0, 0);
        glVertex3f(visualSize, visualSize, 0);
        glVertex3f(visualSize, visualSize, visualSize);
        glVertex3f(visualSize, 0, visualSize);
        glVertex3f(visualSize, 0, 0);
```

```
        glEnd();
        glPopMatrix();
    }

    // Отображение куба с смещением по (x, y, z)
    void draw(double x, double y, double z) {
        glPushMatrix();
        glTranslated(x, y, z);
        draw();
        glPopMatrix();
    }

};

#endif /* visualCube_h */
```

File CubeSettings.h

```
#ifndef visualSettings_h
#define visualSettings_h
#define GL_SILENCE_DEPRECATION
#include <glut.h>

ifstream inputStream("input.txt");
ofstream outputStream("output.txt");

enum cubeColors { // Определение перечисления цветов кубика Рубика
    yellow = 0xFFD700,
    blue = 0x0000FF,
    green = 0x32CD32,
    red = 0xFF0000,
    orange = 0xFF8C00,
    white = 0xFFFFFF
};

const int ROTATE_SPEED_STEP = 10, ROTATE_START_VALUE = 30; // Константы для
вращения кубика Рубика
GLfloat CUBE_SIZE = 12; // Размер кубика Рубика

// Проекция угла поворота на ось
int xRot = 25, yRot = -45;

// Перемещение по оси Z
GLfloat translateZ = -35;

#endif /* visualSettings_h */
```

File GLUTMenu.h

```
#include <glut.h>
#ifndef RUBIKCUBE_GLUTMENU_H
#define RUBIKCUBE_GLUTMENU_H

// Менюшка
void createGLUTMenus() {
    glutCreateMenu(processMenu);
    glutAddMenuEntry("Read Cube from file", 1);
    glutAddMenuEntry("Print Cube in console", 2);
    glutAddMenuEntry("Print Cube in file", 3);
    glutAddMenuEntry("Shuffle", 4);
    glutAddMenuEntry("Find Solution", 5);
    glutAddMenuEntry("Create solved Cube", 6);
```

```
    glutAddMenuEntry("Print solving information in console", 7);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

// Действия
void processMenu(int action) {
    try {
        switch (action) {
            case 1:
                Cube.ReadRubikCube(inputStream);
                break;

            case 2:
                cout << "\nCube's unfolding:\n\n";
                Cube.PrintRubikCube();
                break;

            case 3:
                outputStream << "\nCube's unfolding:\n\n";
                Cube.PrintRubikCube(outputStream);
                break;

            case 4:
                Cube.Shuffle(false);
                cout << "\n\nCube's unfolding after shuffling:\n\n";
                Cube.PrintRubikCube();
                break;

            case 5:
                Cube.FindSolution();
                outputStream << "\n\nCube's unfolding after solving:\n\n";
                Cube.PrintRubikCube(outputStream);
                break;

            case 6:
                Cube.CreateRubikCube();
                Cube.setVisualCube(CUBE_SIZE, colors);
                break;

            case 7:
                Cube.PrintCubeInfo();
                break;

            default:
                break;
        }

    } catch (exception &e) {
        cout << e.what();
    }
}

#endif //RUBIKCUBE_GLUTMENU_H
```

File HelpingFunctions.h

```
#ifndef HELPING_FUNCTIONS_H
#define HELPING_FUNCTIONS_H

#include <iostream>
#include <fstream>
using namespace std;
```

```
const unsigned long maxWordSize = 1;

void PrintColor(const string &color, ostream &outputStream = cout) {
    string result(maxWordSize, ' '), strColor;
    unsigned long lengthField = result.size() + 1, step = (lengthField -
color.size()) / 2;
    strColor = color;
    copy(strColor.begin(), strColor.end(), result.begin() + step);
    outputStream << result << " ";
}

int setRGBColor(const string &color) {
    int result = 0;
    if (color[0] == 'G' or color[0] == 'g') result = 0x32CD32; // Если первый
символ color равен 'G' или 'g', присваиваем result значение 0x32CD32
    else if (color[0] == 'B' or color[0] == 'b') result = 0x0000FF; // Если
первый символ color равен 'B' или 'b', присваиваем result значение 0x0000FF
    else if (color[0] == 'Y' or color[0] == 'y') result = 0xFFD700; // Если
первый символ color равен 'Y' или 'y', присваиваем result значение 0xFFD700
    else if (color[0] == 'W' or color[0] == 'w') result = 0xFFFFFF; // Если
первый символ color равен 'W' или 'w', присваиваем result значение 0xFFFFFF
    else if (color[0] == 'O' or color[0] == 'o') result = 0xFF8C00; // Если
первый символ color равен 'O' или 'o', присваиваем result значение 0xFF8C00
    else if (color[0] == 'R' or color[0] == 'r') result = 0xFF0000; // Если
первый символ color равен 'R' или 'r', присваиваем result значение 0xFF0000
    return result;
}

#endif /* HELPING_FUNCTIONS_H */
```

File MiniCube.h

```
#ifndef RUBIKCUBE_MINICUBE_H
#define RUBIKCUBE_MINICUBE_H

class MiniCube {
private:
    // Переменные для хранения цветов кубика по разным сторонам (up, down,
left, right, front, back), изначально установлены в "black"
    string up = "black", down = "black", left = "black", right = "black",
front = "black", back = "black";

public:
    MiniCube() = default;

    MiniCube(string &up, string &down, string &left, string &right, string
&front, string &back) : up(up), down(down), left(left), right(right),
front(front), back(back) {}

    // Геттеры для цветов кубика
    string UpColor() const {
        return this->up;

    }

    string DownColor() const {
        return this->down;
    }

    string LeftColor() const {
        return this->left;
    }
```

```
    string RightColor() const {
        return this->right;
    }

    string FrontColor() const {
        return this->front;
    }

    string BackColor() const {
        return this->back;
    }

    // Сеттеры для цветов кубика
    void setUpColor(string color) {
        this->up = move(color);
    }

    void setDownColor(string color) {
        this->down = move(color);
    }

    void setLeftColor(string color) {
        this->left = move(color);
    }

    void setRightColor(string color) {
        this->right = move(color);
    }

    void setFrontColor(string color) {
        this->front = move(color);
    }

    void setBackColor(string color) {
        this->back = move(color);
    }
};


#endif //RUBIKCUBE_MINICUBE_H
```

File RubikCube.h:

```
#ifndef RUBIKCUBE_RUBIKCUBE_H
#define RUBIKCUBE_RUBIKCUBE_H
#pragma once

#include "../Other/HelpingFunctions.h"
#include "MiniCube.h"
#include "CubeSettings.h"
#include "VisualCube.h"
#include <iostream>
#include <vector>
#include <string>

#define Plane vector<vector<MiniCube*>>
#define RightCenter RightPlane[1][1]->RightColor()
#define LeftCenter LeftPlane[1][1]->LeftColor()
#define UpCenter UpPlane[1][1]->UpColor()
#define DownCenter DownPlane[1][1]->DownColor()
#define FrontCenter FrontPlane[1][1]->FrontColor()
#define BackCenter BackPlane[1][1]->BackColor()
```

```
int CubeCounter = 0;

class RubikCube {
public:
    RubikCube() {
        arr.resize(3, vector<vector<MiniCube >>(3, vector<MiniCube>(3)));
        PushInPlaneVector();
        CreateRubikCube();
    }

    // Функция для задания начальных (правильных) цветов граней кубика
    void CreateRubikCube() {
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j) {
                UpPlane[i][j]->setUpColor("W");
                DownPlane[i][j]->setDownColor("Y");
                LeftPlane[i][j]->setLeftColor("O");
                RightPlane[i][j]->setRightColor("R");
                FrontPlane[i][j]->setFrontColor("G");
                BackPlane[i][j]->setBackColor("B");
            }
    }

    // Функция для чтения раскладки кубика из файла или консоли
    void ReadRubikCube(istream& streamIn = cin) {

        string color;

        // Чтение верхней грани
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j) {
                streamIn >> color;
                UpPlane[i][j]->setUpColor(color);
                visualColors[j][2][i].setColor(3, setRGBColor(color));
            }

        // Чтение левой грани
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j) {
                streamIn >> color;
                LeftPlane[i][j]->setLeftColor(color);
                visualColors[0][2 - i][j].setColor(4, setRGBColor(color));
            }

        // Чтение передней грани
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j) {
                streamIn >> color;
                FrontPlane[i][j]->setFrontColor(color);
                visualColors[j][2 - i][2].setColor(0, setRGBColor(color));
            }

        // Чтение правой грани
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j) {
                streamIn >> color;
                RightPlane[i][j]->setRightColor(color);
                visualColors[2][2 - i][2 - j].setColor(5,
setRGBColor(color));
            }

        // Чтение задней грани
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j) {
```

```
                streamIn >> color;
                BackPlane[i][j]->setBackColor(color);
                visualColors[2 - j][2 - i][0].setColor(1,
setRGBColor(color));
            }

        // Чтение нижней грани
        for (int i = 0, ik = 2; i < 3; ++i, --ik)
            for (int j = 0; j < 3; ++j) {
                streamIn >> color;
                DownPlane[i][j]->setDownColor(color);
                visualColors[j][0][ik].setColor(2, setRGBColor(color));
            }

    }

    void PrintRubikCube(ostream &streamOut = cout) const {
        for (int i = 0; i < 3; ++i) {
            streamOut << "\t\t | ";
            for (int j = 0; j < 3; ++j)
                PrintColor(UpPlane[i][j]->UpColor(), streamOut);
            streamOut << "| \n";
        }

        streamOut << "\n | ";
        for (int j = 0; j < 3; ++j)
            PrintColor(LeftPlane[0][j]->LeftColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(FrontPlane[0][j]->FrontColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(RightPlane[0][j]->RightColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(BackPlane[0][j]->BackColor(), streamOut);
        streamOut << "| \n | ";

        for (int j = 0; j < 3; ++j)
            PrintColor(LeftPlane[1][j]->LeftColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(FrontPlane[1][j]->FrontColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(RightPlane[1][j]->RightColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(BackPlane[1][j]->BackColor(), streamOut);
        streamOut << "| \n | ";

        for (int j = 0; j < 3; ++j)
            PrintColor(LeftPlane[2][j]->LeftColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(FrontPlane[2][j]->FrontColor(), streamOut);
        streamOut << "| ";
```

```
        for (int j = 0; j < 3; ++j)
            PrintColor(RightPlane[2][j]->RightColor(), streamOut);
        streamOut << "| ";

        for (int j = 0; j < 3; ++j)
            PrintColor(BackPlane[2][j]->BackColor(), streamOut);
        streamOut << "| " << "\n\n";

        for (int i = 0; i < 3; ++i) {
            streamOut << "\t\t | ";
            for (int j = 0; j < 3; ++j)
                PrintColor(DownPlane[i][j]->DownColor(), streamOut);
            streamOut << "| \n";
        }
        streamOut << "\n";
    }

    [[nodiscard]] bool isCubeCompleted() const {
        if (!CheckIfCubeCorrect())
            throw exception();

        bool isFifthCompleted = isFifthStepCompleted();

        bool isFrontCornersCompleted =
                FrontPlane[0][0]->FrontColor() == FrontCenter &&
FrontPlane[0][2]->FrontColor() == FrontCenter;

        bool isBackCornersCompleted =
                BackPlane[0][0]->BackColor() == BackCenter &&
BackPlane[0][2]->BackColor() == BackCenter;

        bool isLeftCornersCompleted =
                LeftPlane[0][0]->LeftColor() == LeftCenter &&
LeftPlane[0][2]->LeftColor() == LeftCenter;

        bool isRightCornersCompleted =
                RightPlane[0][0]->RightColor() == RightCenter &&
RightPlane[0][2]->RightColor() == RightCenter;

        bool isCompleted =
                isFifthCompleted && isFrontCornersCompleted &&
isBackCornersCompleted && isLeftCornersCompleted &&
                isRightCornersCompleted;

        return isCompleted;

    }

    void PrintCubeInfo(ostream &output = cout) {
        if (CheckIfCubeCorrect() && FifthStepCounter != 0) {
            output << "\n\n";
            output << "+-----------------------------------------------------
+\n";
            output << "|           Amount of iterations by steps:
|\n";
            output << "+-----------------------------------------------------
+\n\n";
            output << "1st step) " << FirstStepCounter << " iterations" <<
endl;
            output << "2nd step) " << SecondStepCounter << " iterations" <<
endl;
            output << "3rd step) " << ThirdStepCounter << " iterations" <<
endl;
```

```
            output << "4th step) " << FourthStepCounter << " iterations" <<
endl;
            output << "5th step) " << FifthStepCounter << " iterations" <<
endl;
            output << "6th step) " << SixthStepCounter << " iterations" <<
endl;
            output << "7th step) " << SeventhStepCounter << " iterations" <<
endl;

        }

        (isCubeCompleted()) ?
        output << "\n\nRotates it took to solve cube: " << RotatesCounter <<
"\n\n" :
        output << "\n\n" << RotatesCounter << " rotates done, but cube isn't
solved\n\n";
    }

    void Shuffle(bool isWriteToConsole = true, int countTurns = 20 + rand() %
5) {
        for (int i = 0; i < countTurns; ++i) {
            int temp = 1 + rand() % 4;
            switch (temp) {
                case (1):
                    RightAlgorithm(isWriteToConsole);
                    break;
                case (2):
                    LeftAlgorithm(isWriteToConsole);
                    break;
                case (3):
                    UpAlgorithm(isWriteToConsole);
                    break;
                case (4):
                    DownAlgorithm(isWriteToConsole);
                    break;
                default:
                    break;
            }
        }
        if (isCubeCompleted())
            Shuffle(isWriteToConsole);
    }

    void FindSolution(bool isWriteToConsole = true) {
        if (!CheckIfCubeCorrect())
            throw logic_error("\n\nCube have a wrong centers, try to put
another one!\n");

        if (isCubeCompleted())
            throw logic_error("\n\nCube is already solved!\n");

        try {
            if (isWriteToConsole)
                cout << "\n\nSolving steps:\n";

            RotatesCounter = 0;
            FirstStep(0, isWriteToConsole);
            SecondStep(0, isWriteToConsole);
            ThirdStep(0, isWriteToConsole);
            FourthStep(isWriteToConsole);
            FifthStep(0, isWriteToConsole);
            SixthStep(0, isWriteToConsole);
            SeventhStep(0, false, isWriteToConsole);
```

```
            if (isCubeCompleted() && isWriteToConsole)
                cout << "\nAmount of rotates for solving: " <<
getRotatesCounter() << "\n";

        } catch (exception &e) {
            cout << e.what();
        }
    }

private:
    friend void specialKeys(int, int, int);
    friend void SolveCubeArray(int, bool);
    friend void display();
    friend void init();
    friend void processMenu(int);
    friend void timer(int);

    unsigned int FirstStepCounter = 0, SecondStepCounter = 0,
ThirdStepCounter = 0, FourthStepCounter = 0, FifthStepCounter = 0,
SixthStepCounter = 0, SeventhStepCounter = 0;
    unsigned int RotatesCounter = 0;

    // В arr мы храним слои нашего куба, в каждом из которых у нас есть 3
вектора с 3 мини-кубами
    vector<vector<vector<MiniCube>>> arr;

    // Плоскости
    Plane UpPlane;
    Plane DownPlane;
    Plane LeftPlane;
    Plane RightPlane;
    Plane FrontPlane;
    Plane BackPlane;

    // Заполнение одной плоскости (вектора указателей) мини-кубами по адресу
    void FillPlaneArr(Plane &tempPlane) {
        for (int i = 0; i < 3; ++i) {
            vector<MiniCube *> tempArr(3);
            for (int j = 0; j < 3; ++j)
                tempArr[j] = &arr[i][2][j];
            tempPlane.push_back(tempArr);
        }
    }

    // Распределение мини-кубов от векторов к плоскостям
    void PushInPlaneVector() {
        FillPlaneArr(UpPlane);
        FillPlaneArr(DownPlane);
        FillPlaneArr(LeftPlane);
        FillPlaneArr(RightPlane);
        FillPlaneArr(FrontPlane);
        FillPlaneArr(BackPlane);
    }

    unsigned int getRotatesCounter() const {
        return this->RotatesCounter;
    }

    // Проверка выполнения шагов
    [[nodiscard]] bool isFirstStepCompleted() const {
        bool isDownCompleted = (DownPlane[1][0]->DownColor() == DownCenter)
&& (DownPlane[1][2]->DownColor() == DownCenter) && (DownPlane[0][1]-
>DownColor() == DownCenter) && (DownPlane[2][1]->DownColor() == DownCenter);
        bool isLeftCompleted = LeftCenter == LeftPlane[2][1]->LeftColor();
```

```
        bool isRightCompleted = RightCenter == RightPlane[2][1]->RightColor();
        bool isFrontCompleted = FrontCenter == FrontPlane[2][1]->FrontColor();
        bool isBackCompleted = BackCenter == BackPlane[2][1]->BackColor();
        bool isCompleted = isDownCompleted && isLeftCompleted && isRightCompleted && isFrontCompleted && isBackCompleted;
        return isCompleted;
    }

    [[nodiscard]] bool isSecondStepCompleted() const {
        bool isFirstCompleted = isFirstStepCompleted();
        bool isDownCompleted = (DownPlane[0][0]->DownColor() == DownCenter) && (DownPlane[0][2]->DownColor() == DownCenter) && (DownPlane[2][0]->DownColor() == DownCenter) && (DownPlane[2][2]->DownColor() == DownCenter);
        bool isLeftCompleted = LeftPlane[2][0]->LeftColor() == LeftCenter && LeftPlane[2][2]->LeftColor() == LeftCenter;
        bool isRightCompleted = RightPlane[2][0]->RightColor() == RightCenter && RightPlane[2][2]->RightColor() == RightCenter;
        bool isFrontCompleted = FrontPlane[2][0]->FrontColor() == FrontCenter && FrontPlane[2][2]->FrontColor() == FrontCenter;
        bool isBackCompleted = BackPlane[2][0]->BackColor() == BackCenter && BackPlane[2][2]->BackColor() == BackCenter;
        bool isCompleted = isFirstCompleted && isDownCompleted && isLeftCompleted && isRightCompleted && isFrontCompleted && isBackCompleted;
        return isCompleted;
    }

    [[nodiscard]] bool isThirdStepCompleted() const {
        bool isSecondCompleted = isSecondStepCompleted();
        bool isLeftCompleted = LeftPlane[1][0]->LeftColor() == LeftCenter && LeftPlane[1][2]->LeftColor() == LeftCenter;
        bool isFrontCompleted = FrontPlane[1][0]->FrontColor() == FrontCenter && FrontPlane[1][2]->FrontColor() == FrontCenter;
        bool isRightCompleted = RightPlane[1][0]->RightColor() == RightCenter && RightPlane[1][2]->RightColor() == RightCenter;
        bool isBackCompleted = BackPlane[1][0]->BackColor() == BackCenter && BackPlane[1][2]->BackColor() == BackCenter;
        bool isCompleted = isSecondCompleted && isLeftCompleted && isRightCompleted && isFrontCompleted && isBackCompleted;
        return isCompleted;
    }

    [[nodiscard]] bool isFourthStepCompleted() const {
        bool isThirdCompleted = isThirdStepCompleted();
        bool isUpCompleted = UpPlane[0][1]->UpColor() == UpCenter && UpPlane[1][0]->UpColor() == UpCenter && UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]->UpColor() == UpCenter;
        bool isCompleted = isThirdCompleted && isUpCompleted;
        return isCompleted;
    }

    [[nodiscard]] bool isFifthStepCompleted() const {
        bool isFourthCompleted = isFourthStepCompleted();
        bool isLeftCompleted = LeftPlane[0][1]->LeftColor() == LeftCenter;
        bool isRightCompleted = RightPlane[0][1]->RightColor() == RightCenter;
        bool isFrontCompleted = FrontPlane[0][1]->FrontColor() == FrontCenter;
        bool isBackCompleted = BackPlane[0][1]->BackColor() == BackCenter;
        bool isCompleted = isFourthCompleted && isLeftCompleted && isRightCompleted && isFrontCompleted && isBackCompleted;
        return isCompleted;
    }
```

```
    [[nodiscard]] bool isSixthStepCompleted() const {
        bool isFifthCompleted = isFifthStepCompleted();
        bool isLeftCornersCompleted = (LeftPlane[0][0]->LeftColor() ==
LeftCenter ||
                                       LeftPlane[0][0]->LeftColor() ==
BackCenter || LeftPlane[0][0]->LeftColor() == UpCenter) &&
                                      (LeftPlane[0][2]->LeftColor() ==
LeftCenter || LeftPlane[0][2]->LeftColor() == FrontCenter ||
                                       LeftPlane[0][2]->LeftColor() ==
UpCenter);
        bool isBackCornersCompleted = (BackPlane[0][0]->BackColor() ==
BackCenter ||
                                       BackPlane[0][0]->BackColor() ==
RightCenter || BackPlane[0][0]->BackColor() == UpCenter) &&
                                      (BackPlane[0][2]->BackColor() ==
BackCenter || BackPlane[0][2]->BackColor() == LeftCenter ||
                                       BackPlane[0][2]->BackColor() ==
UpCenter);
        bool isRightCornersCompleted = (RightPlane[0][0]->RightColor() ==
RightCenter ||
                                        RightPlane[0][0]->RightColor() ==
FrontCenter || RightPlane[0][0]->RightColor() == UpCenter) &&
                                       (RightPlane[0][2]->RightColor() ==
RightCenter || RightPlane[0][2]->RightColor() == BackCenter ||
                                        RightPlane[0][2]->RightColor() ==
UpCenter);
        bool isFrontCornersCompleted = (FrontPlane[0][0]->FrontColor() ==
FrontCenter ||
                                        FrontPlane[0][0]->FrontColor() ==
LeftCenter || FrontPlane[0][0]->FrontColor() == UpCenter) &&
                                       (FrontPlane[0][2]->FrontColor() ==
FrontCenter || FrontPlane[0][2]->FrontColor() == RightCenter ||
                                        FrontPlane[0][2]->FrontColor() ==
UpCenter);
        bool isUpCornersCompleted =
                UpPlane[0][0]->UpColor() != FrontCenter && UpPlane[0][0]-
>UpColor() != RightCenter &&
                (UpPlane[0][0]->UpColor() == LeftCenter || UpPlane[0][0]-
>UpColor() == BackCenter ||
                 UpPlane[0][0]->UpColor() == UpCenter) &&
                UpPlane[0][2]->UpColor() != FrontCenter && UpPlane[0][2]-
>UpColor() != LeftCenter &&
                (UpPlane[0][2]->UpColor() == RightCenter || UpPlane[0][2]-
>UpColor() == BackCenter ||
                 UpPlane[0][2]->UpColor() == UpCenter) &&
                UpPlane[2][0]->UpColor() != BackCenter && UpPlane[2][0]-
>UpColor() != RightCenter &&
                (UpPlane[2][0]->UpColor() == LeftCenter || UpPlane[2][0]-
>UpColor() == FrontCenter ||
                 UpPlane[2][0]->UpColor() == UpCenter) &&
                UpPlane[2][2]->UpColor() != BackCenter && UpPlane[2][2]-
>UpColor() != LeftCenter &&
                (UpPlane[2][2]->UpColor() == RightCenter || UpPlane[2][2]-
>UpColor() == FrontCenter ||
                 UpPlane[2][2]->UpColor() == UpCenter);
        bool isCompleted = isFifthCompleted && isLeftCornersCompleted &&
isRightCornersCompleted && isFrontCornersCompleted && isBackCornersCompleted
&& isUpCornersCompleted;
        return isCompleted;
    }

    [[nodiscard]] bool CheckIfCubeCorrect() const {
        bool ifUpAndDownCentersCorrect = ((UpCenter == "W" || UpCenter ==
```

```
"white") && (DownCenter == "Y" || DownCenter == "yellow")) || ((UpCenter ==
"Y" || UpCenter == "yellow") && (DownCenter == "W" || DownCenter ==
"white"));

        bool ifOtherCentersCorrect = ((LeftCenter == "O" || LeftCenter ==
"orange") && (FrontCenter == "G" || FrontCenter == "green") &&
                                      (RightCenter == "R" || RightCenter ==
"red") && (BackCenter == "B" || BackCenter == "blue")) ||
                                     ((LeftCenter == "B" || LeftCenter ==
"blue") && (FrontCenter == "O" || FrontCenter == "orange") &&
                                      (RightCenter == "G" || RightCenter ==
"green") && (BackCenter == "R" || BackCenter == "red")) ||
                                     ((LeftCenter == "R" || LeftCenter ==
"red") && (FrontCenter == "B" || FrontCenter == "blue") &&
                                      (RightCenter == "O" || RightCenter ==
"orange") && (BackCenter == "G" || BackCenter == "green")) ||
                                     ((LeftCenter == "G" || LeftCenter ==
"green") && (FrontCenter == "R" || FrontCenter == "red") &&
                                      (RightCenter == "B" || RightCenter ==
"blue") && (BackCenter == "O" || BackCenter == "orange"));
        bool result = ifUpAndDownCentersCorrect && ifOtherCentersCorrect;
        return result;
    }

    // Этапы решения
    void FirstStep(int count = 0, bool isWriteToConsole = true) {
        if (FirstStepCounter++ > 128) throw logic_error("\nError while
solving cube number " + to_string(CubeCounter) + " (1st step)\n");

        if (!isFirstStepCompleted()) {
            if (FrontCenter == FrontPlane[0][1]->FrontColor() &&
UpPlane[2][1]->UpColor() == DownCenter) {
                RotateMachineGun("F F ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (RightCenter == RightPlane[0][1]->RightColor() &&
UpPlane[1][2]->UpColor() == DownCenter) {
                RotateMachineGun("R R ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (LeftCenter == LeftPlane[0][1]->LeftColor() && UpPlane[1][0]-
>UpColor() == DownCenter) {
                RotateMachineGun("L L ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (BackCenter == BackPlane[0][1]->BackColor() && UpPlane[0][1]-
>UpColor() == DownCenter) {
                RotateMachineGun("B B ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (FrontPlane[0][1]->FrontColor() == DownCenter &&
UpPlane[2][1]->UpColor() == FrontCenter) {
                RotateMachineGun("U'R'F R ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (UpPlane[1][2]->UpColor() == RightCenter && RightPlane[0][1]-
>RightColor() == DownCenter) {
                RotateMachineGun("U'B'R B ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (LeftPlane[0][1]->LeftColor() == DownCenter && UpPlane[1][0]-
>UpColor() == LeftCenter) {
                RotateMachineGun("U'F'L F ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
```

```
            }
            if (BackPlane[0][1]->BackColor() == DownCenter && UpPlane[0][1]-
>UpColor() == BackCenter) {
                RotateMachineGun("U'L'B L ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (FrontPlane[1][2]->FrontColor() == DownCenter) {
                RotateMachineGun("F'U'F ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (RightPlane[1][2]->RightColor() == DownCenter) {
                RotateMachineGun("R'U'R ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (LeftPlane[1][2]->LeftColor() == DownCenter) {
                RotateMachineGun("L'U'L ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (BackPlane[1][2]->BackColor() == DownCenter) {
                RotateMachineGun("B'U'B ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (FrontPlane[1][0]->FrontColor() == DownCenter) {
                RotateMachineGun("F U'F'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (RightPlane[1][0]->RightColor() == DownCenter) {
                RotateMachineGun("R U'R'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (LeftPlane[1][0]->LeftColor() == DownCenter) {
                RotateMachineGun("L U'L'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (BackPlane[1][0]->BackColor() == DownCenter) {
                RotateMachineGun("B U'B'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (FrontPlane[2][1]->FrontColor() == DownCenter) {
                RotateMachineGun("F F U'F F ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (RightPlane[2][1]->RightColor() == DownCenter) {
                RotateMachineGun("R R U'R R ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (LeftPlane[2][1]->LeftColor() == DownCenter) {
                RotateMachineGun("L L U'L L ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (BackPlane[2][1]->BackColor() == DownCenter) {
                RotateMachineGun("B B U'B B ", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (DownPlane[0][1]->DownColor() == DownCenter &&
FrontPlane[2][1]->FrontColor() != FrontCenter) {
                RotateMachineGun("F'F'U'F'F'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (DownPlane[1][2]->DownColor() == DownCenter &&
RightPlane[2][1]->RightColor() != RightCenter) {
                RotateMachineGun("R'R'U'R'R'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
```

```
            if (DownPlane[1][0]->DownColor() == DownCenter &&
LeftPlane[2][1]->LeftColor() != LeftCenter) {
                RotateMachineGun("L'L'U'L'L'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }
            if (DownPlane[2][1]->DownColor() == DownCenter &&
BackPlane[2][1]->BackColor() != BackCenter) {
                RotateMachineGun("B'B'U'B'B'", isWriteToConsole);
                FirstStep(0, isWriteToConsole);
            }

            if (!isFirstStepCompleted() && count < 4) {
                RotateMachineGun("U ", isWriteToConsole);
                FirstStep(++count, isWriteToConsole);
            } else if (!isFirstStepCompleted() && count == 4) {
                Shuffle(isWriteToConsole, 3);
                FirstStep(0, isWriteToConsole);
            }
        }
    }

    void SecondStep(int count = 0, bool isWriteToConsole = true) {
        if (SecondStepCounter++ > 128) throw logic_error("\nError while
solving cube number " + to_string(CubeCounter) + " (2nd step)\n");

        if (!isSecondStepCompleted()) {
            if (FrontPlane[2][1]->FrontColor() == FrontCenter &&
FrontPlane[0][2]->FrontColor() == DownCenter &&
                UpPlane[2][2]->UpColor() == FrontCenter && RightPlane[0][0]-
>RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter) {
                RotateMachineGun("F'U'F ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (FrontPlane[0][0]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                UpPlane[2][0]->UpColor() == LeftCenter && LeftPlane[0][2]-
>LeftColor() == DownCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter) {
                RotateMachineGun("L'U'L ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[0][0]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                UpPlane[0][2]->UpColor() == RightCenter && RightPlane[0][2]-
>RightColor() == DownCenter &&
                RightPlane[2][1]->RightColor() == RightCenter) {
                RotateMachineGun("R'U'R ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[0][2]->BackColor() == DownCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                LeftPlane[0][0]->LeftColor() == LeftCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                UpPlane[0][0]->UpColor() == BackCenter) {
                RotateMachineGun("B'U'B ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (FrontPlane[0][2]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                RightPlane[0][0]->RightColor() == DownCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                UpPlane[2][2]->UpColor() == RightCenter) {
                RotateMachineGun("R U R'", isWriteToConsole);
```

```
                SecondStep(0, isWriteToConsole);
            }
            if (FrontPlane[0][0]->FrontColor() == DownCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                UpPlane[2][0]->UpColor() == FrontCenter && LeftPlane[0][2]-
>LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter) {
                RotateMachineGun("F U F'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[0][0]->BackColor() == DownCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                UpPlane[0][2]->UpColor() == BackCenter && RightPlane[0][2]-
>RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter) {
                RotateMachineGun("B U B'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[0][2]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                LeftPlane[0][0]->LeftColor() == DownCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                UpPlane[0][0]->UpColor() == LeftCenter) {
                RotateMachineGun("L U L'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (FrontPlane[0][2]->FrontColor() == RightCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                RightPlane[0][0]->RightColor() == FrontCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                UpPlane[2][2]->UpColor() == DownCenter) {
                RotateMachineGun("R U'R'U U R U R'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (FrontPlane[0][0]->FrontColor() == LeftCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                UpPlane[2][0]->UpColor() == DownCenter && RightPlane[0][2]-
>RightColor() == FrontCenter &&
                RightPlane[2][1]->RightColor() == RightCenter) {
                RotateMachineGun("F U'F'U U F U F'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[0][0]->BackColor() == RightCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                UpPlane[0][2]->UpColor() == DownCenter && RightPlane[0][2]-
>RightColor() == BackCenter &&
                RightPlane[2][1]->RightColor() == RightCenter) {
                RotateMachineGun("B U'B'U U B U B'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[0][2]->BackColor() == LeftCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                LeftPlane[0][0]->LeftColor() == BackCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                UpPlane[0][0]->UpColor() == DownCenter) {
                RotateMachineGun("L U'L'U U L U L'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (FrontPlane[2][1]->FrontColor() == FrontCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                (FrontPlane[2][2]->FrontColor() == DownCenter ||
RightPlane[2][0]->RightColor() == DownCenter)) {
                RotateMachineGun("F'U'F ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
```

```
            }
            if (FrontPlane[2][1]->FrontColor() == FrontCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                (LeftPlane[2][2]->LeftColor() == DownCenter ||
FrontPlane[2][0]->FrontColor() == DownCenter)) {
                RotateMachineGun("L'U'L ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[2][1]->BackColor() == BackCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                (RightPlane[2][2]->RightColor() == DownCenter ||
BackPlane[2][0]->BackColor() == DownCenter)) {
                RotateMachineGun("R'U'R ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (BackPlane[2][1]->BackColor() == BackCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                (BackPlane[2][2]->BackColor() == DownCenter ||
LeftPlane[2][0]->LeftColor() == DownCenter)) {
                RotateMachineGun("B'U'B ", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
            if (LeftPlane[2][1]->LeftColor() == LeftCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                DownPlane[0][0]->DownColor() == DownCenter &&
DownPlane[0][1]->DownColor() == DownCenter &&
                DownPlane[1][0]->DownColor() == DownCenter &&
DownPlane[1][2]->DownColor() == DownCenter &&
                DownPlane[2][1]->DownColor() == DownCenter &&
(FrontPlane[2][0]->FrontColor() != FrontCenter ||

LeftPlane[2][2]->LeftColor() != LeftCenter)) {
                RotateMachineGun("F U'F'", isWriteToConsole);
                SecondStep(0, isWriteToConsole); // попробовать поставить XOR
            }
            if (RightPlane[2][1]->RightColor() == RightCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                DownPlane[0][1]->DownColor() == DownCenter &&
DownPlane[0][2]->DownColor() == DownCenter &&
                DownPlane[1][0]->DownColor() == DownCenter &&
DownPlane[1][2]->DownColor() == DownCenter &&
                DownPlane[2][1]->DownColor() == DownCenter &&
(FrontPlane[2][2]->FrontColor() != FrontCenter ||

RightPlane[2][0]->RightColor() != RightCenter)) {
                RotateMachineGun("R U'R'", isWriteToConsole);
                SecondStep(0, isWriteToConsole); // попробовать поставить XOR
            }
            if (BackPlane[2][1]->BackColor() == BackCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                DownPlane[2][2]->DownColor() == DownCenter &&
DownPlane[2][1]->DownColor() == DownCenter &&
                DownPlane[1][2]->DownColor() == DownCenter &&
DownPlane[1][0]->DownColor() == DownCenter &&
                DownPlane[0][1]->DownColor() == DownCenter &&
(BackPlane[2][0]->BackColor() != BackCenter ||

RightPlane[2][2]->RightColor() != RightCenter)) {
                RotateMachineGun("B U'B'", isWriteToConsole);
                SecondStep(0, isWriteToConsole); // попробовать поставить XOR
            }
            if (BackPlane[2][1]->BackColor() == BackCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                DownPlane[2][0]->DownColor() == DownCenter &&
```

```
DownPlane[2][1]->DownColor() == DownCenter &&
                DownPlane[1][2]->DownColor() == DownCenter &&
DownPlane[1][0]->DownColor() == DownCenter &&
                DownPlane[0][1]->DownColor() == DownCenter &&
(BackPlane[2][2]->BackColor() != BackCenter ||

LeftPlane[2][0]->LeftColor() != LeftCenter)) {
                RotateMachineGun("L U'L'", isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }

            if (!isSecondStepCompleted() && count < 4) {
                RotateMachineGun("U ", isWriteToConsole);
                SecondStep(++count, isWriteToConsole);
            } else if (!isSecondStepCompleted() && count == 4) {
                Shuffle(isWriteToConsole, 3);
                FirstStep(0, isWriteToConsole);
                SecondStep(0, isWriteToConsole);
            }
        }
    }

    void ThirdStep(int count = 0, bool isWriteToConsole = true) {
        if (ThirdStepCounter++ > 128) logic_error("\nError while solving cube
number " + to_string(CubeCounter) + " (3rd step)\n");

        if (!isThirdStepCompleted()) {
            if (FrontPlane[2][2]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
FrontPlane[0][1]->FrontColor() == FrontCenter &&
                LeftPlane[2][2]->LeftColor() == LeftCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                LeftPlane[2][0]->LeftColor() == LeftCenter && UpPlane[2][1]-
>UpColor() == LeftCenter) {
                RotateMachineGun("U'L'U L U F U'F'", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
            if (FrontPlane[2][2]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[0][1]->RightColor() == RightCenter &&
UpPlane[1][2]->UpColor() == FrontCenter) {
                RotateMachineGun("U'F'U F U R U'R'", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
            if (BackPlane[2][2]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                BackPlane[2][0]->BackColor() == BackCenter &&
BackPlane[0][1]->BackColor() == BackCenter &&
                RightPlane[2][2]->RightColor() == RightCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                RightPlane[2][0]->RightColor() == RightCenter &&
UpPlane[0][1]->UpColor() == RightCenter) {
                RotateMachineGun("U'R'U R U B U'B'", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
            if (BackPlane[2][2]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                BackPlane[2][0]->BackColor() == BackCenter &&
LeftPlane[0][1]->LeftColor() == LeftCenter &&
```

```
                LeftPlane[2][2]->LeftColor() == LeftCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                LeftPlane[2][0]->LeftColor() == LeftCenter && UpPlane[1][0]-
>UpColor() == BackCenter) {
                RotateMachineGun("U'B'U B U L U'L'", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
            if (RightPlane[2][2]->RightColor() == RightCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                RightPlane[2][0]->RightColor() == RightCenter &&
FrontPlane[0][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][2]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
UpPlane[2][1]->UpColor() == RightCenter) {
                RotateMachineGun("U R U'R'U'F'U F ", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
            if (FrontPlane[2][2]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
LeftPlane[0][1]->LeftColor() == LeftCenter &&
                LeftPlane[2][2]->LeftColor() == LeftCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                LeftPlane[2][0]->LeftColor() == LeftCenter && UpPlane[1][0]-
>UpColor() == FrontCenter) {
                RotateMachineGun("U F U'F'U'L'U L ", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
            if (LeftPlane[2][2]->LeftColor() == LeftCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                LeftPlane[2][0]->LeftColor() == LeftCenter &&
BackPlane[0][1]->BackColor() == BackCenter &&
                BackPlane[2][2]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                BackPlane[2][0]->BackColor() == BackCenter && UpPlane[0][1]-
>UpColor() == LeftCenter) {
                RotateMachineGun("U L U'L'U'B'U B ", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
            if (BackPlane[2][2]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                BackPlane[2][0]->BackColor() == BackCenter &&
RightPlane[0][1]->RightColor() == RightCenter &&
                RightPlane[2][2]->RightColor() == RightCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                RightPlane[2][0]->RightColor() == RightCenter &&
UpPlane[1][2]->UpColor() == BackCenter) {
                RotateMachineGun("U B U'B'U'R'U R ", isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }

            if (!isThirdStepCompleted() && count < 4) {
                RotateMachineGun("U ", isWriteToConsole);
                ThirdStep(++count, isWriteToConsole);
            } else if (count == 4 && !isThirdStepCompleted()) {
                if (FrontPlane[2][2]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                    FrontPlane[2][0]->FrontColor() == FrontCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                    RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                    (FrontPlane[1][2]->FrontColor() != FrontCenter ||
RightPlane[1][0]->RightColor() != RightCenter)) {
```

```
                    RotateMachineGun("U R U'R'U'F'U F ", isWriteToConsole);
                    ThirdStep(0, isWriteToConsole);
                } else if (
                        FrontPlane[2][2]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                        FrontPlane[2][0]->FrontColor() == FrontCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                        LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                        (FrontPlane[1][0]->FrontColor() != FrontCenter ||
LeftPlane[1][2]->LeftColor() != LeftCenter)) {
                    RotateMachineGun("U F U'F'U'L'U L ", isWriteToConsole);
                    ThirdStep(0, isWriteToConsole);
                } else if (
                        BackPlane[2][2]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                        BackPlane[2][0]->BackColor() == BackCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                        LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                        (BackPlane[1][2]->BackColor() != BackCenter ||
LeftPlane[1][0]->LeftColor() != LeftCenter)) {
                    RotateMachineGun("U L U'L'U'B'U B ", isWriteToConsole);
                    ThirdStep(0, isWriteToConsole);
                } else if (
                        BackPlane[2][2]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                        BackPlane[2][0]->BackColor() == BackCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                        RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                        (BackPlane[1][0]->BackColor() != BackCenter ||
RightPlane[1][2]->RightColor() != RightCenter)) {
                    RotateMachineGun("U B U'B'U'R'U R ", isWriteToConsole);
                    ThirdStep(0, isWriteToConsole);
                }
            }

            if (!isThirdStepCompleted() and count == 4) {
                Shuffle(isWriteToConsole, 3);
                SecondStep(0, isWriteToConsole);
                ThirdStep(0, isWriteToConsole);
            }
        }
    }

    void FourthStep(bool isWriteToConsole = true) {
        if (FourthStepCounter++ > 64) throw logic_error("\nError while
solving cube number " + to_string(CubeCounter) + " (4th step)\n");

        if (!isFourthStepCompleted()) {
            if (FrontPlane[0][1]->FrontColor() == UpCenter &&
FrontPlane[1][0]->FrontColor() == FrontCenter &&
                FrontPlane[1][2]->FrontColor() == FrontCenter &&
FrontPlane[2][0]->FrontColor() == FrontCenter &&
                FrontPlane[2][1]->FrontColor() == FrontCenter &&
FrontPlane[2][2]->FrontColor() == FrontCenter &&
                RightPlane[0][1]->RightColor() == UpCenter &&
RightPlane[1][0]->RightColor() == RightCenter &&
                RightPlane[1][2]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                UpPlane[0][1]->UpColor() == UpCenter && UpPlane[1][0]-
```

```
>UpColor() == UpCenter) {
                RotateMachineGun("F U R U'R'F'", isWriteToConsole);
                FourthStep(isWriteToConsole);
            }
            if (FrontPlane[0][1]->FrontColor() == UpCenter &&
FrontPlane[1][0]->FrontColor() == FrontCenter &&
                FrontPlane[1][2]->FrontColor() == FrontCenter &&
FrontPlane[2][0]->FrontColor() == FrontCenter &&
                FrontPlane[2][1]->FrontColor() == FrontCenter &&
FrontPlane[2][2]->FrontColor() == FrontCenter &&
                LeftPlane[0][1]->LeftColor() == UpCenter && LeftPlane[1][0]-
>LeftColor() == LeftCenter &&
                LeftPlane[1][2]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                UpPlane[0][1]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter) {
                RotateMachineGun("L U F U'F'L'", isWriteToConsole);
                FourthStep(isWriteToConsole);
            }
            if (BackPlane[0][1]->BackColor() == UpCenter && BackPlane[1][0]-
>BackColor() == BackCenter &&
                BackPlane[1][2]->BackColor() == BackCenter &&
BackPlane[2][0]->BackColor() == BackCenter &&
                BackPlane[2][1]->BackColor() == BackCenter &&
BackPlane[2][2]->BackColor() == BackCenter &&
                RightPlane[0][1]->RightColor() == UpCenter &&
RightPlane[1][0]->RightColor() == RightCenter &&
                RightPlane[1][2]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                UpPlane[1][0]->UpColor() == UpCenter && UpPlane[2][1]-
>UpColor() == UpCenter) {
                RotateMachineGun("R U B U'B'R'", isWriteToConsole);
                FourthStep(isWriteToConsole);
            }
            if (BackPlane[0][1]->BackColor() == UpCenter && BackPlane[1][0]-
>BackColor() == BackCenter &&
                BackPlane[1][2]->BackColor() == BackCenter &&
BackPlane[2][0]->BackColor() == BackCenter &&
                BackPlane[2][1]->BackColor() == BackCenter &&
BackPlane[2][2]->BackColor() == BackCenter &&
                LeftPlane[0][1]->LeftColor() == UpCenter && LeftPlane[1][0]-
>LeftColor() == LeftCenter &&
                LeftPlane[1][2]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]-
>UpColor() == UpCenter) {
                RotateMachineGun("B U L U'L'B'", isWriteToConsole);
                FourthStep(isWriteToConsole);
            }
            if (FrontPlane[0][1]->FrontColor() == UpCenter &&
FrontPlane[1][0]->FrontColor() == FrontCenter &&
                FrontPlane[1][2]->FrontColor() == FrontCenter &&
FrontPlane[2][0]->FrontColor() == FrontCenter &&
                FrontPlane[2][1]->FrontColor() == FrontCenter &&
FrontPlane[2][2]->FrontColor() == FrontCenter &&
                RightPlane[1][0]->RightColor() == RightCenter &&
RightPlane[1][2]->RightColor() == RightCenter &&
                RightPlane[2][0]->RightColor() == RightCenter &&
```

```
RightPlane[2][1]->RightColor() == RightCenter &&
                RightPlane[2][2]->RightColor() == RightCenter &&
UpPlane[1][0]->UpColor() == UpCenter &&
                UpPlane[1][2]->UpColor() == UpCenter && BackPlane[0][1]-
>BackColor() == UpCenter) {
                RotateMachineGun("F R U R'U'F'", isWriteToConsole);
                FourthStep(isWriteToConsole);
            }
            if (FrontPlane[1][0]->FrontColor() == FrontCenter &&
FrontPlane[1][2]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][2]->FrontColor() == FrontCenter &&
RightPlane[0][1]->RightColor() == UpCenter &&
                RightPlane[1][0]->RightColor() == RightCenter &&
RightPlane[1][2]->RightColor() == RightCenter &&
                RightPlane[2][0]->RightColor() == RightCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                RightPlane[2][2]->RightColor() == RightCenter &&
UpPlane[0][1]->UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter && LeftPlane[0][1]-
>LeftColor() == UpCenter) {
                RotateMachineGun("R B U B'U'R'", isWriteToConsole);
                FourthStep(isWriteToConsole);
            }
            if (FrontPlane[1][0]->FrontColor() == FrontCenter &&
FrontPlane[1][2]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][2]->FrontColor() == FrontCenter &&
RightPlane[1][0]->RightColor() == RightCenter &&
                RightPlane[1][2]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                FrontPlane[0][1]->FrontColor() == UpCenter &&
RightPlane[0][1]->RightColor() == UpCenter &&
                LeftPlane[0][1]->LeftColor() == UpCenter && BackPlane[0][1]-
>BackColor() == UpCenter) {
                RotateMachineGun("F U R U'R'F'R B U B'U'R'",
isWriteToConsole);
                FourthStep(isWriteToConsole);
            }

            if (!isFourthStepCompleted()) {
                Shuffle(isWriteToConsole, 3);
                ThirdStep(0, isWriteToConsole);
                FourthStep(isWriteToConsole);
            }
        }
    }

    void FifthStep(int count = 0, bool isWriteToConsole = true) {
        if (FifthStepCounter++ > 32) throw logic_error("\nError while solving
cube number " + to_string(CubeCounter) + " (5th step)\n");

        if (!isFifthStepCompleted()) {
            if (FrontPlane[1][0]->FrontColor() == FrontCenter &&
FrontPlane[1][2]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][2]->FrontColor() == FrontCenter &&
RightPlane[1][0]->RightColor() == RightCenter &&
                RightPlane[1][2]->RightColor() == RightCenter &&
```

```
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                RightPlane[0][1]->RightColor() == FrontCenter &&
UpPlane[0][1]->UpColor() == UpCenter &&
                UpPlane[1][0]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter) {
                RotateMachineGun("U L'U'U'L U L'U L ", isWriteToConsole);
                FifthStep(0, isWriteToConsole);
            }
            if (FrontPlane[1][0]->FrontColor() == FrontCenter &&
FrontPlane[1][2]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][2]->FrontColor() == FrontCenter &&
LeftPlane[1][0]->LeftColor() == LeftCenter &&
                LeftPlane[1][2]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                LeftPlane[0][1]->LeftColor() == FrontCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][0]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter) {
                RotateMachineGun("U B'U'U'B U B'U B ", isWriteToConsole);
                FifthStep(0, isWriteToConsole);
            }
            if (FrontPlane[1][0]->FrontColor() == FrontCenter &&
FrontPlane[1][2]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][2]->FrontColor() == FrontCenter &&
RightPlane[1][0]->RightColor() == RightCenter &&
                RightPlane[1][2]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                BackPlane[0][1]->BackColor() == FrontCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][0]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter) {
                RotateMachineGun("U R'U'U'R U R'U R U B'U'U'B U B'U B U
R'U'U'R U R'U R ", isWriteToConsole);
                FifthStep(0, isWriteToConsole);
            }
            if (BackPlane[1][0]->BackColor() == BackCenter &&
BackPlane[1][2]->BackColor() == BackCenter &&
                BackPlane[2][0]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                BackPlane[2][2]->BackColor() == BackCenter &&
BackPlane[0][1]->BackColor() == RightCenter &&
                RightPlane[1][0]->RightColor() == RightCenter &&
RightPlane[1][2]->RightColor() == RightCenter &&
                RightPlane[2][0]->RightColor() == RightCenter &&
RightPlane[2][1]->RightColor() == RightCenter &&
                RightPlane[2][2]->RightColor() == RightCenter &&
UpPlane[0][1]->UpColor() == UpCenter &&
                UpPlane[1][0]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter) {
                RotateMachineGun("U F'U'U'F U F'U F ", isWriteToConsole);
```

```
                FifthStep(0, isWriteToConsole);
            }
            if (BackPlane[1][0]->BackColor() == BackCenter &&
BackPlane[1][2]->BackColor() == BackCenter &&
                BackPlane[2][0]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                BackPlane[2][2]->BackColor() == BackCenter &&
RightPlane[1][0]->RightColor() == RightCenter &&
                RightPlane[1][2]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                LeftPlane[0][1]->LeftColor() == RightCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][0]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter) {
                RotateMachineGun("U B'U'U'B U B'U B U L'U'U'L U L'U L U
B'U'U'B U B'U B ", isWriteToConsole);
                FifthStep(0, isWriteToConsole);
            }
            if (BackPlane[1][0]->BackColor() == BackCenter &&
BackPlane[1][2]->BackColor() == BackCenter &&
                BackPlane[2][0]->BackColor() == BackCenter &&
BackPlane[2][1]->BackColor() == BackCenter &&
                BackPlane[2][2]->BackColor() == BackCenter &&
LeftPlane[1][0]->LeftColor() == LeftCenter &&
                LeftPlane[1][2]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                LeftPlane[0][1]->LeftColor() == BackCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][0]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter) {
                RotateMachineGun("U R'U'U'R U R'U R ", isWriteToConsole);
                FifthStep(0, isWriteToConsole);
            }

            if (!isFifthStepCompleted() && count < 4) {
                FifthStep(++count, isWriteToConsole);
            } else if (!isFifthStepCompleted() && count == 4) {
                Shuffle(isWriteToConsole, 3);
                FourthStep(isWriteToConsole);
                FifthStep(0, isWriteToConsole);
            }
        }
    }

    void SixthStep(int count = 0, bool isWriteToConsole = true) {
        if (SixthStepCounter++ > 32) throw logic_error("\nError while solving
cube number " + to_string(CubeCounter) + " (6th step)\n");

        if (!isSixthStepCompleted()) {
            if (FrontPlane[0][1]->FrontColor() == FrontCenter &&
FrontPlane[1][0]->FrontColor() == FrontCenter &&
                FrontPlane[1][2]->FrontColor() == FrontCenter &&
FrontPlane[2][0]->FrontColor() == FrontCenter &&
                FrontPlane[2][1]->FrontColor() == FrontCenter &&
FrontPlane[2][2]->FrontColor() == FrontCenter &&
                LeftPlane[0][1]->LeftColor() == LeftCenter &&
LeftPlane[1][0]->LeftColor() == LeftCenter &&
                LeftPlane[1][2]->LeftColor() == LeftCenter &&
```

```
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                UpPlane[0][1]->UpColor() == UpCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]-
>UpColor() == UpCenter &&
                (LeftPlane[0][2]->LeftColor() == FrontCenter ||
LeftPlane[0][2]->LeftColor() == RightCenter ||
                 LeftPlane[0][2]->LeftColor() == UpCenter) &&
(FrontPlane[0][0]->FrontColor() == RightCenter ||

FrontPlane[0][0]->FrontColor() == UpCenter || FrontPlane[0][0]->FrontColor()
== FrontCenter) &&
                (UpPlane[2][0]->UpColor() == UpCenter || UpPlane[2][0]-
>UpColor() == RightCenter ||
                 UpPlane[2][0]->UpColor() == FrontCenter)) {

                RotateMachineGun("F'L'F R'F'L F R ", isWriteToConsole);
                SixthStep(0, isWriteToConsole);
            }
            if (BackPlane[0][1]->BackColor() == BackCenter &&
BackPlane[1][0]->BackColor() == BackCenter &&
                BackPlane[1][2]->BackColor() == BackCenter &&
BackPlane[2][0]->BackColor() == BackCenter &&
                BackPlane[2][1]->BackColor() == BackCenter &&
BackPlane[2][2]->BackColor() == BackCenter &&
                RightPlane[0][1]->RightColor() == RightCenter &&
RightPlane[1][0]->RightColor() == RightCenter &&
                RightPlane[1][2]->RightColor() == RightCenter &&
RightPlane[2][0]->RightColor() == RightCenter &&
                RightPlane[2][1]->RightColor() == RightCenter &&
RightPlane[2][2]->RightColor() == RightCenter &&
                UpPlane[0][1]->UpColor() == UpCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]-
>UpColor() == UpCenter &&
                (RightPlane[0][2]->RightColor() == FrontCenter ||
RightPlane[0][2]->RightColor() == RightCenter ||
                 RightPlane[0][2]->RightColor() == UpCenter) &&
                (BackPlane[0][0]->BackColor() == RightCenter ||
BackPlane[0][0]->BackColor() == UpCenter ||
                 BackPlane[0][0]->BackColor() == FrontCenter) &&
                (UpPlane[0][2]->UpColor() == UpCenter || UpPlane[0][2]-
>UpColor() == RightCenter ||
                 UpPlane[0][2]->UpColor() == FrontCenter)) {
                RotateMachineGun("R'F'L'F R F'L F ", isWriteToConsole);
                SixthStep(0, isWriteToConsole);
            }
            if (BackPlane[0][1]->BackColor() == BackCenter &&
BackPlane[1][0]->BackColor() == BackCenter &&
                BackPlane[1][2]->BackColor() == BackCenter &&
BackPlane[2][0]->BackColor() == BackCenter &&
                BackPlane[2][1]->BackColor() == BackCenter &&
BackPlane[2][2]->BackColor() == BackCenter &&
                LeftPlane[0][1]->LeftColor() == LeftCenter &&
LeftPlane[1][0]->LeftColor() == LeftCenter &&
                LeftPlane[1][2]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                UpPlane[0][1]->UpColor() == UpCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]-
```

```
>UpColor() == UpCenter &&
                (BackPlane[0][2]->BackColor() == FrontCenter ||
BackPlane[0][2]->BackColor() == UpCenter ||
                 BackPlane[0][2]->BackColor() == RightCenter) &&
                (LeftPlane[0][0]->LeftColor() == RightCenter ||
LeftPlane[0][0]->LeftColor() == UpCenter ||
                 LeftPlane[0][0]->LeftColor() == FrontCenter) &&
                (UpPlane[0][0]->UpColor() == UpCenter || UpPlane[0][0]-
>UpColor() == RightCenter ||
                 UpPlane[0][0]->UpColor() == FrontCenter)) {
                RotateMachineGun("F'L'B'L F L'B L ", isWriteToConsole);
                SixthStep(0, isWriteToConsole);
            }
            if (FrontPlane[0][1]->FrontColor() == FrontCenter &&
                FrontPlane[1][0]->FrontColor() == FrontCenter &&
FrontPlane[1][2]->FrontColor() == FrontCenter &&
                FrontPlane[2][0]->FrontColor() == FrontCenter &&
FrontPlane[2][1]->FrontColor() == FrontCenter &&
                FrontPlane[2][2]->FrontColor() == FrontCenter &&
LeftPlane[0][1]->LeftColor() == LeftCenter &&
                LeftPlane[1][0]->LeftColor() == LeftCenter &&
LeftPlane[1][2]->LeftColor() == LeftCenter &&
                LeftPlane[2][0]->LeftColor() == LeftCenter &&
LeftPlane[2][1]->LeftColor() == LeftCenter &&
                LeftPlane[2][2]->LeftColor() == LeftCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[0][1]->UpColor() == UpCenter && UpPlane[1][2]-
>UpColor() == UpCenter &&
                UpPlane[2][1]->UpColor() == UpCenter &&
                (LeftPlane[0][2]->LeftColor() == RightCenter ||
LeftPlane[0][2]->LeftColor() == BackCenter ||
                 LeftPlane[0][2]->LeftColor() == UpCenter) &&
                (FrontPlane[0][0]->FrontColor() == BackCenter ||
FrontPlane[0][0]->FrontColor() == RightCenter ||
                 FrontPlane[0][0]->FrontColor() == UpCenter) &&
                (UpPlane[2][0]->UpColor() == UpCenter || UpPlane[2][0]-
>UpColor() == RightCenter ||
                 UpPlane[2][0]->UpColor() == BackCenter)) {
                RotateMachineGun("B'R'B L'B'R B L ", isWriteToConsole);
                SixthStep(0, isWriteToConsole);
            }
            if (BackPlane[0][1]->BackColor() == BackCenter &&
BackPlane[1][0]->BackColor() == BackCenter &&
                BackPlane[1][2]->BackColor() == BackCenter &&
BackPlane[2][0]->BackColor() == BackCenter &&
                BackPlane[2][1]->BackColor() == BackCenter &&
BackPlane[2][2]->BackColor() == BackCenter &&
                LeftPlane[0][1]->LeftColor() == LeftCenter &&
LeftPlane[1][0]->LeftColor() == LeftCenter &&
                LeftPlane[1][2]->LeftColor() == LeftCenter &&
LeftPlane[2][0]->LeftColor() == LeftCenter &&
                LeftPlane[2][1]->LeftColor() == LeftCenter &&
LeftPlane[2][2]->LeftColor() == LeftCenter &&
                UpPlane[0][1]->UpColor() == UpCenter && UpPlane[0][1]-
>UpColor() == UpCenter &&
                UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]-
>UpColor() == UpCenter &&
                (BackPlane[0][2]->BackColor() == RightCenter ||
BackPlane[0][2]->BackColor() == BackCenter ||
                 BackPlane[0][2]->BackColor() == UpCenter) &&
                (LeftPlane[0][0]->LeftColor() == BackCenter ||
LeftPlane[0][0]->LeftColor() == UpCenter ||
                 LeftPlane[0][0]->LeftColor() == RightCenter) &&
                (UpPlane[0][0]->UpColor() == UpCenter || UpPlane[0][0]-
```

```
>UpColor() == BackCenter ||
                UpPlane[0][0]->UpColor() == RightCenter)) {
                RotateMachineGun("L'B'R'B L B'R B ", isWriteToConsole);
                SixthStep(0, isWriteToConsole);
            }

            if (!isSixthStepCompleted() && count < 4) {
                SixthStep(++count, isWriteToConsole);
            } else if (!isSixthStepCompleted() && count == 4) {
                Shuffle(isWriteToConsole, 3);
                FifthStep(0, isWriteToConsole);
                SixthStep(0, isWriteToConsole);
            }

        }
    }

    void SeventhStep(int count = 0, bool isSequenceStarted = false, bool
isWriteToConsole = true) {
        if (SeventhStepCounter++ > 16) throw logic_error("\nError while
solving cube number " + to_string(CubeCounter) + " (7th step)\n");

        if (!isCubeCompleted()) {
            if (UpPlane[0][1]->UpColor() == UpCenter && UpPlane[1][0]-
>UpColor() == UpCenter &&
                UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]-
>UpColor() == UpCenter &&
                RightPlane[0][0]->RightColor() == UpCenter) {
                RotateMachineGun("F'R F R'F'R F R'", isWriteToConsole);
                if (isSequenceStarted)
                    RotateMachineGun("U ", isWriteToConsole);
                else isSequenceStarted = true;

            } else if (UpPlane[0][1]->UpColor() == UpCenter && UpPlane[1][0]-
>UpColor() == UpCenter &&
                       UpPlane[1][2]->UpColor() == UpCenter && UpPlane[2][1]-
>UpColor() == UpCenter &&
                       FrontPlane[0][2]->FrontColor() == UpCenter) {
                RotateMachineGun("R F'R'F R F'R'F ", isWriteToConsole);
                if (isSequenceStarted)
                    RotateMachineGun("U ", isWriteToConsole);
                else isSequenceStarted = true;
            }

            if (!isCubeCompleted()) {
                RotateMachineGun("U ", isWriteToConsole);
                SeventhStep(++count, isSequenceStarted, isWriteToConsole);
            }
        }
    }

    // Rotates
    void RotateUpPlane(const char degree, bool isWriteToConsole = false) {
        RotatesCounter++;
        string right_0_0 = RightPlane[0][0]->RightColor(), right_0_1 =
RightPlane[0][1]->RightColor(), right_0_2 = RightPlane[0][2]->RightColor();

        if (degree == '+') {
            if (isWriteToConsole)
                cout << "U ";

            string up_0_0 = UpPlane[0][0]->UpColor(),
                   up_1_0 = UpPlane[1][0]->UpColor();
```

```
            UpPlane[0][0]->setUpColor(UpPlane[2][0]->UpColor());
            UpPlane[1][0]->setUpColor(UpPlane[2][1]->UpColor());
            UpPlane[2][0]->setUpColor(UpPlane[2][2]->UpColor());
            UpPlane[2][1]->setUpColor(UpPlane[1][2]->UpColor());
            UpPlane[2][2]->setUpColor(UpPlane[0][2]->UpColor());
            UpPlane[1][2]->setUpColor(UpPlane[0][1]->UpColor());
            UpPlane[0][2]->setUpColor(up_0_0);
            UpPlane[0][1]->setUpColor(up_1_0);

            RightPlane[0][0]->setRightColor(BackPlane[0][0]->BackColor());
            RightPlane[0][1]->setRightColor(BackPlane[0][1]->BackColor());
            RightPlane[0][2]->setRightColor(BackPlane[0][2]->BackColor());

            BackPlane[0][0]->setBackColor(LeftPlane[0][0]->LeftColor());
            BackPlane[0][1]->setBackColor(LeftPlane[0][1]->LeftColor());
            BackPlane[0][2]->setBackColor(LeftPlane[0][2]->LeftColor());

            LeftPlane[0][0]->setLeftColor(FrontPlane[0][0]->FrontColor());
            LeftPlane[0][1]->setLeftColor(FrontPlane[0][1]->FrontColor());
            LeftPlane[0][2]->setLeftColor(FrontPlane[0][2]->FrontColor());

            FrontPlane[0][0]->setFrontColor(right_0_0);
            FrontPlane[0][1]->setFrontColor(right_0_1);
            FrontPlane[0][2]->setFrontColor(right_0_2);

            visualRotateMiniMachineGun(3, ROTATE_START_VALUE, -1);

        } else if (degree == '-') {
            if (isWriteToConsole)
                cout << "U' ";

            string up_0_2 = UpPlane[0][2]->UpColor(),
                   up_1_2 = UpPlane[1][2]->UpColor();

            UpPlane[0][2]->setUpColor(UpPlane[2][2]->UpColor());
            UpPlane[1][2]->setUpColor(UpPlane[2][1]->UpColor());
            UpPlane[2][2]->setUpColor(UpPlane[2][0]->UpColor());
            UpPlane[2][1]->setUpColor(UpPlane[1][0]->UpColor());
            UpPlane[2][0]->setUpColor(UpPlane[0][0]->UpColor());
            UpPlane[1][0]->setUpColor(UpPlane[0][1]->UpColor());
            UpPlane[0][1]->setUpColor(up_1_2);
            UpPlane[0][0]->setUpColor(up_0_2);

            RightPlane[0][0]->setRightColor(FrontPlane[0][0]->FrontColor());
            RightPlane[0][1]->setRightColor(FrontPlane[0][1]->FrontColor());
            RightPlane[0][2]->setRightColor(FrontPlane[0][2]->FrontColor());

            FrontPlane[0][0]->setFrontColor(LeftPlane[0][0]->LeftColor());
            FrontPlane[0][1]->setFrontColor(LeftPlane[0][1]->LeftColor());
            FrontPlane[0][2]->setFrontColor(LeftPlane[0][2]->LeftColor());

            LeftPlane[0][0]->setLeftColor(BackPlane[0][0]->BackColor());
            LeftPlane[0][1]->setLeftColor(BackPlane[0][1]->BackColor());
            LeftPlane[0][2]->setLeftColor(BackPlane[0][2]->BackColor());

            BackPlane[0][0]->setBackColor(right_0_0);
            BackPlane[0][1]->setBackColor(right_0_1);
            BackPlane[0][2]->setBackColor(right_0_2);

            visualRotateMiniMachineGun(3, ROTATE_SPEED_STEP, 1);
        }
    }

    void RotateDownPlane(const char degree, bool isWriteToConsole = true) {
```

```
        RotatesCounter++;
        string right_2_0 = RightPlane[2][0]->RightColor(), right_2_1 =
RightPlane[2][1]->RightColor(), right_2_2 = RightPlane[2][2]->RightColor();

        if (degree == '+') {
            if (isWriteToConsole)
                cout << "D ";

            string down_0_0 = DownPlane[0][0]->DownColor(),
                   down_1_0 = DownPlane[1][0]->DownColor();

            DownPlane[0][0]->setDownColor(DownPlane[2][0]->DownColor());
            DownPlane[1][0]->setDownColor(DownPlane[2][1]->DownColor());
            DownPlane[2][0]->setDownColor(DownPlane[2][2]->DownColor());
            DownPlane[2][1]->setDownColor(DownPlane[1][2]->DownColor());
            DownPlane[2][2]->setDownColor(DownPlane[0][2]->DownColor());
            DownPlane[1][2]->setDownColor(DownPlane[0][1]->DownColor());
            DownPlane[0][2]->setDownColor(down_0_0);
            DownPlane[0][1]->setDownColor(down_1_0);

            RightPlane[2][0]->setRightColor(FrontPlane[2][0]->FrontColor());
            RightPlane[2][1]->setRightColor(FrontPlane[2][1]->FrontColor());
            RightPlane[2][2]->setRightColor(FrontPlane[2][2]->FrontColor());

            FrontPlane[2][0]->setFrontColor(LeftPlane[2][0]->LeftColor());
            FrontPlane[2][1]->setFrontColor(LeftPlane[2][1]->LeftColor());
            FrontPlane[2][2]->setFrontColor(LeftPlane[2][2]->LeftColor());

            LeftPlane[2][0]->setLeftColor(BackPlane[2][0]->BackColor());
            LeftPlane[2][1]->setLeftColor(BackPlane[2][1]->BackColor());
            LeftPlane[2][2]->setLeftColor(BackPlane[2][2]->BackColor());

            BackPlane[2][0]->setBackColor(right_2_0);
            BackPlane[2][1]->setBackColor(right_2_1);
            BackPlane[2][2]->setBackColor(right_2_2);

            visualRotateMiniMachineGun(2, ROTATE_SPEED_STEP, 1);

        } else if (degree == '-') {
            if (isWriteToConsole)
                cout << "D' ";

            string down_0_2 = DownPlane[0][2]->DownColor(), down_1_2 =
DownPlane[1][2]->DownColor();

            DownPlane[0][2]->setDownColor(DownPlane[2][2]->DownColor());
            DownPlane[1][2]->setDownColor(DownPlane[2][1]->DownColor());
            DownPlane[2][2]->setDownColor(DownPlane[2][0]->DownColor());
            DownPlane[2][1]->setDownColor(DownPlane[1][0]->DownColor());
            DownPlane[2][0]->setDownColor(DownPlane[0][0]->DownColor());
            DownPlane[1][0]->setDownColor(DownPlane[0][1]->DownColor());
            DownPlane[0][1]->setDownColor(down_1_2);
            DownPlane[0][0]->setDownColor(down_0_2);

            RightPlane[2][0]->setRightColor(BackPlane[2][0]->BackColor());
            RightPlane[2][1]->setRightColor(BackPlane[2][1]->BackColor());
            RightPlane[2][2]->setRightColor(BackPlane[2][2]->BackColor());

            BackPlane[2][0]->setBackColor(LeftPlane[2][0]->LeftColor());
            BackPlane[2][1]->setBackColor(LeftPlane[2][1]->LeftColor());
            BackPlane[2][2]->setBackColor(LeftPlane[2][2]->LeftColor());

            LeftPlane[2][0]->setLeftColor(FrontPlane[2][0]->FrontColor());
            LeftPlane[2][1]->setLeftColor(FrontPlane[2][1]->FrontColor());
```

```
            LeftPlane[2][2]->setLeftColor(FrontPlane[2][2]->FrontColor());

            FrontPlane[2][0]->setFrontColor(right_2_0);
            FrontPlane[2][1]->setFrontColor(right_2_1);
            FrontPlane[2][2]->setFrontColor(right_2_2);

            visualRotateMiniMachineGun(2, ROTATE_SPEED_STEP, -1);
        }
    }

    void RotateLeftPlane(const char degree, bool isWriteToConsole = true) {
        RotatesCounter++;
        string front_0_0 = FrontPlane[0][0]->FrontColor(), front_1_0 =
FrontPlane[1][0]->FrontColor(), front_2_0 = FrontPlane[2][0]->FrontColor();

        if (degree == '+') {
            if (isWriteToConsole)
                cout << "L ";

            string left_0_0 = LeftPlane[0][0]->LeftColor(),
                   left_1_0 = LeftPlane[1][0]->LeftColor();

            LeftPlane[0][0]->setLeftColor(LeftPlane[2][0]->LeftColor());
            LeftPlane[1][0]->setLeftColor(LeftPlane[2][1]->LeftColor());
            LeftPlane[2][0]->setLeftColor(LeftPlane[2][2]->LeftColor());
            LeftPlane[2][1]->setLeftColor(LeftPlane[1][2]->LeftColor());
            LeftPlane[2][2]->setLeftColor(LeftPlane[0][2]->LeftColor());
            LeftPlane[1][2]->setLeftColor(LeftPlane[0][1]->LeftColor());
            LeftPlane[0][2]->setLeftColor(left_0_0);
            LeftPlane[0][1]->setLeftColor(left_1_0);

            FrontPlane[0][0]->setFrontColor(UpPlane[0][0]->UpColor());
            FrontPlane[1][0]->setFrontColor(UpPlane[1][0]->UpColor());
            FrontPlane[2][0]->setFrontColor(UpPlane[2][0]->UpColor());

            UpPlane[0][0]->setUpColor(BackPlane[2][2]->BackColor());
            UpPlane[1][0]->setUpColor(BackPlane[1][2]->BackColor());
            UpPlane[2][0]->setUpColor(BackPlane[0][2]->BackColor());

            BackPlane[0][2]->setBackColor(DownPlane[2][0]->DownColor());
            BackPlane[1][2]->setBackColor(DownPlane[1][0]->DownColor());
            BackPlane[2][2]->setBackColor(DownPlane[0][0]->DownColor());

            DownPlane[0][0]->setDownColor(front_0_0);
            DownPlane[1][0]->setDownColor(front_1_0);
            DownPlane[2][0]->setDownColor(front_2_0);

            visualRotateMiniMachineGun(4, ROTATE_SPEED_STEP, 1);

        } else if (degree == '-') {
            if (isWriteToConsole)
                cout << "L' ";

            string left_0_0 = LeftPlane[0][0]->LeftColor(), left_0_1 =
LeftPlane[0][1]->LeftColor();

            LeftPlane[0][0]->setLeftColor(LeftPlane[0][2]->LeftColor());
            LeftPlane[0][1]->setLeftColor(LeftPlane[1][2]->LeftColor());
            LeftPlane[0][2]->setLeftColor(LeftPlane[2][2]->LeftColor());
            LeftPlane[1][2]->setLeftColor(LeftPlane[2][1]->LeftColor());
            LeftPlane[2][2]->setLeftColor(LeftPlane[2][0]->LeftColor());
            LeftPlane[2][1]->setLeftColor(LeftPlane[1][0]->LeftColor());
            LeftPlane[2][0]->setLeftColor(left_0_0);
            LeftPlane[1][0]->setLeftColor(left_0_1);
```

```
            FrontPlane[0][0]->setFrontColor(DownPlane[0][0]->DownColor());
            FrontPlane[1][0]->setFrontColor(DownPlane[1][0]->DownColor());
            FrontPlane[2][0]->setFrontColor(DownPlane[2][0]->DownColor());

            DownPlane[0][0]->setDownColor(BackPlane[2][2]->BackColor());
            DownPlane[1][0]->setDownColor(BackPlane[1][2]->BackColor());
            DownPlane[2][0]->setDownColor(BackPlane[0][2]->BackColor());

            BackPlane[0][2]->setBackColor(UpPlane[2][0]->UpColor());
            BackPlane[1][2]->setBackColor(UpPlane[1][0]->UpColor());
            BackPlane[2][2]->setBackColor(UpPlane[0][0]->UpColor());

            UpPlane[0][0]->setUpColor(front_0_0);
            UpPlane[1][0]->setUpColor(front_1_0);
            UpPlane[2][0]->setUpColor(front_2_0);

            visualRotateMiniMachineGun(4, ROTATE_SPEED_STEP, -1);
        }
    }

    void RotateRightPlane(const char degree, bool isWriteToConsole = true) {
        RotatesCounter++;
        string front_0_2 = FrontPlane[0][2]->FrontColor(), front_1_2 =
FrontPlane[1][2]->FrontColor(), front_2_2 = FrontPlane[2][2]->FrontColor();

        if (degree == '+') {
            if (isWriteToConsole)
                cout << "R ";

            string right_0_0 = RightPlane[0][0]->RightColor(), right_1_0 =
RightPlane[1][0]->RightColor();

            RightPlane[0][0]->setRightColor(RightPlane[2][0]->RightColor());
            RightPlane[1][0]->setRightColor(RightPlane[2][1]->RightColor());
            RightPlane[2][0]->setRightColor(RightPlane[2][2]->RightColor());
            RightPlane[2][1]->setRightColor(RightPlane[1][2]->RightColor());
            RightPlane[2][2]->setRightColor(RightPlane[0][2]->RightColor());
            RightPlane[1][2]->setRightColor(RightPlane[0][1]->RightColor());
            RightPlane[0][2]->setRightColor(right_0_0);
            RightPlane[0][1]->setRightColor(right_1_0);

            FrontPlane[0][2]->setFrontColor(DownPlane[0][2]->DownColor());
            FrontPlane[1][2]->setFrontColor(DownPlane[1][2]->DownColor());
            FrontPlane[2][2]->setFrontColor(DownPlane[2][2]->DownColor());

            DownPlane[0][2]->setDownColor(BackPlane[2][0]->BackColor());
            DownPlane[1][2]->setDownColor(BackPlane[1][0]->BackColor());
            DownPlane[2][2]->setDownColor(BackPlane[0][0]->BackColor());

            BackPlane[0][0]->setBackColor(UpPlane[2][2]->UpColor());
            BackPlane[1][0]->setBackColor(UpPlane[1][2]->UpColor());
            BackPlane[2][0]->setBackColor(UpPlane[0][2]->UpColor());

            UpPlane[0][2]->setUpColor(front_0_2);
            UpPlane[1][2]->setUpColor(front_1_2);
            UpPlane[2][2]->setUpColor(front_2_2);

            visualRotateMiniMachineGun(5, ROTATE_SPEED_STEP, -1);

        } else if (degree == '-') {
            if (isWriteToConsole)
                cout << "R' ";
```

```
            string right_0_1 = RightPlane[0][1]->RightColor(), right_0_2 =
RightPlane[0][2]->RightColor();

            RightPlane[0][1]->setRightColor(RightPlane[1][2]->RightColor());
            RightPlane[0][2]->setRightColor(RightPlane[2][2]->RightColor());
            RightPlane[1][2]->setRightColor(RightPlane[2][1]->RightColor());
            RightPlane[2][2]->setRightColor(RightPlane[2][0]->RightColor());
            RightPlane[2][1]->setRightColor(RightPlane[1][0]->RightColor());
            RightPlane[2][0]->setRightColor(RightPlane[0][0]->RightColor());
            RightPlane[1][0]->setRightColor(right_0_1);
            RightPlane[0][0]->setRightColor(right_0_2);

            FrontPlane[0][2]->setFrontColor(UpPlane[0][2]->UpColor());
            FrontPlane[1][2]->setFrontColor(UpPlane[1][2]->UpColor());
            FrontPlane[2][2]->setFrontColor(UpPlane[2][2]->UpColor());

            UpPlane[0][2]->setUpColor(BackPlane[2][0]->BackColor());
            UpPlane[1][2]->setUpColor(BackPlane[1][0]->BackColor());
            UpPlane[2][2]->setUpColor(BackPlane[0][0]->BackColor());

            BackPlane[0][0]->setBackColor(DownPlane[2][2]->DownColor());
            BackPlane[1][0]->setBackColor(DownPlane[1][2]->DownColor());
            BackPlane[2][0]->setBackColor(DownPlane[0][2]->DownColor());

            DownPlane[0][2]->setDownColor(front_0_2);
            DownPlane[1][2]->setDownColor(front_1_2);
            DownPlane[2][2]->setDownColor(front_2_2);

            visualRotateMiniMachineGun(5, ROTATE_SPEED_STEP, 1);
        }
    }

    void RotateFrontPlane(const char degree, bool isWriteToConsole = true) {
        RotatesCounter++;
        string up_2_0 = UpPlane[2][0]->UpColor(), up_2_1 = UpPlane[2][1]-
>UpColor(), up_2_2 = UpPlane[2][2]->UpColor();

        if (degree == '+') {
            if (isWriteToConsole)
                cout << "F ";

            string front_0_0 = FrontPlane[0][0]->FrontColor(), front_1_0 =
FrontPlane[1][0]->FrontColor();

            FrontPlane[0][0]->setFrontColor(FrontPlane[2][0]->FrontColor());
            FrontPlane[1][0]->setFrontColor(FrontPlane[2][1]->FrontColor());
            FrontPlane[2][0]->setFrontColor(FrontPlane[2][2]->FrontColor());
            FrontPlane[2][1]->setFrontColor(FrontPlane[1][2]->FrontColor());
            FrontPlane[2][2]->setFrontColor(FrontPlane[0][2]->FrontColor());
            FrontPlane[1][2]->setFrontColor(FrontPlane[0][1]->FrontColor());
            FrontPlane[0][2]->setFrontColor(front_0_0);
            FrontPlane[0][1]->setFrontColor(front_1_0);

            UpPlane[2][0]->setUpColor(LeftPlane[2][2]->LeftColor());
            UpPlane[2][1]->setUpColor(LeftPlane[1][2]->LeftColor());
            UpPlane[2][2]->setUpColor(LeftPlane[0][2]->LeftColor());

            LeftPlane[0][2]->setLeftColor(DownPlane[0][0]->DownColor());
            LeftPlane[1][2]->setLeftColor(DownPlane[0][1]->DownColor());
            LeftPlane[2][2]->setLeftColor(DownPlane[0][2]->DownColor());

            DownPlane[0][0]->setDownColor(RightPlane[2][0]->RightColor());
            DownPlane[0][1]->setDownColor(RightPlane[1][0]->RightColor());
            DownPlane[0][2]->setDownColor(RightPlane[0][0]->RightColor());
```

```
            RightPlane[0][0]->setRightColor(up_2_0);
            RightPlane[1][0]->setRightColor(up_2_1);
            RightPlane[2][0]->setRightColor(up_2_2);

            visualRotateMiniMachineGun(1, ROTATE_SPEED_STEP, -1);

        } else if (degree == '-') {
            if (isWriteToConsole)
                cout << "F' ";

            string front_0_0 = FrontPlane[0][0]->FrontColor(), front_0_1 =
FrontPlane[0][1]->FrontColor();

            FrontPlane[0][0]->setFrontColor(FrontPlane[0][2]->FrontColor());
            FrontPlane[0][1]->setFrontColor(FrontPlane[1][2]->FrontColor());
            FrontPlane[0][2]->setFrontColor(FrontPlane[2][2]->FrontColor());
            FrontPlane[1][2]->setFrontColor(FrontPlane[2][1]->FrontColor());
            FrontPlane[2][2]->setFrontColor(FrontPlane[2][0]->FrontColor());
            FrontPlane[2][1]->setFrontColor(FrontPlane[1][0]->FrontColor());
            FrontPlane[2][0]->setFrontColor(front_0_0);
            FrontPlane[1][0]->setFrontColor(front_0_1);

            UpPlane[2][0]->setUpColor(RightPlane[0][0]->RightColor());
            UpPlane[2][1]->setUpColor(RightPlane[1][0]->RightColor());
            UpPlane[2][2]->setUpColor(RightPlane[2][0]->RightColor());

            RightPlane[0][0]->setRightColor(DownPlane[0][2]->DownColor());
            RightPlane[1][0]->setRightColor(DownPlane[0][1]->DownColor());
            RightPlane[2][0]->setRightColor(DownPlane[0][0]->DownColor());

            DownPlane[0][0]->setDownColor(LeftPlane[0][2]->LeftColor());
            DownPlane[0][1]->setDownColor(LeftPlane[1][2]->LeftColor());
            DownPlane[0][2]->setDownColor(LeftPlane[2][2]->LeftColor());

            LeftPlane[0][2]->setLeftColor(up_2_2);
            LeftPlane[1][2]->setLeftColor(up_2_1);
            LeftPlane[2][2]->setLeftColor(up_2_0);

            visualRotateMiniMachineGun(1, ROTATE_SPEED_STEP, 1);
        }
    }

    void RotateBackPlane(const char degree, bool isWriteToConsole = true) {
        RotatesCounter++;
        string up_0_0 = UpPlane[0][0]->UpColor(), up_0_1 = UpPlane[0][1]-
>UpColor(), up_0_2 = UpPlane[0][2]->UpColor();

        if (degree == '+') {
            if (isWriteToConsole)
                cout << "B ";

            string back_0_0 = BackPlane[0][0]->BackColor(), back_1_0 =
BackPlane[1][0]->BackColor();

            BackPlane[0][0]->setBackColor(BackPlane[2][0]->BackColor());
            BackPlane[1][0]->setBackColor(BackPlane[2][1]->BackColor());
            BackPlane[2][0]->setBackColor(BackPlane[2][2]->BackColor());
            BackPlane[2][1]->setBackColor(BackPlane[1][2]->BackColor());
            BackPlane[2][2]->setBackColor(BackPlane[0][2]->BackColor());
            BackPlane[1][2]->setBackColor(BackPlane[0][1]->BackColor());
            BackPlane[0][2]->setBackColor(back_0_0);
            BackPlane[0][1]->setBackColor(back_1_0);
```

```
            UpPlane[0][0]->setUpColor(RightPlane[0][2]->RightColor());
            UpPlane[0][1]->setUpColor(RightPlane[1][2]->RightColor());
            UpPlane[0][2]->setUpColor(RightPlane[2][2]->RightColor());

            RightPlane[0][2]->setRightColor(DownPlane[2][2]->DownColor());
            RightPlane[1][2]->setRightColor(DownPlane[2][1]->DownColor());
            RightPlane[2][2]->setRightColor(DownPlane[2][0]->DownColor());

            DownPlane[2][0]->setDownColor(LeftPlane[0][0]->LeftColor());
            DownPlane[2][1]->setDownColor(LeftPlane[1][0]->LeftColor());
            DownPlane[2][2]->setDownColor(LeftPlane[2][0]->LeftColor());

            LeftPlane[2][0]->setLeftColor(up_0_0);
            LeftPlane[1][0]->setLeftColor(up_0_1);
            LeftPlane[0][0]->setLeftColor(up_0_2);

            visualRotateMiniMachineGun(0, ROTATE_SPEED_STEP, 1);

        } else if (degree == '-') {
            if (isWriteToConsole)
                cout << "B' ";

            string back_0_0 = BackPlane[0][0]->BackColor(), back_0_1 =
BackPlane[0][1]->BackColor();

            BackPlane[0][0]->setBackColor(BackPlane[0][2]->BackColor());
            BackPlane[0][1]->setBackColor(BackPlane[1][2]->BackColor());
            BackPlane[0][2]->setBackColor(BackPlane[2][2]->BackColor());
            BackPlane[1][2]->setBackColor(BackPlane[2][1]->BackColor());
            BackPlane[2][2]->setBackColor(BackPlane[2][0]->BackColor());
            BackPlane[2][1]->setBackColor(BackPlane[1][0]->BackColor());
            BackPlane[2][0]->setBackColor(back_0_0);
            BackPlane[1][0]->setBackColor(back_0_1);

            UpPlane[0][0]->setUpColor(LeftPlane[2][0]->LeftColor());
            UpPlane[0][1]->setUpColor(LeftPlane[1][0]->LeftColor());
            UpPlane[0][2]->setUpColor(LeftPlane[0][0]->LeftColor());

            LeftPlane[0][0]->setLeftColor(DownPlane[2][0]->DownColor());
            LeftPlane[1][0]->setLeftColor(DownPlane[2][1]->DownColor());
            LeftPlane[2][0]->setLeftColor(DownPlane[2][2]->DownColor());

            DownPlane[2][0]->setDownColor(RightPlane[2][2]->RightColor());
            DownPlane[2][1]->setDownColor(RightPlane[1][2]->RightColor());
            DownPlane[2][2]->setDownColor(RightPlane[0][2]->RightColor());

            RightPlane[0][2]->setRightColor(up_0_0);
            RightPlane[1][2]->setRightColor(up_0_1);
            RightPlane[2][2]->setRightColor(up_0_2);

            visualRotateMiniMachineGun(0, ROTATE_SPEED_STEP, -1);
        }
    }

    void RotateMachineGun(string commandsSeq, bool isWriteToConsole = true) {
        unsigned long size = commandsSeq.size();
        for (int i = 0; i < size; i += 2) {
            if (commandsSeq[i] == 'U') {
                if (commandsSeq[i + 1] == '+' || commandsSeq[i + 1] == ' ')
                    RotateUpPlane('+', isWriteToConsole);
                else
                    RotateUpPlane('-', isWriteToConsole);
                continue;
            }
```

```
            if (commandsSeq[i] == 'D') {
                if (commandsSeq[i + 1] == '+' || commandsSeq[i + 1] == ' ')
                    RotateDownPlane('+', isWriteToConsole);
                else
                    RotateDownPlane('-', isWriteToConsole);
                continue;
            }

            if (commandsSeq[i] == 'L') {
                if (commandsSeq[i + 1] == '+' || commandsSeq[i + 1] == ' ')
                    RotateLeftPlane('+', isWriteToConsole);
                else
                    RotateLeftPlane('-', isWriteToConsole);
                continue;
            }

            if (commandsSeq[i] == 'R') {
                if (commandsSeq[i + 1] == '+' || commandsSeq[i + 1] == ' ')
                    RotateRightPlane('+', isWriteToConsole);
                else
                    RotateRightPlane('-', isWriteToConsole);
                continue;
            }

            if (commandsSeq[i] == 'F') {
                if (commandsSeq[i + 1] == '+' || commandsSeq[i + 1] == ' ')
                    RotateFrontPlane('+', isWriteToConsole);
                else
                    RotateFrontPlane('-', isWriteToConsole);
                continue;
            }

            if (commandsSeq[i] == 'B') {
                if (commandsSeq[i + 1] == '+' || commandsSeq[i + 1] == ' ')
                    RotateBackPlane('+', isWriteToConsole);
                else
                    RotateBackPlane('-', isWriteToConsole);
                continue;
            }
        }
    }

    // Алгоритмы
    void RightAlgorithm(bool isWriteToConsole = true) {
        this->RotateRightPlane('+', isWriteToConsole);
        this->RotateUpPlane('+', isWriteToConsole);
        this->RotateRightPlane('-', isWriteToConsole);
        this->RotateUpPlane('-', isWriteToConsole);
    }

    void LeftAlgorithm(bool isWriteToConsole = true) {
        this->RotateLeftPlane('+', isWriteToConsole);
        this->RotateUpPlane('+', isWriteToConsole);
        this->RotateLeftPlane('-', isWriteToConsole);
        this->RotateUpPlane('-', isWriteToConsole);
    }

    void UpAlgorithm(bool isWriteToConsole = true) {
        this->RotateUpPlane('+', isWriteToConsole);
        this->RotateLeftPlane('+', isWriteToConsole);
        this->RotateUpPlane('-', isWriteToConsole);
        this->RotateLeftPlane('-', isWriteToConsole);
    }
```

```
    void DownAlgorithm(bool isWriteToConsole = true) {
        this->RotateDownPlane('-', isWriteToConsole);
        this->RotateRightPlane('-', isWriteToConsole);
        this->RotateDownPlane('+', isWriteToConsole);
        this->RotateLeftPlane('+', isWriteToConsole);
    }


    /*** ВИЗУАЛИЗАЦИЯ ***/
    // Количество плоскостей для поворота
    int currentPlane;

    // Проекция угла поворота на ось и расстояния в окне
    const GLfloat CUBE_SIZE = 12;
    int xRot = 25, yRot = -45;
    GLfloat translateZ = -35;

    bool isVisualCubeUsed[3][3][3];
    VisualCube tmp[3][3], visualColors[3][3][3];
    // Угол поворота для каждой плоскости
    int rotateAngle[6];
    // Размер кубика в окне
    GLfloat visualSize;

    void display() {
        glPushMatrix();
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(1, 0, 0);
        glTranslatef(0, 0, translateZ);
        glRotatef(xRot, 1, 0, 0);
        glRotatef(yRot, 0, 1, 0);
        glTranslatef(CUBE_SIZE / -2, CUBE_SIZE / -2, CUBE_SIZE / -2);
        draw();
        glPopMatrix();
        glutSwapBuffers();
    }

    void visualRotate90(int planeId, int degree) {
        // если знак = -1, то повернём 3 раза (против часовой стрелки)
        if (degree == -1)
            degree = 3;

        while (degree--) {
            // BLUE
            if (planeId == 0) {
                // копирование повернутой на 90 градусов верхней/нижней
плоскости в temp, а затем присвоение плоскостей temp
                // так же нужно повернуть каждую деталь каждой плоскости
                for (int i = 0, k = 0; i < 3; ++i)
                    for (int j = 0; j < 3; ++j)
                        tmp[j][2 - i] = visualColors[i][j][k];

                for (int i = 0, k = 0; i < 3; ++i)
                    for (int j = 0; j < 3; ++j) {
                        tmp[i][j].rotateORb();
                        visualColors[i][j][k] = tmp[i][j];
                    }
            }
                // GREEN
            else if (planeId == 1) {
                for (int i = 0, k = 2; i < 3; ++i)
                    for (int j = 0; j < 3; ++j)
                        tmp[j][2 - i] = visualColors[i][j][k];
```

```
            for (int i = 0, k = 2; i < 3; ++i)
                for (int j = 0; j < 3; ++j) {
                    tmp[i][j].rotateRR();
                    visualColors[i][j][k] = tmp[i][j];
                }
        }
            // WHITE
        else if (planeId == 2) {
            for (int i = 0, j = 0; i < 3; ++i)
                for (int k = 0; k < 3; ++k)
                    tmp[k][2 - i] = visualColors[i][j][k];

            for (int i = 0, j = 0; i < 3; ++i)
                for (int k = 0; k < 3; ++k) {
                    tmp[i][k].rotateGD();
                    visualColors[i][j][k] = tmp[i][k];
                }
        }
            // YELLOW
        else if (planeId == 3) {
            for (int i = 0, j = 2; i < 3; ++i)
                for (int k = 0; k < 3; ++k)
                    tmp[k][2 - i] = visualColors[i][j][k];

            for (int i = 0, j = 2; i < 3; ++i)
                for (int k = 0; k < 3; ++k) {
                    tmp[i][k].rotateGU();
                    visualColors[i][j][k] = tmp[i][k];
                }
        }
            // GL
            // RED
        else if (planeId == 4) {
            for (int j = 0, i = 0; j < 3; ++j)
                for (int k = 0; k < 3; ++k)
                    tmp[k][2 - j] = visualColors[i][j][k];

            for (int j = 0, i = 0; j < 3; ++j)
                for (int k = 0; k < 3; ++k) {
                    tmp[j][k].rotateGR();
                    visualColors[i][j][k] = tmp[j][k];
                }
        }
            // GR
            // ORANGE
        else if (planeId == 5) {
            for (int j = 0, i = 2; j < 3; ++j)
                for (int k = 0; k < 3; ++k)
                    tmp[k][2 - j] = visualColors[i][j][k];

            for (int j = 0, i = 2; j < 3; ++j)
                for (int k = 0; k < 3; ++k) {
                    tmp[j][k].rotateGR();
                    visualColors[i][j][k] = tmp[j][k];
                }
        }
    }
}

void timer(int degree, int) {
    if (currentPlane == -1)
        return;
```

```
        display();
        visualRotateMiniMachineGun(currentPlane, ROTATE_SPEED_STEP, degree);
    }

    void visualRotateMiniMachineGun(int idx, int angle, int degree) {
        if (currentPlane == -1 || currentPlane == idx) {
            if (degree == -1)
                rotateAngle[idx] -= angle;
            else if (degree == 1)
                rotateAngle[idx] += angle;

            if (rotateAngle[idx] % 90 != 0) {
                currentPlane = idx;
                timer(degree, 0);
            } else {
                // если угол стал кратен 90, повернитесь по вектору
                if ((rotateAngle[idx] < 0) ^ (currentPlane == 2 ||
currentPlane == 3))
                    visualRotate90(idx, 1);
                else
                    visualRotate90(idx, -1);

                rotateAngle[idx] = 0;
                currentPlane = -1;
            }
        }
    }

    void setVisualCube(GLfloat size, unsigned int *color) {
        memset(rotateAngle, 0, sizeof(rotateAngle));
        this->visualSize = size;
        currentPlane = -1;

        // Top
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j)
                visualColors[i][j][2].setColor(0, color[0]);

        // Bottom
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j)
                visualColors[i][j][0].setColor(1, color[1]);

        // Front
        for (int k = 0; k < 3; ++k)
            for (int j = 0; j < 3; ++j)
                visualColors[j][0][k].setColor(2, color[2]);

        // Back
        for (int k = 0; k < 3; ++k)
            for (int j = 0; j < 3; ++j)
                visualColors[j][2][k].setColor(3, color[3]);

        // Left
        for (int i = 0; i < 3; ++i)
            for (int k = 0; k < 3; ++k)
                visualColors[0][k][i].setColor(4, color[4]);

        // Right
        for (int i = 0; i < 3; ++i)
            for (int k = 0; k < 3; ++k)
                visualColors[2][k][i].setColor(5, color[5]);

        // Размеры маленьких деталек
```

```
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j)
                for (int k = 0; k < 3; ++k)
                    visualColors[i][j][k].visualSize = (size / 3) * 0.95;

    }

    // Отрисовка кубика в окне
    void draw() {
        const GLfloat K = 0.25;
        // K - черный куб, размер которого равен K * visualSize
        glPushMatrix();
        glColor3f(0, 0, 0);
        glTranslatef(((1 - K) / 2) * visualSize + K * visualSize / 2, ((1 -
K) / 2) * visualSize + K * visualSize / 2, ((1 - K) / 2) * visualSize + K *
visualSize / 2);
        glutSolidCube(visualSize * K);
        glPopMatrix();

        memset(isVisualCubeUsed, true, sizeof(isVisualCubeUsed));
        if (currentPlane != -1) {
            glPushMatrix();
            if (currentPlane == 0 || currentPlane == 1) {
                int k = (currentPlane & 1) * 2;
                for (int i = 0; i < 3; ++i)
                    for (int j = 0; j < 3; ++j)
                        isVisualCubeUsed[i][j][k] = false;

                glTranslated(visualSize / 2, visualSize / 2, 0);   // перевод
в центр
                glRotatef(rotateAngle[currentPlane], 0, 0, 1);   // поворот
                glTranslated(-visualSize / 2, -visualSize / 2, 0); // перевод
в исходное положение

                // отрисовка
                for (int i = 0; i < 3; ++i)
                    for (int j = 0; j < 3; ++j)
                        visualColors[i][j][k].draw(visualSize / 3 * i,
visualSize / 3 * j, visualSize / 3 * k);

            } else if (currentPlane == 2 || currentPlane == 3) {
                int j = (currentPlane & 1) * 2;
                for (int i = 0; i < 3; ++i)
                    for (int k = 0; k < 3; ++k)
                        isVisualCubeUsed[i][j][k] = false;

                glTranslated(visualSize / 2, 0, visualSize / 2);
                glRotatef(rotateAngle[currentPlane], 0, 1, 0);
                glTranslated(-visualSize / 2, 0, -visualSize / 2);
                for (int i = 0; i < 3; ++i)
                    for (int k = 0; k < 3; ++k)
                        visualColors[i][j][k].draw(visualSize / 3 * i,
visualSize / 3 * j, visualSize / 3 * k);

            } else if (currentPlane == 4 || currentPlane == 5) {
                int i = (currentPlane & 1) * 2;
                for (int j = 0; j < 3; ++j)
                    for (int k = 0; k < 3; ++k)
                        isVisualCubeUsed[i][j][k] = false;

                glTranslated(0, visualSize / 2, visualSize / 2);
                glRotatef(rotateAngle[currentPlane], 1, 0, 0);
                glTranslated(0, -visualSize / 2, -visualSize / 2);
                for (int j = 0; j < 3; ++j)
```

```
                    for (int k = 0; k < 3; ++k)
                        visualColors[i][j][k].draw(visualSize / 3 * i,
visualSize / 3 * j, visualSize / 3 * k);
            }
            glPopMatrix();
        }

        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3; ++j)
                for (int k = 0; k < 3; ++k)
                    if (isVisualCubeUsed[i][j][k])
                        visualColors[i][j][k].draw(visualSize / 3 * i,
visualSize / 3 * j, visualSize / 3 * k);
    }
};

static RubikCube Cube;

#endif //RUBIKCUBE_RUBIKCUBE_H
```