МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Нижегородский государственный технический университет
им. Р.Е. Алексеева» (НГТУ)
Кафедра: «Цифровая экономика»

Дисциплина: «Численные методы»

Руководство программиста

Разработал:
студент 3-го курса группы 21-САИ
Краличев Игорь Евгеньевич
Подпись:_____

Нижний Новгород, 2024

Содержание

Аннотация

Приводится руководство программиста программного обеспечения для моделирования потока идеального газа на ограниченной сетке с препятствием.

Программный продукт должен быть разработан для любого пользователя и предназначен для моделирования потока идеального газа на замкнутой сетке с находящимся внутри препятствием в форме круга и прямоугольника. Модель должна учитывать физические и математические аспекты движения газа, его взаимодействия с препятствием, и быть реализованной с использованием численных методов.

В руководстве программиста рассматриваются назначение, характеристики, условия, необходимые для выполнения программы.

Назначение и условия применения программы

Назначение программы

Программа предназначена для моделирования потока идеального газа на замкнутой сетке с находящимся внутри препятствием в форме круга и прямоугольника. Модель должна учитывать физические и математические аспекты движения газа, его взаимодействия с препятствием, и быть реализованной с использованием численных методов.

Функции, выполняемые программой

1) Ввод входных данных пользователем.
2) Визуализация моделирования потока газа на экран устройства.
3) Сохранение результатов вычислений в текстовый файл.
4) Выбор режима отображения потока.
5) Возможность изменения параметров моделирования.

Требования к программному обеспечению

Для корректного выполнения программы на компьютере пользователя должна быть установлена 64-разрядная операционная системы Windows не ниже Windows 7, также Java SE версии не ниже 16.0.1.

Требования к персоналу (программисту)

Программист должен обладать практическими навыками работы с графическим пользовательским интерфейсов операционной системы, иметь квалификацию «Программист».

Характеристики программы

Режим работы программы

Программа выполняется в интерактивном режиме работы. У пользователя есть возможность взаимодействия с системой.

Средства контроля правильности выполнения программы

Контроль правильности выполнения программы осуществляется вручную с помощью выгрузки вычисленного решения в текстовый файл.

Инструменты разработки

При решении работы были использованы следующие инструменты:

1. Apache NetBeans (Product Version**:** Apache NetBeans IDE 12.4, Java**:** 16.0.1; OpenJDK 64-Bit Server VM 16.0.1+9-24, разрядность: x64);
2. Visual Studio IDE (версия: 17.7.3, разрядность: x64).

Apache NetBeans — это бесплатная интегрированная среда разработки с открытым исходным кодом для создания программного обеспечения в операционных системах Windows, macOS, Linux и Solaris. Она позволяет создавать веб-приложения, корпоративное, десктопное и мобильное программное обеспечение. Изначально разработчики создавали NetBeans для Java, одна затем список поддерживаемых языков программирования был расширен, включив в себя Python, PHP, HTML5, JavaScript, C, C++, «Ада», Ruby (не поддерживается в последних версиях) и некоторых другие.

Visual Studio IDE— это интегрированная среда разработки (IDE) от компании Microsoft, предназначенная для разработки программного обеспечения. В Visual Studio предоставляются различные инструменты и функциональные возможности, упрощающие процесс разработки, отладки и тестирования приложений.

Visual Studio поддерживает множество языков программирования, включая C++, C#, Visual Basic, F#, JavaScript и другие. Для разработки на C++ в Visual Studio используется компилятор Microsoft C++, который обеспечивает мощные возможности компиляции и оптимизации кода.

Visual Studio обеспечивает разработчиков на C++ всеми необходимыми инструментами для создания высококачественного программного обеспечения. Отличительной чертой Visual Studio является его обширная функциональность и поддержка различных платформ и технологий, что делает его одним из популярных выборов для разработки на C++.

Обращение к программе

Запуск программы

Для запуска программы откройте исполняемый файл SimulationGas.exe. Исходный код запуска программы представлен в приложении 1.

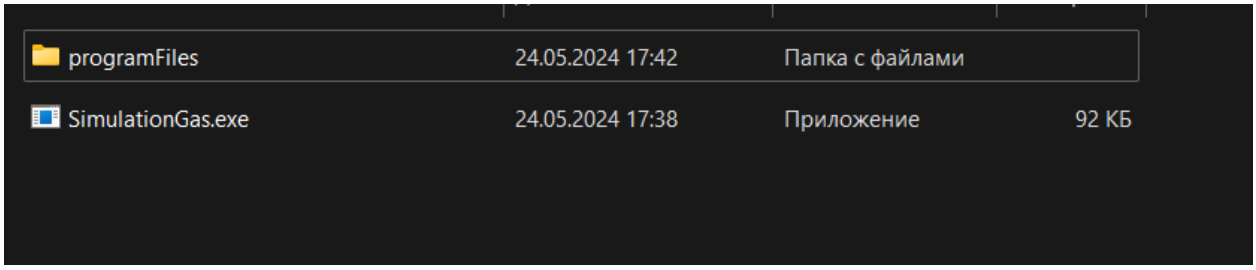

Рисунок 1. Папка с исполняемым файлом

Выбор входных данных

После запуска исполняемого файла появится окно с выбором входных данных для моделирования:

- Препятствие (круг или прямоугольник);
- Размер препятствия (маленький или большой);
- Время моделирования (ползунком выбрать время от 20 до 500);
- Формат отрисовки (линии тока или векторы скоростей);
- Выгрузка данных в файл;

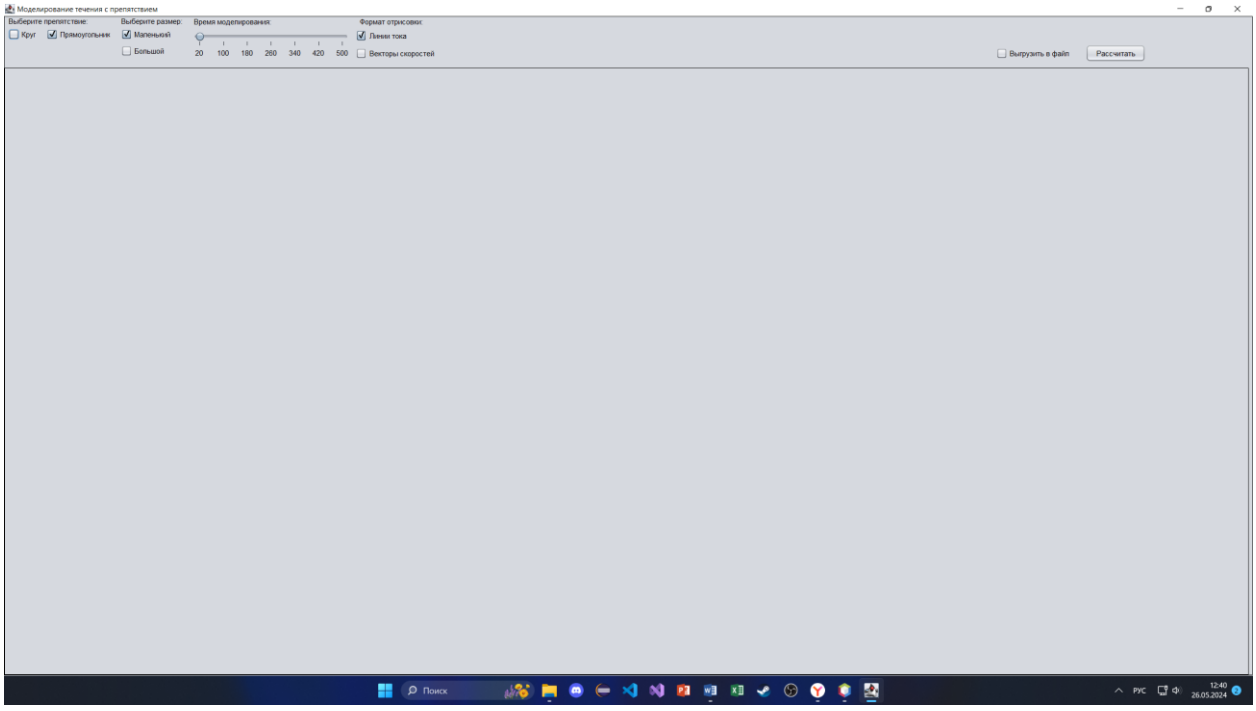

Рисунок 2. Окно выбора входных данных

Моделирование потока

После выбора начальных условий нажать на кнопку «Рассчитать», которая после нажатия станет неактивной и ожидать окончания моделирования.

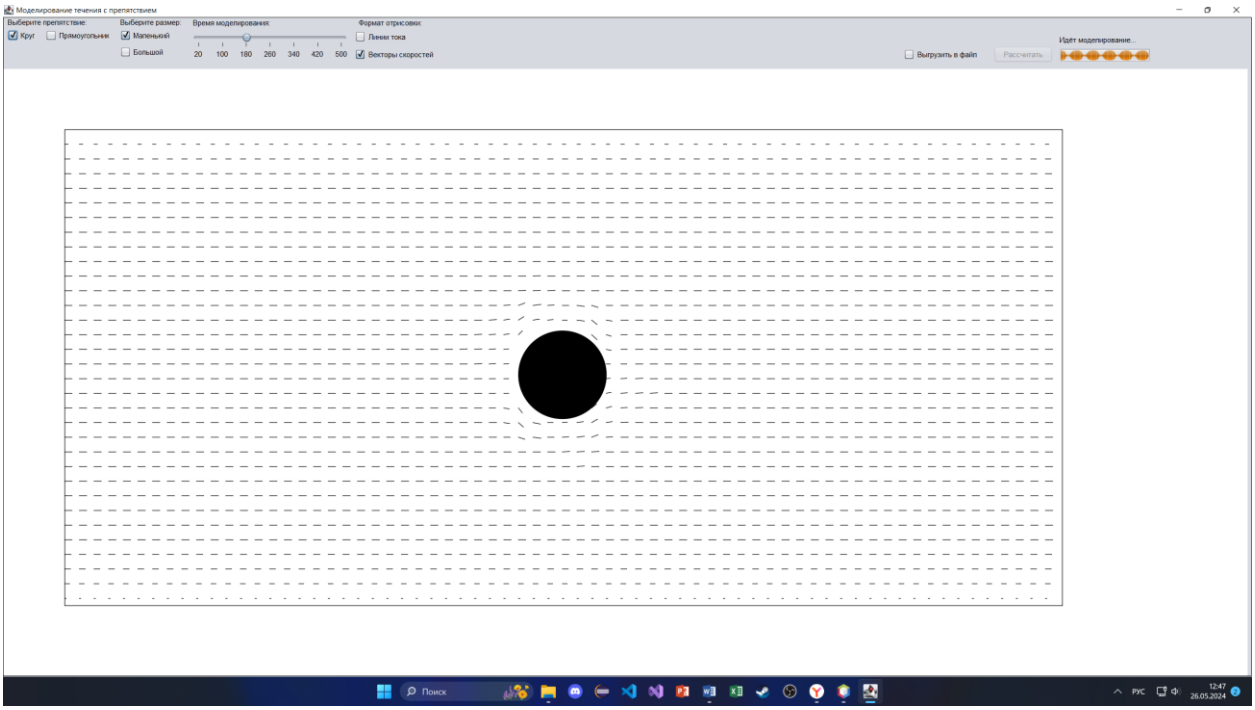

Рисунок 3. Процесс моделирования потока газа

По окончанию моделирования, когда пропадёт строка состояния «Идёт моделирование» и кнопка «Рассчитать» снова станет активной, можно снова выбрать входные данные или закрыть программу.

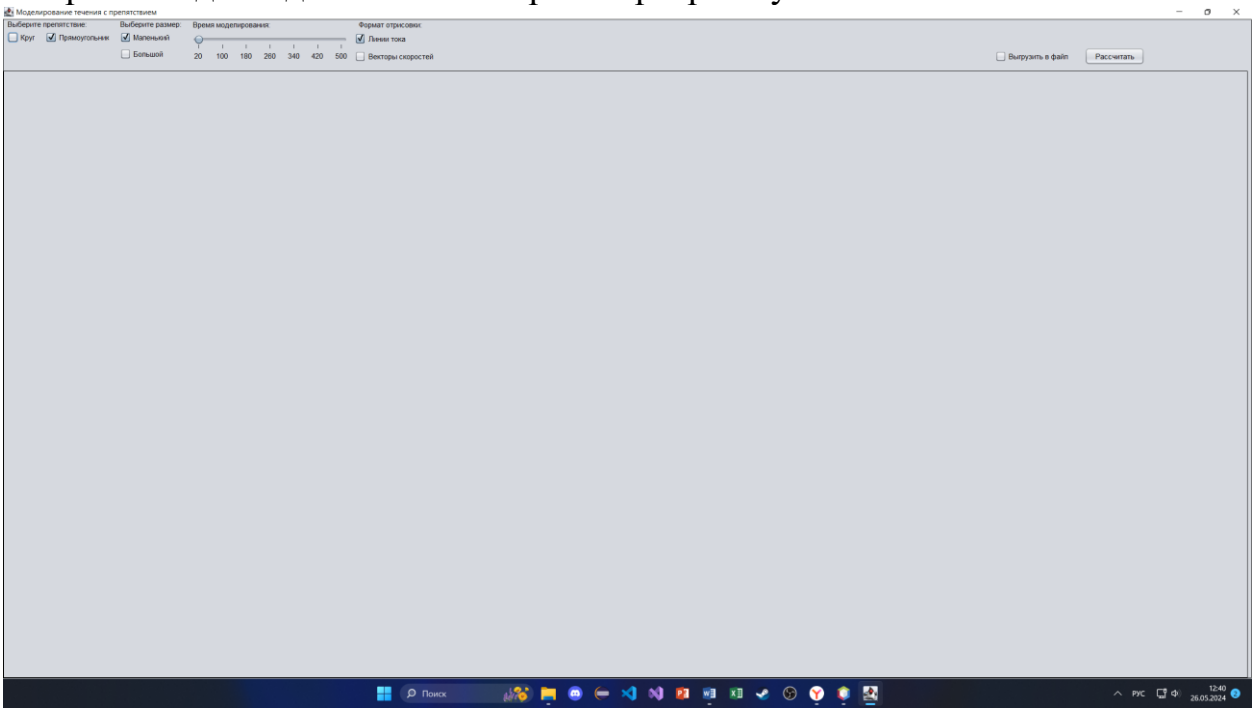

Рисунок 4. Окно выбора входных данных

В случае выбора выгрузки в файл, они создаются на диске с программой и имеют названия MyP.txt, MyU.txt, MyV.txt.

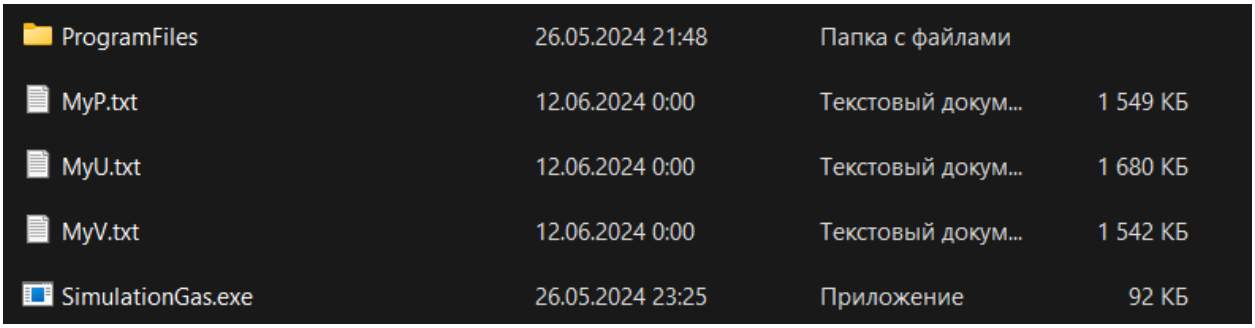

Рисунок 5. Файлы с численным результатом моделирования

Исходный код программы моделирования потока газа представлен в приложении 2.

Входные и выходные данные

Организация используемой входной информации

Входная информация организована в виде выбора пользователем соответствующих флаговых кнопок и ползунков.

Организация используемой выходной информации

Выходная информация реализована в виде отображения потока газа, который обтекает препятствие внутри ограниченной сетки. Также есть возможность выгрузить численные значения моделирования в текстовый файл.

Приложение 1

Код программы на C++

```
#include <windows.h>
#include <iostream>
#include <string>

int main() {
    // Получаем путь к текущему исполняемому файлу
    char currentPath[MAX_PATH];
    GetModuleFileNameA(NULL, currentPath, MAX_PATH);

    // Извлекаем путь к папке, где находится исполняемый файл
    std::string exePath = currentPath;
    size_t lastSlashPos = exePath.rfind('\\');
    if (lastSlashPos == std::string::npos) {
        std::cerr << "Не удалось извлечь путь к исполняемому файлу." << std::endl;
        return 1;
    }
    std::string folderPath = exePath.substr(0, lastSlashPos);

    // Формируем полный путь к .jar файлу
    std::string jarPath = folderPath + "\\ProgramFiles\\dist\\ProgramFiles.jar";

    // Открываем .jar файл
    int result = (int)ShellExecuteA(NULL, "open", jarPath.c_str(), NULL, NULL,
SW_SHOWNORMAL);

    if (result <= 32) {
        std::cerr << "Ошибка при открытии .jar файла. Код ошибки: " << result << std::endl;
        return 1;
    }

    return 0;
}
```

Приложение 2

Код программы на Java

```
package simulationGas;

//package org.jzy3d.demos.scatter;


import com.sun.tools.javac.Main;

import java.awt.*;

import java.awt.geom.Line2D;

import javax.swing.*;

import java.awt.geom.Ellipse2D;
```

```
import java.awt.geom.Rectangle2D;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Random;

/**
 *
 * @author IGOR
 */
public class SimulationForm extends javax.swing.JFrame {

    /**
     * Creates new form NewJFrame
     */
    public SimulationForm() {
        initComponents();
        jProgressBar1.setVisible(false); //Скрываю прогресс бар
        jLabel3.setVisible(false); // Убираю надпись о моделировании
        //setSize(800, 600);
    }
```

```
int obsctacleType, drawChoice, obstacleSize, choiceFile;


public class MyClass {


    private JPanel jPanel2;
    private JProgressBar jProgressBar;


    // функция округления числа
    private static double round(double value, int places) {
        if (places < 0) {
            throw new IllegalArgumentException();
        }


        BigDecimal bd = new BigDecimal(Double.toString(value));
        bd = bd.setScale(places, RoundingMode.HALF_UP);
        return bd.doubleValue();
    }


    // Диаметр круга для отрисовки
    public static int findDiameterCircle(int left_border, int multiplier, int x, int y, double[][]
obsctacle, int obstacleSize) {
        int diameter = 0;
        int iCircleStart = x, iCircleEnd = 0;
        for (int j = y - 2; j > 0; j = j - 1) {
            for (int i = 0; i < x; i = i + 1) {
                if (insideObstacle(obsctacle, i, j) == true) {


                    if (i < iCircleStart && obsctacle[i][j] == 0) {
                        iCircleStart = i;
                        //System.out.println(iCircleStart);
                    }
                    if (i > iCircleEnd && obsctacle[i][j] == 0) {
```

```
                    iCircleEnd = i;
                    //System.out.println(iCircleEnd);
                }
            }
        }
    }
    if (obstacleSize == 1) {
        diameter = ((left_border + (iCircleEnd) * multiplier) - (left_border + (iCircleStart) * multiplier)) + 25;
    }
    if (obstacleSize == 2) {
        diameter = ((left_border + (iCircleEnd) * multiplier) - (left_border + (iCircleStart) * multiplier)) + 25;
    }
    return diameter;
}

// Отрисовка векторов с препятствием
public static void drawVectors(double[][][] u, double[][][] v, int x, int y, int t, double[][] obsctacle, int obsctacleType, int obstacleSize, JPanel jPanel2) {
    int tmin = 15;
    int multiplier = 12;
    int left_border = 100;
    int up_border = 100;
    double arrow_mult = multiplier / 1.1;
    double direction = 0.5;
    double currenty;
    int starty;
    int prevI = 0;
    for (int n = tmin; n < t; n = n + 10) {

        Graphics2D g2d = (Graphics2D) jPanel2.getGraphics();
        g2d.setColor(Color.WHITE);
```

```
            // Очистка панели
            g2d.fillRect(0, 0, jPanel2.getWidth(), jPanel2.getHeight());
            // Включение антиалиасинга для более гладкой отрисовки
            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
            g2d.setColor(Color.BLACK);
            // Отрисока коробки
            g2d.drawRect(left_border, up_border, x * multiplier + 4, (y - 1) * multiplier);
            g2d.setColor(Color.BLACK);

            for (int j = y - 3; j > 0; j = j - 2) {
               for (int i = 0; i < x; i = i + 2) {
                  g2d.setColor(Color.BLACK);
                  if (insideObstacle(obsctacle, i, j) == true) {
                     if (i - prevI > 1 || prevI - i < -1) {
                        g2d.setColor(Color.WHITE);
                     }
                  }
                  int newX = (int) (left_border + i * multiplier + (u[i][j][n] * arrow_mult));
                  int newY = (int) (up_border + (y - 1 - j) * multiplier - (v[i][j][n] * arrow_mult));
                  g2d.drawLine(left_border + i * multiplier, up_border + (y - 1 - j) * multiplier,
newX, newY);
                  // Рисуем кружок в конце вектора
                  //g2d.fillOval(newX - 3, newY - 3, 3, 3);
                  prevI = i;
               }
            }

            // отрисовка препятствия
            int checkExit = 0;
            int widthCircle, heightCircle;
```

```
            for (int j = y - 2; j > 0; j = j - 1) {
                for (int i = 0; i < x; i = i + 1) {
                    if (insideObstacle(obsctacle, i, j) == true) {
                        if (obsctacleType == 1) { // окружность
                            widthCircle = findDiameterCircle(left_border, multiplier, x, y, obsctacle,
obstacleSize);
                            heightCircle = findDiameterCircle(left_border, multiplier, x, y, obsctacle,
obstacleSize);
                            //System.out.println("w=" + widthCircle);
                            //System.out.println("h=" + heightCircle);
                            // Отрисовка круга
                            if (obsctacle[i][j] == 0 && checkExit != 1) {
                                //System.out.println("n=" + n + ",i=" + i + ",j=" + j);
                                if (obstacleSize == 1) {
                                    g2d.fillOval(left_border + (i - 2) * multiplier, up_border + (y - 2 - j) *
multiplier + 5, widthCircle, heightCircle);
                                }
                                if (obstacleSize == 2) {
                                    g2d.fillOval(left_border + (i - 5) * multiplier, up_border + (y - 2 - j) *
multiplier + 5, widthCircle, heightCircle);
                                }
                                //System.out.println((left_border + (iCircleStart) * multiplier) -
(left_border + (iCircleEnd) * multiplier));
                                // Закрашиваем область внутри круга
                                //g2d.fillRect(left_border + i * multiplier, up_border + (y - 2 - j) *
multiplier, multiplier, multiplier);
                                checkExit = 1;
                            }
                        } else if (obsctacleType == 2 && checkExit != 1) { // прямоугольник
                            // Отрисовка прямоугольника
                            if (obstacleSize == 1) {
                                g2d.fillRect(left_border + (i) * multiplier, up_border + (y - 2 - j) *
multiplier, multiplier, multiplier + 1);
                            }
```

```
                    if (obstacleSize == 2) {
                        g2d.fillRect(left_border + (i) * multiplier, up_border + (y - 2 - j) *
multiplier + 5, multiplier, multiplier);
                    }
                }
            }
        }
        // Пауза отрисовка
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

// Отрисовка линий тока с препятствием
public static void drawForces(double[][][] u, double[][][] v, int x, int y, int t, double[][]
obsctacle, int obsctacleType, int obstacleSize, JPanel jPanel2) {
    int tmin = 15; // минимальное время для отрисовки
    int multiplier = 12; // множитель длины
    int left_border = 100;  // левая граница с отступом
    int up_border = 100;    // верхняя граница с отступом
    double arrow_mult = multiplier / 1.1;   //множитель длины для линий
    double direction = 0.5; // число сравнения направления
    double currenty;    // нахождение кисти в настоящий момент
    int starty;
    for (int n = tmin; n < t; n = n + 10) {

        Graphics2D g2d = (Graphics2D) jPanel2.getGraphics();
        g2d.setColor(Color.WHITE);
```

```
            // Очистка панели
            g2d.fillRect(0, 0, jPanel2.getWidth(), jPanel2.getHeight());
            // Включение антиалиасинга для более гладкой отрисовки
            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
            g2d.setColor(Color.BLACK);
            // Отрисока коробки
            g2d.drawRect(left_border, up_border, x * multiplier + 1, (y - 1) * multiplier);

            // Линии тока
            for (int j = y - 3; j > 1; j = j - 2) {
               //g2d.setColor(Color.BLACK);
               starty = j;
               currenty = (y - 1 - starty) * multiplier;
               int prevEndX = (int) (left_border + 0 * multiplier + u[0][starty][n] * arrow_mult);
               int prevEndY = (int) (up_border + currenty);
               int prevI = 0;
               //g2d.drawLine(left_border, up_border + (y - 1 - j) * multiplier, left_border,
up_border + (y - 1 - j) * multiplier);
               for (int i = 0; i < x - 1; i = i + 1) {
                  if (insideObstacle(obsctacle, i, starty) == false) {
                     g2d.setColor(Color.BLACK);
                     //System.out.println("i=" + i);

                     if (i - prevI > 1 || prevI - i < -1) {
                        g2d.setColor(Color.WHITE);
                     }

                     if (u[i][starty][n] == 0) {
                        g2d.setColor(Color.WHITE);
                     }
```

```
                if (v[i][starty][n] > direction) {
                    int nextX = (int) (left_border + (i + 1) * multiplier + u[i + 1][starty + 1][n] *
arrow_mult);
                    int nextY = (int) (up_border + currenty - v[i + 1][starty + 1][n] *
arrow_mult);
                    if (!insideObstacle(obsctacle, i + 1, starty + 1)) {
                        g2d.drawLine(prevEndX, prevEndY, nextX, nextY);
                        prevEndX = nextX;
                        prevEndY = nextY;
                        starty++;
                    }
                } else if (v[i][starty][n] < -direction) {
                    int nextX = (int) (left_border + (i + 1) * multiplier + u[i + 1][starty - 1][n] *
arrow_mult);
                    int nextY = (int) (up_border + currenty - v[i + 1][starty - 1][n] *
arrow_mult);
                    if (!insideObstacle(obsctacle, i + 1, starty + 1)) {
                        g2d.drawLine(prevEndX, prevEndY, nextX, nextY);
                        prevEndX = nextX;
                        prevEndY = nextY;
                        starty--;
                    }
                } else if (v[i][starty][n] > -direction && v[i][starty][n] < direction) {
                    int nextX = (int) (left_border + (i + 1) * multiplier + u[i + 1][starty][n] *
arrow_mult);
                    int nextY = (int) (up_border + currenty - v[i + 1][starty][n] * arrow_mult);
                    if (!insideObstacle(obsctacle, i + 1, starty + 1)) {
                        g2d.drawLine(prevEndX, prevEndY, nextX, nextY);
                        prevEndX = nextX;
                        prevEndY = nextY;
                    }
                }
                prevI = i;
```

```
                }
            }
            //Проверка направления по вектору
            /*
        for (int i = 0; i < x; i = i + 1) {
            if (insideObstacle(obsctacle, i, j) == false) {
                g2d.drawLine((int)(left_border + i * multiplier), (int)(up_border + (y - 1 - j) *
multiplier), (int)(left_border + i * multiplier + u[i][j][n] * arrow_mult), (int)(up_border + (y - 1 -
j) * multiplier - v[i][j][n] * arrow_mult));
            }
        }*/
            }
            // отрисовка препятствия
            int checkExit = 0;
            int widthCircle, heightCircle;
            for (int j = y - 2; j > 0; j = j - 1) {
                for (int i = 0; i < x; i = i + 1) {
                    if (insideObstacle(obsctacle, i, j) == true) {
                        if (obsctacleType == 1) { // окружность
                            widthCircle = findDiameterCircle(left_border, multiplier, x, y, obsctacle,
obstacleSize);
                            heightCircle = findDiameterCircle(left_border, multiplier, x, y, obsctacle,
obstacleSize);
                            //System.out.println("w=" + widthCircle);
                            //System.out.println("h=" + heightCircle);
                            // Отрисовка круга
                            if (obsctacle[i][j] == 0 && checkExit != 1) {
                                //System.out.println("n=" + n + ",i=" + i + ",j=" + j);
                                if (obstacleSize == 1) {
                                    g2d.fillOval(left_border + (i - 2) * multiplier, up_border + (y - 2 - j) *
multiplier + 5, widthCircle, heightCircle);
                                }
                                if (obstacleSize == 2) {
```

```
                            g2d.fillOval(left_border + (i - 5) * multiplier, up_border + (y - 2 - j) *
multiplier + 5, widthCircle, heightCircle);
                        }
                        //System.out.println((left_border + (iCircleStart) * multiplier) -
(left_border + (iCircleEnd) * multiplier));
                        // Закрашиваем область внутри круга
                        //g2d.fillRect(left_border + i * multiplier, up_border + (y - 2 - j) *
multiplier, multiplier, multiplier);
                        checkExit = 1;
                    }
                } else if (obsctacleType == 2 && checkExit != 1) { // прямоугольник
                    // Отрисовка прямоугольника
                    if (obstacleSize == 1) {
                        g2d.fillRect(left_border + (i) * multiplier, up_border + (y - 2 - j) *
multiplier + 5, multiplier + 15, multiplier + 1);
                    }
                    if (obstacleSize == 2) {
                        g2d.fillRect(left_border + (i) * multiplier, up_border + (y - 2 - j) *
multiplier + 5, multiplier + 15, multiplier + 1);
                    }
                }
            }
        }
        // Пауза отрисовка
        try {
            Thread.sleep(300);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
    public static void insertObstacle(int x, int y, int obstacleType, int obstacleSize, double[][]
obstacle) {
        // задание пока пустого поля
        for (int i = 0; i < x; i++) {
            for (int j = 0; j < y; j++) {
                obstacle[i][j] = 1;
            }
        }
        if (obstacleType == 1) { // окружность
            int multiplier = x / 10;
            int cx = (x - 1) / 2;
            int cy = (y - 1) / 2;
            int r = 6;
            //Маленький
            if (obstacleSize == 1) {
                r = 6;
            }
            // Большой
            if (obstacleSize == 2) {
                r = 10;
            }
            // Заполнение
            for (int i = 1; i < x - 1; i++) {
                for (int j = 1; j < y - 1; j++) {
                    if (((i - cx) * (i - cx) + (j - cy) * (j - cy)) < r * r) {
                        obstacle[i][j] = 0;
                    }
                }
            }
        }

        if (obstacleType == 2) { // прямоугольник
```

```
        int startX = ((x - 1) / 2) - y / 4;
        int endX = ((x - 1) / 2 + y / 5) + 1;
        int startY = y / 3;
        int endY = (2 * y) / 4 + 2;
        // Маленький
        if (obstacleSize == 1) {
            startY = y / 3;
            endY = (2 * y) / 4 + 2;
        }
        // Большой
        if (obstacleSize == 2) {
            startY = y / 4;
            endY = (3 * y) / 4 + 2;
        }
        // Заполнение
        for (int i = startX; i < endX; i++) {
            for (int j = startY; j < endY; j++) {
                obstacle[i][j] = 0;
            }
        }
    }

}

// функция проверки нахождения в препятствии
public static boolean insideObstacle(double[][] obstacle, int i, int j) {
    if (obstacle[i][j] == 0) {
        return true; // внутри
    }
    return false; // не внутри объекта
}
```

```
    // подстановка значений на границах сетки
    public static void boundary(double[][][] u, double[][][] v, double[][][] p, int x, int y, int t, double Umax, double[][][] rho) {
        double R = 8.314; // газовая постоянная
        double T = 300; // температура в Кельвинах
        double M = 0.023; // молярная масса
        for (int n = 0; n < t; n++) {
            for (int i = 0; i < x; i++) {
                // начальные скорости
                for (int j = 1; j < y - 1; j++) {
                    u[i][j][n] = Umax;
                    v[i][j][n] = 0;
                }
                //нижняя граница
                u[i][0][n] = 0;
                v[i][0][n] = 0;
                //верхняя граница
                u[i][y - 1][n] = 0;
                v[i][y - 1][n] = 0;
            }
        }
        // давление в спокойствии по Клайперону- Менделеева
        double Pdef = (rho[1][1][1] * T * R) / (M * 100000);
        for (int i = 0; i < x; i++) {
            for (int j = 0; j < y; j++) {
                p[i][j][0] = Pdef;
            }
        }
    }

    // запуск программы
```

```
    public static void start(int x, int y, int t, int shapeType, int obstacleSize, int drawChoice, int choiceFile, JPanel jPanel2) throws IOException {
        // создание массива скорости горизонтальной u
        final double Umax = 1; // 10 // максимальная скорость по U
        final double Pdef = 1.25; //10.25; // Кпа
        double nu = 0.1; // коэффициент вязкости
        double R = 8.314; // газовая постоянная
        double T = 300; // температура в Кельвинах
        double M = 0.023; // молярная масса
        double F = 1; // внешние силы
        double dx = 0.1; // шаг по x
        double dy = 0.05; // шаг по y
        double dt = 0.005; // шаг по t
        int numStart = 15; // время начала отсчёта, когда появляется препятствие
        double koef = 1.0; // значение коррекции

        // создание массива горизонтальной скорости u
        double[][][] u = new double[x][][];
        for (int i = 0; i < x; i++) {
            u[i] = new double[y][t];
        }
        // создание массива вертикальной скорости v
        double[][][] v = new double[x][][];
        for (int i = 0; i < x; i++) {
            v[i] = new double[y][t];
        }
        // создание массива плотности
        double[][][] rho = new double[x][][];
        for (int i = 0; i < x; i++) {
            rho[i] = new double[y][t];
        }
        // создание массива давлений
```

```
double[][][] p = new double[x][][];
for (int i = 0; i < x; i++) {
   p[i] = new double[y][t];
}
// массив для поля и фигуры
double[][] obstacle = new double[x][];
for (int i = 0; i < x; i++) {
   obstacle[i] = new double[y];
}
boundary(u, v, p, x, y, t, Umax, rho);
insertObstacle(x, y, shapeType, obstacleSize, obstacle);

for (int n = 0; n < t - 1; n++) {
   for (int i = 1; i < x - 1; i++) {
      for (int j = 1; j < y - 1; j++) {
         rho[i][j][n] = 1.4;
      }
   }
}
for (int n = 0; n < t - 1; n++) {
   // обнуляются узлы фигуры
   if (n > numStart) {
      for (int i = 1; i < x - 1; i++) {
         for (int j = 1; j < y - 1; j++) {
            // заполнение нулями внутри объекта
            if (insideObstacle(obstacle, i, j) == true) {
               u[i][j][n] = 0;
               v[i][j][n] = 0;
               p[i][j][n] = 0;
            }
            if (n == t - 2) {
```

```
                    u[i][j][n + 1] = 0;
                    v[i][j][n + 1] = 0;
                    p[i][j][n + 1] = 0;
                }
            }
        }
    }
    // считается давление по Новью-Стокса
    for (int i = 1; i < x - 1; i++) {
        for (int j = 1; j < y - 1; j++) {
            if (insideObstacle(obstacle, i, j) == true && n > numStart) {
                continue;




            }
            p[i][j][n + 1] = (((p[i + 1][j][n] + p[i - 1][j][n]) * dy * dy +
                    (p[i][j + 1][n] + p[i][j - 1][n]) * dx * dx) / (2. * (dx * dx + dy * dy))) -
                    ((rho[i][j][n] * dx * dx * dy * dy) / (2. * (dx * dx + dy * dy))) * ((1. / (dt *
1)) * (((u[i + 1][j][n] - u[i - 1][j][n]) / (2. * dx)) +
                    ((v[i][j + 1][n] - v[i][j - 1][n]) / (2. * dy))) -
                    (((u[i + 1][j][n] - u[i - 1][j][n]) / (2. * dx)) * ((u[i + 1][j][n] - u[i - 1][j][n]) /
(2. * dx))) -
                    2. * (((u[i][j + 1][n] - u[i][j - 1][n]) / (2. * dy)) * ((v[i + 1][j][n] - v[i -
1][j][n]) / (2. * dx))) -
                    (((v[i][j + 1][n] - v[i][j - 1][n]) / (2. * dy)) * ((v[i][j + 1][n] - v[i][j - 1][n]) /
(2. * dy))));
        }
    }
    // в крайних точках по x
    for (int j = 1; j < y - 1; j++) {
        p[x - 1][j][n + 1] = (((p[0][j][n] + p[x - 2][j][n]) * dy * dy +
                (p[x - 1][j + 1][n] + p[x - 1][j - 1][n]) * dx * dx) / (2. * (dx * dx + dy * dy))) -
```

```
                ((rho[1][1][1] * dx * dx * dy * dy) / (2. * (dx * dx + dy * dy))) * ((1. / dt) *
(((u[0][j][n] - u[x - 2][j][n]) / (2. * dx)) +
                ((v[x - 1][j + 1][n] - v[x - 1][j - 1][n]) / (2. * dy))) -
                (((u[0][j][n] - u[x - 2][j][n]) / (2. * dx)) * ((u[0][j][n] - u[x - 2][j][n]) / (2. *
dx))) -
                2. * (((u[x - 1][j + 1][n] - u[x - 1][j - 1][n]) / (2. * dy)) * ((v[0][j][n] - v[x -
2][j][n]) / (2. * dx))) -
                (((v[x - 1][j + 1][n] - v[x - 1][j - 1][n]) / (2. * dy)) * ((v[x - 1][j + 1][n] - v[x -
1][j - 1][n]) / (2. * dy))));
        p[0][j][n + 1] = (((p[1][j][n] + p[x - 1][j][n]) * dy * dy + (p[0][j + 1][n] + p[0][j -
1][n]) * dx * dx) / (2. * (dx * dx + dy * dy))) -
                ((rho[1][1][1] * dx * dx * dy * dy) / (2. * (dx * dx + dy * dy))) * ((1. / dt) *
(((u[1][j][n] - u[x - 1][j][n]) / (2. * dx)) +
                ((v[0][j + 1][n] - v[0][j - 1][n]) / (2. * dy))) -
                (((u[1][j][n] - u[x - 1][j][n]) / (2. * dx)) * ((u[1][j][n] - u[x - 1][j][n]) / (2. *
dx))) -
                2. * (((u[0][j + 1][n] - u[0][j - 1][n]) / (2. * dy)) * ((v[1][j][n] - v[x - 1][j][n]) /
(2. * dx))) -
                (((v[0][j + 1][n] - v[0][j - 1][n]) / (2. * dy)) * ((v[0][j + 1][n] - v[0][j - 1][n]) /
(2. * dy))));
        }
        //
        for (int i = 0; i < x; i++) {
            p[i][0][n + 1] = p[i][1][n + 1];
            p[i][y - 1][n + 1] = p[i][y - 2][n + 1];
        }
        // считаются скорости по Новью-Стоксу
        for (int i = 1; i < x - 1; i++) {
            for (int j = 1; j < y - 1; j++) {
                if (insideObstacle(obstacle, i, j) == true && n > numStart) {
                    continue;
                }
                u[i][j][n + 1] = u[i][j][n] + dt * (nu * (((u[i + 1][j][n] - 2 * u[i][j][n] + u[i -
1][j][n]) / (dx * dx)) +
                        ((u[i][j + 1][n] - 2 * u[i][j][n] + u[i][j - 1][n]) / (dy * dy))) -
```

```
                    u[i][j][n] * ((u[i][j][n] - u[i - 1][j][n]) / dx) - v[i][j][n] * ((u[i][j][n] - u[i][j -
1][n]) / dy) -
                    ((p[i + 1][j][n] - p[i - 1][j][n]) / (rho[i][j][n] * 2 * dx)) + F);
                v[i][j][n + 1] = v[i][j][n] + dt * (nu * (((v[i + 1][j][n] - 2 * v[i][j][n] + v[i -
1][j][n]) / (dx * dx)) +
                    ((v[i][j + 1][n] - 2 * v[i][j][n] + v[i][j - 1][n]) / (dy * dy))) -
                    u[i][j][n] * ((v[i][j][n] - v[i - 1][j][n]) / dx) - v[i][j][n] * ((v[i][j][n] - v[i][j -
1][n]) / dy) -
                    ((p[i][j + 1][n] - p[i][j - 1][n]) / (rho[i][j][n] * 2 * dy)));
                // верхние и нижние точки объектов
                if ((n > numStart + 1) && insideObstacle(obstacle, i - 1, j) == true) {
                    if (insideObstacle(obstacle, i, j + 1) == true) {
                        v[i][j][n + 1] += koef;
                    }
                    if (insideObstacle(obstacle, i, j - 1) == true) {
                        v[i][j][n + 1] -= koef;
                    }
                }
                // угловые точки
                if ((n > numStart + 1) && insideObstacle(obstacle, i + 1, j) == true) {
                    if (insideObstacle(obstacle, i, j + 1) == false) {
                        v[i][j][n + 1] += koef;
                    }
                    /*
                    if (insideObstacle(obstacle, i - 2, j + 2) == false) {
                        v[i][j][n + 1] += koef;
                    }
                     */
                    if (insideObstacle(obstacle, i, j - 1) == false) {
                        v[i][j][n + 1] -= koef;
                    }
                    /*
                    if (insideObstacle(obstacle, i - 2, j - 2) == false) {
```

```
                    v[i][j][n + 1] -= koef;
                }
                */
            }
        }
    }
    if (n > numStart) {
        koef = koef * ((n - 1.0) / n);
    }

    // в крайних точках по x
    for (int j = 1; j < y - 1; j++) {
        u[0][j][n + 1] = u[0][j][n] + dt * (nu * (((u[1][j][n] - 2 * u[0][j][n] + u[x - 1][j][n]) / (dx * dx)) +
                ((u[0][j + 1][n] - 2 * u[0][j][n] + u[0][j - 1][n]) / (dy * dy))) - u[0][j][n] * ((u[0][j][n] - u[x - 1][j][n]) / dx) -
                v[0][j][n] * ((u[0][j][n] - u[0][j - 1][n]) / dy) - ((p[1][j][n] - p[x - 1][j][n]) / (rho[1][1][1] * 2 * dx)) + F);
        v[0][j][n + 1] = v[0][j][n] + dt * (nu * (((v[1][j][n] - 2 * v[0][j][n] + v[x - 1][j][n]) / (dx * dx)) +
                ((v[0][j + 1][n] - 2 * v[0][j][n] + v[0][j - 1][n]) / (dy * dy))) - u[0][j][n] * ((v[0][j][n] - v[x - 1][j][n]) / dx) -
                v[0][j][n] * ((v[0][j][n] - v[0][j - 1][n]) / dy) - ((p[0][j + 1][n] - p[0][j - 1][n]) / (rho[1][j][1] * 2 * dy)));
        u[x - 1][j][n + 1] = u[x - 1][j][n] + dt * (nu * (((u[0][j][n] - 2 * u[x - 1][j][n] + u[x - 2][j][n]) / (dx * dx)) +
                ((u[x - 1][j + 1][n] - 2 * u[x - 1][j][n] + u[x - 1][j - 1][n]) / (dy * dy))) -
                u[x - 1][j][n] * ((u[x - 1][j][n] - u[x - 2][j][n]) / dx) - v[x - 1][j][n] * ((u[x - 1][j][n] - u[x - 1][j - 1][n]) / dy) -
                ((p[0][j][n] - p[x - 2][j][n]) / (rho[1][1][1] * 2 * dx)) + F);
        v[x - 1][j][n + 1] = v[x - 1][j][n] + dt * (nu * (((v[0][j][n] - 2 * v[x - 1][j][n] + v[x - 2][j][n]) / (dx * dx)) +
                ((v[x - 1][j + 1][n] - 2 * v[x - 1][j][n] + v[x - 1][j - 1][n]) / (dy * dy))) -
                u[x - 1][j][n] * ((v[x - 1][j][n] - v[x - 2][j][n]) / dx) - v[x - 1][j][n] * ((v[x - 1][j][n] - v[x - 1][j - 1][n]) / dy) -
```

```
                ((p[x - 1][j + 1][n] - p[x - 1][j - 1][n]) / (rho[1][1][1] * 2 * dy)));
            }

            // обнуляем точки на верхней и нижней границе
            for (int i = 0; i < x; i++) {
                u[i][y - 1][n + 1] = 0;
                u[i][0][n + 1] = 0;
                v[i][y - 1][n + 1] = 0;
                v[i][0][n + 1] = 0;
            }

        }
        // Линии тока
        if (drawChoice == 1) {
            drawForces(u, v, x, y, t, obstacle, shapeType, obstacleSize, jPanel2);
        } // Векторы скоростей
        else {
            drawVectors(u, v, x, y, t, obstacle, shapeType, obstacleSize, jPanel2);
        }
        // Запись в файл
        if (choiceFile == 1) {
            // Открытие файлов
            String jarPath =
Main.class.getProtectionDomain().getCodeSource().getLocation().getPath();
            Path path = Paths.get(jarPath);
            String folderPath = path.getParent().toString();
            String uFileName = "MyU.txt";
            String vFileName = "MyV.txt";
            String pFileName = "MyP.txt";

            try {
                File folder = new File(folderPath);
```

```
            if (!folder.exists()) {
                folder.mkdirs();
            }
            FileWriter uWriter = new FileWriter(folderPath + uFileName);
            FileWriter vWriter = new FileWriter(folderPath + vFileName);
            FileWriter pWriter = new FileWriter(folderPath + pFileName);

            // Запись массива u
            uWriter.write("Скорость u:");
            uWriter.write(System.lineSeparator());
            uWriter.write(System.lineSeparator());
            for (int n = numStart; n < t; n++) {
                uWriter.write("t=" + n);
                uWriter.write(System.lineSeparator());
                for (int j = y - 1; j > -1; j--) {
                    for (int i = 0; i < x; i++) {
                        uWriter.write(round(u[i][j][n], 3) + "  ");
                    }
                    uWriter.write(System.lineSeparator());
                    uWriter.write(System.lineSeparator());
                }
            }
            uWriter.close();
            System.out.println("Файл MyU.txt создан или перезаписан в папке 'данные' на диске D");

            // Запись массива v
            vWriter.write("Скорость v:");
            vWriter.write(System.lineSeparator());
            vWriter.write(System.lineSeparator());
            for (int n = numStart; n < t; n++) {
                vWriter.write("t=" + n);
```

```
                vWriter.write(System.lineSeparator());
                for (int j = y - 1; j > -1; j--) {
                    for (int i = 0; i < x; i++) {
                        vWriter.write(round(v[i][j][n], 3) + "  ");
                    }
                    vWriter.write(System.lineSeparator());
                    vWriter.write(System.lineSeparator());
                }
            }
            vWriter.close();
            System.out.println("Файл MyV.txt создан или перезаписан в папке 'данные' на диске D");

            // Запись массива p
            pWriter.write("Давление P (в Па):");
            pWriter.write(System.lineSeparator());
            pWriter.write(System.lineSeparator());
            for (int n = 0; n < t; n++) {
                pWriter.write("t=" + n);
                pWriter.write(System.lineSeparator());
                for (int j = y - 1; j > -1; j--) {
                    for (int i = numStart; i < x; i++) {
                        pWriter.write(round(p[i][j][n], 3) + "  ");
                    }
                    pWriter.write(System.lineSeparator());
                    pWriter.write(System.lineSeparator());
                }
            }
            pWriter.close();
            System.out.println("Файл MyP.txt создан или перезаписан в папке 'данные' на диске D");
        } catch (IOException e) {
```

```
                System.out.println("Ошибка при создании или сохранении файлов.");
                e.printStackTrace();
            }

        }
    }
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jPanel1 = new javax.swing.JPanel();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jSlider1 = new javax.swing.JSlider();
    jCheckBox1 = new javax.swing.JCheckBox();
    jCheckBox2 = new javax.swing.JCheckBox();
    jButton1 = new javax.swing.JButton();
    jPanel2 = new javax.swing.JPanel();
    jLabel3 = new javax.swing.JLabel();
    jProgressBar1 = new javax.swing.JProgressBar();
    jCheckBox4 = new javax.swing.JCheckBox();
    jCheckBox3 = new javax.swing.JCheckBox();
    jLabel4 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jCheckBox5 = new javax.swing.JCheckBox();
    jCheckBox6 = new javax.swing.JCheckBox();
    jCheckBox7 = new javax.swing.JCheckBox();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

```
        setTitle("Моделирование течения с препятствием");
        setBounds(new java.awt.Rectangle(0, 0, 0, 0));
        setExtendedState(6);
        setMinimumSize(new java.awt.Dimension(800, 600));
        setSize(new java.awt.Dimension(1173, 600));

        jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0,
0)));
        jPanel1.setPreferredSize(new java.awt.Dimension(1173, 618));

        jLabel1.setText("Выберите препятствие:");

        jLabel2.setText("Время моделирования:");

        jSlider1.setMajorTickSpacing(80);
        jSlider1.setMaximum(500);
        jSlider1.setMinimum(20);
        jSlider1.setPaintLabels(true);
        jSlider1.setPaintTicks(true);
        jSlider1.setToolTipText("");
        jSlider1.setValue(20);

        jCheckBox1.setText("Круг");
        jCheckBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBox1ActionPerformed(evt);
            }
        });

        jCheckBox2.setSelected(true);
        jCheckBox2.setText("Прямоугольник");
        jCheckBox2.addActionListener(new java.awt.event.ActionListener() {
```

```
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jCheckBox2ActionPerformed(evt);
        }
    });

    jButton1.setText("Рассчитать");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });

    jPanel2.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0,
0)));
    jPanel2.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
    jPanel2.setLayout(new java.awt.BorderLayout());

    jLabel3.setText("Идёт моделирование...");

    jCheckBox4.setText("Большой");
    jCheckBox4.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jCheckBox4ActionPerformed(evt);
        }
    });

    jCheckBox3.setSelected(true);
    jCheckBox3.setLabel("Маленький");
    jCheckBox3.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jCheckBox3ActionPerformed(evt);
        }
```

```
        });

        jLabel4.setText("Выберите размер:");

        jLabel5.setText("Формат отрисовки:");

        jCheckBox5.setSelected(true);
        jCheckBox5.setText("Линии тока");
        jCheckBox5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBox5ActionPerformed(evt);
            }
        });

        jCheckBox6.setText("Векторы скоростей");
        jCheckBox6.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBox6ActionPerformed(evt);
            }
        });

        jCheckBox7.setText("Выгрузить в файл");

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
```

```
                .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 153,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addComponent(jCheckBox1)
                    .addGap(18, 18, 18)
                    .addComponent(jCheckBox2)))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
                .addComponent(jLabel4)
                .addComponent(jCheckBox3, javax.swing.GroupLayout.PREFERRED_SIZE, 81,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jCheckBox4, javax.swing.GroupLayout.PREFERRED_SIZE, 81,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
                .addComponent(jLabel2)
                .addComponent(jSlider1, javax.swing.GroupLayout.PREFERRED_SIZE, 254,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(6, 6, 6)
                    .addComponent(jLabel5)
                    .addGap(446, 446, 446))
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
                        .addComponent(jCheckBox6, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jCheckBox5, javax.swing.GroupLayout.PREFERRED_SIZE,
113, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(27, 27, 27)
```

```
                .addComponent(jCheckBox7)
                .addGap(27, 27, 27)
                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 98,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(10, 10, 10)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
                    .addComponent(jProgressBar1,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel3))))
            .addGap(165, 165, 165))
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addContainerGap())
    );
    jPanel1Layout.setVerticalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel2)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jSlider1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel5)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                            .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELAT
ED)
                        .addComponent(jCheckBox5)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELAT
ED)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                            .addComponent(jCheckBox6)
                            .addComponent(jCheckBox7)
                            .addComponent(jButton1)))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELAT
ED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jLabel3)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELAT
ED)
                        .addComponent(jProgressBar1,
javax.swing.GroupLayout.PREFERRED_SIZE, 24,
javax.swing.GroupLayout.PREFERRED_SIZE)))))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATE
D)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.BASELINE)
                            .addComponent(jCheckBox1)
                            .addComponent(jCheckBox2)
                            .addComponent(jCheckBox3)))
                    .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
                .addComponent(jCheckBox4, javax.swing.GroupLayout.PREFERRED_SIZE, 24,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addGap(10, 10, 10)
            .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE, 541,
Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, 1295,
Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );

        pack();
    }// </editor-fold>

    private void jCheckBox6ActionPerformed(java.awt.event.ActionEvent evt) {
        if (jCheckBox6.isSelected() == true) {
            jCheckBox5.setSelected(false);
        }
    }

    private void jCheckBox5ActionPerformed(java.awt.event.ActionEvent evt) {
        if (jCheckBox5.isSelected() == true) {
            jCheckBox6.setSelected(false);
        }
```

```
}

private void jCheckBox3ActionPerformed(java.awt.event.ActionEvent evt) {
    if (jCheckBox3.isSelected() == true) {
        jCheckBox4.setSelected(false);
    }
}

private void jCheckBox4ActionPerformed(java.awt.event.ActionEvent evt) {
    if (jCheckBox4.isSelected() == true) {
        jCheckBox3.setSelected(false);
    }
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    //jPanel2.setBackground(Color.WHITE);
    //GraphicPanel GP = new GraphicPanel();
    //jPanel2.add(GP);
    //jPanel2.revalidate();
    //jPanel2.repaint();

    if (jCheckBox1.isSelected() == true) {
        obsctacleType = 1;
    }
    if (jCheckBox2.isSelected() == true) {
        obsctacleType = 2;
    }
    if (jCheckBox3.isSelected() == true) {
        obstacleSize = 1; //- маленький объект
    }
    if (jCheckBox4.isSelected() == true) {
```

```
            obstacleSize = 2; //- большой объект
        }
        if (jCheckBox5.isSelected() == true) {
            drawChoice = 1; // - линии тока
        }
        if (jCheckBox6.isSelected() == true) {
            drawChoice = 2; // - векторы скоростей
        }
        if (jCheckBox7.isSelected() == true) {
            choiceFile = 1; // - запись в файл
        }
        if (jCheckBox7.isSelected() != true) {
            choiceFile = 0; // - запись в файл не выбрана
        }
        int panelX = jPanel2.getWidth();
        //System.out.println(panelX);
        int panelY = jPanel2.getHeight();
        //System.out.println(panelY);
        int time = jSlider1.getValue();
        //System.out.println(time);
        //MyClass.start(panelX / 15, panelY / 15, time, obsctacleType, jPanel2, jProgressBar1);
        jProgressBar1.setVisible(true); //Включаю прогресс бар
        jProgressBar1.setIndeterminate(true); // Устанавливаю неопределенное состояние
        jLabel3.setVisible(true);//Паявляется надпись о моделировании
        jButton1.setEnabled(false);//Кнопка блокируется
        // Запускаем отрисовку в отдельном потоке, чтобы не блокировать EDT
        new Thread(() -> {
            try {
                MyClass.start(panelX / 15, panelY / 15, time, obsctacleType, obstacleSize,
drawChoice, choiceFile, jPanel2);
            } catch (IOException ex) {
                Logger.getLogger(SimulationForm.class.getName()).log(Level.SEVERE, null, ex);
```

```
        } finally {
            // После завершения отрисовки, сбрасываем неопределенное состояние в EDT
            SwingUtilities.invokeLater(() -> {
                jProgressBar1.setIndeterminate(false);
                // Можно установить определенное значение, если известно
                jProgressBar1.setVisible(false); // Убираем прогресс бар, потому что всё завершено
                jLabel3.setVisible(false); // Убираю надпись о моделировании
                jButton1.setEnabled(true);// Активирую кнопку
            });
        }
    }).start();
  }

  private void jCheckBox2ActionPerformed(java.awt.event.ActionEvent evt) {
    if (jCheckBox2.isSelected() == true) {
      jCheckBox1.setSelected(false);
    }
  }

  private void jCheckBox1ActionPerformed(java.awt.event.ActionEvent evt) {
    if (jCheckBox1.isSelected() == true) {
      jCheckBox2.setSelected(false);
    }
  }

  /**
   * @param args the command line arguments
   */
  public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
```

```
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(SimulationForm.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(SimulationForm.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(SimulationForm.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(SimulationForm.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new SimulationForm().setVisible(true);
```

```
        }
    });

}

// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JCheckBox jCheckBox1;
private javax.swing.JCheckBox jCheckBox2;
private javax.swing.JCheckBox jCheckBox3;
private javax.swing.JCheckBox jCheckBox4;
private javax.swing.JCheckBox jCheckBox5;
private javax.swing.JCheckBox jCheckBox6;
private javax.swing.JCheckBox jCheckBox7;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JProgressBar jProgressBar1;
private javax.swing.JSlider jSlider1;
// End of variables declaration
}
```