Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина:** Алгоритмы и структуры данных

**Тема:** Разработка GUI приложения для алгоритма нахождения кратчайшего пути в лабиринте из точки Start (S) в точку Finish (F)

Выполнила

студентка гр. №3530901/80003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Рубша А. И.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глухих М.И.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт-Петербург   
2019

Оглавление

[Техническое задание 3](#_Toc26876923)

[Метод решения 4](#_Toc26876924)

Листинг программа…………………………………………………………….....6

[**Main.java** 6](#_Toc26876926)

[**Controller.java** 7](#_Toc26876927)

[**Logic.java** 22](#_Toc26876928)

[**error1.fxml** 27](#_Toc26876929)

[**about.fxml** 28](#_Toc26876930)

**labirint.fxml**………………………………………………………………………………………………….....29

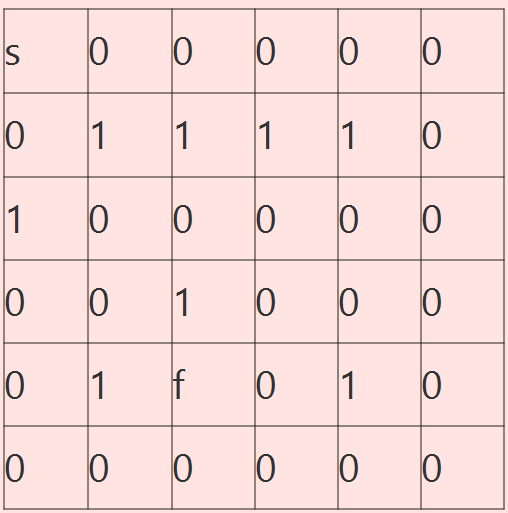
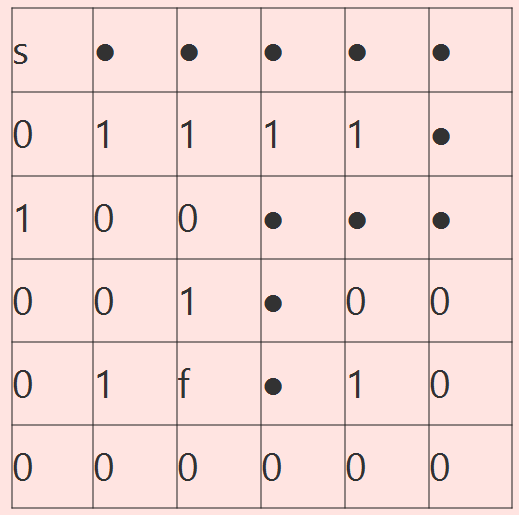
**noWays.fxml**…………………………………………………………………………………………………….31

[Тестирование 32](#_Toc26876931)

[Скриншоты работы программы 34](#_Toc26876932)

# Техническое задание

Лабиринт задан матрицей N x M со значениями 0 или 1. Цифра 0 означает, что дверь открыта и проход возможен, в то время как 1 означает, что двери закрыты и проход через эту клетку невозможен. Можно делать шаги только вверх, вниз, вправо или влево. По диагонали ходить нельзя. Требуется найти самый короткий путь из точки S в точку F.



Ссылка на репозиторий: https://github.com/Valkiriya228/labirint

# **Метод решения**

В проекте использована концепция MVC (Model-View-Controller) для структурирования кода и отделения методов логики от визуализации.

Класс Main - осуществляет запуск приложения.

Класс Logic - осуществляет логику программы. В нем присутствует только метод logicOfGame(), представляющий собой волновой алгоритм поиска всевозможных решений и кратчайшего по итогу.

labirint.fxml - графическое представление главного окна, меню.

Класс Controller - осуществляет обработку событий (открытие разделов Help -> About, File -> Open, File -> Exit, то есть открытие файла, его чтение, открытие окна описывающего работу данного приложения, реализация выхода из приложения).

**Порядок работы программы:**

При запуске программы в меню главного окна выбирается входной файл (File -> Open).

С помощью регулярных выражений проверяется первая строчка (задание размеров матрицы) на правильность ввода. Также идет проверка следующих строк на правильность введенных в нее чисел (проверяется индивидуальность точек S и F, а также наличие в строках цифр только вида 0 или 1, проверяется вариант, когда все цифры в заданной матрице равны единице)

При неправильном вводе данных в файл, несовпадении числа строк/столбцов, заданных нами в файле с количеством цифр, введенных нами в файл, выбрасывается окно – ошибка.

Если же ошибок нет, программа запоминает значения строк и столбцов (строка = stroka, столбец = stolbets), а также запоминает все строки нашей матрицы в ArrayList<String> allelements. Затем программа преобразовывает список allelements, разбивая отдельно по элементам, в ArrayList<String> all. (то есть all – список всех элементов исходной матрицы).

Далее идет отрисовка таблицы (gridPane), в ячейках которой будут храниться элементы, заданные пользователем изначально.

После запускается метод logicOfGame() класса Logic, реализующий нахождение наикратчайшего пути из точки S в точку F.

Данный метод принимает на своем входе лист всех изначальных элементов матрицы (то есть ArrayList<String> all), координаты точек старта и финиша (то есть int startX, startY, finishX, finishY).

Нахождение наикратчайшего пути представляет собой волновой алгоритм, то есть алгоритм, который распространяет во все возможные стороны волны, то бишь шаги. Программа создает 3 двумерных массива (Map[][], cMap[][] и Map1[][]).

Map[][] – двумерный массив, в который в начале программы заносятся все элементы, поданные пользователем изначально, то есть данный массив содержит 0, 1, s/f.

cMap[][] – двумерный массив, который сначала заполняется значениями «-2» и «-1», то есть если дверь закрыта в данной точке, программа ставит значение -2 (блокируется проход), и если индикатор еще не ступал на данную точку, ставится -1.

Затем с помощью цикла while мы осуществляем распространение волны на каждую соседнюю клетку при наличии прохода, то есть если ее значение -1, мы присваиваем данной ячейке значение, равное значению step. Step изначально равен 0, и с каждым проходом цикла будет увеличиваться на 1. Когда решение будет найдено, то есть наш алгоритм дойдет до точки f, программа выходит из цикла while.

Map1[][] – конечный двумерный массив. Одновременно работая с массивом cMap[][], заполняем наш итоговый массив Map[][]. То есть если индикатор не ступал на данную клетку (ее значение равно -1) или же в данной клетке заблокирован проход (ее значение равно -2), то массив Map[][] заполняется 0 (если -1) или 1(если -2). Затем если значение в данной ячейке больше -1, то есть индикатор ступал в данную клетку и присвоил ей определенное значение (значение шага), то массив Map[][] присваивает данной клетке значение «\*(значение шага в данной клетке)\*».

Далее мы работаем одновременно с массивом Map[][] и Map1[][]. Программа идет обратно, от точки f к точке s, то есть прокладывает один из путей, и данный путь отражается в массиве Map[][]

Как итог, мы имеем массив Map[][], в котором присутствуют 0, 1, старт и финиш и также наикратчайший путь, который проложен символами “●”

Результатом метода logicOfGame() является ArrayList<String> array, состоящий из всех конечных элементов, в том числе символов, означающих наикратчайший путь (“●”).

Если наикратчайший путь существует, то список, содержащий порядковые номера всех элементов матрицы, через которые проходил путь, форматируется и выводится в таблицу, которую мы снова отрисовываем в дальнейшем(gridPane), если же путей нет, выводится сообщение об этом.

**Листинг программы.**

**Main.java**

package sample;  
  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
import java.awt.\*;  
  
  
  
public class Main extends Application {  
  
 public Main() throws AWTException {  
 }  
  
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 Parent root = FXMLLoader.*load*(getClass().getResource("/labirint.fxml"));  
 primaryStage.setTitle("LABIRINT");  
 Scene scene = (new Scene(root, 1187, 853));  
 primaryStage.setScene(scene);  
  
 primaryStage.setResizable(true);  
 primaryStage.show();  
 }  
  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }

# **Controller.java**

# package sample;

# import javafx.application.Platform;

# import javafx.event.ActionEvent;

# import javafx.fxml.FXML;

# import javafx.fxml.FXMLLoader;

# import javafx.scene.Parent;

# import javafx.scene.Scene;

# import javafx.scene.control.Button;

# import javafx.scene.control.Label;

# import javafx.scene.control.MenuItem;

# import javafx.scene.layout.\*;

# import javafx.scene.text.Font;

# import javafx.stage.FileChooser;

# import javafx.stage.Stage;

# import java.io.BufferedReader;

# import java.io.File;

# import java.io.FileReader;

# import java.io.IOException;

# import java.util.ArrayList;

# import java.util.Arrays;

# import java.util.Objects;

# import java.util.ResourceBundle;

# public class Controller {

# @FXML

# private ResourceBundle resources;

# @FXML

# public Button start;

# @FXML

# public MenuItem Open;

# @FXML

# public MenuItem Exit;

# @FXML

# public GridPane gridPane = null;

# private int stroka;

# private int stolbets;

# private ArrayList<String> allelements = new ArrayList<>(); //список, содержащий все строки в матрице

# private int startX;

# private int startY;

# private int finishX;

# private int finishY;

# @FXML

# public MenuItem About;

# @FXML

# private AnchorPane AnchorPane;

# @FXML

# public Label label;

# @FXML

# public void initialize() throws IOException {

# Exit.setOnAction(event -> Platform.exit()); // Закрытие программы из меню

# About.setOnAction(event -> { //Открытие окошка правил работы программы

# Parent root1;

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("about.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Rules of the game");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# });

# }

# private ArrayList<String> lines = new ArrayList<>();

# private ArrayList<String> all = new ArrayList<>();

# private ArrayList<String> finish = new ArrayList<>();

# public void OpenAction(ActionEvent actionEvent) throws Exception {

# lines.clear(); //очищаем все листы от старых данных для заполнения новыми

# finish.clear();

# all.clear();

# allelements.clear();

# FileChooser fc = new FileChooser(); // диалоговое окно для выбора исходного файла

# fc.setTitle("Select file");

# FileChooser.ExtensionFilter extensionFilter = new FileChooser.ExtensionFilter("txt files (\*.txt)", "\*.txt"); // проверка исходного файла на верность типа

# fc.getExtensionFilters().add(extensionFilter);

# Stage stage2 = (Stage) AnchorPane.getScene().getWindow();

# File file = fc.showOpenDialog(stage2);

# try {

# FileReader fileReader = new FileReader(file); //чтение входного файла

# BufferedReader reader = new BufferedReader(fileReader);

# String line;

# while ((line = reader.readLine()) != null) {

# lines.add(line);

# }

# if (!lines.get(0).matches("\\d+[x]\\d+")) { //проверка правильности ввода размеров матрицы

# Parent root1; //окно ошибки в случае неверного ввода входных данных

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# throw new Exception();

# }

# String[] line1 = lines.get(0).split("x"); //отделяем цифру количества строк от цифры количества столбцов

# stroka = Integer.parseInt(line1[0]); //присваивание значений размеров матрицы

# stolbets = Integer.parseInt(line1[1]);

# for (int i = 1; i < lines.size(); i++) { //проверка на соответствие размера матрицы и введенной ее пользователем

# int k = 0;

# String x = lines.get(i).replace(" ", "");

# if (!lines.get(i).matches("([sf01]\\s)+[01sf]")) { //проверка на правильность ввода данных в матрице

# Parent root1; //окно ошибки в случае неверного ввода данных

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# throw new Exception();

# }

# if (lines.size() - 1 != stroka) { // проверка на соответсвие заданного значения строк в файле

# Parent root1; //окно ошибки в случае несовпадения количества строк с заданной нами матрицей

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# throw new Exception();

# }

# for (int j = 0; j < x.length(); j++) {

# k++;

# }

# if (k != stolbets) { // проверка на соответсвие заданного значения столбцов в файле

# // и количества элементов в строке файла, при несоответствии выбрасываем исключение

# Parent root1; //окно ошибки в случае несовпадения количества столбцов с заданной нами матрицей

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# throw new Exception();

# }

# allelements.add(x); //добавление всех строк в данный лист

# }

# int cS = 0; //количество "s", содержащихся в матрице

# int cF = 0; //количество "f", содержащихся в матрице

# int countzero = 0; //количество "0", содержащихся в матрице

# int countone = 0; //количество "1", содержащихся в матрице

# for (int l = 0; l < allelements.size(); l++) {

# String[] element = allelements.get(l).split(" "); //преобразования массива для получения итогового списка всех элементов по отдельности

# String a = Arrays.toString(element).replace("[", "").replace("]", "");

# String[] b = a.split(""); //b - итоговый массив всех элементов

# for (int i1 = 0; i1 < b.length; i1++) { // в данном цикле подсчитывается количество 0, 1, s, f

# all.add(b[i1]); // и при наличии несоответствий с правилами, выбрасываем исключение.

# if (b[i1].equals("s")) {

# cS++;

# continue;

# }

# if (b[i1].equals("f")) {

# cF++;

# continue;

# }

# if (b[i1].equals("0")) {

# countzero++;

# } else if (b[i1].equals("1")) {

# countone++;

# } else {

# Parent root1; //окно ошибки в случае наличия в матрице других элементов, помимо 0, 1, s, f

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# throw new Exception();

# }

# }

# }

# if ((cS != 1) || (cF != 1)) { //проверка на индивидуальность старта (s) и финиша (f)

# Parent root1; //окно ошибки в случае неиндивидуальности s/f

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# throw new Exception();

# }

# if (countzero == 0) { // проверка на отсутствие 0, т.е все поле заполнено 1 => путей нет, ибо двери все закрыты

# Parent root1; //окно ошибки в случае отсутствия 0 на поле

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# }

# try {

# Stage stage = new Stage();

# if (gridPane == null) { // делаю изначально проверку на наличие на основном окне таблицы,

# // если она есть, то стирается и появляется новая, которую мы создаем в дальнейшем

# gridPane = new GridPane();

# } else {

# gridPane.getChildren().clear();

# gridPane = new GridPane();

# }

# ColumnConstraints cc = new ColumnConstraints(); //отрисовка таблицы

# cc.setFillWidth(true);

# cc.setHgrow(Priority.ALWAYS);

# for (int h = 0; h < stolbets; h++) { // создаю столбцы и добавляю новые в таблицу

# gridPane.getColumnConstraints().add(cc);

# }

# RowConstraints rc = new RowConstraints();

# rc.setFillHeight(true);

# rc.setVgrow(Priority.ALWAYS);

# for (int b = 0; b < stroka; b++) { // создаю строки и добавляю новые в таблицу

# gridPane.getRowConstraints().add(rc);

# }

# gridPane.setHgap(0);// свожу расстояние границ вертикали и горизонтали к нулю

# gridPane.setVgap(0);

# gridPane.setGridLinesVisible(true);// делаю видимыми для пользователя границы матрицы

# int y = 0;

# int[][] arr = new int[stroka][stolbets];

# for (int c = 0; c < arr.length; c++) { // ввод наших данных в ячейки итоговой таблицы

# for (int j = 0; j < arr[c].length; j++) {

# Label lb = new Label(all.get(y));

# lb.setFont(Font.font(40));

# gridPane.add(lb, j, c);

# y++;

# }

# }

# //настраиваю характеристики таблицы, т е ее расположение, ее вид

# gridPane.setLayoutX(375); // размещаю в окне матрицу в середине для удобства пользователя

# gridPane.setLayoutY(100);

# gridPane.setMinWidth(500);

# gridPane.setMinHeight(500);

# AnchorPane.getChildren().add(gridPane); // отображаю матрицу на наше основное окно

# } catch (Exception e) {

# e.printStackTrace();

# }

# } catch (Exception e) {

# e.printStackTrace();

# }

# if (allelements.isEmpty()) { //проверка на наличие пустых строк в изначальной заданной нами матрице

# Parent root1; //окно ошибки в случае наличия пустых строк в заданной нами матрице

# try {

# root1 = FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().getResource("error1.fxml")), resources);

# Stage stage1 = new Stage();

# stage1.setTitle("Error");

# stage1.setScene(new Scene(root1, 600, 556));

# stage1.show();

# } catch (Exception e) {

# e.printStackTrace();

# }

# }

# }

# Logic log = new Logic();

# private void logicOfGame1() throws Exception {

# ArrayList array;

# array = log.logicOfGame(allelements, startX, startY, finishX, finishY);

# try { // создание и отрисовка новой таблицы с итоговой матрицей

# Stage stage = new Stage();

# if (gridPane == null) { // делаю изначально проверку на наличие на основном окне таблицы,

# // если она есть, то стирается и появляется новая, которую мы создаем в дальнейшем

# gridPane = new GridPane();

# } else {

# gridPane.getChildren().clear();

# gridPane = new GridPane();

# }

# ColumnConstraints cc = new ColumnConstraints();

# cc.setFillWidth(true);

# cc.setHgrow(Priority.ALWAYS);

# for (int h = 0; h < stolbets; h++) {

# gridPane.getColumnConstraints().add(cc);

# }

# RowConstraints rc = new RowConstraints();

# rc.setFillHeight(true);

# rc.setVgrow(Priority.ALWAYS);

# for (int b = 0; b < stroka; b++) {// создаю строки и добавляю новые в таблицу

# gridPane.getRowConstraints().add(rc);

# }

# gridPane.setHgap(0);// свожу расстояние границ вертикали и горизонтали к нулю

# gridPane.setVgap(0);

# gridPane.setGridLinesVisible(true);// делаю видимыми для пользователя границы матрицы

# int z = 0;

# for (int c = 0; c < stroka; c++) { // вношу наши конечные значения в ячейки матрицы

# for (int j = 0; j < stolbets; j++) {

# Label lb = new Label(String.valueOf(array.get(z)));

# lb.setFont(Font.font(40));

# gridPane.add(lb, j, c);

# z++;

# }

# }

# //настраиваю характеристики таблицы, т е ее расположение, ее вид

# gridPane.setLayoutX(375); // размещаю в окне матрицу в середине для удобства пользователя

# gridPane.setLayoutY(100);

# gridPane.setMinWidth(500);

# gridPane.setMinHeight(500);

# AnchorPane.getChildren().add(gridPane); // отображаю матрицу на наше основное окно

# } catch (Exception e) {

# e.printStackTrace();

# }

# }

# public void logicOfGame(ActionEvent actionEvent) throws Exception {

# logicOfGame1();

# }

# }

# **Logic.java**

package sample;

import java.util.ArrayList;

import java.util.Arrays;

public class Logic {

public ArrayList<String> logicOfGame(ArrayList<String> allelements, int startX, int startY, int finishX, int finishY) throws Exception {

ArrayList<String> allel = new ArrayList<>(); //лист всех итоговых элементов

ArrayList<String> array = new ArrayList<>(); //лист всех элементов конечного двумерного массива

for (String allelement : allelements) { //занесение всех элементов в один лист

String[] element = allelement.split(" ");

String a = Arrays.toString(element).replace("[", "").replace("]", "");

String[] b = a.split("");

allel.addAll(Arrays.asList(b)); //allel - итоговый список со всеми элементами

}

int stolbets = allelements.get(0).length(); //присваивание значений столбцу

int stroka = allelements.size(); //присваивание значений строке

String[][] Map1 = new String[stroka][stolbets]; //итоговый двумерный массив, который мы будем выводить в конце

String[][] Map = new String[stroka][stolbets]; //имеем двумерный массив с нашими изначальными данными, то бишь 0, 1 и s/f

String[][] cMap = new String[stroka][stolbets]; //двумерный массив, в процессе работы мы работаем с ним и его преобразовываем (ставим индикаторы и распространяем волну)

int o = 0;

for (int x = 0; x < stroka; x++) {

for (int y = 0; y < stolbets; y++) {

Map[x][y] = (allel.get(o));

cMap[x][y] = "0";

if (Map[x][y].equals("s")) { // заносим в память координаты точек старта (s)

startY = x;

startX = y;

}

if (Map[x][y].equals("f")) { // заносим в память координаты точек финиша (f)

finishY = x;

finishX = y;

}

o++;

}

}

boolean add = true;

int step = 0;

for (int y = 0; y < stroka; y++) {

for (int x = 0; x < stolbets; x++) {

if ("1".equals(Map[y][x])) {

cMap[y][x] = "-2"; //индикатор стены (блокирует проход)

} else {

cMap[y][x] = "-1"; // индикатор еще не ступал сюда

}

}

}

cMap[finishY][finishX] = String.valueOf(0); //инициализация точки финиша как "0"

//будем в дальнейшем для подсчета кратчайшего пути идти от финиша к старту

while (add) {

add = false;

for (int y = 0; y < stroka; y++) { //начинаем уже само распространение волны, пока есть возможности

for (int x = 0; x < stolbets; x++) {

if (cMap[y][x].equals(String.valueOf(step))) { //смотрим все соседние клетки по вертикали и горизонтали и распространяем волну

if (y >= 1 && cMap[y - 1][x].equals("-1")) cMap[y - 1][x] = String.valueOf(step + 1);

if (x >= 1 && cMap[y][x - 1].equals("-1")) cMap[y][x - 1] = String.valueOf(step + 1);

if (y < stroka - 1 && cMap[y + 1][x].equals("-1")) cMap[y + 1][x] = String.valueOf(step + 1);

if (x < stolbets - 1 && cMap[y][x + 1].equals("-1")) cMap[y][x + 1] = String.valueOf(step + 1);

}

}

}

step++; //увеличение шага для дальнейшего хода

if (cMap[startY][startX].equals(String.valueOf(-1))) add = true; //если волна все еще не распространилась на стартовую точку, то продолжаем искать путь дальше

else add = false; //решение найдено, выходим из цикла while

if (step > stroka \* stolbets) add = false; //решение не найдено => решений нет, выходим из цикла while

}

for (int y = 0; y < stroka; y++) { //заносим в конечный двумерный массив точки старта и финиша а также закрытые двери (1)

for (int x = 0; x < stolbets; x++) { // и двери, в которые индикатор не ступал и не будет ступать (0)

if (cMap[y][x].equals(String.valueOf(-1))) Map1[y][x] = String.valueOf(0);

else if (cMap[y][x].equals(String.valueOf(-2))) Map1[y][x] = String.valueOf(1);

else if (y == startY && x == startX) {

Map1[y][x] = "s";

}

else if (y == finishY && x == finishX) {

Map1[y][x] = "f";

}

else if (Integer.parseInt(cMap[y][x]) > (-1)) Map1[y][x] = ("\*" + cMap[y][x] + "\*"); //ввод в матрицу ход со значением шага (step)

}

}

int k = startY;

int s = startX;

int stepp = step - 1;

while (stepp != 0) { // идем обратно от финиша к старту и оставляем только единственный наикратчайший путь

// (откидываем все остальные лишние шаги и пути, оставляем только один наикратчайший)

if ((k >= 1 && Map1[k - 1][s].equals("\*" + (stepp)+ "\*"))) { //итоговый путь будет обозначен знаком "●"

Map[k - 1][s] = "●";

k = k - 1;

} else if ((s >= 1 && Map1[k][s - 1].equals("\*" + (stepp)+ "\*"))) {

Map[k][s - 1] = "●";

s = s - 1;

} else if ((k < stroka - 1 && Map1[k + 1][s].equals("\*" + (stepp)+ "\*"))) {

Map[k + 1][s] = "●";

k = k + 1;

} else if ((s < stolbets - 1 && Map1[k][s + 1].equals("\*" + (stepp)+ "\*"))) {

Map[k][s + 1] = "●";

s = s + 1;

}

stepp--; //уменьшение шага для дальнейшего поиска

}

for (int y = 0; y < stroka; y++) { //занесение в итоговый список элементов значения

array.addAll(Arrays.asList(Map[y]).subList(0, stolbets));

}

return array;

}

}

**error1.fxml**

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.text.\*?>  
<?import javafx.scene.control.\*?>  
<?import java.lang.\*?>  
<?import javafx.scene.layout.\*?>  
  
  
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">  
 <children>  
 <Label layoutX="36.0" layoutY="100.0" prefHeight="286.0" prefWidth="528.0" text="Неверный формат введенного файла или отсутствие данных. Проверьте, что данные введены или формат входных данных удовлетворяет правилам, приведенным в разделе Help" textAlignment="CENTER" wrapText="true">  
 <font>  
 <Font name="Bookman Old Style" size="32.0" />  
 </font>  
 </Label>  
 <Label layoutX="219.0" layoutY="38.0" prefHeight="72.0" prefWidth="249.0" text="Error!">  
 <font>  
 <Font name="Constantia" size="49.0" />  
 </font>  
 </Label>  
 </children>  
</AnchorPane>

# **about.fxml**

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.text.\*?>  
<?import javafx.scene.control.\*?>  
<?import java.lang.\*?>  
<?import javafx.scene.layout.\*?>  
  
<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">  
 <center>  
 <Label fx:id="aboutwindow" alignment="TOP\_CENTER" prefHeight="356.0" prefWidth="564.0" text="Основная задача игры Лабиринт - найти самый короткий путь их точки S к точке F. Но на пути будут встречаться двери: 0 - открытые, 1 - закрытые. 1 строка вводится в виде матрицы NxM, а последующие строки содержат в себе ряд чисел, идущих через пробел. В каждой строке должно быть одинаковое количество элементов. Элементы S и F должны храниться в единственном экземпляре. Удачи! " textAlignment="CENTER" textOverrun="CENTER\_WORD\_ELLIPSIS" wrapText="true" BorderPane.alignment="CENTER">  
 <font>  
 <Font name="Cambria Math" size="25.0" />  
 </font></Label>  
 </center>  
</BorderPane>

# **labirint.fxml**

# <?xml version="1.0" encoding="UTF-8"?>

# <?import javafx.scene.text.\*?>

# <?import javafx.scene.control.\*?>

# <?import java.lang.\*?>

# <?import javafx.scene.layout.\*?>

# <AnchorPane fx:id="AnchorPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="853.0" prefWidth="1187.0" style="-fx-background-color: #FFE4E1;" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller">

# <children>

# <MenuBar layoutY="-2.0" prefHeight="32.0" prefWidth="1187.0"> <menus>

# <Menu fx:id="File" mnemonicParsing="false" text="File"> <items>

# <MenuItem fx:id="Open" mnemonicParsing="false" onAction="#OpenAction" text="Open" />

# <MenuItem fx:id="Exit" mnemonicParsing="false" text="Exit" /> </items>

# </Menu>

# <Menu fx:id="Help" mnemonicParsing="false" text="Help"> <items>

# <MenuItem fx:id="About" mnemonicParsing="false" text="About" /> </items>

# </Menu>

# </menus>

# </MenuBar>

# <Label fx:id="label" layoutX="135.0" layoutY="112.0" prefHeight="645.0" prefWidth="937.0" />

# <Button fx:id="start" layoutX="529.0" layoutY="757.0" mnemonicParsing="false" onAction="#logicOfGame" prefHeight="46.0" prefWidth="130.0" text="START">

# <font>

# <Font name="Constantia Bold" size="26.0" />

# </font>

# </Button>

# </children>

# </AnchorPane>

**noWays.fxml**

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.\*?>

<?import javafx.scene.control.\*?>

<?import java.lang.\*?>

<?import javafx.scene.layout.\*?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">

<children>

<Label layoutX="36.0" layoutY="100.0" prefHeight="286.0" prefWidth="528.0" text="Нет возможных путей!" textAlignment="CENTER" wrapText="true">

<font>

<Font name="Bookman Old Style" size="32.0" />

</font>

</Label>

<Label layoutX="219.0" layoutY="38.0" prefHeight="72.0" prefWidth="249.0" text="Error!">

<font>

<Font name="Constantia" size="49.0" />

</font>

</Label>

</children>

</AnchorPane>

# **Тестирование**

Класс test тестирует исключительно класс Logic и его единственный метод logicOfGame().

package sample;  
  
  
import org.junit.Test;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Collection;  
  
import static org.junit.Assert.*assertEquals*;  
  
public class test {  
  
 Collection<String> input = new ArrayList<>(Arrays.*asList*("00000","00110","1s110","00010","01f00"));  
 Collection<String> output = new ArrayList<>(Arrays.*asList*(  
 "0","0","0","0","0",  
 "0","0","1","1","0",  
 "1","s","1","1","0",  
 "0","●","●","1","0",  
 "0","1","f","0","0"));  
  
 Collection<String> input1 = new ArrayList<>(Arrays.*asList*("00000f1","0111101","1110001","0010101","0110001","0000111", "01s0111"));  
 Collection<String> output1 = new ArrayList<>(Arrays.*asList*(  
 "0","0","0","0","0","f","1",  
 "0","1","1","1","1","●","1",  
 "1","1","1","●","●","●","1",  
 "0","0","1","●","1","0","1",  
 "0","1","1","●","0","0","1",  
 "0","0","●","●","1","1","1",  
 "0","1","s","0","1","1","1"));  
 Collection<String> input2 = new ArrayList<>(Arrays.*asList*("000f","0010","1010","0000","0s10"));

Collection<String> output2 = new ArrayList<>(Arrays.*asList*(  
 "0","●","●","f",  
 "0","●","1","0",  
 "1","●","1","0",  
 "0","●","0","0",  
 "0","s","1","0"));  
 Logic log = new Logic();  
  
 @Test  
 public void logicOfGame1() throws Exception {  
 *assertEquals*(output, log.logicOfGame((ArrayList<String>) input, 1, 2, 2 , 4));  
 *assertEquals*(output1, log.logicOfGame((ArrayList<String>) input1, 5, 0, 2 , 6));  
 *assertEquals*(output2, log.logicOfGame((ArrayList<String>) input2, 3, 0, 1 , 4));  
  
 }  
  
}

# **Скриншоты работы программы**

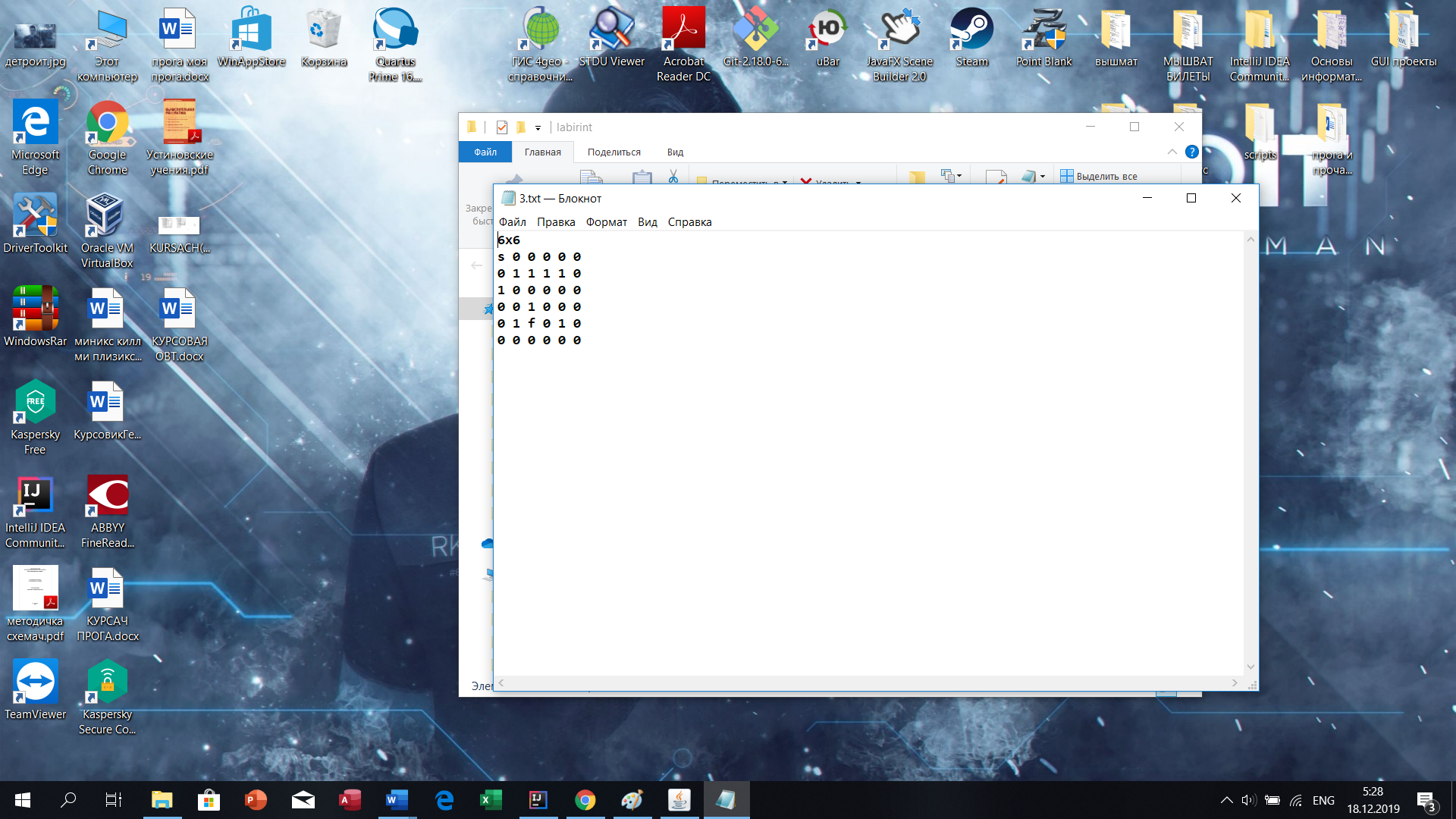
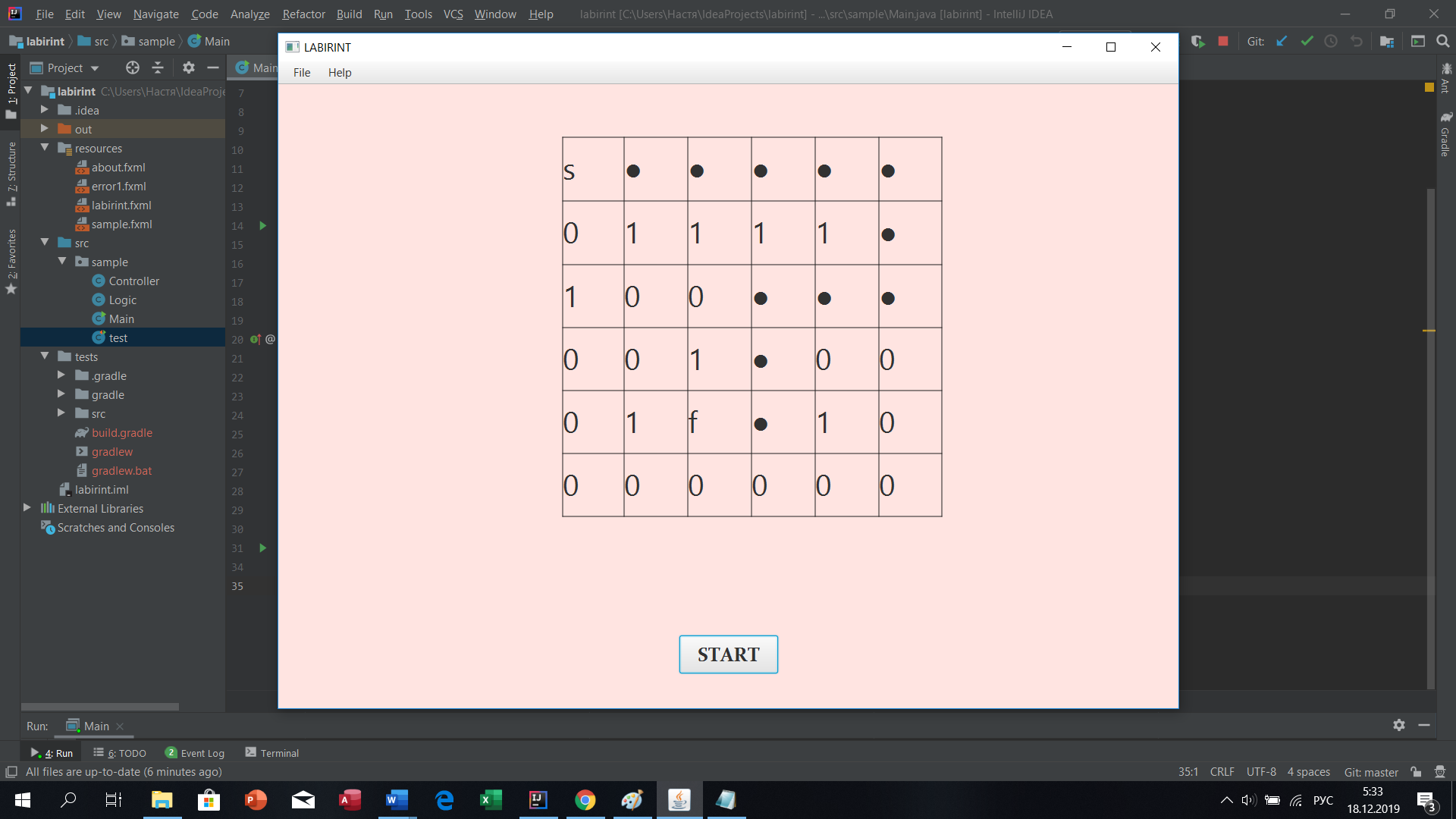
Рис.1 Главное окно и меню программы

Рис.2 Входной файл верного формата



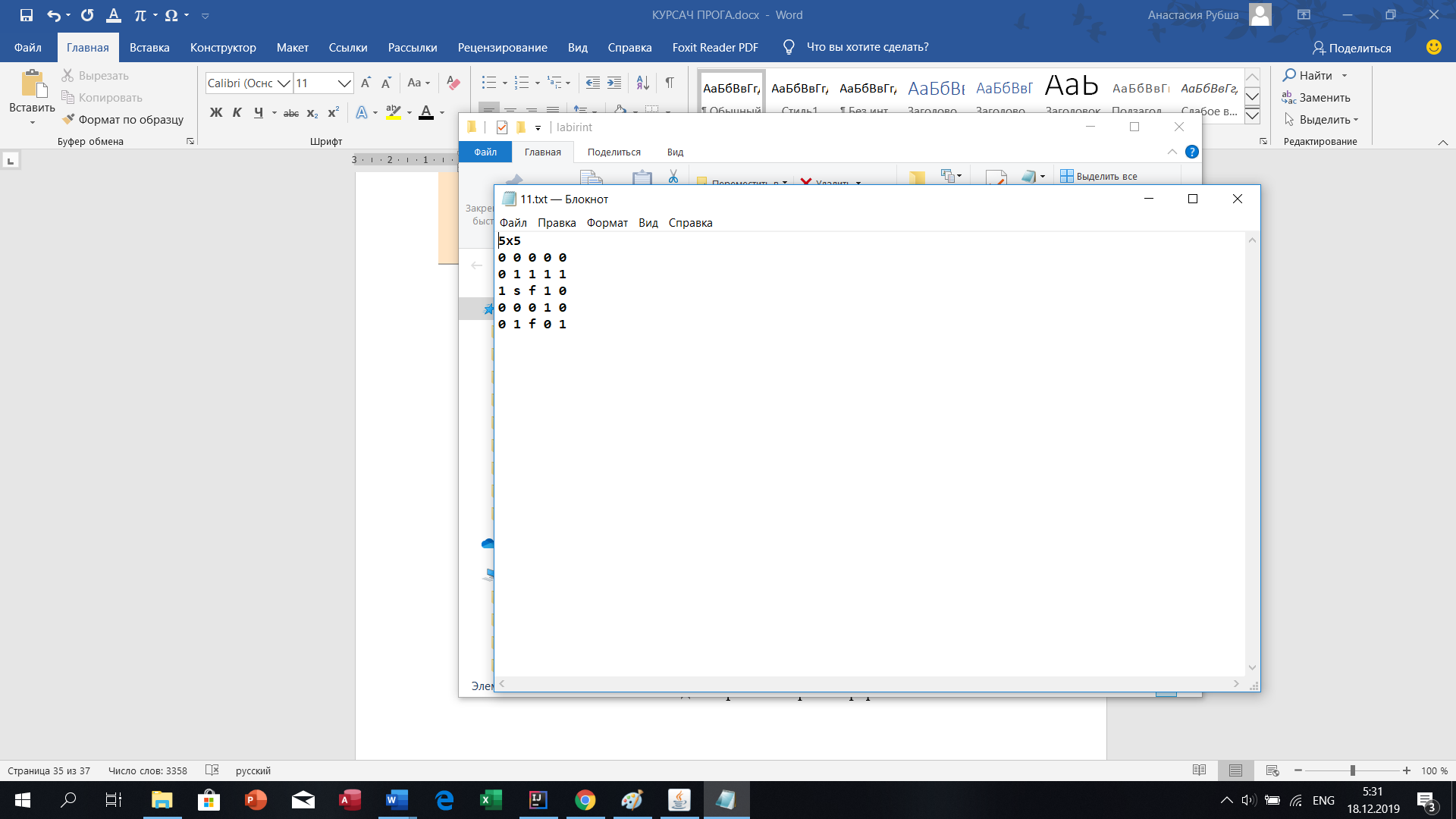
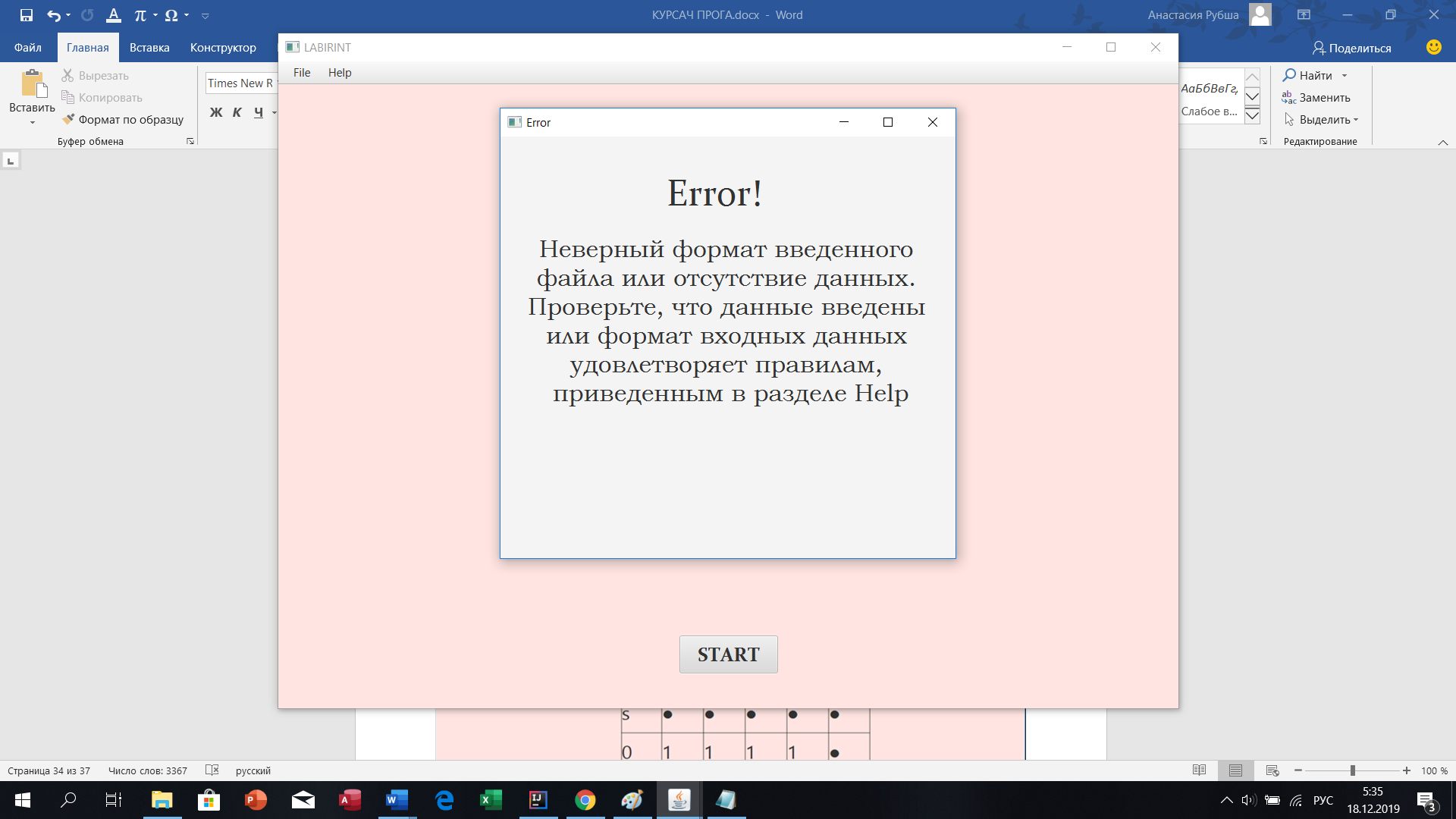
Рис. 3 Результат работы программы для входных данных, показанных на рисунке 2

Рис. 4 Входной файл неверного формата (2 финиша)

Рис. 5 Предупреждение о неверном формате входного файла

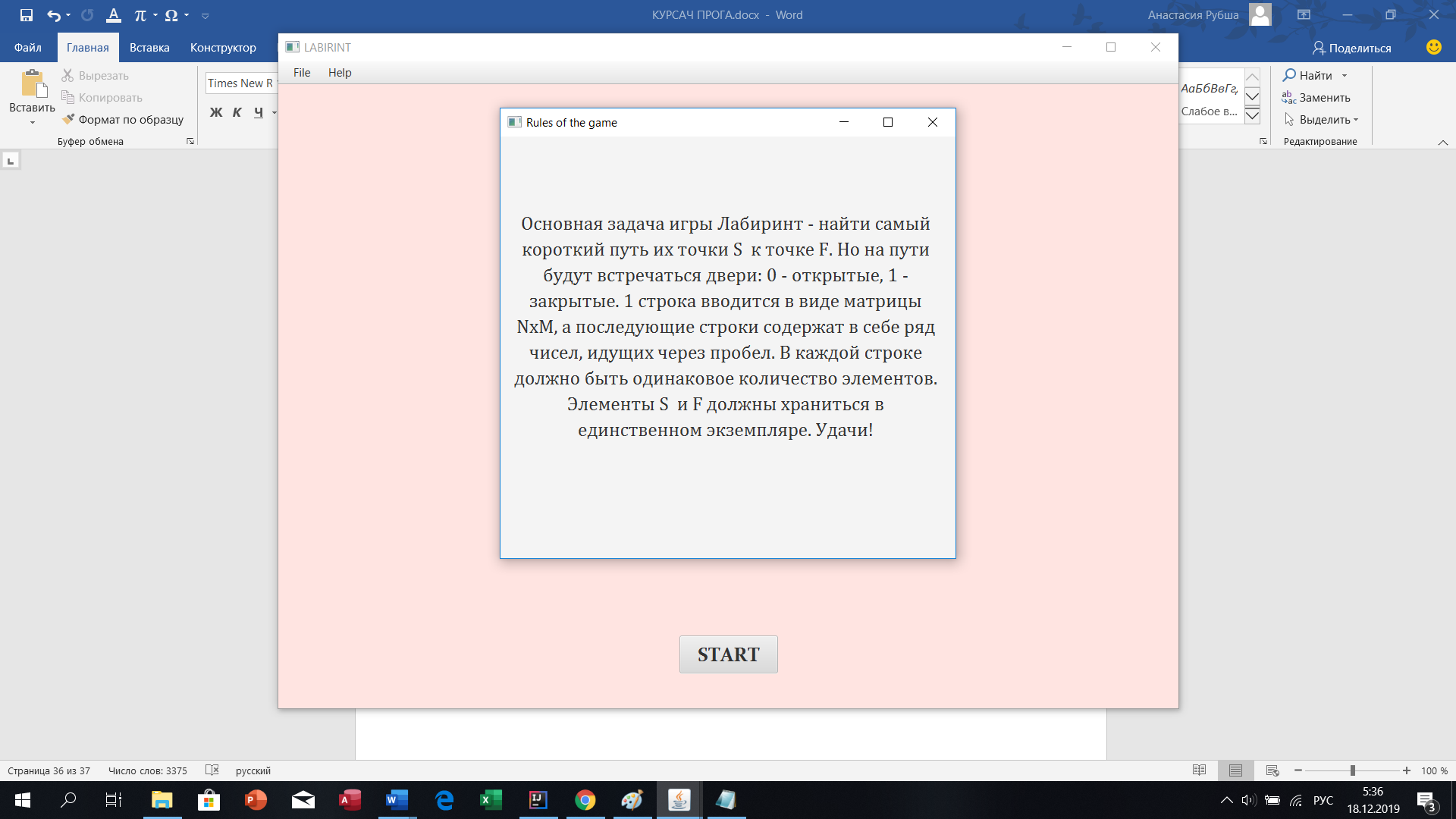
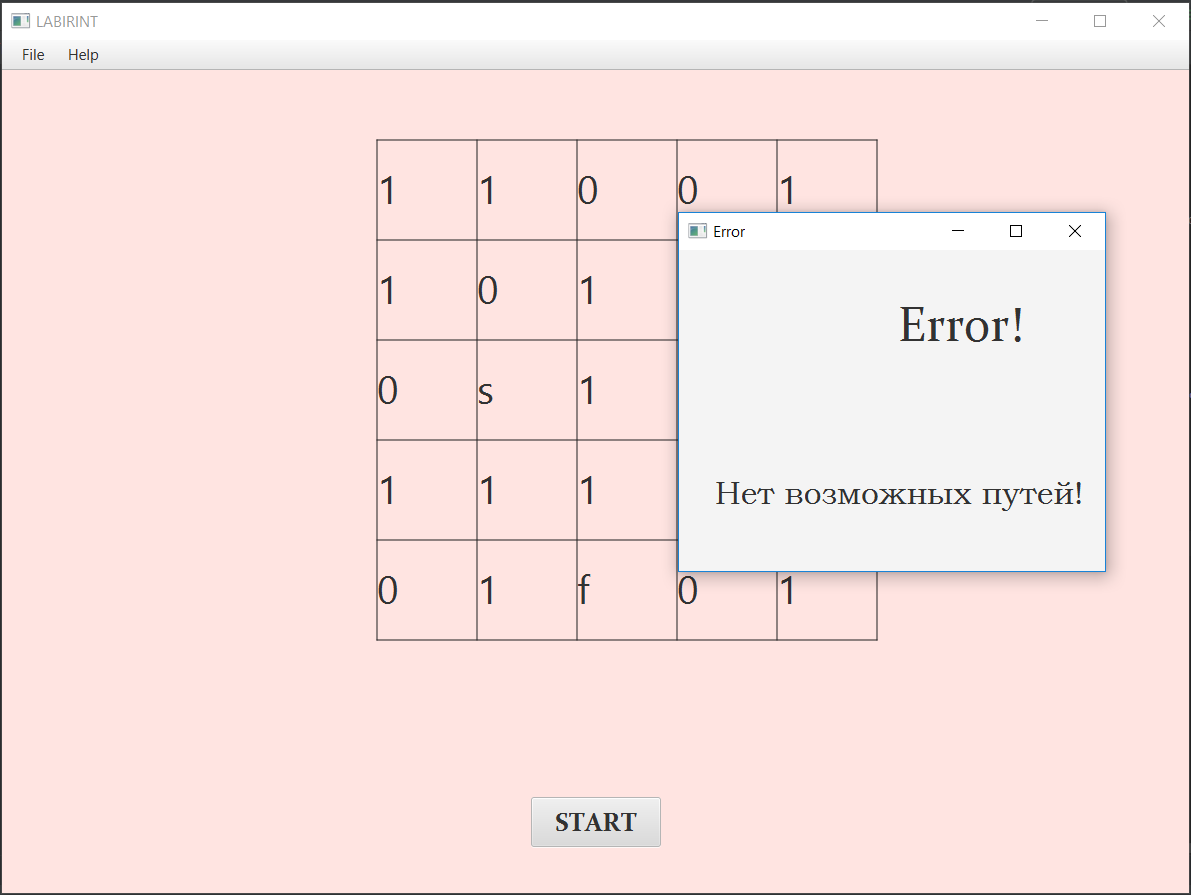


Рис. 6 Окно с правилами. (Help -> About)

Рис. 7 Предупреждение об отсутствии всевозможных путей в лабиринте.