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 и его решателя на языке Java

Выполнил

студент гр. 3530903/80003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Викторов И.А.

(подпись)

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

(подпись)

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

Санкт-Петербург   
2019

**Оглавление**

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc25151727)

[**МЕТОД РЕШЕНИЯ** 4](#_Toc25151728)

[**ЛИСТИНГ ПРОГРАММЫ** 7](#_Toc25151729)

[**Main.java** 7](#_Toc25151730)

[**MainMenuController.java** 8](#_Toc25151731)

[**ActionController.java** 13](#_Toc25151732)

[**HelpAboutController.java** 18](#_Toc25151733)

[**ExceptionController.java** 20](#_Toc25151734)

[**Parsing.java** 21](#_Toc25151735)

[**NodeCircle.java** 23](#_Toc25151736)

[**AlgorithmFindRoute.java** 24](#_Toc25151737)

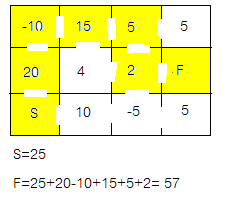
[**ОШИБКИ И ПРЕДУПРЕЖДЕНИЯ** 29](#_Toc25151738)

[**ТЕСТЫ** 30](#_Toc25151739)

[**СКРИНШОТЫ ПРОГРАММЫ** 32](#_Toc25151740)

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Лабиринт представляет собой матрицу NxM комнат. Между некоторыми из них есть двери. У человека есть исходная сумма денег. Когда человек входит в комнату, сумма его денег изменяется (может увеличиться или уменьшиться) на величину заданную для этой комнаты. Найти путь от Start (S) до Finish (F), чтобы собрать максимальное количество денег (или потерять минимальное).



GitHub репозиторий проекта: <https://github.com/Victorov-I-A/Cash-Labyrinth>

# **МЕТОД РЕШЕНИЯ**

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

Содержимое пакета view:

* View – класс, отвечающий за графическое представление основного окна

Cодержимое пакета controller:

* Main - запуск приложения
* Controller – контроллер, отвечающий за обработку событий происходящих в основном окне с игровым полем. Помимо этого, этот класс отвечает за визуализацию входных данных и алгоритма решателя.

Содержимое пакета model:

* Logic - алгоритм решателя
* Graph – класс, отвечающий за создание графа из ячеек игрового поля
* Room – класс, представляющий игровую ячейку
* Triplet – вспомогательный класс, предназначенный для создания более оптимального алгоритма
* Pair – вспомогательный класс, предназначенный для более удобного хранения данных

Более подробная работа программы:

В главном меню пользователь с помощью тулбара может открыть проводник и с его помощью создать, либо редактировать уже готовый текстовый файл.

За это отвечает метод clickFileChooser().

В текстовом файле он задает входные данные согласно инструкции изложенной в Help -> About.

Для выбора рабочего текстового файла пользователю необходимо нажать на кнопку “Choose File”. В случае, если в текстовом файле находятся некорректные входные данные пользователю вылетит предупреждение об этом.

За это отвечает метод clickChoose(), который в свою очередь вызывает метод scanText() из класса Parsing, который проверяет входную строку на корректность входных данных, после чего записывает количество узлов графа и сами узлы в матрицу смежности.

После выбора необходимого текстового файла, пользователю необходимо нажать на кнопку “Start”, которая открывает следующую сцену. Так же в случае если пользователь предварительно не выбрал необходимый файл, вылезает предупреждение об этом.

За это отвечает метод clickStart().

На открытой новой сцене сразу инициализирована отрисовка графа, кнопка “Solve it” и надпись “Current progress”.

За отрисовку графа отвечает метод draw() класса ActionController, который рисует по следующему алгоритму:

Узлы располагаются как точки по окружности с заданным центром и радиусом.

Радиус константен. Центр окружности - центр сцены.

Шаг между двумя узлами равен углу.

360° делим на количество узлов - получаем шаг.

Узлы располагаются по формуле:

узел(центр сцены по X + синус(шаг) \* радиус, центр сцены по Y + косинус(шаг) \* радиус)

Далее, для запуска решателя, пользователю необходимо нажать на кнопку “Solve it”. Перед запуском решателя совершаются две проверки.

1. Проверка на связность. Если у графа есть хоть один узел не имеющий связей, то вылетает предупреждение о том, что маршрут у данного графа построить невозможно.
2. Так как алгоритм обхода графа, проходя по каждому ребру только по разу, давно известен как Эйлеров алгоритм, то проверка звучит как “проверить, является ли граф Эйлеровым”, т.е проверить не содержит ли граф больше двух нечетных узлов.

За это отвечают два метода allConnections() и isEuler() из класса AlgorithmFindRoute.

В случае, если все два условия удовлетворены, то запускается решатель.

Алгоритм построения Эйлерова цикла:

Чтобы построить Эйлеров путь, нужно запустить алгоритм из вершины с нечетной степенью.  
Алгоритм напоминает поиск в ширину. Главное отличие состоит в том, что пройденными помечаются не вершины, а ребра графа. Начиная со стартовой вершины v строим путь, добавляя на каждом шаге не пройденное еще ребро, смежное с текущей вершиной. Вершины пути накапливаются. Когда наступает такой момент, что для текущей вершины все инцидентные ей ребра уже пройдены, записываем накопленные вершины в ответ, пока не встретим вершину, которой инцидентны не пройденные еще ребра. Далее продолжаем обход по не посещенным ребрам.

За это отвечает метод printEulerTour() из класса AlgorithmFindRoute.

За отрисовку решателя отвечает метод printPassedRibs() из класса ActionController.

# **ЛИСТИНГ ПРОГРАММЫ**

## **Main.java**

package labyrinth.controller;  
  
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import labyrinth.view.View;  
  
public class Main extends Application {  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) {  
 Controller controller = new Controller();  
  
 View view = new View(controller);  
 view.createMain();  
  
 {  
 Group secondRoot = new Group();  
  
 secondRoot.getChildren().add(controller.errorLabel);  
  
 controller.errorWindow.setTitle("Error");  
 controller.errorWindow.setScene(new Scene(secondRoot, 300, 80));  
  
 controller.errorWindow.initModality(Modality.*WINDOW\_MODAL*);  
 controller.errorWindow.initOwner(primaryStage);  
 }  
  
 {  
 Group root = new Group();  
  
 root.getChildren().add(controller.pane);  
  
 primaryStage.setTitle("Cash labyrinth");  
 primaryStage.setScene(new Scene(root, 1200, 1000));  
 primaryStage.show();  
 }  
 }  
}

**Controller.java**

package labyrinth.controller;  
  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.Pane;  
import javafx.scene.layout.VBox;  
import javafx.stage.Stage;  
import labyrinth.model.Logic;  
import labyrinth.model.Pair;  
  
public class Controller {  
 private Logic logic;  
 private int[][] horizonDoors = new int[7][8];  
 private int[][] verticalDoors = new int[8][7];  
 private boolean isStart = false;  
 private boolean isEnd = false;  
 private TextField start;  
 private TextField end;  
 private int cash;  
  
 public TextField[][] matrixOfTField = new TextField[8][8];  
 public Button[][] horizonButtons = new Button[7][8];  
 public Button[][] verticalButtons = new Button[8][7];  
  
 public final Pane pane = new Pane();  
 public final TextField cashField = new TextField();  
 public final Button startButton = new Button();  
 public final Button reset = new Button();  
 public final VBox playField = new VBox();  
  
 public final Label errorLabel = new Label();  
 public final Stage errorWindow = new Stage();  
  
 public Controller() {  
 pane.getChildren().addAll(playField, cashField, startButton, reset);  
  
 for (int i = 0; i < 15; i++) {  
  
 HBox hBox = new HBox();  
 hBox.prefHeight(770);  
 hBox.prefWidth(70);  
  
 if (i % 2 == 0) {  
 for (int j = 0; j < 8; j++) {  
 TextField textField = new TextField();  
 textField.setPrefSize(70, 70);  
  
 hBox.getChildren().add(textField);  
 matrixOfTField[i / 2][j] = textField;  
  
 if (j != 7) {  
 Button button = new Button();  
 button.setPrefSize(30, 70);  
  
 hBox.getChildren().add(button);  
 verticalButtons[i / 2][j] = button;  
 verticalDoors[i / 2][j] = 0;  
 }  
 }  
 }  
 if (i % 2 == 1) {  
 hBox.setSpacing(30);  
  
 for (int j = 0; j < 8; j++) {  
 Button button = new Button();  
 button.setPrefSize(70, 30);  
  
 hBox.getChildren().add(button);  
 horizonButtons[i / 2][j] = button;  
 horizonDoors[i / 2][j] = 0;  
 }  
 }  
 playField.getChildren().add(hBox);  
 }  
 }  
  
 public void horizonDoorAction(int x, int y) {  
 if (horizonDoors[x][y] == 0) {  
 horizonButtons[x][y].setStyle("-fx-background-color: white");  
 horizonDoors[x][y] = 1;  
 } else {  
 horizonButtons[x][y].setStyle("-fx-background-color");  
 horizonDoors[x][y] = 0;  
 }  
 }  
  
 public void verticalDoorAction(int x, int y) {  
 if (verticalDoors[x][y] == 0) {  
 verticalButtons[x][y].setStyle("-fx-background-color: white");  
 verticalDoors[x][y] = 1;  
 } else {  
 verticalButtons[x][y].setStyle("-fx-background-color");  
 verticalDoors[x][y] = 0;  
 }  
 }  
  
 public void roomAction(int x, int y) {  
 if (isStart == false && matrixOfTField[x][y] != end) {  
  
 matrixOfTField[x][y].setStyle("-fx-background-color: yellow");  
 matrixOfTField[x][y].setText("start");  
 matrixOfTField[x][y].setEditable(false);  
  
 isStart = true;  
 start = matrixOfTField[x][y];  
 return;  
 }  
 if (isStart == true && matrixOfTField[x][y] == start) {  
  
 matrixOfTField[x][y].setStyle("-fx-background-color");  
 matrixOfTField[x][y].setText("");  
 matrixOfTField[x][y].setEditable(true);  
  
 isStart = false;  
 start = null;  
 return;  
 }  
 if (isEnd == false && matrixOfTField[x][y] != start) {  
 matrixOfTField[x][y].setStyle("-fx-background-color: yellow");  
 matrixOfTField[x][y].setText("end");  
 matrixOfTField[x][y].setEditable(false);  
  
 isEnd = true;  
 end = matrixOfTField[x][y];  
 return;  
 }  
 if (isEnd == true && matrixOfTField[x][y] == end) {  
  
 matrixOfTField[x][y].setStyle("-fx-background-color");  
 matrixOfTField[x][y].setText("");  
 matrixOfTField[x][y].setEditable(true);  
  
 isEnd = false;  
 end = null;  
 }  
 }  
  
 public void resetAction() {  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 matrixOfTField[i][j].setStyle("-fx-background-color");  
 matrixOfTField[i][j].setText("");  
 matrixOfTField[i][j].setEditable(true);  
  
 if (i != 7) {  
 horizonButtons[i][j].setStyle("-fx-background-color");  
 horizonButtons[i][j].setDisable(false);  
  
 horizonDoors[i][j] = 0;  
 }  
 if (j != 7) {  
 verticalButtons[i][j].setStyle("-fx-background-color");  
 verticalButtons[i][j].setDisable(false);  
  
 verticalDoors[i][j] = 0;  
 }  
  
 }  
 startButton.setText("Старт!");  
 startButton.setOnMouseClicked(e -> startAction());  
  
 isEnd = false;  
 isStart = false;  
 start = null;  
 end = null;  
  
 cash = 0;  
 cashField.setText("");  
 }  
  
 public void startAction() {  
 if (!isStart) {  
 errorLabel.setText("Выберите поле старта");  
 errorWindow.show();  
 return;  
  
 }  
  
 if (!isEnd) {  
 errorLabel.setText("Выберите поле выхода");  
 errorWindow.show();  
 return;  
  
 }  
  
 try {  
 cash = Integer.*parseInt*(cashField.getText());  
 } catch (NumberFormatException e) {  
 errorLabel.setText("Поле с начальной суммой не заполнено");  
 errorWindow.show();  
 return;  
  
 }  
  
 int[][] matrixOfRoom = new int[8][8];  
 Pair startCoordinate = null;  
 Pair endCoordinate = null;  
  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 if (matrixOfTField[i][j] == start) {  
 matrixOfRoom[i][j] = 0;  
 startCoordinate = new Pair(i, j);  
 } else if (matrixOfTField[i][j] == end) {  
 matrixOfRoom[i][j] = 0;  
 endCoordinate = new Pair(i, j);  
 } else try {  
 matrixOfRoom[i][j] = Integer.*parseInt*(matrixOfTField[i][j].getText());  
 } catch (NumberFormatException e) {  
 matrixOfTField[i][j].setText("0");  
 matrixOfRoom[i][j] = 0;  
 }  
 matrixOfTField[i][j].setEditable(false);  
 if (i != 7)  
 horizonButtons[i][j].setDisable(true);  
 if (j != 7)  
 verticalButtons[i][j].setDisable(true);  
 }  
  
 logic = new Logic(matrixOfRoom, horizonDoors, verticalDoors, startCoordinate, endCoordinate);  
 logic.findWay();  
  
 if (logic.getFinalWay().isEmpty()) {  
 errorLabel.setText("Поле задано неверно");  
 errorWindow.show();  
 redactionAction();  
 return;  
 }  
  
 for (Pair coordinate : logic.getFinalWay()) {  
 matrixOfTField[coordinate.getX()][coordinate.getY()].setStyle("-fx-background-color: #B0E0E6");  
 }  
  
 cashField.setText(String.*valueOf*(cash + logic.getFinalCost()));  
  
 startButton.setText("Редактировать");  
 startButton.setOnMouseClicked(e -> redactionAction());  
 }  
  
 public void redactionAction() {  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 if (matrixOfTField[i][j] != end && matrixOfTField[i][j] != start) {  
 matrixOfTField[i][j].setStyle("-fx-background-color");  
 matrixOfTField[i][j].setEditable(true);  
 }  
  
 if (i != 7) {  
 horizonButtons[i][j].setDisable(false);  
 }  
 if (j != 7) {  
 verticalButtons[i][j].setDisable(false);  
 }  
  
 }  
 cashField.setText(String.*valueOf*(cash));  
  
 startButton.setText("Старт!");  
 startButton.setOnMouseClicked(e -> startAction());  
 }  
}

**View.java**

package labyrinth.view;  
  
import labyrinth.controller.Controller;  
  
public class View {  
 private Controller controller;  
  
  
 public View(Controller controller) {  
 this.controller = controller;  
 }  
  
  
 public void createMain() {  
 controller.errorLabel.setPrefSize(250, 40);  
 controller.errorLabel.setLayoutX(25);  
 controller.errorLabel.setLayoutY(15);  
  
 controller.playField.setStyle("-fx-background-color: linear-gradient(#f2f2f2, #d6d6d6),\n" +  
 "linear-gradient(#fcfcfc 0%, #d9d9d9 20%, #d6d6d6 100%),\n" +  
 "linear-gradient(#dddddd 0%, #f6f6f6 50%);"  
 );  
  
 controller.cashField.setPromptText("Начальная сумма");  
 controller.cashField.setPrefSize(120, 25);  
 controller.cashField.setLayoutX(1030);  
 controller.cashField.setLayoutY(50);  
  
 controller.startButton.setText("Старт!");  
 controller.startButton.setPrefSize(70, 70);  
 controller.startButton.setLayoutX(1080);  
 controller.startButton.setLayoutY(880);  
  
 controller.reset.setText("Сброс");  
 controller.reset.setPrefSize(70, 70);  
 controller.reset.setLayoutX(1000);  
 controller.reset.setLayoutY(880);  
  
  
 controller.playField.setPrefSize(770, 770);  
 controller.playField.setLayoutX(215);  
 controller.playField.setLayoutY(115);  
  
 for (int i = 0; i < 8; i++){  
 for (int j = 0; j < 8; j++) {  
 int x = i;  
 int y = j;  
 if (i != 7)  
 controller.horizonButtons[x][y].setOnMouseClicked( e -> controller.horizonDoorAction(x, y));  
 if (j != 7)  
 controller.verticalButtons[x][y].setOnMouseClicked( e -> controller.verticalDoorAction(x, y));  
 }  
 }  
  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 int x = i;  
 int y = j;  
 controller.matrixOfTField[x][y].setOnMouseClicked( e -> controller.roomAction(x, y));  
 }  
  
 controller.startButton.setOnMouseClicked( e -> controller.startAction());  
  
 controller.reset.setOnMouseClicked(e -> controller.resetAction());  
 }  
}

**Logic.java**

package labyrinth.model;  
  
import java.util.\*;  
  
public class Logic {  
 private HashSet<Room> finalWay = new HashSet<>();  
 private int finalCost;  
  
 private Room startRoom;  
 private Room endRoom;  
  
 public Logic(int[][] matrix, int[][] horizonDoors, int[][] verticalDoors, Pair startCoordinate, Pair endCoordinate) {  
 Graph graph = new Graph(matrix, startCoordinate, endCoordinate);  
 graph.createGraph(horizonDoors,verticalDoors);  
 startRoom = graph.getStartRoom();  
 endRoom = graph.getEndRoom();  
 }  
  
 public void findWay() {  
 HashSet<Room> finalWay = new HashSet<>();  
 int finalCost = Integer.*MIN\_VALUE*;  
  
 ArrayDeque<Triplet> allWays = new ArrayDeque<>();  
  
 allWays.addFirst(new Triplet(new HashSet<>(), startRoom, 0));  
 allWays.getFirst().getWay().add(startRoom);  
  
 while (!allWays.isEmpty()) {  
 Triplet previous = allWays.getFirst();  
  
 for (Room neighbor: previous.getRoom().getNeighbors()) {  
 if (neighbor == endRoom && previous.getCost() > finalCost) {  
 finalWay = previous.getWay();  
 finalCost = previous.getCost();  
 } else if  
 (!previous.getWay().contains(neighbor) && neighbor != endRoom) {  
 allWays.addLast(previous.nextStep(neighbor));  
 }  
 }  
  
 allWays.removeFirst();  
 }  
 finalWay.remove(startRoom);  
 this.finalWay = finalWay;  
 this.finalCost = finalCost;  
 }  
  
 public LinkedList<Pair> getFinalWay() {  
 LinkedList<Pair> coordinates = new LinkedList<>();  
  
 finalWay.forEach(room ->  
 coordinates.add(room.getCoordinate())  
 );  
 return coordinates;  
  
 }  
  
 public int getFinalCost() {  
 return finalCost;  
 }  
}

**Graph.java**

package labyrinth.model;  
  
public class Graph {  
 private Room[][] matrixOfRoom = new Room[8][8];  
 private Room startRoom;  
 private Room endRoom;  
  
  
 public Graph(int[][] matrix, Pair startCoordinate, Pair endCoordinate) {  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 matrixOfRoom[i][j] = new Room(matrix[i][j], i, j);  
 }  
 startRoom = matrixOfRoom[startCoordinate.getX()][startCoordinate.getY()];  
 endRoom = matrixOfRoom[endCoordinate.getX()][endCoordinate.getY()];  
 }  
  
 public void createGraph(int[][] horizonDoors, int[][] verticalDoors) {  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 Room firstRoom = matrixOfRoom[i][j];  
 Room secondRoom;  
  
 if (j != 7 && verticalDoors[i][j] == 1) {  
 secondRoom = matrixOfRoom[i][j + 1];  
 firstRoom.addToNeighbors(secondRoom);  
 secondRoom.addToNeighbors(firstRoom);  
 }  
 if (i != 7 && horizonDoors[i][j] == 1) {  
 secondRoom = matrixOfRoom[i + 1][j];  
 firstRoom.addToNeighbors(secondRoom);  
 secondRoom.addToNeighbors(firstRoom);  
 }  
 }  
 }  
  
 public Room getEndRoom() {  
 return endRoom;  
 }  
  
 public Room getStartRoom() {  
 return startRoom;  
 }  
  
 public Room[][] getMatrixOfRoom() {  
 return matrixOfRoom;  
 }  
}

**Room.java**

package labyrinth.model;  
  
import java.util.\*;  
  
public class Room {  
  
 private int cost;  
 private Pair coordinate;  
 private List<Room> neighbors = new ArrayList<>(); //соседние клетки, в которые можно попасть  
  
 public Room(int cost, int x, int y) {  
 this.cost = cost;  
 coordinate = new Pair(x, y);  
 }  
  
 public int getCost() {  
 return cost;  
 }  
  
 public Pair getCoordinate() {  
 return coordinate;  
 }  
  
 public List<Room> getNeighbors() {  
 return neighbors;  
 }  
  
 public void addToNeighbors(Room room) {  
 neighbors.add(room);  
 }  
}

**Triplet.java**

package labyrinth.model;  
  
import java.util.HashSet;  
  
public class Triplet {  
 private HashSet<Room> way;  
 private Room room;  
 private int cost;  
  
 public Triplet(HashSet<Room> way, Room room, int cost) {  
 this.way = way;  
 this.room = room;  
 this.cost = cost;  
 }  
  
 public HashSet<Room> getWay() {  
 return way;  
 }  
  
 public int getCost() {  
 return cost;  
 }  
  
 public Room getRoom() {  
 return room;  
 }  
  
 public Triplet nextStep(Room room) {  
 HashSet<Room> newWay = new HashSet<>(getWay());  
 newWay.add(room);  
  
 return new Triplet(newWay, room, getCost() + room.getCost());  
 }  
}

**Pair.java**

package labyrinth.model;  
  
public class Pair {  
 private int x;  
 private int y;  
  
 public Pair(int x, int y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public int getY() {  
 return y;  
 }  
  
 public int getX() {  
 return x;  
 }  
}

# **ТЕСТЫ**

Были созданы тесты в классе Test.java, проверяющие работу программы.

package test;  
  
import labyrinth.model.Graph;  
import labyrinth.model.Logic;  
import labyrinth.model.Pair;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.junit.jupiter.api.Assertions.*assertTrue*;  
  
class Test {  
  
 @org.junit.jupiter.api.Test  
 void graphTest() {  
 int[][] matrix = new int[8][8];  
 int[][] horizonDoors = new int[7][8];  
 int[][] verticalDoors = new int[8][7];  
  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 matrix[i][j] = 1;  
 if (i != 7)  
 horizonDoors[i][j] = 0;  
 if (j != 7)  
 verticalDoors[i][j] = 0;  
 }  
  
 horizonDoors[0][0] = 1;  
 verticalDoors[0][0] = 1;  
  
 Graph graph = new Graph(matrix, new Pair(0,0), new Pair(0, 1));  
 graph.createGraph(horizonDoors, verticalDoors);  
  
 *assertTrue*(graph.getStartRoom().getNeighbors().contains(graph.getMatrixOfRoom()[1][0]));  
 *assertTrue*(graph.getStartRoom().getNeighbors().contains(graph.getEndRoom()));  
 }  
  
 @org.junit.jupiter.api.Test  
 void logicTest() {  
 int[][] matrix = new int[8][8];  
 int[][] horizonDoors = new int[7][8];  
 int[][] verticalDoors = new int[8][7];  
  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 matrix[i][j] = 8;  
 if (i != 7)  
 horizonDoors[i][j] = 0;  
 if (j != 7 && i == 0)  
 verticalDoors[i][j] = 1;  
 else if (j != 7)  
 verticalDoors[i][j] = 0;  
 }  
  
 Logic logic = new Logic(matrix, horizonDoors, verticalDoors, new Pair(0, 0), new Pair(0, 7));  
 logic.findWay();  
 *assertEquals*(48, logic.getFinalCost());  
 }  
  
 @org.junit.jupiter.api.Test  
 void nullTest() {  
 int[][] matrix = new int[8][8];  
 int[][] horizonDoors = new int[7][8];  
 int[][] verticalDoors = new int[8][7];  
  
 for (int i = 0; i < 8; i++)  
 for (int j = 0; j < 8; j++) {  
 matrix[i][j] = 1;  
 if (i != 7)  
 horizonDoors[i][j] = 0;  
 if (j != 7)  
 verticalDoors[i][j] = 0;  
 }  
 Logic logic = new Logic(matrix, horizonDoors, verticalDoors, new Pair(0, 0), new Pair(0, 7));  
 logic.findWay();  
 *assertTrue*(logic.getFinalWay().isEmpty());  
 }  
}

# 

# **СКРИНШОТЫ ПРОГРАММЫ**

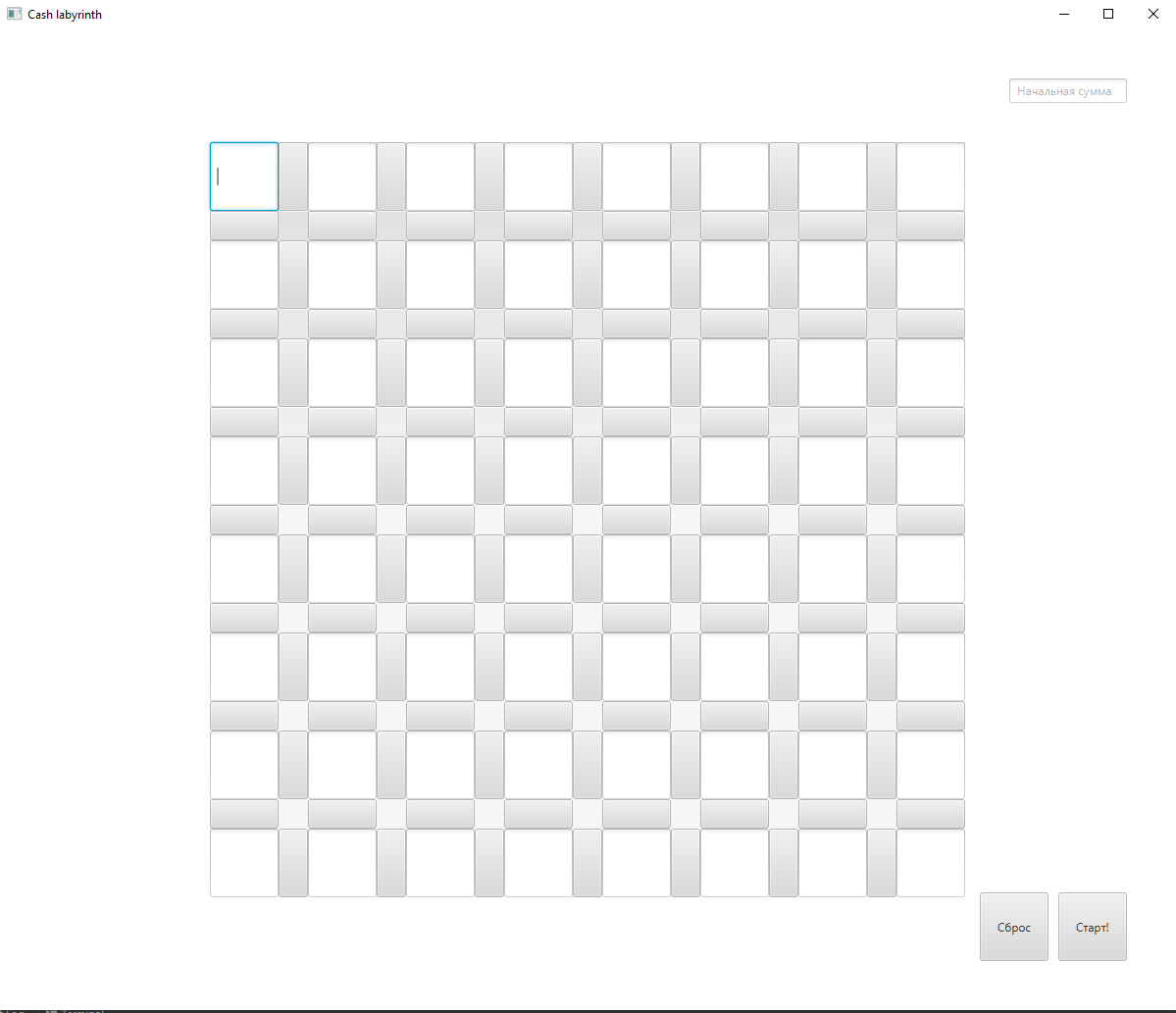
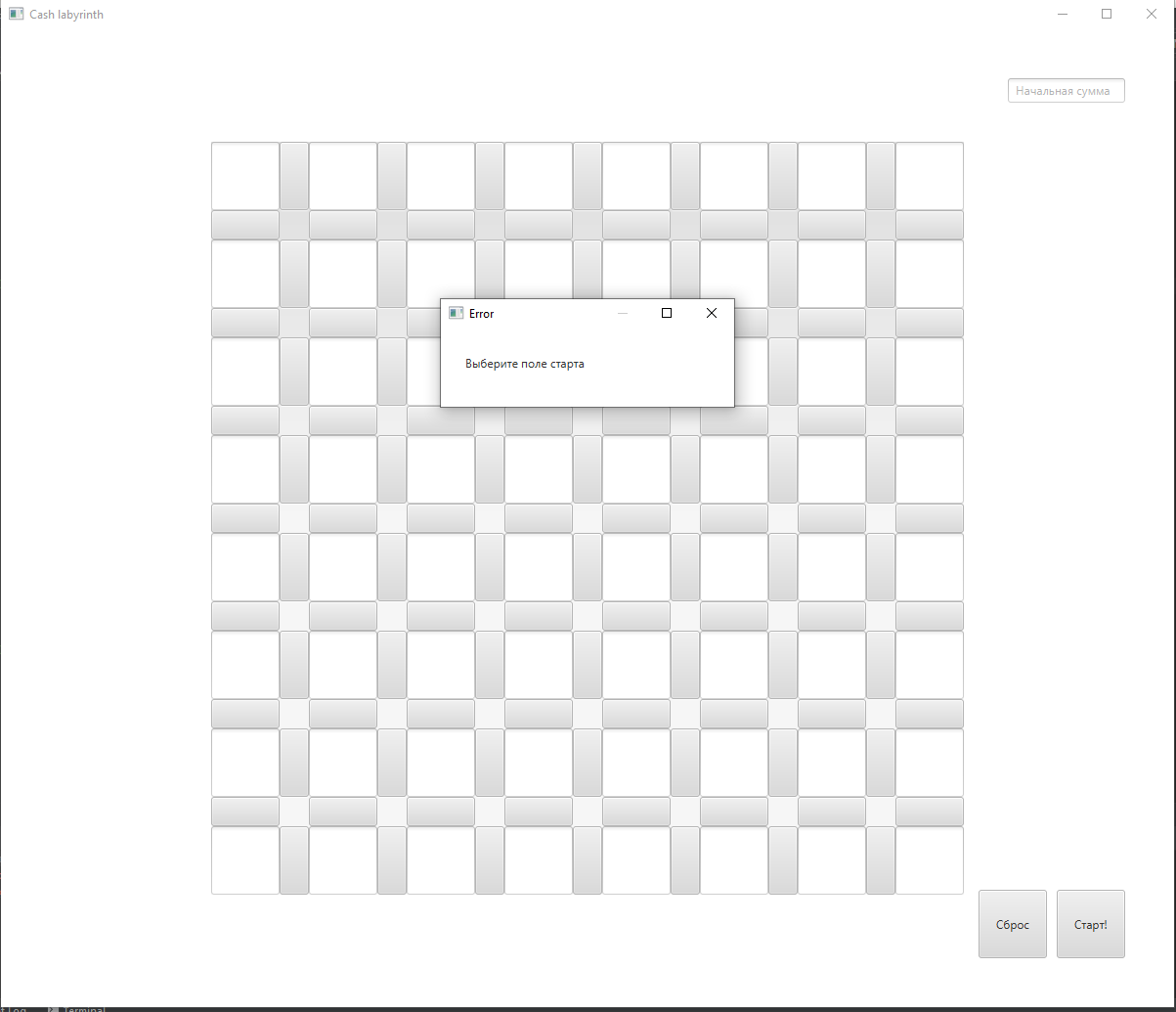
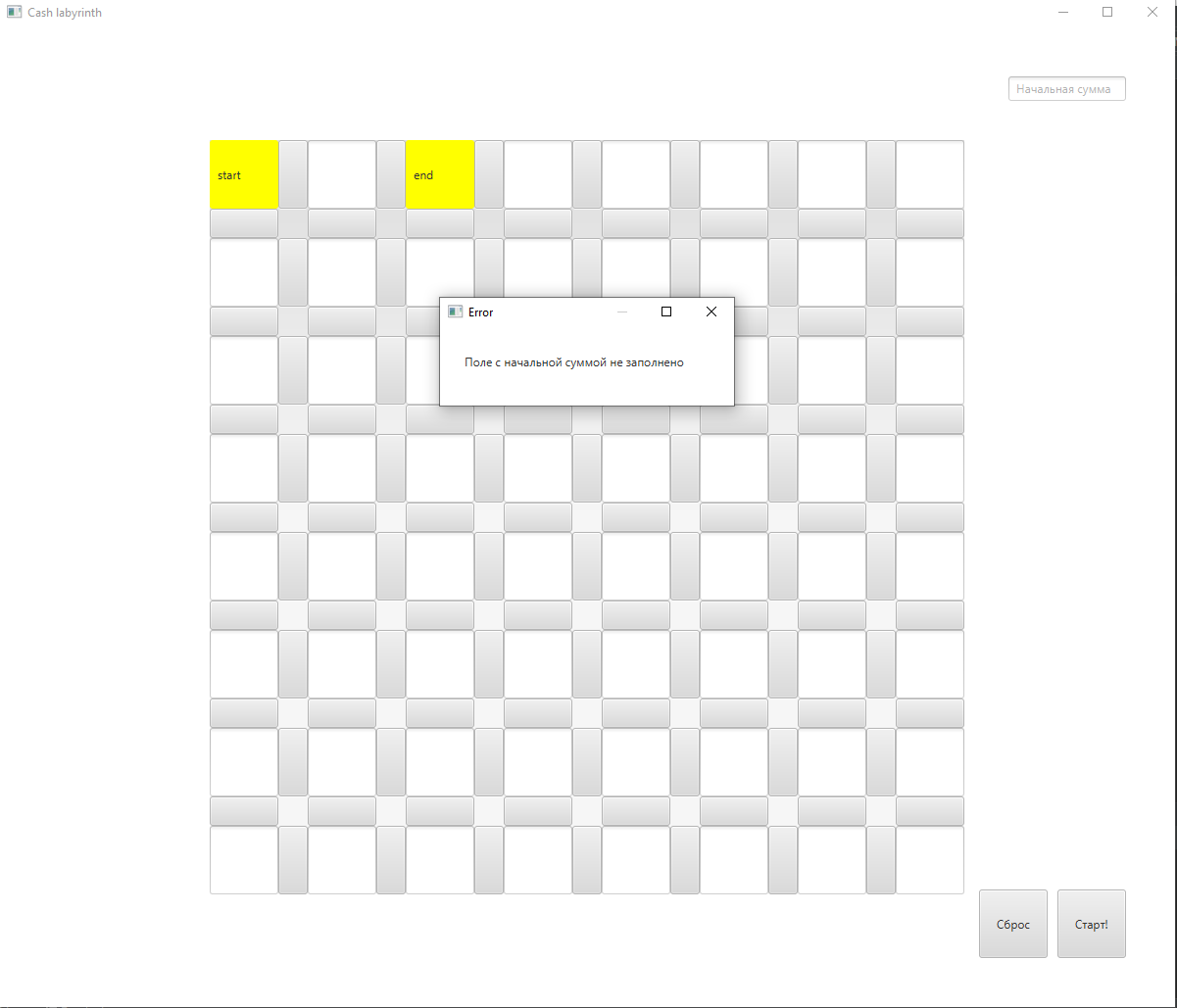


Рис. 1 внешний вид

приложения

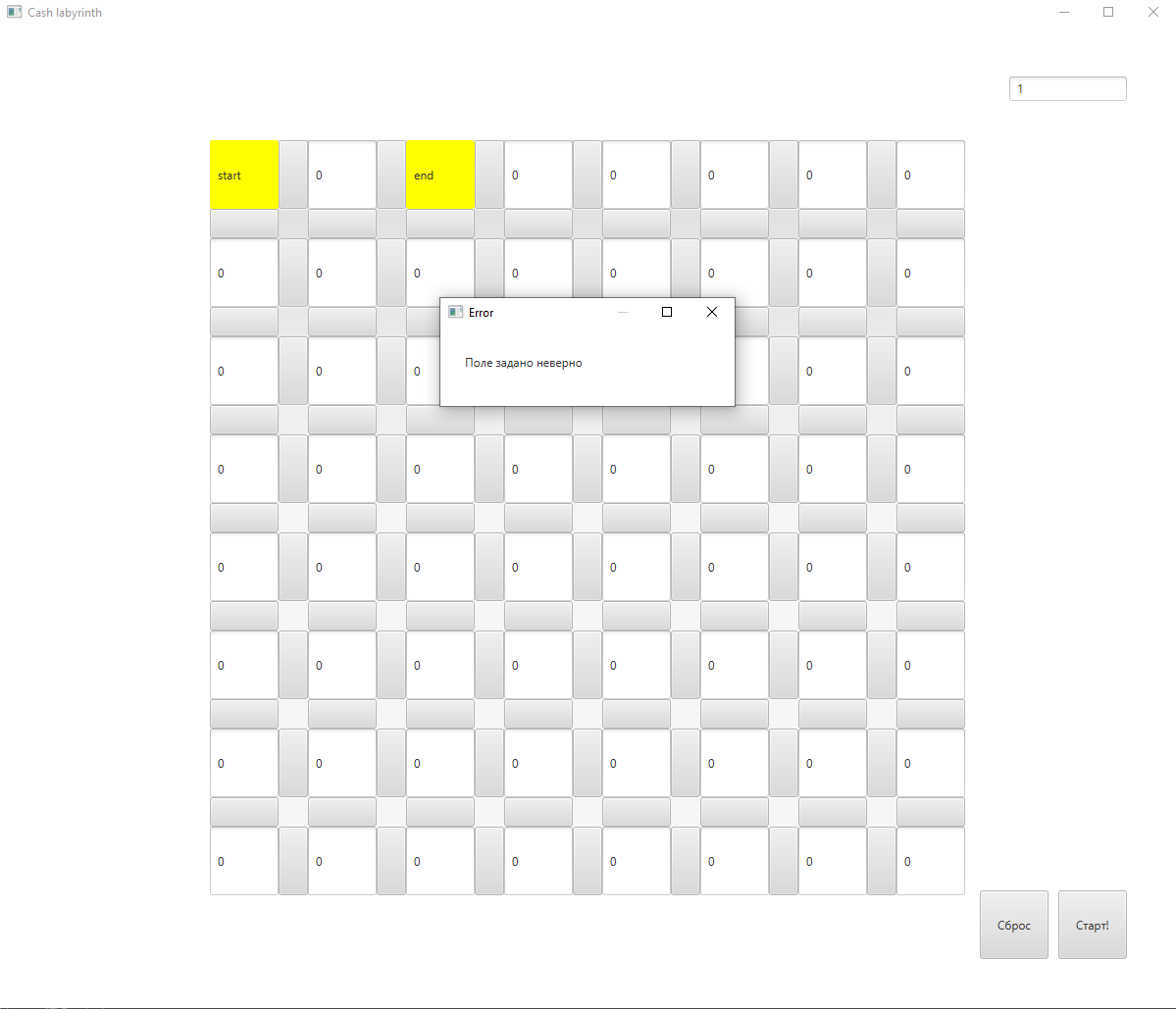
Рис. 2.1 Ошибка при неверном вводе данных пользователем

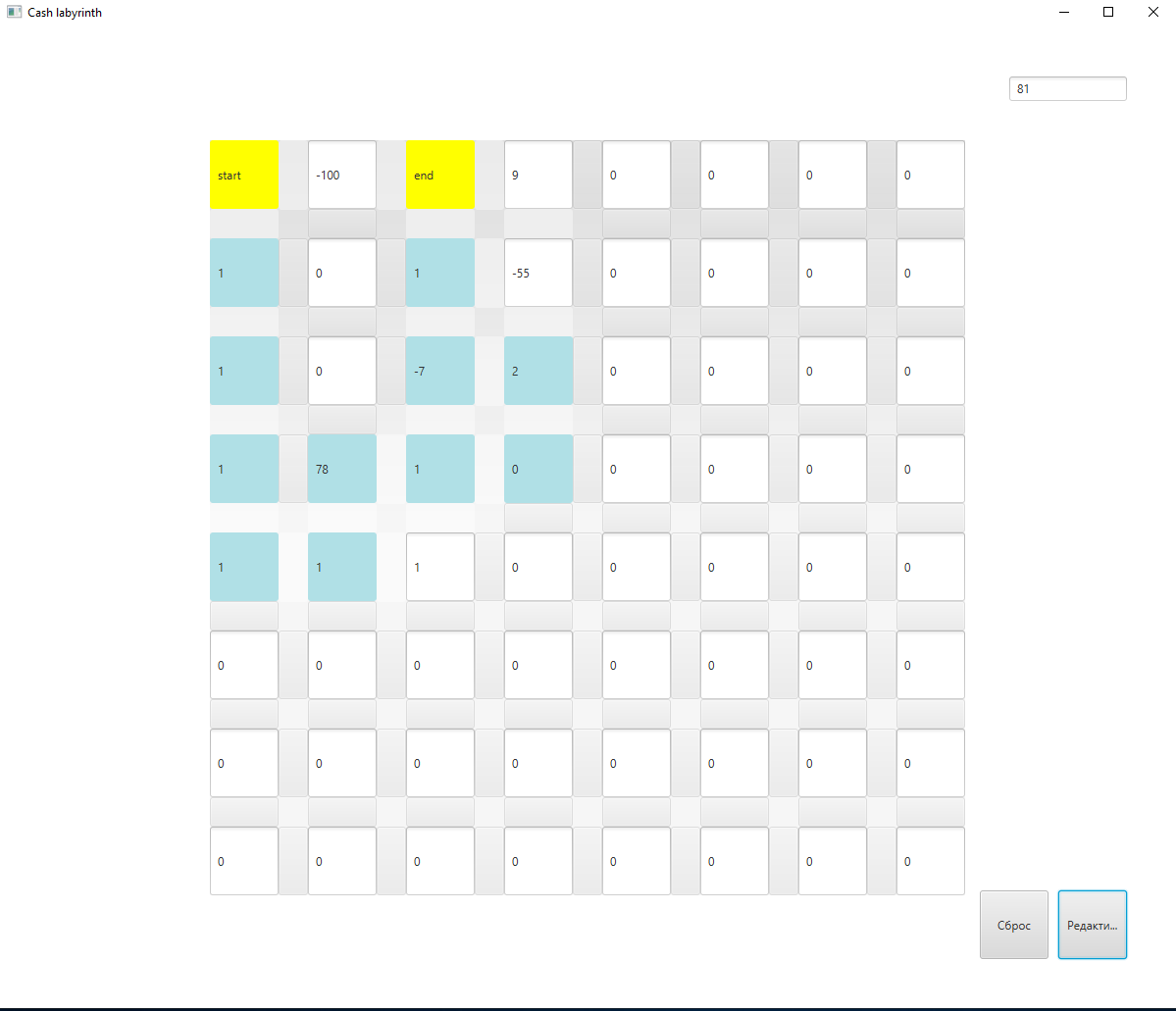
Рис. 2.2 Ошибка при неверном вводе

данных пользователем

Рис. 2.3 Ошибка при неверном вводе данных пользователем

Рис. 3 Игровое поле с выведенным результатом работы алгоритма

Рис. 2.3 Ошибка при неверном вводе данных пользователем

Рис. 3 Игровое поле с выведенным

результатом работы алгоритма