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 генерации случайного лабиринта и его решения.

Выполнил

студент гр. 3530903/80003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Танашкин В.А.

(подпись)

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

(подпись)

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

Санкт-Петербург   
2019

**Оглавление**

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc26066265)

[**МЕТОД РЕШЕНИЯ** 4](#_Toc26066266)

[**ЛИСТИНГ ПРОГРАММЫ** 8](#_Toc26066267)

[**Controller.java** 8](#_Toc26066268)

[**Main.java** 10](#_Toc26066269)

[**Reader.java** 10](#_Toc26066270)

[**Writer.java** 12](#_Toc26066271)

[**Cell.java** 13](#_Toc26066272)

[**Field.java** 14](#_Toc26066273)

[**MazeView.java** 16](#_Toc26066274)

[**TypeOfCell.java** 17](#_Toc26066275)

[**Direction.java** 18](#_Toc26066276)

[**Generator.java** 19](#_Toc26066277)

[**MenuFrame.java** 22](#_Toc26066278)

[**ОШИБКИ И ПРЕДУПРЕЖДЕНИЯ** 32](#_Toc26066279)

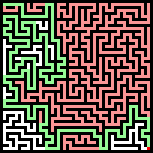
[**ТЕСТЫ** 33](#_Toc26066280)

[**СКРИНШОТЫ ПРОГРАММЫ** 34](#_Toc26066281)

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Заданы размеры и seed для генератора псевдослучайных чисел. Построить лабиринт заданного размера, используя seed для его генерации. Вывести полученный лабиринт на экран. Найти выход в лабиринте и вывести путь на экран.





GitHub репозиторий проекта: <https://github.com/NastyPill/Maze>

# **МЕТОД РЕШЕНИЯ**

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

Содержимое пакета view:

* MenuFrame.java – отвечает за отображение меню, а также за отображение самого лабиринта. В том числе выполняет роль решателя лабиринта.

Cодержимое пакета controller:

* Controller.java – отвечает за связь view и генератора, предает последнему все введенные данные и запускает его работу.
* Reader.java – отвечает за чтение лабиринта из файла и передачу данных на view.
* Writer.java – отвечает за запись лабиринта в файл, получает данные от генератора и записывает их в .txt файл.
* Main.java – запуск приложения.

Содержимое пакета entities:

* Cell.java – класс-представление ячейки лабиринта.
* Field.java – класс-представление поля лабиринта, использует Cell.
* MazeView.java – класс-представление поля лабиринта в виде набора строк, необходим для чтения/записи в файл.
* TypeOfCell.java – класс-перечисление, представляет собой различные типы ячеек поля, например: проход, вход, выход, стена.

Содержимое пакета generator:

* Generator.java – класс, отвечающий за генерацию лабиринта.
* Direction.java – класс-перечисление, содержит константы четырех направлений: север, юг, запад, восток.

Более подробная работа программы:

В главном меню пользователь с помощью панели меню может открыть проводник и с его помощью выбрать файл для чтения, либо записи.

В меню он задает входные данные согласно инструкции изложенной в Help.

Возможно создание своего собственного лабиринта в текстовом файле, а затем чтение из него, легенда символов содержится в Help.

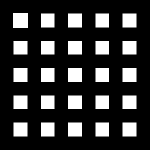
В случае, если в текстовом файле находятся некорректные входные данные пользователю вылетит предупреждение об этом.

За это отвечает метод read(), который проверяет длины строк в файле, а так же, check() из класса Controller, который проверяет входные строки на корректность входных данных, после чего передает их в MenuFrame.

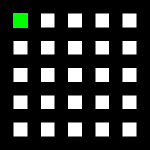
После выбора необходимого текстового файла, либо ввода размера и(или) seed’а, пользователю необходимо нажать на кнопку “Start”, которая открывает следующее окно, с сгенерированным лабиринтом.

После нажатия “Start” генератор начнет свою работу по следующему алгоритму:

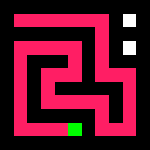
Имеем подобную матрицу:



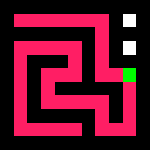
1)Начальная клетка становится посещенной



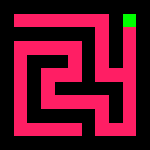
2)Мы идем в случайного соседа, пока не останется ни одного не посещенного



3)Поднимаемся по стеку перемещений вверх, пока не будет хоть один непосещенный сосед, если таковой имеется см. пункт 2.



4)Все клетки посещены, генерация завершена.



За генерацию отвечает метод generate из класса Generator.

В открытом окне, пользователь увидит лабиринт, фиолетовая точка – вход, красная – выход, черные – стены. Пользователь может делать шаги с использованием стрелок на клавиатуре.

За отрисовку лабиринта отвечает метод createMazePanel() класса MenuFrame, который рисует по следующему алгоритму:

На вход приходит массив строк, считанных из файла, программа преобразует из в массив квадратных разноцветных панелей и располагает их в окне.

Размер окна вычисляется как максимальная возможная размерность при текущем кол-ве ячеек по вертикали и горизонтали, но меньшая, чем 90% максимально возможной высоты экрана.

Нажатие кнопки (“s”/”ы”) позволяет пользователю увидеть путь от текущего положения фиолетовой точки до выхода. Решается лабиринт по такому алгоритму:

1. Текущая клетка становится посещенной
2. Если есть непосещенные соседи, идем в случайного соседа
3. Если нет непосещенных соседей, идем вверх по стеку, если стек пуст и выход не найден, то выхода нет, либо он недостижим, если стек не пуст, ищем непосещенную клетку, если она найдена, см пункт 2.
4. Выход найден, рисуем путь.

По своей сути решение лабиринта – обход графа в глубину, где развилки – узлы, а обычные проходы – ребра.

# **ЛИСТИНГ ПРОГРАММЫ**

## **Controller.java**

**package** controller;

**import** **entities.Field**;

**import** **entities.MazeView**;

**import** **generator.Generator**;

**import** **javax.imageio.ImageIO**;

**import** **java.awt.\***;

**import** **java.awt.image.BufferedImage**;

**import** **java.io.File**;

**import** **java.io.IOException**;

**public** **class** **Controller** {

**private** Generator generator;

**private** **int** sizeX;

**private** **int** sizeY;

**private** Long seed;

**private** String fileIn;

**private** String fileOut;

**public** **Controller**() {

fileOut = "";

fileIn = "";

}

**public** **void** **setSeed**(**long** seed) {

**this**.seed = seed;

}

**public** **void** **setSizeX**(**int** sizeX) {

**this**.sizeX = sizeX;

}

**public** **void** **setSizeY**(**int** sizeY) {

**this**.sizeY = sizeY;

}

**public** **void** **setFileIn**(String fileIn) {

**this**.fileIn = fileIn;

}

**public** **void** **setFileOut**(String fileOut) {

**this**.fileOut = fileOut;

}

**private** String[] **check**(Reader reader) **throws** IllegalArgumentException {

**try** {

String[] s = reader.read();

**for** (String str : s) {

**if** (!str.matches("[\\.|\\+|\\\*|0|X]\*")) {

**throw** **new** **IllegalArgumentException**("File is not properly formatted");

}

}

**return** s;

} **catch** (IOException ex) {

**return** **null**;

}

}

**public** MazeView **start**() **throws** IllegalArgumentException {

**try** {

Reader reader;

String[] s;

**if** (!fileIn.isEmpty()) {

reader = **new** Reader(fileIn);

s = check(reader);

**if** (s != **null**) {

**return** **new** **MazeView**(reader.getHeight(), reader.getWidth(), s);

}

}

**if** (seed != **null**) {

System.out.println(seed);

generator = **new** Generator(sizeX, sizeY, seed);

} **else** {

generator = **new** Generator(sizeX, sizeY);

}

generator.generate();

**if** (fileOut.isEmpty()) {

fileOut = getClass().getProtectionDomain().getCodeSource().getLocation().toString().substring(**6**) + "txt.txt";

setFileOut(fileOut);

}

Writer writer = **new** Writer(fileOut);

writer.write(generator.getField().toString());

writer.close();

reader = **new** Reader(fileOut);

String[] s123 = reader.read();

reader.close();

savePic();

**return** **new** **MazeView**(reader.getHeight(), reader.getWidth(), s123);

} **catch** (IOException ex) {

ex.printStackTrace();

}

**return** **null**;

}

**public** **void** **imageMakerInvoke**() {

generator = **new** Generator(sizeX, sizeY, seed);

generator.generate();

ImageMaker imageMaker = **new** ImageMaker(generator.getField().toString().split("\n"));

imageMaker.start();

}

**private** **void** **savePic**() {

System.out.println("start");

BufferedImage image = **new** BufferedImage(sizeY \* **3**, sizeX \* **3**, BufferedImage.TYPE\_INT\_RGB);

Graphics2D graphics2D = image.createGraphics();

Field field = generator.getField();

String[] fieldString = field.toString().split("\n");

**for** (**int** j = **0**; j < fieldString.length; j++) {

**for** (**int** i = **0**; i < fieldString[j].length(); i++) {

**switch** (fieldString[j].charAt(i)) {

**case** '.':

graphics2D.setColor(Color.WHITE);

**break**;

**case** '+':

graphics2D.setColor(Color.BLACK);

**break**;

**case** '\*':

graphics2D.setColor(Color.CYAN);

**break**;

**case** 'X':

graphics2D.setColor(Color.RED);

**break**;

}

graphics2D.fillRect(i \* **3**, j \* **3**, **3**, **3**);

}

}

System.out.println("ok");

File file = **new** File(getClass().getProtectionDomain().getCodeSource().getLocation().toString().substring(**6**) + "img.png");

**try** {

System.out.println("print");

System.out.println(file);

ImageIO.write(image, "png", file);

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

## **Main.java**

**package** controller;

**import** **view.MenuFrame**;

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

MenuFrame frame = **new** MenuFrame();

}

}

## **Reader.java**

**package** controller;

**import** **java.io.BufferedReader**;

**import** **java.io.Closeable**;

**import** **java.io.FileReader**;

**import** **java.io.IOException**;

**public** **class** **Reader** **implements** Closeable{

**private** BufferedReader br;

**private** **int** height;

**private** **int** width;

**private** String file;

**public** **Reader**(String file) **throws** IOException {

**this**.file = file;

br = **new** BufferedReader(**new** FileReader(file));

String tempString = br.readLine();

width = tempString.length();

**while** (tempString != **null**) {

height++;

tempString = br.readLine();

}

}

**public** String[] **read**() **throws** IOException, IllegalArgumentException {

String typeOfCell[] = **new** String[height];

br = **new** BufferedReader(**new** FileReader(file));

String tempString = br.readLine();

String prevString = tempString;

**int** i = **0**;

**while** (tempString != **null**) {

**if**(prevString.length() != tempString.length()) {

**throw** **new** **IllegalArgumentException**("File not properly formatted");

}

typeOfCell[i] = tempString;

prevString = tempString;

tempString = br.readLine();

i++;

}

**return** typeOfCell;

}

**public** **int** **getHeight**() {

**return** height;

}

**public** **int** **getWidth**() {

**return** width;

}

**@Override**

**public** **void** **close**() **throws** IOException {

br.close();

br = **null**;

}

}

## **Writer.java**

**package** controller;

**import** **java.io.\***;

**public** **class** **Writer** **implements** Closeable {

BufferedWriter bw;

**public** **Writer**(String file) **throws** IOException {

File f = **new** File(file);

**if** (!f.exists())

f.createNewFile();

bw = **new** BufferedWriter(**new** FileWriter(file));

}

**public** **void** **write**(String content) **throws** IOException {

bw.write(content);

}

**@Override**

**public** **void** **close**() **throws** IOException {

bw.close();

bw = **null**;

}

}

## **Cell.java**

**package** entities;

**public** **class** **Cell** {

**boolean** visited;

TypeOfCell type;

**int** x;

**int** y;

**public** **Cell**(**int** x, **int** y, TypeOfCell type) {

**this**.type = type;

**this**.x = x;

**this**.y = y;

**this**.visited = **false**;

}

**@Override**

**public** String **toString**() {

**return** type.toString();

}

**public** **int** **getX**() {

**return** x;

}

**public** **void** **setX**(**int** x) {

**this**.x = x;

}

**public** **int** **getY**() {

**return** y;

}

**public** **void** **setY**(**int** y) {

**this**.y = y;

}

**public** **void** **setVisited**(**boolean** visited) {

**this**.visited = visited;

}

**public** **boolean** **isVisited**() {

**return** visited;

}

}

## **Field.java**

**package** entities;

**import** **javafx.util.Pair**;

**public** **class** **Field** {

Cell field[][];

**int** width;

**int** height;

Pair<Integer, Integer> entrance;

**public** **Field**(**int** height, **int** width) {

**this**.width = width;

**this**.height = height;

field = **new** Cell[height][width];

}

**public** **void** **init**() {

**for** (**int** y = **0**; y < height; y++) {

**for** (**int** x = **0**; x < width; x++) {

**this**.initField(x, y);

}

}

}

**@Override**

**public** String **toString**() {

StringBuilder result = **new** StringBuilder();

**for** (**int** i = **0**; i < height; i++) {

**for** (**int** j = **0**; j < width; j++) {

result.append(field[i][j].toString());

}

result.append("\n");

}

**return** result.toString();

}

**private** **void** **initField**(**int** x, **int** y) {

field[y][x] = **new** Cell(x, y, TypeOfCell.WALL);

}

**public** **void** **setWall**(**int** x, **int** y, Boolean isWall) {

**if** (x >= **0** && x < width && y >= **0** && y < height)

**if** (isWall)

field[y][x] = **new** Cell(x, y, TypeOfCell.WALL);

**else**

field[y][x] = **new** Cell(x, y, TypeOfCell.HOLE);

}

**public** Boolean **setEndPoints**(**int** x, **int** y, Boolean isExit) {

**if** (!isExit) {

**if** (x == **1** && y == **1**) {

field[y][x] = **new** Cell(x, y, TypeOfCell.ENTRANCE);

entrance = **new** Pair<>(x, y);

**return** **true**;

}

} **else** {

**if** (x == width - **2** && y == height - **2**) {

field[y][x] = **new** Cell(x, y, TypeOfCell.EXIT);

**return** **true**;

}

}

**return** **false**;

}

**public** Cell **getCell**(**int** x, **int** y) {

**return** field[y][x];

}

**public** Cell[][] **getField**() {

**return** field;

}

}

## **MazeView.java**

**package** entities;

**public** **class** **MazeView** {

**private** **int** height;

**private** **int** width;

**private** String[] field;

**public** **MazeView**(**int** height, **int** width, String[] field) {

**this**.height = height;

**this**.width = width;

**this**.field = field;

}

**public** **int** **getHeight**() {

**return** height;

}

**public** **int** **getWidth**() {

**return** width;

}

**public** String[] **getField**() {

**return** field;

}

}

## **TypeOfCell.java**

**package** entities;

**public** **enum** TypeOfCell {

WALL("+"), EXIT("X"), ENTRANCE("\*"), EMPTY\_CELL("0"), HOLE(".");

String string;

TypeOfCell(String name) {

**this**.string = name;

}

**@Override**

**public** String **toString**() {

**return** string;

}

}

## **Direction.java**

**package** generator;

**import** **javafx.util.Pair**;

**import** **java.util.ArrayList**;

**public** **enum** Direction {

NORTH(**0**, -**1**), WEST(-**1**, **0**), EAST(**1**, **0**), SOUTH(**0**, **1**);

**private** **int** x;

**private** **int** y;

//shifts in x,y for cells i should check

**private** ArrayList<Pair<Integer, Integer>> cellsAround;

Direction(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

**public** Pair<Integer, Integer> getCoordsShift() {

**return** **new** Pair<>(x, y);

}

}

## **Generator.java**

**package** generator;

**import** **entities.Cell**;

**import** **entities.Field**;

**import** **java.util.\***;

**public** **class** **Generator** {

**private** **int** width;

**private** **int** height;

**private** Random random;

**private** Field field;

**private** **long** unvisited;

**public** **Generator**(**int** height, **int** width, Long seed) {

dimensionsCheck(width, height);

init(width, height, seed);

}

**public** **Generator**(**int** width, **int** height) {

dimensionsCheck(width, height);

init(width, height, System.currentTimeMillis());

}

**private** **void** **init**(**int** width, **int** height, Long seed) {

System.out.println(seed);

unvisited = **0**;

field = **new** Field(height, width);

field.init();

**this**.width = width;

**this**.height = height;

random = **new** Random(seed);

}

**private** **void** **dimensionsCheck**(**int** width, **int** height) {

**if** (width < **15** || width > **999999** || height < **15** || height > **999999**) {

**throw** **new** **IllegalArgumentException**("Dimensions should be in 15..3001");

}

}

**private** **void** **generateField**() {

field.init();

**for** (**int** i = **1**; i < field.getField().length; i++) {

**for** (**int** j = **1**; j < field.getField().length; j++) {

**if**(i % **2** == **1** && j % **2** == **1** && i != field.getField().length - **1** && j != field.getField().length - **1**) {

field.setWall(j, i, **false**);

unvisited++;

}

}

}

}

**private** **void** **generateMaze**() {

Cell[][] cells = field.getField();

Cell currentCell = cells[**1**][**1**];

Cell neighbourCell;

Stack<Cell> stack = **new** Stack<>();

List<Cell> cellsUnvisited = getUnvisitedCells();

**do** {

cellsUnvisited.remove(currentCell);

List<Cell> neighbours = getNeighbours(currentCell);

**if** (neighbours.size() != **0**) {

**int** randNum = random.nextInt(neighbours.size());

neighbourCell = neighbours.get(randNum);

stack.push(currentCell);

removeWall(currentCell, neighbourCell);

currentCell = neighbourCell;

currentCell.setVisited(**true**);

unvisited--;

} **else** **if** (stack.size() > **0**) {

currentCell = stack.pop();

} **else** {

**int** randNum = random.nextInt(cellsUnvisited.size() - **1**);

currentCell = cellsUnvisited.get(randNum);

}

}

**while** (unvisited > **0**);

}

**private** List<Cell> **getUnvisitedCells**() {

Cell[][] cells = field.getField();

List<Cell> cellArrayList = **new** LinkedList<>();

**for** (**int** i = **0**; i < cells.length; i++) {

**for** (**int** j = **0**; j < cells.length; j++) {

**if**(cells[i][j].toString().equals(".") && !cells[i][j].isVisited()) {

cellArrayList.add(cells[i][j]);

}

}

}

**return** cellArrayList;

}

**private** List<Cell> **getNeighbours**(Cell cell) {

List<Cell> neighbours = **new** ArrayList<>();

**for**(Direction direction : Direction.values()){//для каждого направдения

**if**(cell.getX() + direction.getCoordsShift().getKey() \* **2** > **0** &&

cell.getX() + direction.getCoordsShift().getKey() \* **2** < width &&

cell.getY() + direction.getCoordsShift().getValue() \* **2** > **0** &&

cell.getY() + direction.getCoordsShift().getValue() \* **2** < height){

Cell mazeCellCurrent = field.getCell(cell.getX() + direction.getCoordsShift().getKey() \* **2**, cell.getY() + direction.getCoordsShift().getValue() \* **2**);

**if**(mazeCellCurrent.toString().equals(".") && !mazeCellCurrent.isVisited()){

neighbours.add(mazeCellCurrent); //записать в массив;

}

}

}

**return** neighbours;

}

**private** **void** **removeWall**(Cell currentCell, Cell neighbourCell) {

**int** xDiff = neighbourCell.getX() - currentCell.getX();

**int** yDiff = neighbourCell.getY() - currentCell.getY();

**int** addX, addY;

addX = (xDiff != **0**) ? (xDiff / Math.abs(xDiff)) : **0**;

addY = (yDiff != **0**) ? (yDiff / Math.abs(yDiff)) : **0**;

**int** newX = currentCell.getX() + addX; //координаты стенки

**int** newY = currentCell.getY() + addY;

field.setWall(newX, newY, **false**);

field.getCell(newX, newY).setVisited(**true**);

}

**public** **void** **generate**() {

generateField();

generateMaze();

field.setEndPoints(**1**, **1**, **false**);

field.setEndPoints(field.getField().length - **2**, field.getField().length - **2**, **true**);

}

**public** Field **getField**() {

**return** field;

}

}

## **MenuFrame.java**

**package** view;

**import** **controller.Controller**;

**import** **entities.Cell**;

**import** **entities.MazeView**;

**import** **entities.TypeOfCell**;

**import** **generator.Direction**;

**import** **javafx.util.Pair**;

**import** **javax.imageio.ImageIO**;

**import** **javax.swing.\***;

**import** **javax.swing.border.BevelBorder**;

**import** **java.awt.\***;

**import** **java.awt.event.KeyAdapter**;

**import** **java.awt.event.KeyEvent**;

**import** **java.awt.event.MouseAdapter**;

**import** **java.awt.event.MouseEvent**;

**import** **java.awt.image.BufferedImage**;

**import** **java.io.BufferedReader**;

**import** **java.io.File**;

**import** **java.io.FileReader**;

**import** **java.io.IOException**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**import** **java.util.Random**;

**import** **java.util.Stack**;

**public** **class** **MenuFrame** **extends** JFrame {

**private** JPanel panel;

**private** JTextArea sizeArea;

**private** JTextArea seedArea;

**private** JMenuBar bar;

**private** JLabel label;

**private** JButton startButton;

**private** Controller controller;

**private** JFileChooser fileChooser;

**private** Boolean haveFile = **false**;

**private** JPanel cells[][];

**private** Pair<Integer, Integer> exit;

**private** Pair<Integer, Integer> entrance;

**private** **boolean** haveHole;

**private** Cell[][] field;

**private** String[] typeOfCell;

**public** **MenuFrame**() {

createMenuPanel();

JMenu fileMenu = **new** JMenu("File");

JMenuItem fileToItem = **new** JMenuItem("Select file to write");

JMenuItem fileFromItem = **new** JMenuItem("Select file to read");

fileMenu.add(fileFromItem);

fileMenu.add(fileToItem);

fileFromItem.addActionListener(actionEvent -> {

fileChooser = **new** JFileChooser();

fileChooser.setCurrentDirectory(**new** File("."));

fileChooser.showOpenDialog(panel);

**if** (fileChooser.getSelectedFile() != **null** && fileChooser.getSelectedFile().getName().contains(".txt")) {

controller.setFileIn(fileChooser.getSelectedFile().getAbsolutePath());

haveFile = **true**;

**try** {

controller.start();

} **catch** (IllegalArgumentException ex) {

haveFile = **false**;

controller.setFileIn("");

JOptionPane.showMessageDialog(**null**, ex.getMessage());

}

} **else** {

**if** (fileChooser.getSelectedFile() == **null**) {

JOptionPane.showMessageDialog(**null**, "File has not been chosen");

} **else** {

JOptionPane.showMessageDialog(**null**, "It is not txt file: " + fileChooser.getSelectedFile().getAbsolutePath());

}

}

});

fileToItem.addActionListener(actionEvent -> {

fileChooser = **new** JFileChooser();

fileChooser.setCurrentDirectory(**new** File("."));

fileChooser.showOpenDialog(panel);

**if** (fileChooser.getSelectedFile() != **null** && fileChooser.getSelectedFile().getName().contains(".txt"))

controller.setFileOut(fileChooser.getSelectedFile().getAbsolutePath());

});

JMenu settingsMenu = **new** JMenu("Settings");

JMenuItem language = **new** JMenuItem("Language");

JMenuItem colorScheme = **new** JMenuItem("Color scheme");

settingsMenu.add(colorScheme);

settingsMenu.add(language);

//settingsMenu.add(solveMenuItem);

JMenuItem helpMenuItem = **new** JMenu("Help");

JMenu exitMenu = **new** JMenu("Exit");

JMenuItem sure = **new** JMenuItem("Are you sure?");

JMenuItem yes = **new** JMenuItem("YES");

JMenuItem no = **new** JMenuItem("NO");

exitMenu.add(sure);

exitMenu.add(yes);

exitMenu.add(no);

sure.setEnabled(**false**);

helpMenuItem.addMouseListener(**new** MouseAdapter() {

**@Override**

**public** **void** **mouseClicked**(MouseEvent e) {

**try** {

String res = getClass().getClassLoader().getResource("help.txt").toString().substring(**6**);

BufferedReader br = **new** BufferedReader(**new** FileReader(res));

String s = br.readLine();

StringBuilder sb = **new** StringBuilder();

**while** (s != **null**) {

sb.append(s);

sb.append("\n");

s = br.readLine();

}

JOptionPane.showMessageDialog(**null**, sb.toString());

} **catch** (IOException ex) {

JOptionPane.showMessageDialog(**null**, ex.getMessage());

}

}

});

yes.addActionListener(actionEvent -> {

System.exit(**0**);

});

bar = **new** JMenuBar();

bar.add(fileMenu);

bar.add(settingsMenu);

bar.add(helpMenuItem);

bar.add(exitMenu);

bar.setBackground(**new** Color(**160**, **160**, **160**));

bar.setBorder(BorderFactory.createLineBorder(**new** Color(**0**), **0**));

**this**.setSize(**300**, **300**);

**this**.setContentPane(panel);

**this**.setResizable(**false**);

**this**.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

**this**.setJMenuBar(bar);

**this**.setBackground(**new** Color(**0**));

**this**.setVisible(**true**);

}

**public** **void** **createMenuPanel**() {

controller = **new** Controller();

startButton = **new** JButton("Start");

label = **new** JLabel("MAZE");

label.setFont(**new** Font("Arial", Font.BOLD, **74**));

panel = **new** JPanel();

seedArea = **new** JTextArea("Enter seed here");

seedArea.addMouseListener(**new** MouseAdapter() {

**@Override**

**public** **void** **mouseClicked**(MouseEvent e) {

seedArea.setText("");

}

});

sizeArea = **new** JTextArea("Enter size (number in 15..200)");

sizeArea.addMouseListener(**new** MouseAdapter() {

**@Override**

**public** **void** **mouseClicked**(MouseEvent e) {

sizeArea.setText("");

}

});

panel.setLayout(**null**);

panel.add(sizeArea);

panel.add(startButton);

panel.add(seedArea);

panel.add(label);

startButton.setBounds(**50**, **165**, **180**, **60**);

sizeArea.setBounds(**50**, **130**, **180**, **20**);

sizeArea.setBorder(**new** BevelBorder(**1**));

seedArea.setBounds(**50**, **100**, **180**, **20**);

seedArea.setBorder(**new** BevelBorder(**1**));

label.setBounds(**35**, **30**, **250**, **60**);

startButton.setBorder(**new** BevelBorder(**0**));

startButton.setFont(**new** Font("Arial", Font.BOLD, **50**));

startButton.setForeground(**new** Color(**205**, **205**, **205**));

startButton.setBackground(**new** Color(**60**, **60**, **60**));

sizeArea.setBackground(**new** Color(**60**, **60**, **60**));

sizeArea.setForeground(**new** Color(**205**, **205**, **205**));

seedArea.setBackground(**new** Color(**60**, **60**, **60**));

seedArea.setForeground(**new** Color(**205**, **205**, **205**));

label.setForeground(**new** Color(**205**, **205**, **205**));

panel.setBackground(**new** Color(**60**, **60**, **60**));

JButton picButton = **new** JButton("Pic");

picButton.setBorder(**new** BevelBorder(**0**));

picButton.setFont(**new** Font("Arial", Font.BOLD, **20**));

picButton.setForeground(**new** Color(**205**, **205**, **205**));

picButton.setBackground(**new** Color(**60**, **60**, **60**));

picButton.setBounds(**0**, **165**, **40**, **50**);

panel.add(picButton);

picButton.addMouseListener(**new** MouseAdapter() {

**@Override**

**public** **void** **mouseClicked**(MouseEvent e) {

String s = sizeArea.getText();

**int** dim = -**1**;

**if** (s.matches("\\d{2,6}")) {

dim = Integer.parseInt(s);

**if** (dim % **2** == **0**) {

dim++;

}

} **else** {

sizeArea.setText("Illegal format");

**return**;

}

controller.setSizeX(dim);

controller.setSizeY(dim);

**if** (seedArea.getText().equals("Illegal format!") || seedArea.getText().equals("Enter seed here")) {

controller.setSeed(System.currentTimeMillis());

} **else** {

**try** {

controller.setSeed(Long.parseLong(s));

} **catch** (NumberFormatException ex) {

seedArea.setText("Illegal format!");

}

}

controller.imageMakerInvoke();

}

});

startButton.addMouseListener(**new** MouseAdapter() {

**@Override**

**public** **void** **mouseClicked**(MouseEvent e) {

**if** (haveFile) {

**try** {

createMazePanel(controller.start());

} **catch** (IllegalArgumentException ex) {

controller.setFileIn("");

haveFile = **false**;

JOptionPane.showMessageDialog(**null**, ex.getMessage());

}

} **else** {

String s = sizeArea.getText();

**int** dim = -**1**;

**if** (s.matches("\\d{2,3}")) {

dim = Integer.parseInt(s);

**if** (dim > **202**) {

sizeArea.setText("Size should be 15-200");

} **else** {

**if** (dim % **2** == **0**) {

dim++;

}

**if** (dim < **15** || dim > **202**) {

sizeArea.setText("Size should be 15-200");

**return**;

}

}

} **else** {

sizeArea.setText("Illegal format");

**return**;

}

controller.setSizeX(dim);

controller.setSizeY(dim);

**if** (seedArea.getText().equals("Illegal format!") || seedArea.getText().equals("Enter seed here")) {

controller.setSeed(System.currentTimeMillis());

} **else** {

**try** {

controller.setSeed(Long.parseLong(seedArea.getText()));

} **catch** (NumberFormatException ex) {

seedArea.setText("Illegal format!");

}

}

**try** {

createMazePanel(controller.start());

} **catch** (IllegalArgumentException ex) {

JOptionPane.showMessageDialog(**null**, ex.getMessage());

}

}

}

});

**this**.getContentPane().requestFocus();

}

**public** **void** **createMazePanel**(MazeView mazeView) {

**int** rows = mazeView.getHeight();

**int** cols = mazeView.getWidth();

typeOfCell = mazeView.getField();

entrance = **null**;

exit = **null**;

cells = **new** JPanel[rows][cols];

JPanel panel = **new** JPanel();

panel.setLayout(**new** GridLayout(rows, cols));

**for** (**int** i = **0**; i < rows; i++) {

**for** (**int** j = **0**; j < cols; j++) {

cells[i][j] = **new** JPanel();

**switch** (typeOfCell[i].charAt(j)) {

**case** '+':

cells[i][j].setBackground(Color.BLACK);

**break**;

**case** '.':

cells[i][j].setBackground(Color.WHITE);

haveHole = **true**;

**break**;

**case** '0':

cells[i][j].setBackground(Color.GREEN);

**break**;

**case** 'X': {

cells[i][j].setBackground(Color.RED);

**if** (exit == **null**)

exit = **new** Pair<>(i, j);

**else** {

**throw** **new** **IllegalArgumentException**("There are two exits");

}

}

**break**;

**case** '\*': {

cells[i][j].setBackground(Color.MAGENTA);

**if** (entrance == **null**)

entrance = **new** Pair<>(i, j);

**else** {

**throw** **new** **IllegalArgumentException**("There are two entrances");

}

}

**break**;

}

panel.add(cells[i][j]);

}

}

**if** (entrance == **null**) {

**throw** **new** **IllegalArgumentException**("ERROR: There is no entrance in the maze");

}

**if** (exit == **null**) {

**throw** **new** **IllegalArgumentException**("ERROR: There is no exit in the maze");

}

**if** (!haveHole) {

**throw** **new** **IllegalArgumentException**("ERROR: There is no holes in the maze");

}

**int** multiplier = **100**;

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

**while** (cols \* multiplier > screenSize.getHeight() \* **0.9**) {

multiplier--;

}

**this**.setSize(**new** Dimension(cols \* multiplier + **16**, rows \* multiplier + **60**));

**this**.setBackground(**new** Color(**0**));

**this**.setResizable(**false**);

**this**.setVisible(**true**);

**this**.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

panel.setFocusable(**true**);

panel.addKeyListener(**new** KeyAdapter() {

**@Override**

**public** **void** **keyPressed**(KeyEvent e) {

**int** x = entrance.getValue();

**int** y = entrance.getKey();

**int** tempX = x;

**int** tempY = y;

**switch** (e.getKeyCode()) {

**case** KeyEvent.VK\_S: {

field = **new** Cell[cells.length][cells.length];

**for** (**int** i = **0**; i < cells.length; i++) {

**for** (**int** j = **0**; j < cells.length; j++) {

**switch** (typeOfCell[i].charAt(j)) {

**case** '+':

field[i][j] = **new** Cell(i, j, TypeOfCell.WALL);

**break**;

**case** '.':

field[i][j] = **new** Cell(i, j, TypeOfCell.HOLE);

**break**;

**case** '0':

field[i][j] = **new** Cell(i, j, TypeOfCell.EMPTY\_CELL);

**break**;

**case** 'X':

field[i][j] = **new** Cell(i, j, TypeOfCell.EXIT);

**break**;

**case** '\*':

field[i][j] = **new** Cell(i, j, TypeOfCell.ENTRANCE);

**break**;

}

}

}

solveMaze();

savePic();

}

**break**;

**case** KeyEvent.VK\_UP: {

entrance = **new** Pair<>(y - **1**, x);

y = entrance.getKey();

}

**break**;

**case** KeyEvent.VK\_DOWN: {

entrance = **new** Pair<>(y + **1**, x);

y = entrance.getKey();

}

**break**;

**case** KeyEvent.VK\_LEFT: {

entrance = **new** Pair<>(y, x - **1**);

x = entrance.getValue();

}

**break**;

**case** KeyEvent.VK\_RIGHT: {

entrance = **new** Pair<>(y, x + **1**);

x = entrance.getValue();

}

**break**;

}

**if** (y < rows && y >= **0** && x < cols && x >= **0** && !cells[y][x].getBackground().equals(Color.BLACK)) {

cells[tempY][tempX].setBackground(**new** Color(**157**, **157**, **157**));

cells[y][x].setBackground(Color.MAGENTA);

} **else** {

y = tempY;

x = tempX;

entrance = **new** Pair<>(y, x);

}

**if** (exit.getKey() == entrance.getKey() && exit.getValue() == entrance.getValue()) {

**int** result = JOptionPane.showConfirmDialog(**null**, "You win!\n EXIT?", "YOU WIN!", JOptionPane.YES\_NO\_OPTION);

**if** (result == JOptionPane.YES\_OPTION) {

System.exit(**0**);

} **else** {

String s = "PLEASE CLICK YES!";

**int** ctr = **0**;

StringBuilder sb = **new** StringBuilder(s);

**while** (result != JOptionPane.YES\_OPTION) {

ctr++;

result = JOptionPane.showConfirmDialog(**null**, sb.toString(), "YOU WIN!", JOptionPane.YES\_NO\_OPTION);

**if** (ctr % **5** == **0**) {

sb.append("\n");

} **else** {

sb.append(" ");

}

sb.append(s);

}

System.exit(**0**);

}

}

}

});

panel.setBackground(**new** Color(**0**));

**this**.setContentPane(panel);

panel.requestFocus();

}

**private** **void** **savePic**() {

System.out.println("start");

BufferedImage image = **new** BufferedImage(cells.length \* **3**, cells.length \* **3**, BufferedImage.TYPE\_INT\_RGB);

Graphics2D graphics2D = image.createGraphics();

**for** (**int** j = **0**; j < cells.length; j++) {

**for** (**int** i = **0**; i < cells.length; i++) {

graphics2D.setColor(cells[j][i].getBackground());

graphics2D.fillRect(i \* **3**, j \* **3**, **3**, **3**);

}

}

System.out.println("ok");

File file = **new** File(getClass().getProtectionDomain().getCodeSource().getLocation().toString().substring(**6**) + "solvedImg.png");

**try** {

System.out.println("print");

System.out.println(file);

ImageIO.write(image, "png", file);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **void** **solveMaze**() {

Random random = **new** Random();

Cell currentCell = field[entrance.getKey()][entrance.getValue()];

Cell neighbourCell;

Stack<Cell> stack = **new** Stack<>();

**do** {

java.util.List<Cell> neighbours = getNeighbours(currentCell);

**if** (neighbours.size() != **0**) {

cells[currentCell.getX()][currentCell.getY()].setBackground(**new** Color(**150**, **250**, **150**));

**int** randNum = random.nextInt(neighbours.size());

neighbourCell = neighbours.get(randNum);

stack.push(currentCell);

currentCell = neighbourCell;

currentCell.setVisited(**true**);

} **else** **if** (stack.size() > **0**) {

cells[currentCell.getX()][currentCell.getY()].setBackground(**new** Color(**250**, **150**, **150**));

currentCell = stack.pop();

} **else** {

JOptionPane.showMessageDialog(**null**, "There is no exit in the maze");

**return**;

}

}

**while** (currentCell.getX() != exit.getKey() || currentCell.getY() != exit.getValue());

}

**private** List<Cell> **getNeighbours**(Cell cell) {

List<Cell> neighbours = **new** ArrayList<>();

**for** (Direction direction : Direction.values()) {//для каждого направдения

**if** (cell.getX() + direction.getCoordsShift().getKey() > **0** &&

cell.getX() + direction.getCoordsShift().getKey() < field.length &&

cell.getY() + direction.getCoordsShift().getValue() > **0** &&

cell.getY() + direction.getCoordsShift().getValue() < field.length) {

Cell mazeCellCurrent = field[cell.getX() + direction.getCoordsShift().getKey()][cell.getY() + direction.getCoordsShift().getValue()];

**if** ((mazeCellCurrent.toString().equals("X") || mazeCellCurrent.toString().equals(".")) && !mazeCellCurrent.isVisited()) {

neighbours.add(mazeCellCurrent); //записать в массив;

}

}

}

**return** neighbours;

}

}

# **ОШИБКИ И ПРЕДУПРЕЖДЕНИЯ**

Все “ошибки” являются наследниками класса Exception. Они обрабатываются программой и пользователь видит только всплывающие окна, либо текст в полях ввода.

|  |  |
| --- | --- |
| Код ошибки | Описание |
| 1 | File has not been chosen; Файл не выбран. |
| 2 | It is not txt file %filename%; Это не текстовый файл %файл% |
| 3 | Illegal format; Неверный формат ввода |
| 4 | Size should be in x-y; размер должен быть в пределах от х до у |
| 5 | There are two exits; В лабиринте два выхода |
| 6 | There are two entrances; В лабиринте два входа |
| 7 | There is no entrance/exit; В лабиринте нет входа/выхода |
| 8 | There are no holes; В лабиринте нет проходов |
| 9 | File not properly formatted; Файл не отформатирован должным образом |

# **ТЕСТЫ**

Все функции из класса MazeTest тестируют методы класса Generator.

**import** **generator.Generator**;

**import** **org.junit.jupiter.api.Test**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**public** **class** **MazeTest** {

**@Test**

**public** **void** **test**() {

//проверка невалидных значений высоты и ширины

assertThrows(IllegalArgumentException.class, **this**::generatorTest);

//проверка валидных значений высоты и ширины

assertNotNull(generatorNNTest());

Generator g1 = generatorSeedWorksTest();

g1.generate();

Generator g2 = generatorSeedWorksTest();

g2.generate();

//проверка лабиринтов на "одинаковость" при генерации с одним seed'ом

assertEquals(g1.getField().toString(), g2.getField().toString());

g1 = generatorSeedWorksTest();

g1.generate();

g2 = generatorSeedNotWorksTest();

g2.generate();

//проверка лабиринтов на разность при генерации с разными seed'ами

assertNotEquals(g1.getField().toString(), g2.getField().toString());

}

**private** Generator **generatorTest**() {

**return** **new** **Generator**(**99999999**, **99999999**, **123**l);

}

**private** Generator **generatorNNTest**() {

**return** **new** **Generator**(**20**, **20**);

}

**private** Generator **generatorSeedWorksTest**() {

**return** **new** **Generator**(**50**, **50**, **50**l);

}

**private** Generator **generatorSeedNotWorksTest**() {

**return** **new** **Generator**(**50**, **50**, **51**l);

}

}

# **СКРИНШОТЫ ПРОГРАММЫ**

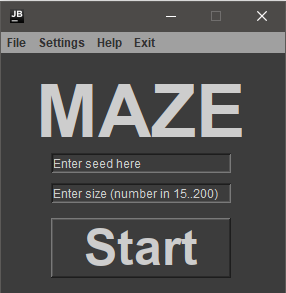


Рис.1 Главное меню

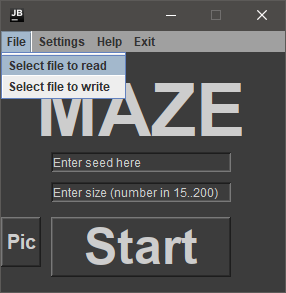


Рис.2.1 Меню File

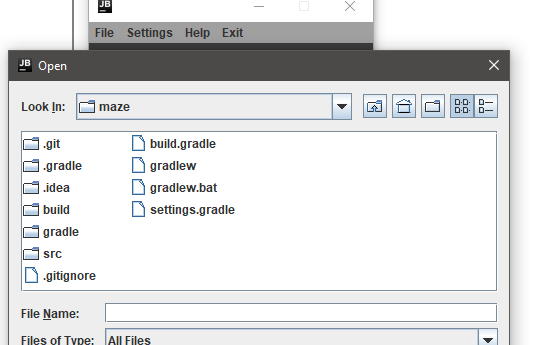


Рис. 2.2 Проводник

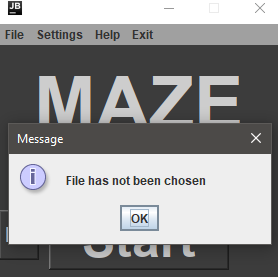


Рис. 3 Ошибка: “Файл не выбран”

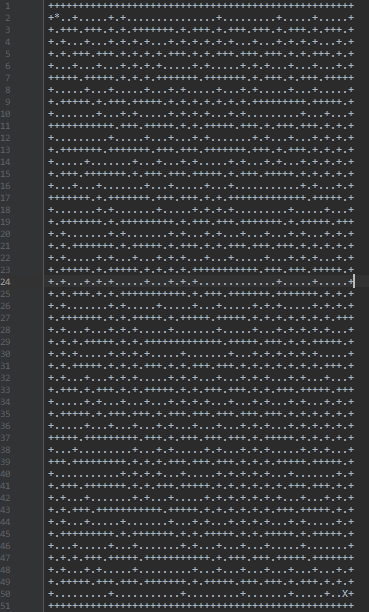


Рис. 4. Текстовое представление лабиринта



Рис. 5 Сгенерированный лабиринт

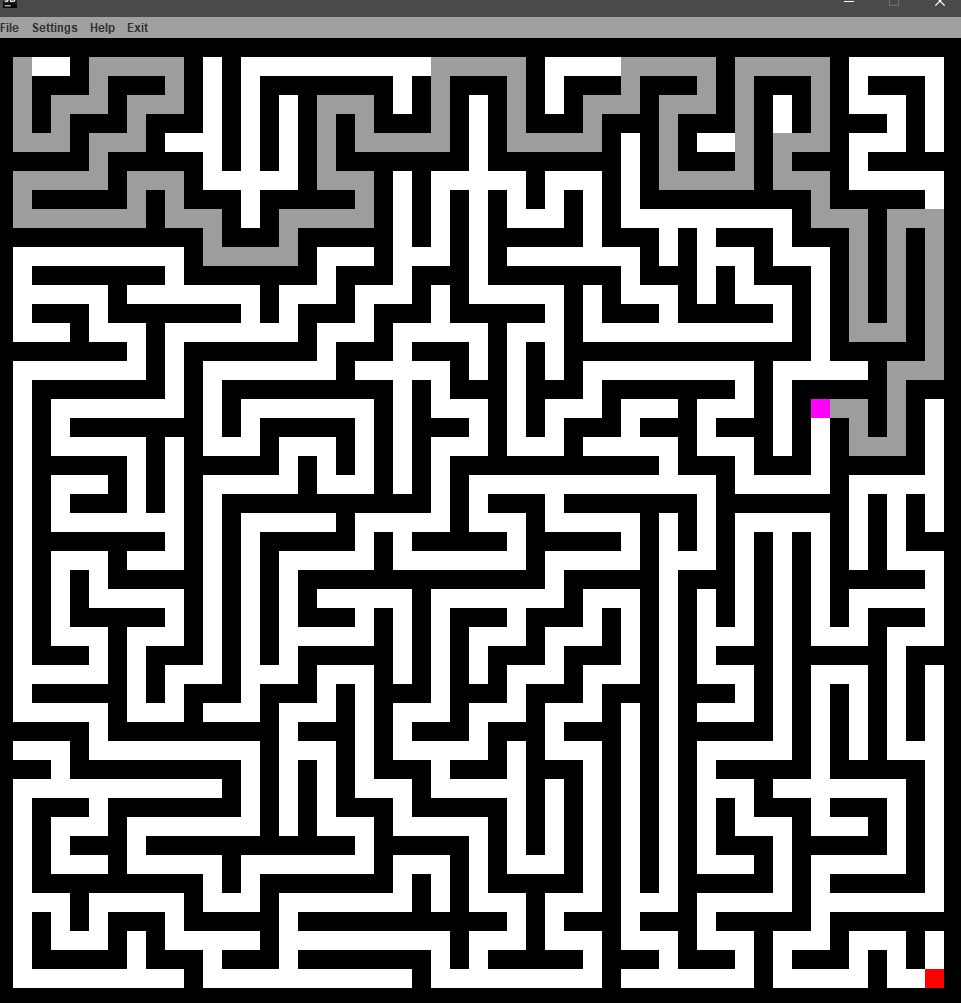
Рис. 6 Часть пути пройдена

Рис. 7 Работа решателя

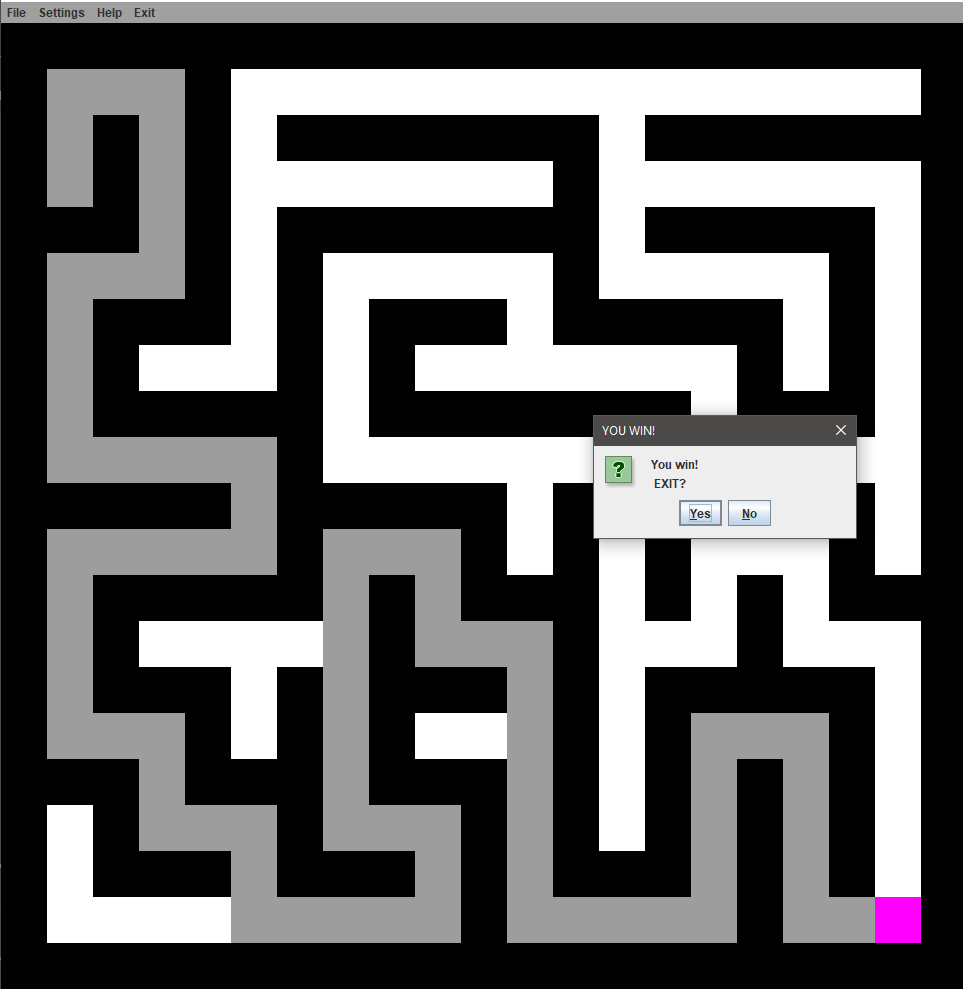


Рис. 8 Лабиринт пройден

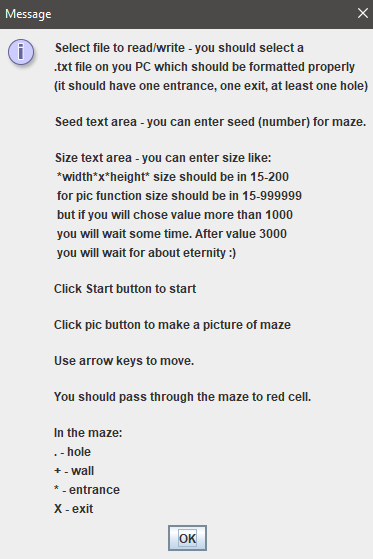


Рис. 9 Help

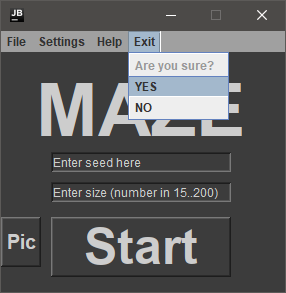


Рис. 10 Меню Exit.