МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Проект лабиринт

Студент гр. 8303 _____ Пушпышев А.И.

Студент гр. 8303 _____ Данилов А.В.

Студент гр. 8303 _____ Стукалев А.И.

Руководитель _____ Фиалковский М.С.

Санкт-Петербург

2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Пушпышев А.И. группы 8303

Студент Данилов А.В. группы 8303

Студент Стукалев А.И, группы 8303

Тема практики: проект Лабиринт

Задание на практику:

Генерируется декартовый лабиринт с закрашенным ячейками, изображение которого визуализировано. Пользователь выбирает точку входа. После нажатия на кнопку “Старт” появляется найденный путь. После этого визуализируется самый короткий найденный путь. Есть возможность пошагово пройти по шагам алгоритма нахождения пути. При невозможности найти выход из лабиринта пользователь получает уведомление. Просмотренные во время работы алгоритма ячейки необходимо выделить полупрозрачным цветом.

Алгоритм: A-star.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 07.07.2020

Дата защиты отчета: 00.07.2020

Студент	_____	Пушпышев А.И.
Студент	_____	Данилов А.В.
Студент	_____	Стукалев А.И.
Руководитель	_____	Фиалковский М.С.

АННОТАЦИЯ

Целью работы является получения навыков работы с такой парадигмой программирования, как объектно-ориентированное программирование. Для получения данных знаний выполняется один из вариантов мини-проекта. В процессе выполнения мини-проекта необходимо реализовать графический интерфейс к данной задаче, организовать ввод и вывод данных с его помощь, реализовать сам алгоритм, научиться работать в команде. В данной работе в качестве мини-проекта выступает генерация декартового лабиринта с расчётом минимального пути от точки входа к точке выхода с возможностями пользователя задать размеры лабиринта самостоятельно, выбрать конкретные точки для входа и выхода для лабиринта. Также при разработке выполняется написание тестирования, для проверки корректности алгоритма.

SUMMARY

The aim of the work is to gain skills in working with such a programming paradigm as object-oriented programming. To obtain this knowledge, one of the mini-project options is performed. In the process of implementing a mini-project, it is necessary to implement a graphical interface to this task, organize data input and output with its help, implement the algorithm itself, learn how to work in a team. In this work, a mini-project is the generation of a Cartesian labyrinth with the calculation of the minimum path from the entry point to the exit point with the user's ability to set the dimensions of the labyrinth independently, select specific points for entry and exit for the labyrinth. Also during development, testing is written to verify the correctness of the algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основная цель практики – реализация мини-проекта, который является визуализацией алгоритма поиска самого короткого пути в лабиринте. Для выполнения данной цели были поставлены задачи: разработка GUI к проекту, генерация самого лабиринта и алгоритм поиска пути (A-star).

Алгоритм генерации лабиринта – лабиринт генерируется случайным образом. Сначала все клетки – стены. Выбираем начальную клетку, затем случайно выбираем направление от клетки и проверяем является ли следующая клетка соседом только для предыдущей, переходим в неё и она больше не является стеной и выполняем до тех пор пока таких клеток не останется. Результат работ данного алгоритма – случайно сгенерированный лабиринт.

Алгоритм A-star - поиск пути из выбранной начальной точки в выбранную конечную, эвристической функцией в данном случае выступает “Манхэтанское расстояние” (длина отрезка между двумя точками). Результатом алгоритма является минимальный путь из точки входа в точку выхода, если таковой имеется.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

1.1.1 Требования к вводу исходных данных

Для корректной работы алгоритма необходимо:

- ввести координаты лабиринта и нажать кнопку “Ок”.

- выбрать кнопкой мыши начало и конец лабиринта, затем нажать кнопку “Старт”.

- переключать шаги алгоритма с помощью кнопок “Назад” и “Вперёд”.

1.1.2 Требования к визуализации

Прототип программы представлен на данных рисунках.

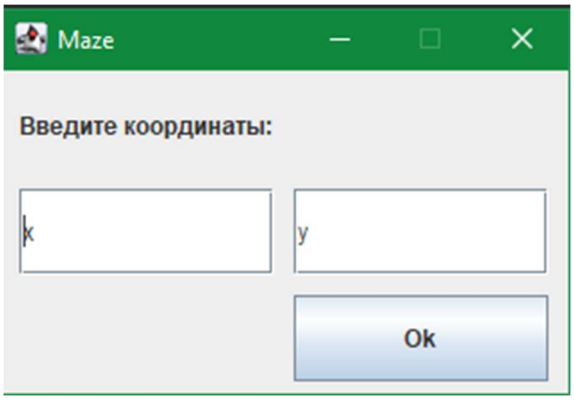

Рисунок 1

Рисунок 2

На Рисунке 1, видно куда пользователю необходимо ввести координаты (размеры) лабиринта, а затем нажать кнопку “Ок”.

На Рисунке 2 видно визуальное отображение лабиринта, в каждой клетке указаны её координаты. Кнопка “Старт” – необходима для запуска работы алгоритма, кнопки “Назад” и “Вперёд” – необходимы для переключения между шагами алгоритма, на шаг вперёд и на шаг назад соответственно.

1.1.3 Требования к выходным данным

Выходными данными является визуальное отображение найденного пути в лабиринте.

1.2. Уточнение требований после первого этапа

1.2.1 Требования к вводу исходных данных

Для корректной работы алгоритма необходимо:

- ввести координаты лабиринта и нажать кнопку “Ок”.
- нажать кнопку “Указать начало” и выбрать начало лабиринта.
- нажать кнопку “Указать конец” и выбрать конец лабиринта.
- переключать шаги алгоритма с помощью кнопок “Назад” и “Вперёд”.

1.2.2 Требования к визуализации

Прототип программы представлен на данных рисунках.

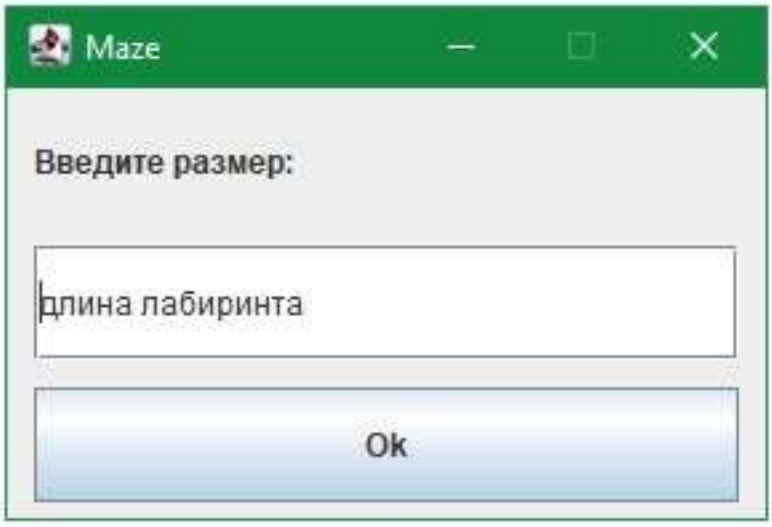

Рисунок 3

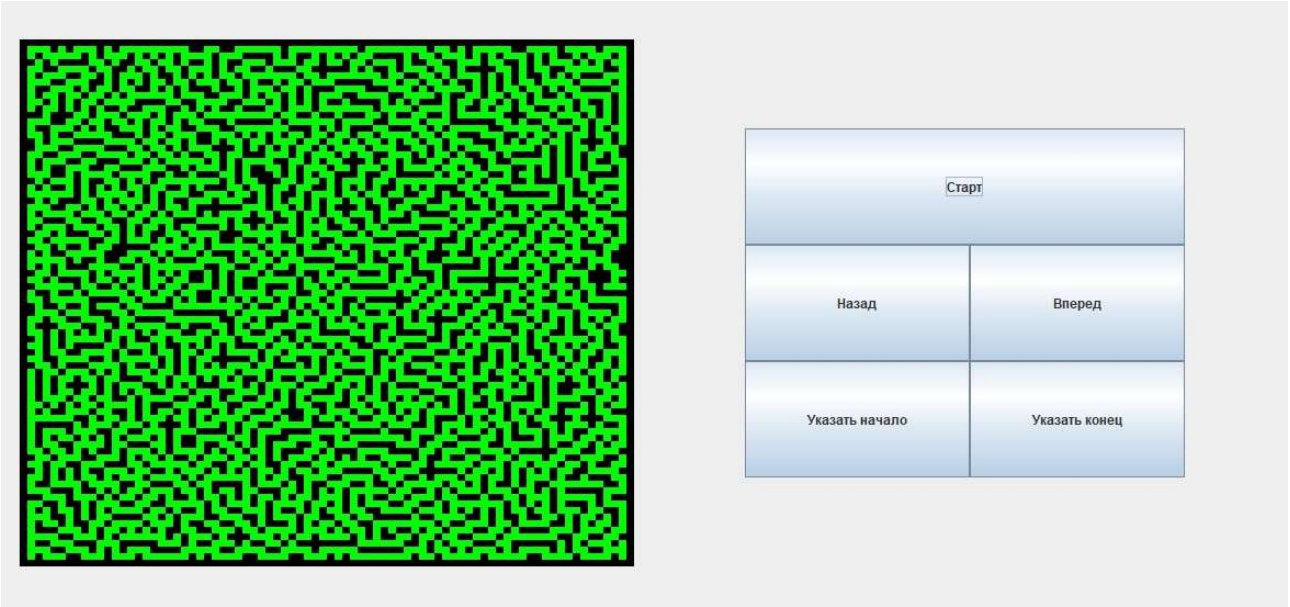

Рисунок 4

На Рисунке 3, видно куда пользователю необходимо ввести координаты (размеры) лабиринта, а затем нажать кнопку “Ок”.

На Рисунке 4 видно визуальное отображение лабиринта. Кнопка “Старт” – необходима для запуска работы алгоритма, кнопки “Назад” и “Вперёд” – необходимы для переключения между шагами алгоритма, на шаг вперёд и на шаг назад соответственно.

Также в данном приложение будет меняться цвет клеток лабиринта в зависимости от того является ли клетка стеной лабиринта или же, например, является ли клетка частью пути от входа к выходу.

С помощью нажатия кнопки “Указать начало” пользователю предлагается возможность самостоятельно выбрать начало лабиринта, а с помощью нажатия кнопки “Указать конец” – выбрать конец лабиринта.

1.2.3 Требования к выходным данным

Выходными данными является визуальное отображение найденного пути в лабиринте.

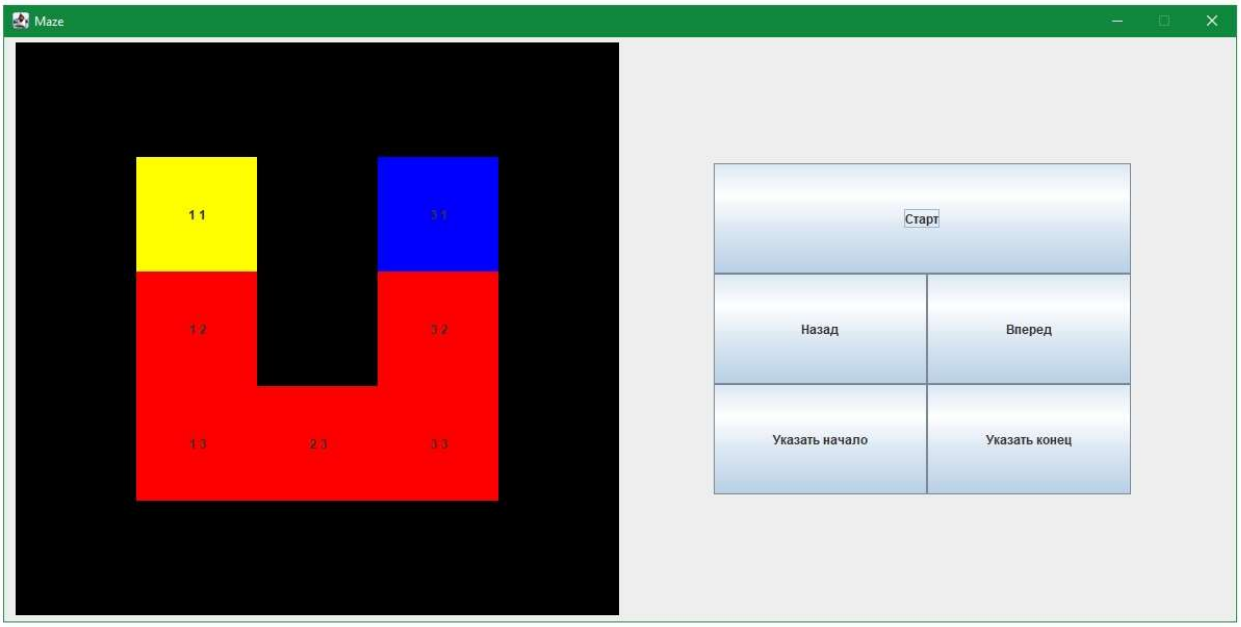

Рисунок 5

На Рисунке 5 - генерация алгоритма, и ход работы алгоритма после нажатия кнопки стар, желтая клетка - начало, голубая - конец пути

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЯ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. К 29 июня 2020 года выбрать тему мини-проекта, распределить роли между участниками бригады, создать примерный план работы.
2. Ко 2 июля 2020 года, выполнить первую итерацию, то есть создать прототип программы, прототип визуального дизайна, а также сделать отчёт по первой итерации.
3. К 3 июля 2020 года исправить все недочеты, обнаруженные при защите на первой итерации.
4. К 4 июля 2020 года реализовать генерацию лабиринта.
5. К 4 июля 2020 года реализовать качественный дизайн интерфейса для программы.
6. К 4 июля 2020 года частично выполнить первые два раздела итогового отчёта по летней практике.
7. К 5 июля 2020 года провести тестирование генерации лабиринта.
8. К 5 июля 2020 года реализовать алгоритм A-star поиска пути в лабиринте, реализовать работу кнопок “Выбрать начало лабиринта” и “Выбрать конец лабиринта”.
9. К 6 июля провести тестирование алгоритма A-star.
10. К 6 июля 2020 года частично выполнить оставшиеся два раздела итогового отчёта по летней практике.
11. К 7 июля 2020 года реализовать возможность просмотра хода пути по лабиринту по шагам.
12. 7 июля провести разбор недочётов после защиты 2-ой итерации
13. К 11 июля 2020 года завершить итоговый отчёт по летней практике.
14. К 11 июля провести улучшение дизайна итоговой программы.

2.2. Распределение ролей в бригаде

- Пушпышев А.И.
    - Алгоритм генерации лабиринта
    - Алгоритм A-star
    - Все побочные вещи, связанные с кодом
- Данилов А.В.
    - Тестирование обоих алгоритмов
    - Контроль качества
- Стукалев А.И.
    - Документация
    - Фронтенд

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Класс Maze реализует объект лабиринт:

Поля:

public final int size – размер лабиринта

public final Vector<Vector<Cell>> labyrinth – матрица клеток лабиринта

Методы:

public Maze(int size) – конструктор экземпляров класса Maze

public Maze(Maze m) – конструктор копирования

private void generateMaze() – метод генерации лабиринта

private void addNeighbours(Cell cell) – вспомогательный метод создания списка соседей клетки

public void clear() – возвращает объект в начальное состояние

Вложенный в класс Maze класс Cell:

Поля:

int x, int y – координаты клетки в матрице

boolean isWall – является ли клетка стеной

boolean wasSeen – была ли клетка просмотрена в алгоритме A-star

boolean isPath – является ли клетка частью пути

Vector<Cell> neighbours – список соседей клетки

int g – расстояние от точки входа до данной клетки

int h – “Манхэтанское расстояние”

int f – приоритет клети в алгоритме A-star

Cell cameFrom – предыдущая клетка пути

Методы:

public Cell(int x, int y)– конструктор экземпляров класса Cell

public Cell(Cell obj)– конструктор копирования

public void clear() – возвращает объект в начальное состояние

Класс A-star реализует работу алгоритма a-star:

Поля:

public static Snapshot snap – экземпляр класса отвечающий за паттерн “Snapshot”

Методы:

public static List<Maze.Cell> search(Maze maze, Maze.Cell sourceCell, Maze.Cell goalCell) – метод, который реализует алгоритма a-star

private static List<Maze.Cell> reconstructPath(Maze.Cell start, Maze.Cell goal) – метод восстановления пути от одной точки к другой

private static int heuristicCost(Maze.Cell a, Maze.Cell b) – функция для “Манхэтэнского расстояния”

public static class CellComparator implements Comparator<Maze.Cell> - реализация интерфейса сравнения для класса Cell

3.2. Основные методы

Основные методы необходимые для эффективной работы программы были написаны в классе Maze, реализующий лабиринт, и классе A-star, реализующий алгоритм a-star:

private void generateMaze() – метода класса Maze, генерирующий лабиринт. Идея алгоритма заключается в следующем: перемещение из клетки осуществляется в случайного соседа, который принадлежит, то есть не является соседом ни одной из клеток, которые входят в текущий путь. Метод заполняет поле матрицы клеток.

public static List<Maze.Cell> search(Maze maze, Maze.Cell sourceCell, Maze.Cell goalCell) – метод класса Astar, реализующий ход алгоритма и записывающий каждую итерацию “Историю” (паттерн Snapshot). Метод принимает в качестве аргументов Maze maze – лабиринт, Cell sourceCell и Cell goalCell – клетки, между которыми ищется путь. Возвращает список клеток, которые входят в путь.

4. ТЕСТИРОВАНИЕ

4.1 Мануальное тестирование программы

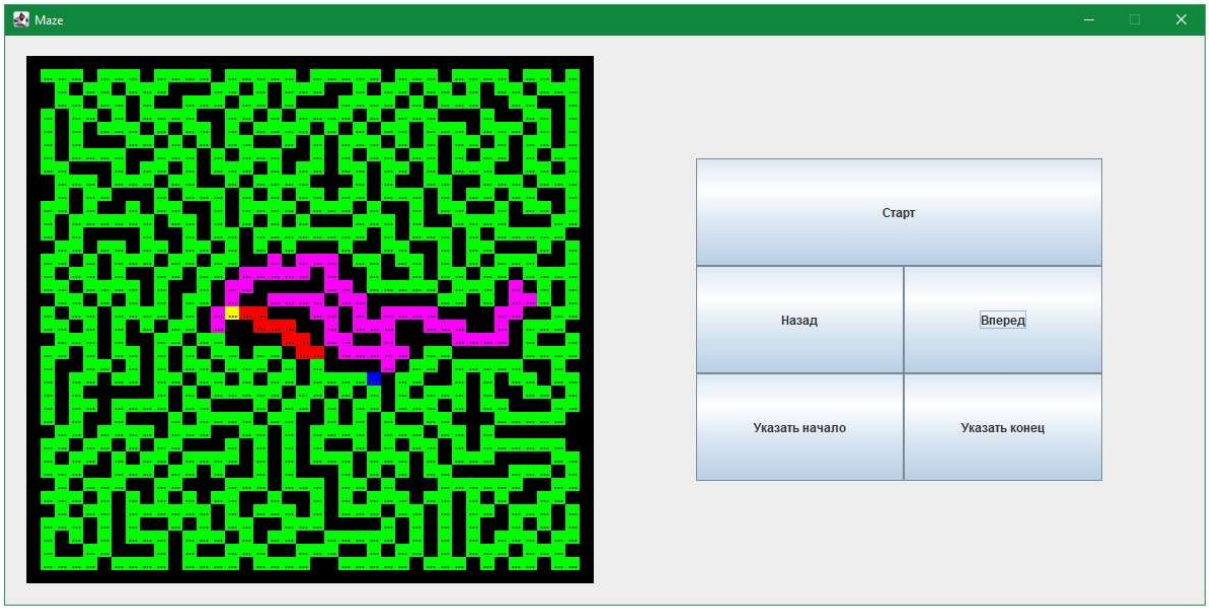

Рисунок 6 "построение хода алгоритма по шагам при помощи кнопок вперед и назад"

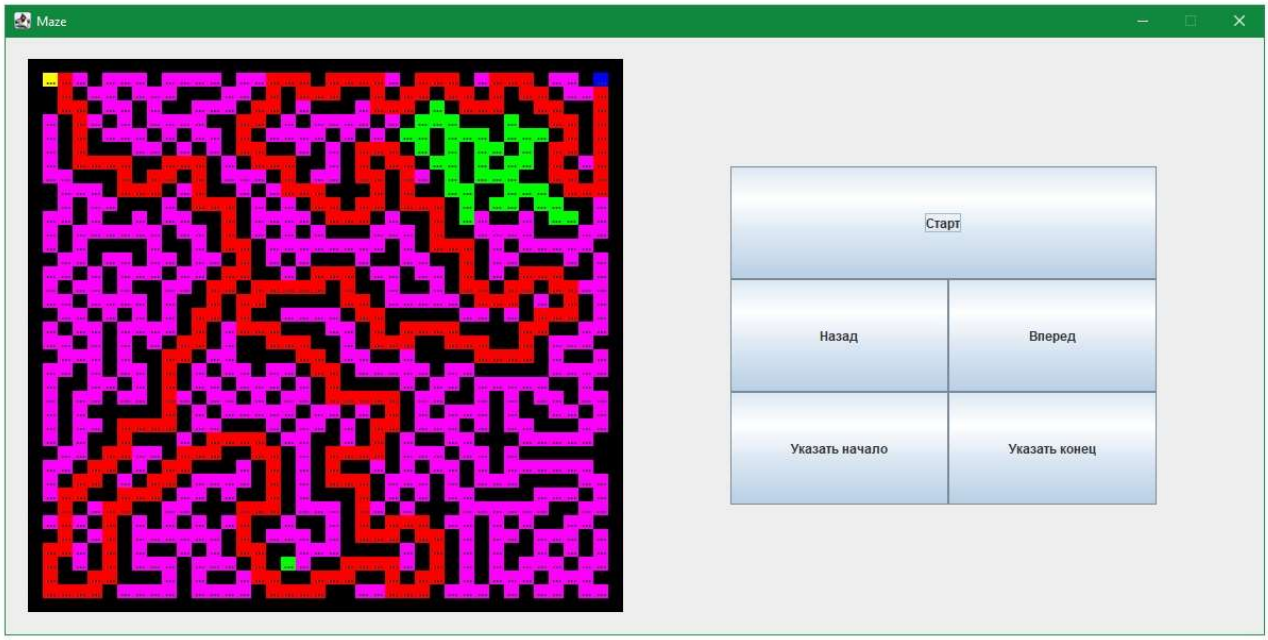

Рисунок 7 “Прямой ход по нажатии старт, фиолетовые ячейки - обработанные алгоритмом, красное - путь, желтое - начало, синее – конец”

Мануальным тестированием был проверен весь функционал программы

4.2. Unit тестирование программы

Для проведение автоматического тестирования программы использовалась библиотека junit 4. UNIT тесты были написаны с целью проверить основные моменты кода, а именно проверить верность нахождения пути в лабиринте.

Для проверки эффективного алгоритма поиска пути в лабиринте A* был написан проверяющий алгоритм поиска в глубину DFS, который является жадным, а следовательно не всегда оптимальным. Но из-за особенностей построения лабиринта, в нем существует единственный путь из точки A в точку B, поэтому в UNIT тестах происходил последовательный запуск алгоритмов A* и DFS, каждый из которых возвращал список вершин - финальный путь из A в B.

Были реализованы 3 теста.

searchBig(), searchMid(), searchSmall(), каждый из которых рандомно генерировал вершины A и B и находил путь с помощью каждого из алгоритмов.Различие этих алгоритмов в размере лабиринтов, которыми они оперируют. Максимальный размер стороны лабиринта для каждого 80, 50 и 17 соответственно. Итогом теста было сравнение итоговых путей методом assertEquals(), представленных в виде списка вершин List<Maze.Cell>.

Итогом теста редко бывает исход java.lang.OutOfMemoryError: Java heap space, причиной которого служит конечный запас оперативной памяти и неидеальность сборщика мусора, используемого в JVM. UNIT тесты представлены в Приложении Б.

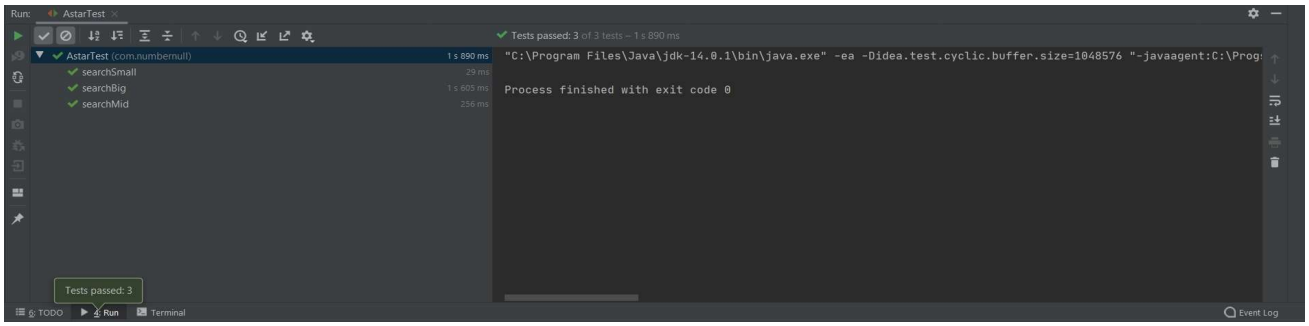

Рисунок 8 "Корректное выполнение Unit-test"

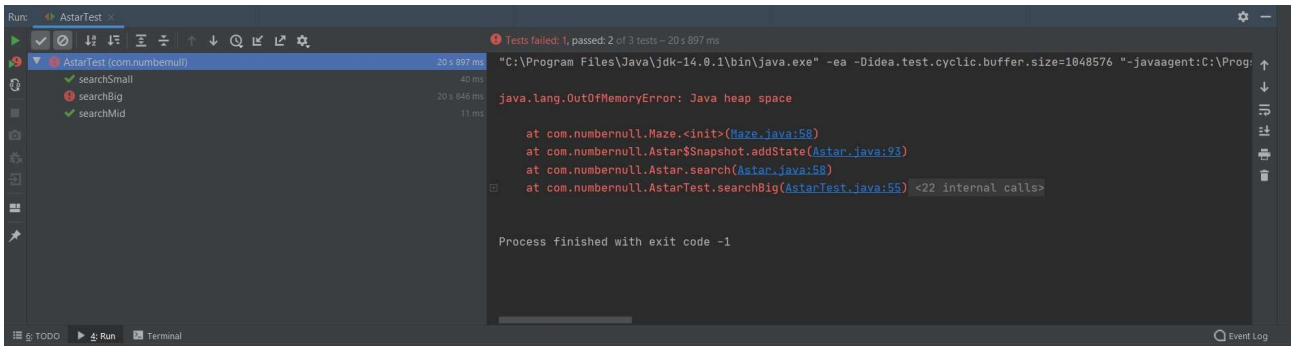

Рисунок 9 "Некорректное завершение теста в результате не достатка места в куче"

ЗАКЛЮЧЕНИЕ

Программа выполнялась в соответствие с планом. В ходе выполнения работы были реализованы алгоритм генерации лабиринта и алгоритм A-star, а также была визуализирована работа данных алгоритмов и создан интерфейс позволяющий управлять работой программы. Поставленные задачи были выполнены полностью и в срок.

В данный мини-проект был добавлены следующие дополнительные требования: возможность выбора входа и выхода из лабиринта с помощью отдельных кнопок, возможность пользователю задать размер лабиринта самостоятельно.

Подводя итог, можно сказать, что поставленные задачи были выполнены успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Учебный курс по основам Java на Stepik: https://stepik.org/course/187/
2. Официальная документация к Java: https://docs.oracle.com/en/java/javase/
3. Java 8. Руководство для начинающих. Герберт Шилдт
4. https://ru.wikipedia.org/wiki/
5. https://habr.com/ru/
6. https://ru.stackoverflow.com/

ПРИЛОЖЕНИЕ А. КОД ПРОГАММЫ

Файл Astar.java

```
package com.numbernull;

import java.util.*;

public class Astar {
   public static Snapshot snap;

   public static ArrayList<Maze.Cell> search(Maze maze, Maze.Cell sourceCell, Maze.Cell goalCell){

      if(sourceCell.isWall || goalCell.isWall)
         return new ArrayList<Maze.Cell>();

      snap = new Snapshot();
      Map<Maze.Cell, Boolean> closedSet = new HashMap<Maze.Cell, Boolean>();
      Comparator<Maze.Cell> comparator = new CellComparator();
      PriorityQueue<Maze.Cell> openSet = new PriorityQueue<Maze.Cell>(10, comparator);
      Vector<Maze.Cell> path = new Vector<Maze.Cell>();
      sourceCell.g = 0;
      sourceCell.h = heuristicCost(sourceCell, goalCell);
      sourceCell.f = sourceCell.g + sourceCell.h;
      openSet.add(sourceCell);
      snap.addState(maze);
      sourceCell.wasSeen = true;
      while(!openSet.isEmpty()){
         Maze.Cell x = openSet.remove();

          if (x == goalCell)
            return reconstructPath(sourceCell, goalCell);

         closedSet.put(x, true);
         for(Maze.Cell i : x.neighbours) {
            if (!i.isWall) {
               if (closedSet.containsKey(i)) {
                  continue;
               }
               int gScore = x.g + 1;
               boolean gBetter = false;
```

```
                if (!openSet.contains(i)) {
                    openSet.add(i);
                    gBetter = true;
                }else {
                    if (gScore < i.g) {
                        gBetter = true;
                    }
                }
                if(gBetter){
                    i.wasSeen = true;
                    i.cameFrom = x;
                    i.g = gScore;
                    i.h = heuristicCost(i,x);
                    i.f = i.g + i.h;
                }
            }
        }
        snap.addState(maze);
    }
    return new ArrayList<Maze.Cell>();
}

public static class Snapshot{
    public ArrayList<Maze> states;

    public Snapshot(){
        states = new ArrayList<>();
    }

    public void addState(final Maze step){
        this.states.add(new Maze(step));
    }

    public Maze getState(Integer iteration){
        return this.states.get(iteration);
    }
}

static ArrayList<Maze.Cell> reconstructPath(Maze.Cell start, Maze.Cell goal){
    ArrayList<Maze.Cell> path = new ArrayList<Maze.Cell>();
    Maze.Cell currentCell = goal;
    while(currentCell != null){
        path.add(currentCell);
```

```
            currentCell.isPath = true;
            currentCell = currentCell.cameFrom;
        }
        return path;
    }

    private static int heuristicCost(Maze.Cell a, Maze.Cell b){
        return (b.x * b.x + b.y * b.y) - (a.x * a.x + a.y * a.y);
    }

    public static class CellComparator implements Comparator<Maze.Cell>{
        @Override
        public int compare(Maze.Cell o1, Maze.Cell o2) {
            return Integer.compare(o2.f, o1.f);
        }
    }
}
```

Файл Main.java

```
package com.numbernull;

import javax.swing.*;

public class Main {

    public static void main(String[] args) {
                SizeInput si = new SizeInput("Maze");
                si.setVisible(true);
                si.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                si.setSize(300, 200);
                si.setResizable(false);
                si.setLocationRelativeTo(null);
                si.getRootPane().setBorder(BorderFactory.createEmptyBorder(5,  10,  5,
10));
        }
}
```

Файл Maze.java

```
package com.numbernull;
```

```
import java.util.*;

public class Maze {
   public final int sizeX;
   public final int sizeY;
   public final Vector<Vector<Cell>> labyrinth;
   public Maze(int sizeX, int sizeY){
      this.sizeX = sizeX;
      this.sizeY = sizeY;
      labyrinth = new Vector< Vector<Cell> >(10, 10);
      for(int i = 0; i < sizeY; i++) {
         Vector<Cell> cells = new Vector<Cell>();
         for (int j = 0; j < sizeX; j++) {
            Cell cell = new Cell(j, i);
            cells.addElement(cell);
            cell.isWall = true;
         }
         labyrinth.addElement(cells);
      }

      for(int i = 1; i < sizeY - 1; i++){
         for(int j = 1; j < sizeX - 1; j++){
            addNeighbours(labyrinth.elementAt(i).elementAt(j));
         }
      }
      generateMaze();
   }

   public Maze(int size){
      this.sizeX = size;
      this.sizeY = size;
      labyrinth = new Vector< Vector<Cell> >(10, 10);
      for(int i = 0; i < sizeY; i++) {
         Vector<Cell> cells = new Vector<Cell>();
         for (int j = 0; j < sizeX; j++) {
            Cell cell = new Cell(j, i);
            cells.addElement(cell);
            cell.isWall = true;
         }
         labyrinth.addElement(cells);
      }
      for(int i = 1; i < sizeY - 1; i++){
         for(int j = 1; j < sizeX - 1; j++){
```

```
            addNeighbours(labyrinth.elementAt(i).elementAt(j));
        }
    }
    generateMaze();
}

public Maze(Maze m){
    this.sizeX = m.sizeX;
    this.sizeY = m.sizeY;
    this.labyrinth = new Vector< Vector<Cell> >();
    for(int i = 0; i < sizeY; i++) {
        Vector<Cell> cells = new Vector<Cell>();
        for (int j = 0; j < sizeX; j++) {
            Cell cell = new Cell(m.labyrinth.elementAt(i).elementAt(j));
            cells.addElement(cell);
        }
        this.labyrinth.addElement(cells);
    }
}

public void printMaze(){
    for(int i = 0; i < sizeY; i++){
        for(int j = 0; j < sizeX; j++){
            if(labyrinth.elementAt(i).elementAt(j).isWall){
                System.out.print("#");
            }else System.out.print("&");
        }
        System.out.print("\n");
    }
    System.out.println();
}

private void generateMaze() {
    Cell currentCell = labyrinth.elementAt(1).elementAt(1);
    Stack<Cell> stack = new Stack<Cell>();
    Map<Cell, Integer> proceccedCells = new HashMap<Cell, Integer>();
    stack.push(currentCell);
    boolean flagStack = false;
    //основной цикл
    while (!stack.empty()) {
        proceccedCells.put(currentCell, -1);//если вершина является дорожкой, то
нет смысла в нее ходить
        currentCell.isWall = false;//ставим в текущую клетку путь
```

```
        //записываем соседей просмотриваемой вершины в массив, и помечаем сколько раз !повторно! мы их встретили
        if(!flagStack) {
            for (Cell i : currentCell.neighbours) {
                if (proceccedCells.containsKey(i)) {
                    if (proceccedCells.get(i) != -1) {
                        proceccedCells.put(i, proceccedCells.get(i) + 1);
                    }
                } else {
                    proceccedCells.put(i, 0);
                }
            }
        }
        Random random = new Random();
        //выбираем рандомного соседа
        Cell randNeighbour = currentCell.neighbours.elementAt(random.nextInt(currentCell.neighbours.size()));

        boolean backFlag = false;
        //если рандомный сосед уже сосед другой просмотренной вершины
        if (proceccedCells.get(randNeighbour) != 0) {
            //проверяем, что есть сосед, который не принадлежит обработанной вершине
            for (Cell i : currentCell.neighbours) {
                backFlag = proceccedCells.get(i) != 0;//false - когда равно 0, то есть потенциальная дорожка
                if(!backFlag){
                    break;
                }
            }
        }
        //если таких вершин нет, тот идем на предыдущую ячейку
        if (backFlag) {
            //смотрим, что не пытаемся очистить пустой стек, и переходим в предыдущую вершину
            stack.pop();
            if(!stack.empty()) {
                currentCell = stack.peek();
                flagStack = true;
            }
        } else {
            //если такие вершины есть, то пребираем соседей до посинения
            while (proceccedCells.get(randNeighbour) != 0) {
```

```
                randNeighbour =
currentCell.neighbours.elementAt(random.nextInt(currentCell.neighbours.size()));
            }
            currentCell = randNeighbour;//нашли нужную вершину, перешли в нее
            stack.push(currentCell);
            flagStack = false;
        }
    }
}


private void addNeighbours(Cell cell) {
    for (int i = cell.y - 1; i <= cell.y + 1; i++) {
        for (int j = cell.x - 1; j <= cell.x + 1; j++) {
            if (i > 0 && i < sizeY - 1 && j > 0 && j < sizeX - 1) {
                if((i != cell.y || j != cell.x) && (Math.abs(cell.x - j) == 1 ^ Math.abs(cell.y
- i) == 1)) {
                    cell.neighbours.addElement(labyrinth.elementAt(i).elementAt(j));
                }
            }
        }
    }
}

private void printNeighbours(Cell cell){
    System.out.println("Main cell is " + cell.x + " " + cell.y);
    for(Cell i : cell.neighbours){
        System.out.println("\tSlave cell is " + i.x + " " + i.y);
    }
}


public void clear(){
    for (int i = 0; i < this.sizeX-1; i++) {
        for (int j = 0; j <this.sizeY-1 ; j++) {
            labyrinth.elementAt(i).elementAt(j).clear();
        }
    }
    for (int i = 0; i < this.sizeX-1; i++) {
        for (int j = 0; j <this.sizeY-1 ; j++) {
            addNeighbours(labyrinth.elementAt(i).elementAt(j));
        }
    }
```

```
}

public class Cell{
    int x;
    int y;
    boolean isWall;
    boolean wasSeen;
    boolean isPath;
    Vector<Cell> neighbours;
    int g;
    int h;
    int f;
    Cell cameFrom;

    public Cell(int x, int y){
        this.x = x;
        this.y = y;
        wasSeen = false;
        isPath = false;
        neighbours = new Vector<Cell>();
    }

    public Cell(Cell obj){
        this.x = obj.x;
        this.y = obj.y;
        this.isWall = obj.isWall;
        this.wasSeen = obj.wasSeen;
        this.isPath = obj.isPath;
        this.neighbours = new Vector<>();
    }

    public void clear(){
        wasSeen = false;
        isPath = false;
        g = 0;
        h = 0;
        f = 0;
        cameFrom = null;
        neighbours.clear();
    }

}
```

```
}
```

Файл MazeWindow.java

```
package com.numbernull;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Vector;

/**
 * TO-DO exceptions
 */

public class MazeWindow extends JFrame {

    boolean isBegin = false;
    boolean isEnd = false;
    boolean isReady = false;

    int beginX;
    int beginY;
    int endX;
    int endY;
    Vector<Vector<JLabel>> mazeField;
    ArrayList<Maze.Cell> path;

    int currentIteration;

    private void updateField(Maze maze) {
        for (int i = 1; i < maze.sizeY - 1; i++) {
            for (int j = 1; j < maze.sizeX - 1; j++) {
                if (maze.labyrinth.elementAt(i).elementAt(j).wasSeen) {
                    boolean onWay = false;
                    for (Maze.Cell tmp : path) {
                        if (tmp.x == j && tmp.y == i) {
                            onWay = true;
                            break;
                        }
                    }
```

```
                if (onWay) {
                    mazeField.elementAt(i).elementAt(j).setBackground(Color.RED);

mazeField.elementAt(i).elementAt(j).setBorder(BorderFactory.createLineBorder(Col
or.RED, 1));
                } else {

mazeField.elementAt(i).elementAt(j).setBackground(Color.MAGENTA);

mazeField.elementAt(i).elementAt(j).setBorder(BorderFactory.createLineBorder(Col
or.MAGENTA, 1));
                }
            } else {
                if (!maze.labyrinth.elementAt(i).elementAt(j).isWall) {

mazeField.elementAt(i).elementAt(j).setBackground(Color.GREEN);

mazeField.elementAt(i).elementAt(j).setBorder(BorderFactory.createLineBorder(Col
or.GREEN, 1));
                }
            }
        }
    }

mazeField.elementAt(beginY).elementAt(beginX).setBackground(Color.YELLOW);

mazeField.elementAt(beginY).elementAt(beginX).setBorder(BorderFactory.createLi
neBorder(Color.YELLOW, 1));
    mazeField.elementAt(endY).elementAt(endX).setBackground(Color.BLUE);

mazeField.elementAt(endY).elementAt(endX).setBorder(BorderFactory.createLineB
order(Color.BLUE, 1));
  }


  public MazeWindow(String s, int x, int y, Maze labyrinth) {
    super(s);

    beginX = -1;
    beginY = -1;
    endX = -1;
    endY = -1;
```

```
        this.path = new ArrayList<>();
        this.currentIteration = 0;

        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(1200, 600);
        setResizable(false);
        setLocationRelativeTo(null);
        getRootPane().setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));

        Container container = getContentPane();
        container.setLayout(new GridLayout(0, 2));

        GridLayout mazeStyle = new GridLayout(x, y, 0, 0);
        JPanel maze = new JPanel();
        //maze.setSize(1000, 600);
        maze.setLayout(mazeStyle);

        GridBagLayout optionsStyle = new GridBagLayout();
        JPanel options = new JPanel();
        options.setLayout(optionsStyle);
        GridBagConstraints layoutOpt = new GridBagConstraints();


        JButton start = new JButton("Старт");

        start.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(beginX == -1 || beginY == -1 || endX == -1 || endY == -1 ){
                    JOptionPane.showMessageDialog(new JFrame(), "Укажите \"Начало\"
и \"Конец\"",
                            "Ошибка", JOptionPane.ERROR_MESSAGE);
                }else {
                    if (e.getSource() == start && !isReady) {
                        path                    =                    Astar.search(labyrinth,
labyrinth.labyrinth.elementAt(beginY).elementAt(beginX),
labyrinth.labyrinth.elementAt(endY).elementAt(endX));
                        isReady = true;
                        updateField(Astar.snap.states.get(Astar.snap.states.size() - 1));
                        currentIteration = Astar.snap.states.size() - 1;
                    }else{
                        if(e.getSource() == start && isReady){
```

```
                currentIteration = Astar.snap.states.size() - 1;
                updateField(Astar.snap.states.get(Astar.snap.states.size() - 1));
            }
        }
    }
}
});

layoutOpt.fill = GridBagConstraints.HORIZONTAL;
layoutOpt.gridx = 0; // № столбца
layoutOpt.gridy = 0; // № строки
layoutOpt.gridwidth = 2; // число ячеек, занимаемых объектом
layoutOpt.ipadx = 80;
layoutOpt.ipady = 80;
layoutOpt.anchor = GridBagConstraints.CENTER; //  задает выравнивание

options.add(start, layoutOpt);

JButton next = new JButton("Вперед");
JButton prev = new JButton("Назад");

next.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == next) {
            if(beginX == -1 || beginY == -1 || endX == -1 || endY == -1 ){
                JOptionPane.showMessageDialog(new JFrame(), "Укажите
\"Начало\" и \"Конец\"",
                        "Ошибка", JOptionPane.ERROR_MESSAGE);
            }else {
                if (!isReady) {
                    path = Astar.search(labyrinth,
labyrinth.labyrinth.elementAt(beginY).elementAt(beginX),
labyrinth.labyrinth.elementAt(endY).elementAt(endX));
                    isReady = true;
                }
                if (currentIteration < Astar.snap.states.size() - 1) {
                    currentIteration++;
                    updateField(Astar.snap.states.get(currentIteration));
                }
            }
        }
    }
```

```
        });

        prev.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == prev){
                    if(beginX == -1 || beginY == -1 || endX == -1 || endY == -1 ){
                        JOptionPane.showMessageDialog(new JFrame(), "Укажите
\"Начало\" и \"Конец\"",
                                "Ошибка", JOptionPane.ERROR_MESSAGE);
                    }else {
                        if (!isReady) {
                            path = Astar.search(labyrinth,
labyrinth.labyrinth.elementAt(beginY).elementAt(beginX),
labyrinth.labyrinth.elementAt(endY).elementAt(endX));
                            isReady = true;
                        }
                        if (currentIteration > 0) {
                            currentIteration--;
                            updateField(Astar.snap.states.get(currentIteration));
                        }
                    }
                }
            }
        });

        layoutOpt.fill = GridBagConstraints.HORIZONTAL;
        layoutOpt.gridx = 0; // № столбца
        layoutOpt.gridy = 1; // № строки
        layoutOpt.gridwidth = 1; // число ячеек, занимаемых объектом
        layoutOpt.anchor = GridBagConstraints.LINE_END; //  задает выравнивание

        options.add(prev, layoutOpt);
        layoutOpt.gridx = 1; // № столбца
        layoutOpt.anchor = GridBagConstraints.LINE_START;
        options.add(next, layoutOpt);

        // настройки для стороны с лабиринтом
        JPanel leftPane = new JPanel();
        leftPane.setLayout(new GridBagLayout());
        GridBagConstraints gbcMaze = new GridBagConstraints();
        gbcMaze.fill = GridBagConstraints.NONE; //стратегию распределения
компоненту свободного пространства
```

```
        gbcMaze.weightx = 1; //выделение пространства для столбцов

        gbcMaze.gridx = 0; // № столбца
        gbcMaze.gridy = 0; // № строки
        gbcMaze.gridwidth = 1; // число ячеек, занимаемых объектом
/*
        gbcMaze.ipadx = 20;
        gbcMaze.ipady = 20;
        gbcMaze.anchor = GridBagConstraints.LINE_END; //  задает выравнивание компонента внутри отведенного для него пространства
        leftPane.add(new JButton("Сохранить лабиринт"), gbcMaze);

        gbcMaze.gridx = 1; // № столбца
        gbcMaze.gridy = 0; // № строки
        gbcMaze.gridwidth = 1; // число ячеек, занимаемых объектом
        gbcMaze.anchor = GridBagConstraints.LINE_START;
        leftPane.add(new JButton("Загрузить лабиринт"), gbcMaze);
*/
        gbcMaze.ipadx = 0;
        gbcMaze.ipady = 0;
        gbcMaze.fill = GridBagConstraints.BOTH;
        gbcMaze.gridwidth = 2;
        //gbcMaze.weightx = 1; //выделение пространства для столбцов
        gbcMaze.weighty = 1; //и строк
        //gbcMaze.gridx = 0; // № столбца
        gbcMaze.gridy = 1; // № строки

        JButton setBegin = new JButton("Указать начало");
        JButton setEnd = new JButton("Указать конец");

        setBegin.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (e.getSource() == setBegin) {
                    if(isReady){
                        cleanTable(labyrinth, endX, endY);
                    }
                    isBegin = true;
                    isEnd = false;
                }
            }
        });
```

```
setEnd.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == setEnd) {
            if(isReady){
                cleanTable(labyrinth, beginX, beginY);
            }
            isEnd = true;
            isBegin = false;
        }
    }
});

layoutOpt.gridx = 0; // № столбца
layoutOpt.gridy = 2;
layoutOpt.anchor = GridBagConstraints.LINE_START;
options.add(setBegin, layoutOpt);
layoutOpt.gridx = 1;
layoutOpt.anchor = GridBagConstraints.LINE_END;
options.add(setEnd, layoutOpt);

mazeField = new Vector<>();


for (int i = 0; i < y; i++) {
    Vector<JLabel> cells = new Vector<>();
    for (int j = 0; j < x; j++) {
        if (!labyrinth.labyrinth.elementAt(i).elementAt(j).isWall) {
            JLabel textLabel = new JLabel();
            textLabel.setBorder(BorderFactory.createLineBorder(Color.GREEN, 1));
            textLabel.setBackground(Color.GREEN);
            textLabel.setOpaque(true);
            textLabel.setHorizontalAlignment(SwingConstants.CENTER);
            textLabel.setText(j + " " + i);
            textLabel.addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent e) {
                    int x;
                    int y;
                    String[] args = textLabel.getText().split(" ");
                    x = Integer.parseInt(args[0]);
                    y = Integer.parseInt(args[1]);
                    if (isBegin) {
```

```
                        beginX = x;
                        beginY = y;
                        isBegin = false;

mazeField.elementAt(beginY).elementAt(beginX).setBackground(Color.YELLOW);

mazeField.elementAt(beginY).elementAt(beginX).setBorder(BorderFactory.createLi
neBorder(Color.YELLOW, 1));
                    }
                    if (isEnd) {
                        endX = x;
                        endY = y;
                        isEnd = false;

mazeField.elementAt(endY).elementAt(endX).setBackground(Color.BLUE);

mazeField.elementAt(endY).elementAt(endX).setBorder(BorderFactory.createLineB
order(Color.BLUE, 1));
                    }
                }
            });
            cells.addElement(textLabel);
            maze.add(textLabel);
        } else {
            JLabel textLabel = new JLabel();
            textLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK,
5));
            textLabel.setBackground(Color.BLACK);
            textLabel.setOpaque(true);
            textLabel.setHorizontalAlignment(SwingConstants.CENTER);
            cells.addElement(textLabel);
            maze.add(textLabel);
        }
      }
      mazeField.addElement(cells);
    }

    leftPane.add(maze, gbcMaze);

    container.add(leftPane);
    container.add(options);
  }
```

```
    private  void cleanTable(Maze m, int saveX, int saveY){
        m.clear();
        isReady = false;
        currentIteration = 0;
        for (int i = 0; i < m.sizeY-1; i++) {
            for (int j = 0; j < m.sizeX-1 ; j++) {
                if(!m.labyrinth.elementAt(i).elementAt(j).isWall){
                    mazeField.elementAt(i).elementAt(j).setBackground(Color.GREEN);

mazeField.elementAt(i).elementAt(j).setBorder(BorderFactory.createLineBorder(Col
or.GREEN, 1));
                }
            }
        }
    }
}
```

SizeInput.java

```
package com.numbernull;

import org.w3c.dom.ranges.RangeException;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;



public class SizeInput extends JFrame{
    private final JTextField yCord;
    private int y = 0 ;
    private final JButton ok;

    public SizeInput(final String s) {
        super(s);
        setLayout(new GridLayout(3, 0, 10, 10));
        this.ok = new JButton("Ok");
        JLabel info = new JLabel("Введите размер:");
        this.yCord = new JTextField("длина лабиринта", 5);
        add(info);
        add(this.yCord);
        add(this.ok);
```

```
        ok.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e) {
                if(e.getSource() == ok){
                    try {
                        y = Integer.parseInt(yCord.getText());
                        if (y < 4 || y > 100) {
                            throw new RangeException((short) 0,"");
                        }
                        setVisible(false);
                        new MazeWindow(s, y, y, new Maze(y));
                    }catch (NumberFormatException | RangeException err){
                        JOptionPane.showMessageDialog(new JFrame(), "Введите целое
значение от 4 до 100",
                                "Ошибка", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        });

        yCord.addMouseListener(new MouseAdapter(){
            @Override
            public void mouseClicked(MouseEvent e){
                yCord.setText("");
            }
        });

    }
}
```

ПРИЛОЖЕНИЕ Б. UNIT-ТЕСТИРОВАНИЕ

Файл AstarTest.java

```
package com.numbernull;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import static org.junit.Assert.*;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class AstarTest {
    Maze Big1, Mid1, Small1 ,Big2, Mid2, Small2;
    List<Maze.Cell> A;
    List<Maze.Cell> D;
    int StartX;
    int StartY;
    int GoalX;
    int GoalY;
    private int[] genCoords(Maze m){
        Random rnd = new Random();
        int[] ans = new int[4];
        StartX = rnd.nextInt(m.sizeX-1);
        StartY = rnd.nextInt(m.sizeX-1);
        GoalX = rnd.nextInt(m.sizeX-1);
        GoalY = rnd.nextInt(m.sizeX-1);
        while (m.labyrinth.elementAt(StartY).elementAt(StartX).isWall){
            StartX = rnd.nextInt(m.sizeX-1);
            StartY = rnd.nextInt(m.sizeX-1);
        }
        while (m.labyrinth.elementAt(GoalY).elementAt(GoalX).isWall){
            GoalX = rnd.nextInt(m.sizeX-1);
            GoalY = rnd.nextInt(m.sizeX-1);
        }
        ans[0] = StartX;
        ans[1] = StartY;
        ans[2] = GoalX;
```

```
    ans[3] = GoalY;
    return ans;
}

@Test
public void searchBig() {
    Random rnd = new Random();
    int sizeB = rnd.nextInt(10)+70;
    for (int j = 0; j < 2; j++) {
        Big1 = new Maze(sizeB);
        int[] tmp = genCoords(Big1);
        StartX = tmp[0];
        StartY = tmp[1];
        GoalX = tmp[2];
        GoalY = tmp[3];

        A = Astar.search(Big1, Big1.labyrinth.elementAt(StartX).elementAt(StartY),
                Big1.labyrinth.elementAt(GoalX).elementAt(GoalY));
        Big1.clear();
        D = DFS.search(Big1, Big1.labyrinth.elementAt(StartX).elementAt(StartY),
                Big1.labyrinth.elementAt(GoalX).elementAt(GoalY));
        assertEquals(A,D);

    }
}
@Test
public void searchMid() {
    Random rnd = new Random();
    int sizeM = rnd.nextInt(10)+30;
    for (int j = 0; j < 2; j++) {
        Mid1 = new Maze(sizeM);
        int[] tmp = genCoords(Mid1);
        StartX = tmp[0];
        StartY = tmp[1];
        GoalX = tmp[2];
        GoalY = tmp[3];
        A = Astar.search(Mid1, Mid1.labyrinth.elementAt(StartX).elementAt(StartY),
                Mid1.labyrinth.elementAt(GoalX).elementAt(GoalY));
        Mid1.clear();
        D = DFS.search(Mid1, Mid1.labyrinth.elementAt(StartX).elementAt(StartY),
                Mid1.labyrinth.elementAt(GoalX).elementAt(GoalY));
        assertEquals(A,D);
```

```
        }
    }
    @Test
    public void searchSmall() {
        Random rnd = new Random();
        int sizeS = rnd.nextInt(10)+6;
        for (int j = 0; j < 2; j++) {
            Small1 = new Maze(sizeS);
            int[] tmp = genCoords(Small1);
            StartX = tmp[0];
            StartY = tmp[1];
            GoalX = tmp[2];
            GoalY = tmp[3];
            A = Astar.search(Small1,
Small1.labyrinth.elementAt(StartX).elementAt(StartY),
                    Small1.labyrinth.elementAt(GoalX).elementAt(GoalY));
            Small1.clear();
            D = DFS.search(Small1,
Small1.labyrinth.elementAt(StartX).elementAt(StartY),
                    Small1.labyrinth.elementAt(GoalX).elementAt(GoalY));
            assertEquals(D,A);

        }
    }
}
```

Файл DFS.java

```
package com.numbernull;
import java.util.*;

public class DFS {
    public static List<Maze.Cell> search(Maze maze, Maze.Cell sourceCell, Maze.Cell
goalCell){

        if(sourceCell.isWall || goalCell.isWall)
            return new ArrayList<Maze.Cell>();

        Stack<Maze.Cell>stack = new Stack<>();
        Maze.Cell curr;
        sourceCell.wasSeen = true;
        stack.push(sourceCell);
        while (!stack.isEmpty()){
            curr = stack.pop();
            if(curr == goalCell)
```

```
                return Astar.reconstructPath(sourceCell, goalCell);
            for (int i = 0; i < hasFriends(curr).size();++i) {
                if (!hasFriends(curr).get(i).wasSeen) {
                    hasFriends(curr).get(i).cameFrom = curr;
                    stack.push(hasFriends(curr).get(i));
                    hasFriends(curr).get(i).wasSeen = true;
                }
            }
        }
        return new ArrayList<Maze.Cell>();
    }
    public static List<Maze.Cell> hasFriends(Maze.Cell curr){
        List<Maze.Cell> ans = new ArrayList<Maze.Cell>();
        for (Maze.Cell i: curr.neighbours){
            if (!i.isWall)
                ans.add(i);
        }
        return ans;
    }
}
```