**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

По лабораторному практикуму

Тема: Поиск пути в лабиринте с использованием бэктрекинга

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6382 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Охримук Е.С |
| гр. 6382 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Кацер И.С. |
| гр.6303  Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Эвергрин П.С.  Фирсов М.А. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студентка Охримук Е.С. группы 6382 | | |
| Студентка Кацер И.С. группы 6382 | | |
| Студент Эвергрин П.С. группы 6303  Тема практики: Поиск пути в лабиринте с использованием бэктрекинга | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма поиска пути в лабиринте с использованием бэктрекинга на Java с графическим интерфейсом. | | |
| Сроки прохождения практики: 27.06.2018 – 10.07.2018 | | |
| Дата сдачи отчета: | | |
| Дата защиты отчета: | | |
|  | | |
| Студентка гр.6382 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Охримук Е.С. |
| Студентка гр.6382 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Кацер И.С. |
| Студент гр.6303 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Эвергрин П.С. |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Фирсов М.А. |

**Аннотация**

В данной работе рассмотрена программа, которая работает с помощью бэктрекинга, ищет выход из лабиринта. Программа разработана в среде IntelliJ IDEA. Язык разработанной программы Java. Программа подробно показывает процесс прохождения лабиринта в поисках выхода из него.

**Summary**

In this work, we consider a program that works by using backtracking, looking for a way out of the labyrinth. The program is developed in the IntelliJ IDEA. The language of the developed Java program. The program details the process of passing the labyrinth in search of an exit from it.

**Введение.**

**Формулировка задания.** Требуется разработать программу, поиск пути в лабиринте с использованием бэктрекинга. При этом должен присутствовать графический интерфейс.

**Поиск с возвратом**, **бэктрекинг**  — общий [метод](https://ru.wikipedia.org/wiki/Метод) нахождения решений задачи, в которой требуется [полный перебор](https://ru.wikipedia.org/wiki/Полный_перебор) всех возможных вариантов в некотором [множестве](https://ru.wikipedia.org/wiki/Множество) М. Как правило позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты …», «Сколько существует способов …», «Есть ли способ …», «Существует ли объект…» и т. п.

Термин *backtracking* был введен в 1950 году американским математиком Дерриком Генри Лемером.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше.

Метод поиска с возвратом является универсальным. Достаточно легко проектировать и программировать алгоритмы решения задач с использованием этого метода.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 4 |
| 1. | Требования к программе | 6 |
| 2. | План разработки и распределение ролей в бригаде | 6 |
| 2.1. | План разработки | 6 |
| 2.2. | Распределение ролей в бригаде | 6 |
| 3. | Особенности реализации | 6 |
| 3.1. | Описание классов | 6 |
| 3.2. | Алгоритм работы | 9 |
| 3.3  3.4 | Входные/выходные данные  Описание интерфейса | 9  10 |
| 4. | Тестирование | 11 |
| 4.1 | Тестирование графического интерфейса | 11 |
| 4.2 | Тестирование кода алгоритма | 12 |
|  | Заключение | 18 |
|  | Список использованных источников | 18 |
|  | Приложение А. Код программы. | 18 |
|  | Приложение B. UML - диаграмма. | 22 |

1. **Требование к программе**

Данная программа должна корректно работать на всевозможных входных данных и выводить маршрут выхода из лабиринта. В случае, если выхода не существует, уведомлять об этом пользователя.

Программа должна подробно демонстрировать найденный путь и наглядно показывать его.

1. **План разработки и распределение ролей в бригаде**
   1. **План разработки:**

К 02.07 разработана спецификация программы.

К 04.07 разработана первая версия программы.

* 1. **Распределение ролей в бригаде:**

Кацер Ирина: разработка класса PaintCanvas, написание отчета, тестирование.

Охримук Екатерина: разработка классов NewJFrame, написание отчета, тестирование.

Эвергрин Проспер: разработка класса Maze, написание отчета, тестирование.

1. **Особенности реализации**

**3.1. Описание классов:**

*Class NewJFrame.*

* privatevoidinitComponents() *-* метод инициализации кнопок и полей
* private void jButton1MouseClicked(java.awt.event.MouseEvent evt) — метод нажатия на jButton1
* private void jSpinner1StateChanged(javax.swing.event.ChangeEvent evt) — метод изменения значений на jSpinner1
* private void jSpinner2StateChanged(javax.swing.event.ChangeEvent evt)— метод изменения значений на jSpinner2
* private void canvas1MouseMoved(java.awt.event.MouseEvent evt) — метод обработки движения мыши на поле лабиринта
* private void jButton2MouseClicked(java.awt.event.MouseEvent evt) — метод обработки нажатия на jButton2
* private void canvas1MouseReleased(java.awt.event.MouseEvent evt) — метод обработки нажатия на поле лабиринта
* private void jButton3MouseClicked(java.awt.event.MouseEvent evt)— метод обработки нажатия на jButton3
* private void jSpinner3StateChanged(javax.swing.event.ChangeEvent evt) — метод изменения значений на jSpinner3
* private void jButton4MouseClicked(java.awt.event.MouseEvent evt)— метод обработки нажатия на jButton4

*Class PaintCanavas*

* void set\_height(int height) — метод установки высоты
* void set\_width(int width) — метод установки ширины
* void paint\_canvas — метод отрисовки поля лабиринта
* void draw\_rect(int x,int y,Color color,Color c\_filling) — метод рисования прямоугольника.

Int x,int y,Color color,Color c\_filling — параметры клетки лабиринта

* void paint\_selection(int x,int y,boolean clicked) — метод рисования выделения.

Int x,int y,boolean clicked — параметры отправной точки

*Class Maze*

* void on\_mouse\_dragged(int x,int y) — метод обработки движения мыши на Canvas.

Int x, int y — координаты мыши

* void set\_delay (int delay) — метод установки задержки.

Int delay — время задержки

* void paint () - метод перерисовки поля
* void bad\_rand\_generarion() - метод случайной генерации лабиринта
* void on\_mouse\_clicked(int x,int y) — метод обработки нажатия мыши на Canvas.

Int x, int y — координаты мыши

* void set\_height(int height) — метод установки высоты.

Int height — высота

* void clear() - метод очистки поля
* void go\_maze() - метод построения маршрута в лабиринте
* boolean check\_left(int x , int y) — метод проверки левой свободной клетки.

int x , int y — координаты отправной точки

* boolean check\_right(int x , int y)— метод проверки правой свободной клетки.

int x , int y — координаты отправной точки

* boolean check\_up(int x , int y)— метод проверки верхней свободной клетки.

int x , int y — координаты отправной точки

* boolean check\_down(int x , int y)— метод проверки нижней свободной клетки.

int x , int y — координаты отправной точки

* void Find\_way\_out(int x, int y) — метод поиска выхода из лабиринта.

int x , int y — координаты отправной точки

**3.2.Алгоритм работы:**

1. Сделайте начальную клетку текущей и отметьте ее как посещенную.2. Пока в стеке есть точки или не найден выход 1) Если текущая клетка имеет не посещённых «соседей»

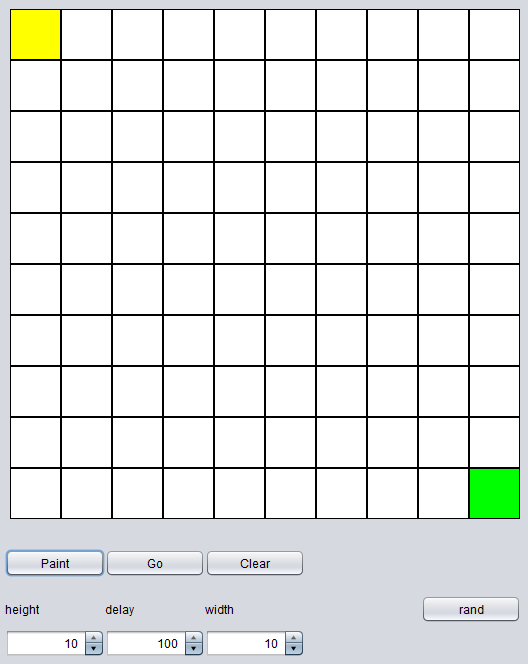
а. Протолкните текущую клетку в стекб. Выберите случайную клетку из соседнихв. Уберите стенку между текущей клеткой и выбраннойг. Сделайте выбранную клетку текущей и отметьте ее как посещенную. 2) Иначе если стек не пуста. Выдерните клетку из стекаб. Сделайте ее текущей3) Иначеа. Выберите не посещённую клетку, сделайте ее текущей и отметьте как посещенную.

**3.3. Входные/выходные данные:**

На *вход* пользователь вводит длину и ширину лабиринта, может выбирать скорость обхода (в миллисекундах), а также сам построить лабиринт.

На *выходе* пользователь получит маршрут выхода из лабиринта, а если такового нет, то получит соответствующее сообщение.

**3.4. Описание интерфейса:**



Пользовательский интерфейс состоит из :

* Поле лабиринта (пользователь составляет лабиринт вручную или случайно генерирует).

Зеленый квадрат показывает место входа.

Желтый квадрат показывает место выхода.

При нажатии на квадрат он окрашивается в другой цвет, показывая тем самым стенку в лабиринте. При повторном нажатии стенка убирается.

* Три окна ввода параметров поля (height, delay, width).
* Кнопка Go, при нажатии на которую начинается обход лабиринта.
* Кнопка Clear, при нажатии на которую происходит очистка поля лабиринта.
* Кнопка Paint, при нажатии на которую происходит отрисовка поля.
* Кнопка rand, при нажатии на которую происходит случайная генерация лабиринта.

**4. Тестирование**

**4.1. Тестирование графического интерфейса**

Все кнопки нашего графического интерфейса работают корректно. В ходе тестирования никаких ошибок выявлено не было.

**4.2. Тестирование кода алгоритма**

1. Случай, когда на поле нет стенок, приведен на Рисунке 1.

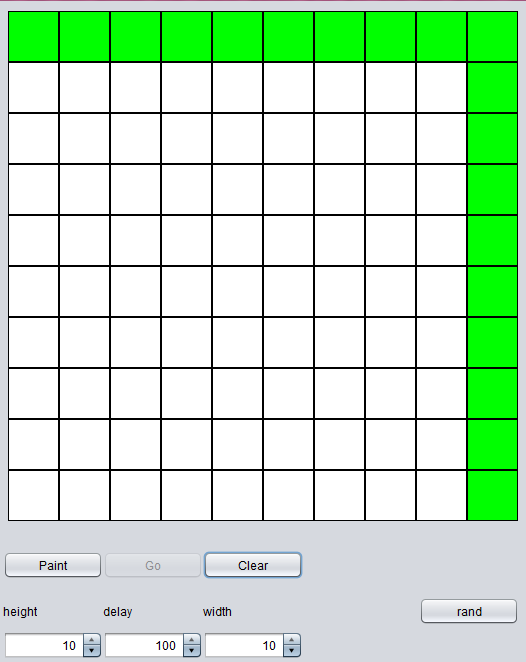


Рисунок 1 – Результат работы программы, когда совсем нет стен

1. Случай, когда на поле расставлены все стенки, приведен на Рисунке 2.

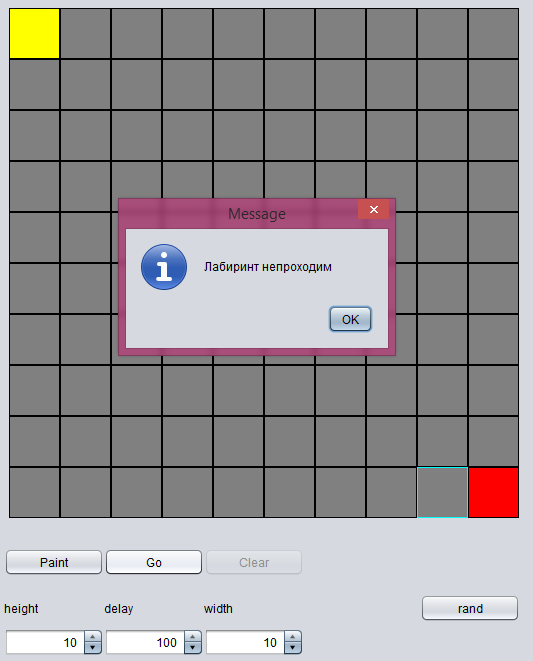


Рисунок 2­ –Результат работы программы, когда расставлены все стенки

1. Случай, когда стенки расставлены вокруг выхода из лабиринта, приведен на Рисунке 3.

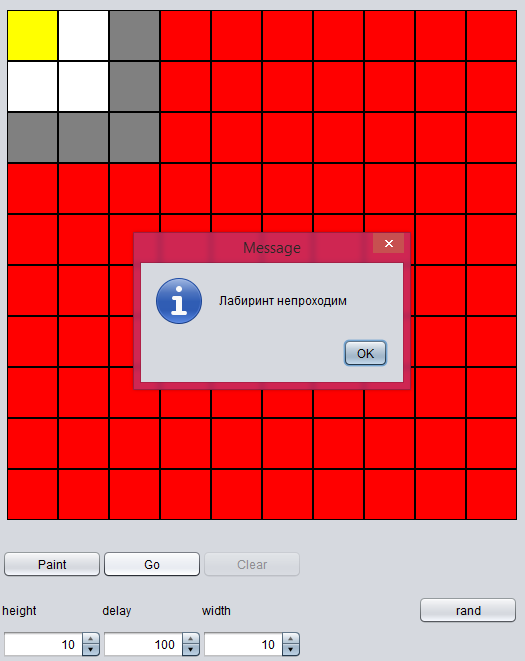


Рисунок 3­ –Результат работы программы, когда стенки расставлены вокруг выхода

1. Случай, когда из лабиринта есть выход (лабиринт составлен пользователем), приведен на Рисунке 4.

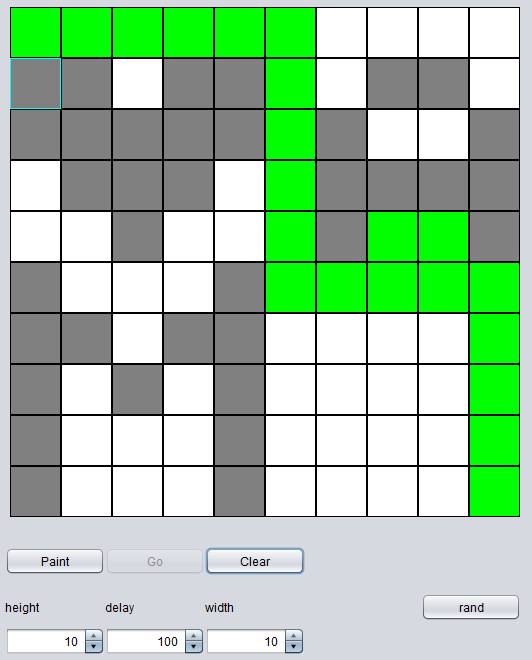


Рисунок 4­ –Результат работы программы, когда из лабиринта есть выход (лабиринт составлен пользователем)

1. Случай, когда из лабиринта есть выход (лабиринт случайно сгенерирован), приведен на Рисунке 5.

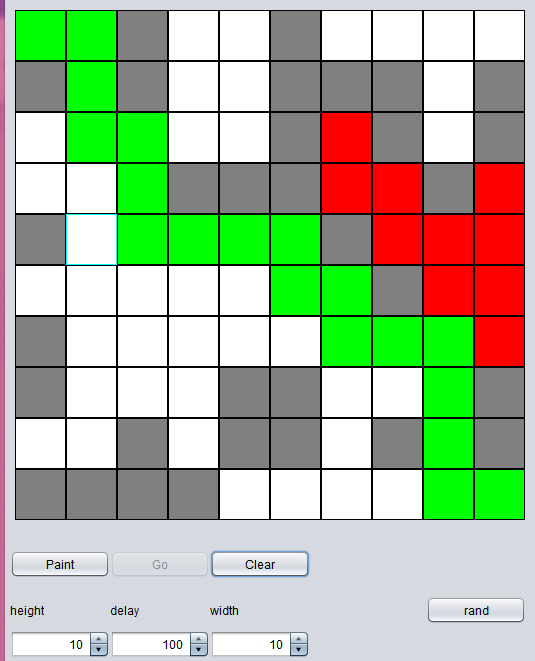


Рисунок 5­ –Результат работы программы, когда из лабиринта есть выход (лабиринт случайно сгенерирован)

1. Случай, когда из лабиринта нет выхода (лабиринт сгенерирован позьзователем), приведен на Рисунке 6.

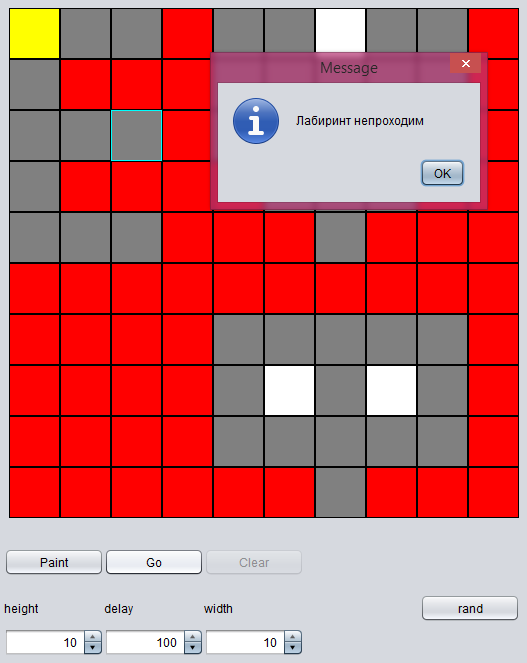


Рисунок 6­ – Результат работы программы, когда из лабиринта нет выхода (лабиринт сгенерирован пользователем)

**Заключение:**

В ходе данной учебной практики были изучены основы программирования на языке Java, пройден интерактивный курс «Java.Базовый курс». После чего была разработана программа, которая с помощью бэктрекинга находит выход из лабиринта, наглядно демонстрируя его пользователю.

Программа соответствует требованиям спецификации и прошла проверку на всевозможных тестах, никаких ошибок выявлено не было.

**Список использованных источников:**

1. Поиск с возвратом.URL: ru.wikipedia.org/wiki/Поиск\_с\_возвратом
2. [Java Базовый курс.URL: stepik.org/course/Java-Базовый-курс-187/syllabus](https://stepik.org/course/Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81-187/syllabus)
3. Статья «Лабиринт».URL: habrahabr.ru/post/262345/

**Приложение А.**

**Код программы.**

Maze.java

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.util.ArrayDeque;  
**import** java.util.Deque;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
  
  
**public class** Maze **extends** JPanel {  
 **static int** *size\_canvas* = 511;*//используется для рассчета размера ячеек* **static int** *thikness* = 1;  
 **int delay**=100;  
 **int width**,**height**;  
 **int**[][] **maze\_wall**;*//массив который хранит наше поле* **boolean dont\_edit**=**false**;*//разрешение рисовки поля(при рандоме нельзя)* PaintCanvas **canvas**;*//для рисовки нашего поля* Deque<Integer[]> **stack** = **new** ArrayDeque<>();*//очередь для прохода по полю* Maze(Canvas canv,**int** height , **int** width){ *//конструктор: разрисовку поля, ширину, высоту* **canvas** = **new** PaintCanvas(canv,height,width);  
 **this**.**height** = height;  
 **this**.**width** = width;  
 **maze\_wall** = **new int**[width][height];  
 paint();  
 }  
 **void** on\_mouse\_dragged(**int** x,**int** y){ *//метод обработки движения мыши на Canvas.  
 //Int x, int y — координаты мыши* **int** coord\_x = (x\***width**)/*size\_canvas*;*//координата в массиве* **int** coord\_y = (y\***height**)/*size\_canvas*;*//координата в массиве* **if** ((coord\_x==0 && coord\_y==0) || (coord\_x==**width**-1 && coord\_y==**height**-1) || (**dont\_edit**)) **return**;  
 **canvas**.paint\_selection(coord\_x, coord\_y,**maze\_wall**[coord\_x][coord\_y]!=0);*//раскрашиваем выбранную ячейку* }  
  
 @Override  
 **public void** paintComponents(Graphics g) {  
 **super**.paintComponents(g);  
 **this**.paint();  
 }  
  
 **void** set\_delay(**int** delay){ *//— метод установки задержки* **this**.**delay** = delay; *// Int delay — время задержки* }  
  
 **void** paint(){ *//разрисовываем поле* **if** (**dont\_edit**) **return**;  
 **canvas**.paint\_canvas(); *//прорисовываем начальное поле(белое с входом и выходом)* **for**(**int** i=0;i<**width**;i++){  
 **for**(**int** j=0;j<**height**;j++){  
 **if** (**maze\_wall**[i][j]!=0) **canvas**.draw\_rect(i, j, Color.***BLACK***,Color.***gray***);*//если в ячейке не ноль,  
 // то там стена и красим ее в серый* }  
 }  
  
 }  
  
  
  
 **void** bad\_rand\_generarion(){ *//метод случайной генерации лабиринта* **maze\_wall** = **new int**[**width**][**height**];*//создается поле* **for**(**int** i=0;i<**width**\***height**\*0.5;i++){  
 **int** x = (**int**) (Math.*random*()\***width**);  
 **int** y = (**int**) (Math.*random*()\***height**);  
 **maze\_wall**[x][y]=1;*//прописывается стена* }  
 **maze\_wall**[0][0]=0;*//конечная точка помечается 0 чтобы не создалась стена* **maze\_wall**[**width**-1][**height**-1]=0;*//начальная точка помечается 0 чтобы не создалась стена* paint();*//рисуем* }  
  
 **void** on\_mouse\_clicked(**int** x,**int** y){*//метод обработки нажатия мыши на Canvas.* **int** coord\_x = (x\***width**)/*size\_canvas*;  
 **int** coord\_y = (y\***height**)/*size\_canvas*;  
 **if** ((coord\_x==0 && coord\_y==0) || (coord\_x==**width**-1 && coord\_y==**height**-1) || (**dont\_edit**)) **return**;  
 **if** (**maze\_wall**[coord\_x][coord\_y]!=0) **maze\_wall**[coord\_x][coord\_y] = 0;  
 **else maze\_wall**[coord\_x][coord\_y] = 1;*//чтобы менять наличие и отсутствие стеныв массиве(с 0 на 1 либо с 1 на 0)* **canvas**.paint\_selection(coord\_x, coord\_y,**maze\_wall**[coord\_x][coord\_y]!=0);*//прорисовываем стену* }  
  
 **void** set\_height(**int** height){ *// метод установки высоты* **this**.**height** = height;*//заносит новую высоту в наш класс* **maze\_wall** = **new int**[**width**][height];*//создает новый массив* **canvas**.set\_height(height);*//передаем новую высоту для отрисовки поля* }  
  
 **void** clear(){ *//очищение поля полностью* **stack**.clear();  
 **dont\_edit**=**false**;*//разрешаем рисовать стены* **maze\_wall** = **new int**[**width**][**height**];*//0 забиваеим массив==очищаем* **canvas**.paint\_canvas();*//прорисует начальное поле* }  
  
 **void** go\_maze(){ *//метод построения маршрута в лабиринте* **try** {  
 **dont\_edit**=**true**;*//непосещенные вершины* **maze\_wall**[**width**-1][**height**-1]=2;*//начальная точка помечается 2* Find\_way\_out(**width**-1, **height**-1);*//ищем путь из начальной точки* **if**(**maze\_wall**[0][0]!=2) {  
 JOptionPane.*showMessageDialog*(**this**, **"Лабиринт непроходим"**);  
 }  
 } **catch** (InterruptedException ex) {  
 Logger.*getLogger*(NewJFrame.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
  
 **boolean** check\_left(**int** x , **int** y){ *//метод проверки возможности движения влево* **if** (x<1) **return false**; *//если ограничения поля* **if** (**maze\_wall**[x-1][y]==0) **return true**;*//может* **else return false**;*//иначе стена* }  
  
  
 **boolean** check\_right(**int** x , **int** y){  
 **if** (x >**width**-2) **return false**;  
 **if** (**maze\_wall**[x+1][y]==0) **return true**;  
 **else return false**;  
 }  
  
  
 **boolean** check\_up(**int** x , **int** y){  
 **if** (y<1) **return false**;  
 **if** (**maze\_wall**[x][y-1]==0) **return true**;  
 **else return false**;  
 }  
  
 **boolean** check\_down(**int** x , **int** y){  
 **if** (y>**height**-2) **return false**;  
 **if** (**maze\_wall**[x][y+1]==0) **return true**;  
 **else return false**;  
 }  
  
  
 **void** Find\_way\_out(**int** x, **int** y) **throws** InterruptedException{ *//метод поиска пути* **if**(x==0 && y==0) **return**; *//если дошли до конца(выхода)* Thread.*sleep*(**delay**);*//задержка* Integer[] arr;  
 arr = **new** Integer[2];  
 arr[0]=x;  
 arr[1]=y;  
 **stack**.push(arr);*//добавили координаты в очередь* **if** (check\_up(x,y)){ *//если можем идти вверх* **maze\_wall**[x][y-1]=2;*//помечаем верхнюю ячейку как посещенную=2* **canvas**.draw\_rect(x,y-1,Color.***BLACK***,Color.***GREEN***);*//раскрашиваем ее в зеленый* Find\_way\_out(x,y-1);*//рекурсивно запускаем для этой ячейки метод поиска пути* **return**;  
 }  
  
 **if** (check\_left(x,y)){  
 **maze\_wall**[x-1][y]=2;  
 **canvas**.draw\_rect(x-1,y,Color.***BLACK***,Color.***GREEN***);  
 Find\_way\_out(x-1,y);  
 **return**;  
 }  
  
 **if** (check\_down(x,y)){  
 **maze\_wall**[x][y+1]=2;  
 **canvas**.draw\_rect(x,y+1,Color.***BLACK***,Color.***GREEN***);  
 Find\_way\_out(x,y+1);  
 **return**;  
 }  
  
 **if** (check\_right(x,y)){  
 **maze\_wall**[x+1][y]=2;  
 **canvas**.draw\_rect(x+1,y,Color.***BLACK***,Color.***GREEN***);  
 Find\_way\_out(x+1,y);  
 **return**;  
 }  
  
 **canvas**.draw\_rect(x,y,Color.***BLACK***,Color.***RED***);*//если идти некуда, раскрашиваем екрасным где стоим* **if** (**stack**.size()>1){ *//если очередь не пуста* arr = **stack**.pop();*//выталкиваем координаты следующей ячейки* arr = **stack**.pop();  
 Find\_way\_out(arr[0],arr[1]);*//запускаем метод поиска пути* }  
 }  
  
  
  
 **void** set\_width(**int** width){ *// метод установки ширины* **this**.**width** = width;  
 **maze\_wall** = **new int**[width][**height**];  
 **canvas**.set\_width(width);  
 }  
  
}

Мы работали на удаленном сервером GitHub. Классы NewJFrame.java и PaintCanvas.java представлены по https://github.com/OhrimukKat/Practice.

**Приложение А.**

**UML – диаграмма.**

Мы UML-диаграмма представлена на Рисунке 7.

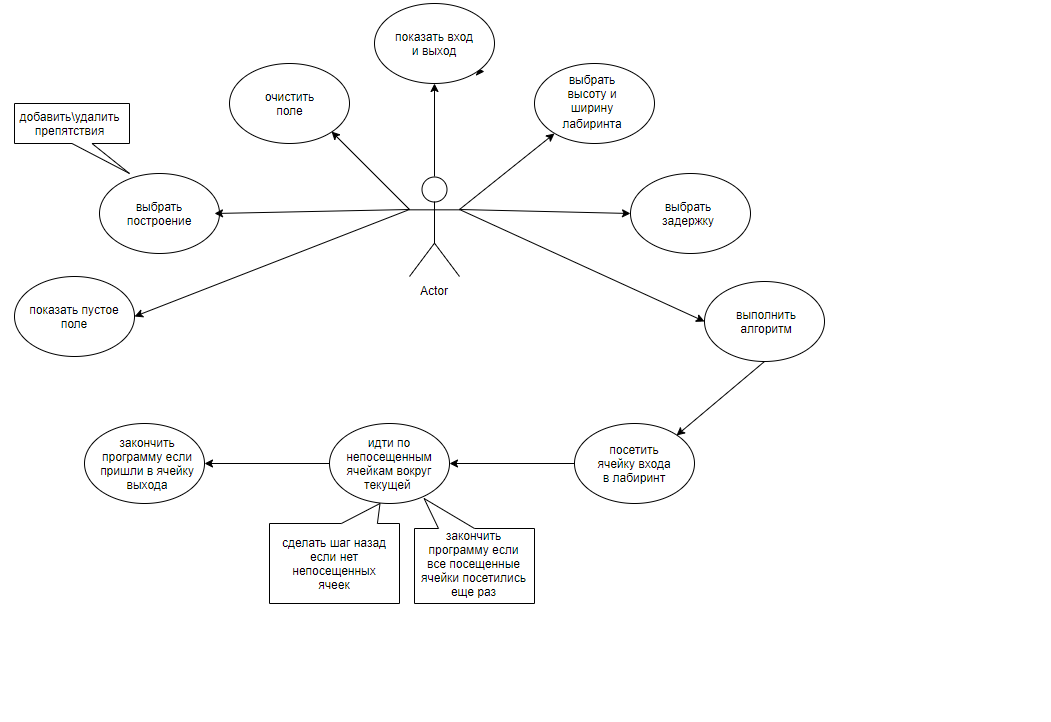


Рисунок 7 – UML – диаграмма