**Форма № Н-6.01**

ДОНБАССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(полное наименование высшего учебного заведения)

СПЕЦИАЛИЗИРОВАННЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ

(полное название кафедры, цикловой комиссии)

**КУРСОВОЙ ПРОЕКТ**

**(РАБОТА)**

по Технологии программирования

(название дисциплины)

по теме: Разработка программы для генерации лабиринта и поиску пути в нём

Студента III курса СКС-15 группы

Направления подготовки:

Компьютерная инженерия

Шишко А.А.

(фамилия и инициалы)

Руководитель зав. каф., к-т наук, доцент Гонтовой С.В.

(должность, ученое звание, ученая степень,фамилия и инициалы)

Национальная шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Количество баллов: \_\_\_\_\_\_\_Оценка: ECTS\_\_\_\_\_

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

г. Алчевск - 2017 год

РЕФЕРАТ

Изм.

Лист

№ докум.

Подпись

Дата

Лист

КР 6.050102.107.01.00.000 ПЗ

Разраб.

Шишко А.А.

Провер.

*Гонтовой С.В.*

Реценз.

Н. Контр.

Утверд.

**Пояснительная записка**

Лит.

Листов

**ДонГТУ СКС-15**

Пояснительная записка содержит: 53 стр., 11 табл., 12 рис. Цель курсового проекта - закрепление знаний и навыков, полученных при изучении курса «Технология программирования», а также демонстрация практических навыков при выполнении работы.

Предмет курсового проекта – разработка компьютерной программы, которая решает конкретную задачу, а также ее тестирование.

В работе изложены методы генерации лабиринта, расписаны все этапы создания программы, а также проведено ее тестирование. Данная программа написана на современном языке программирования высокого уровня Java.

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| 1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ |  |
| 1.1 Введение | 4 |
| 1.2 Требование к программе | 5 |
| 1.3 Условия эксплуатации | 6 |
| 1.4 Требование к программной документации | 7 |
| 1.5 Стадии и этапы разработки | 8 |
| 1.6 Порядок контроля и приёмки | 9 |
| 2 АЛГОРИТМ ЭЙЛЕРА |  |
| 2.1 Алгоритм Эйлера для генерации лабиринта | 10 |
| 2.2 Описание лабиринта | 11 |
| 2.3 Работа алгоритма на примере | 13 |
| 3 ВОЛНОВОЙ АЛГОРИТМ |  |
| 3.1 Понятие и применение | 19 |
| 3.2 Описание алгоритма | 20 |
| 4 РАЗРАБОТКА ПРОГРАММЫ |  |
| 4.1 Модель программы | 23 |
| 4.2 Структура проекта | 24 |
| 4.3 Основные алгоритмы программы | 28 |
| 4.4 Общие требование к программе | 40 |
| 5 ПРИЛОЖЕНИЕ |  |
| 5.1 Интерфейс программы | 41 |
| 5.2 Программный код | 43 |
| СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ | 53 |

1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1 Введение

1.1.1 Наименование программы

Наименование программы «Лабиринт»

1.1.2 Назначение и область применения

Программа предназначена для выполнения курсовой работы по предмету «Технология программирования». Она должна обеспечивать возможность выполнения перечисленных ниже функций:

- Сгенерировать лабиринт по заданному алгоритму

- Найти выход из лабиринта и выдать ответ в виде списка координат

1.2 Требование к программе

1.2.1 Требование к функциональным характеристикам

Генерируемый лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квад­рат определяется его координатами в матрице.

Программа находит проход через лабиринт, двигаясь от заданного входа. После отыскания прохода программа выводит найденный путь в виде координат квад­ратов.

1.2.2 Требование к надёжности

1.2.2.1 Требование к обеспечению надёжного функционирования программы

Программа будет надёжно функционировать при бесперебойном питании и отсутствии вирусов на вычислительной машине.

1.2.2.2 Время восстановления после отказа

Время отказа программы из-за каких-либо технических неполадок некритично. Программу можно запустить заново и продолжить её тестирование.

1.2.2.3 Отказы из-за некорректных действий пользователей системы

Программа полностью защищена от некорректных действий со стороны пользователя через интерфейс.

1.3 Условия эксплуатации

1.3.1 Климатические условия эксплуатации

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

1.3.2 Требование к составу и параметрам технических средств

Для запуска программы на электронно-вычислительной машине должен быть установлен пакет Java 8.

1.4 Требование к программной документации

1.4.1 Предварительный состав программной документации

Предварительный состав программной документации должен включать в себя:

- Техническое задание;

- Теоретические сведения;

- Практическая часть;

- Методика испытаний.

1.5 Стадии и этапы разработки

1.5.1 Стадии разработки

Разработка должна быть проведена в 2 стадии: разработка технического задания; рабочее проектирование.

1.5.2 Этапы разработки

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.   
На стадии рабочего проектирования должны быть выполнены перечисленные этапы работ: разработка программы; разработка программной документации; испытания программы.

5.3 Содержание работ по этапам

На этапе разработки технического задания необходимо определить требование к программе, к техническим средствам, общие стадии и этапы разработки программы. На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы. На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации. На этапе испытания программы необходимо протестировать работоспособность программы.

1.6 Порядок контроля и приёмки

1.6.1 Общие положения

Приёмка программы должна осуществляться заказчиком. Программа считается выполненной, если она удовлетворяет всем пунктам данного технического задания.

2 АЛГОРИТМ ЭЙЛЕРА

2.1 Алгоритм Эйлера для генерации лабиринта

Алгоритм Эйлера создает «совершенные» лабиринты, имеющие только один путь между любыми двумя ячейками, по одной строке за раз. Сам алгоритм невероятно быстрый и гораздо эффективнее памяти, чем другие популярные алгоритмы (такие как Prim и Kruskal), требующие хранения, пропорционального только одной строке. Это позволяет создавать лабиринты неопределенной длины на системах с ограниченной памятью.

2.2 Описание алгоритма

Замечание: мы предполагаем, что самая левая ячейка имеет границу слева, а самая правая — справа.

* + 1. Создайте первую строку. Ни одна ячейка не будет являться частью ни одного множества.
    2. Присвойте ячейкам, не входящим в множество, свое уникальное множество.
    3. Создайте правые границы, двигаясь слева направо:
       1. Случайно решите добавлять границу или нет.
          1. Если текущая ячейка и ячейка справа принадлежат одному множеству, то создайте границу между ними (для предотвращения зацикливаний).
          2. Если вы решили не добавлять границу, то объедините два множества в которых находится текущая ячейка и ячейка справа.
    4. Создайте границы снизу, двигаясь слева направо:
       1. Случайно решите добавлять границу или нет. Убедитесь, что каждое множество имеет хотя бы одну ячейку без нижней границы (для предотвращения изолирования областей).
          1. Если ячейка в своем множестве одна, то не создавайте границу снизу.
          2. Если ячейка одна в своем множестве без нижней границы, то не создавайте нижнюю границу.
    5. Решите, будете ли вы дальше добавлять строки или хотите закончить лабиринт.
       1. Если вы хотите добавить еще одну строку, то:
          1. Выведите текущую строку.
          2. Удалите все правые границы.
          3. Удалите ячейки с нижней границей из их множества.
          4. Удалите все нижние границы.
          5. Продолжайте с шага 2.
       2. Если вы решите закончить лабиринт, то:.
          1. Добавьте нижнюю границу к каждой ячейке.
          2. Двигаясь слева направо:

Если текущая ячейка и ячейка справа члены разных множеств, то:

Удалите правую границу.

Объедините множества текущей ячейки и ячейки справа.

Выведите завершающую строку.

2.3 Работа алгоритма на примере

Пример здесь создаст прямоугольный лабиринт неопределенной длины. Мы создадим лабиринт по одной строке за раз, начиная с верхнего ряда, и двигаемся слева направо между ячейками. Каждая ячейка в строке будет назначена набору. Мы будем представлять это здесь, перечисляя множества и записывая номер набора, принадлежащего ячейке внутри каждой ячейки. Каждая ячейка может иметь стену справа и внизу. Мы предположим, что стена существует слева от самой левой ячейки и справа от самой правой ячейки в каждом ряду и что стена существует в верхней части всех ячеек в первой строке.

2.3.1 Создадим первую строку. Это будет просто пустая строка:

|  |
| --- |
|  |

2.3.2 Присоединим все ячейки, не являющиеся членами набора, к своему уникальному набору.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

2.3.3 Создадим границы справа

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ( 1 | 2 ) | 3 | 4 | 5 | 6 | 7 | 8 |

Если мы решили не создавать границу, то объединим множества

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | ( 1 | 3 ) | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |  |  |
| 1 | 1 | ( 1 | 4 ) | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |

2.3.4 Создадим границы снизу

Убедитесь, что каждый комплект имеет по меньшей мере одну ячейку с нижним переходом (то есть без нижней стенки). Невыполнение этого требования приведёт к изоляции.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |

2.3.5 Создаём новую строку

Предыдущая строка стала текущей.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |

Удалим стенки справа

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |

Если ячейка имеет границу снизу, удалим её из множества

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |
| 1 |  |  | 4 |  | 6 | 6 |  |

Удалим нижние границы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 4 | 4 | 6 | 6 | 6 |
| 1 |  |  | 4 |  | 6 | 6 |  |

Продолжая с пункта 2:

Присоединим ячейки, не принадлежащие множествам к своим уникальным множествам

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 6 | 7 |

Продолжая с пункта 3: добавляем границы справа

Добавлена граница 1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| ( 1 | 2 ) | 3 | 4 | 5 | 6 | 6 | 7 |

Граница не добавлена, объединим множества 2 и 3:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | ( 2 | 3 ) | 4 | 5 | 6 | 6 | 7 |

Добавлена граница 2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 2 | ( 2 | 5 ) | 6 | 6 | 7 |

Добавлена граница 3:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 2 | 2 | ( 5 | 6 ) | 6 | 7 |

Следующие две ячейки являются членами одного и того же набора, поэтому мы ДОЛЖНЫ добавить стену. Невыполнение этого правила создаст петли в нашем лабиринте.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 2 | 2 | 5 | ( 6 | 6 ) | 7 |

Продолжая с пункта 4: создаём нижние границы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 2 | 2 | 5 | 6 | 6 | 6 |

По крайней мере одна ячейка из каждого набора должна иметь нижний проход (т.е. не должна иметь нижнюю стенку).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 1 | 2 | 2 | 2 | 5 | 6 | 6 | 6 |

2.3.6 Завершение лабиринта.

Последний ряд отличается от регулярной строки двумя способами: 1) каждая ячейка имеет нижнюю стенку и 2) каждая ячейка должна быть членом одного и того же набора.

Сделать каждую часть ячейки одного и того же набора просто. Просто удалим стены между ячейками, которые являются членами разных наборов, пока все ячейки не станут частью одного и того же набора. Не удаляйте стену, если она разделяет две ячейки, которые являются членами одного и того же набора.

Начнем с создания обычной строки и добавления нижней границы к каждой ячейке.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 1 | 1 | 3 | 3 | 7 | 7 | 8 | 8 |

Закончим лабиринт, разрушая границы между ячейками, принадлежащими различным множествам и объединяя их в одно.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 1 | ( 1 | 3 ) | 3 | 7 | 7 | 8 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 1 | 1 | 1 | ( 1 | 7 ) | 7 | 8 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 1 | 1 | 1 | 1 | 1 | ( 1 | 8 ) | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

В конце должен получиться идеальный лабиринт, в котором нет циклов (между двумя ячейками есть только один путь) и изолированных частей (ячейки или групп ячеек, которые не связаны с другими частями лабиринта). Теперь мы можем назначить любые две ячейки соответственно «входом» и «выходом». [1]

3 ВОЛНОВОЙ АЛГОРИТМ

3.1 Понятие и применение

Алгоритм волновой трассировки волновой алгоритм, алгоритм Ли — алгоритм поиска пути, алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину

В основном используется при компьютерной трассировке разводке печатных плат, соединительных проводников на поверхности микросхем. Другое применение волнового алгоритма — поиск кратчайшего расстояния на карте в компьютерных стратегических играх

Волновой алгоритм в контексте поиска пути в лабиринте был предложен Э Ф Муром. Ли независимо открыл этот же алгоритм при формализации алгоритмов трассировки печатных плат в 1961 году. [2]

3.2 Описание алгоритма

Алгоритм работает на дискретном рабочем поле ДРП, представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные Множество всех ячеек ДРП разбивается на подмножества: «проходимые» свободные, т е при поиске пути их можно проходить, «непроходимые» препятствия, путь через эту ячейку запрещён, стартовая ячейка источник и финишная приемник Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости6

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки

Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке

Соседние ячейки принято классифицировать двояко: в смысле окрестности Мура и окрестности фон Неймана, отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1 Каждая ячейка, меченая числом шагов от стартовой ячейки становится стартовой и из неё порождаются очередные шаги в соседние ячейки Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек6 Трасс с минимальной числовой длиной пути, как при поиске пути в окрестностях Мура, так и фон Неймана может существовать несколько Выбор окончательного пути в приложениях диктуется другими соображениями, находящимися вне этого алгоритма Например, при трассировке печатных плат — минимумом линейной длины проложенного проводника. Псевдокод алгоритма представлен в таблице 3.1.1. [3]

Таблица 3.2.1 – Псевдокод

|  |  |
| --- | --- |
| Инициализация | |
| Пометить стартовую ячейку как 0  d:=0 | |
| Распространение волны | Восстановление волны |
| ЦИКЛ  ДЛЯ каждой ячейки loc, помеченной числом d:  - пометить все соседние свободные непомеченные ячейки числом d + 1;  КЦ  d := d + 1;  ПОКА (финишная ячейка не помечена) И (есть возможность распространения волны). | ЕСЛИ финишная ячейка помечена  ТО  перейти в финишную ячейку  ЦИКЛ  выбрать среди соседних ячейку, помеченную числом на 1 меньше числа в текущей ячейке  перейти в выбранную ячейку и добавить её к пути  ПОКА текущая ячейка — не стартовая  ВОЗВРАТ путь найден  ИНАЧЕ  ВОЗВРАТ путь не найден |

4 РАЗРАБОТКА ПРОГРАММЫ

4.1 Модель программы

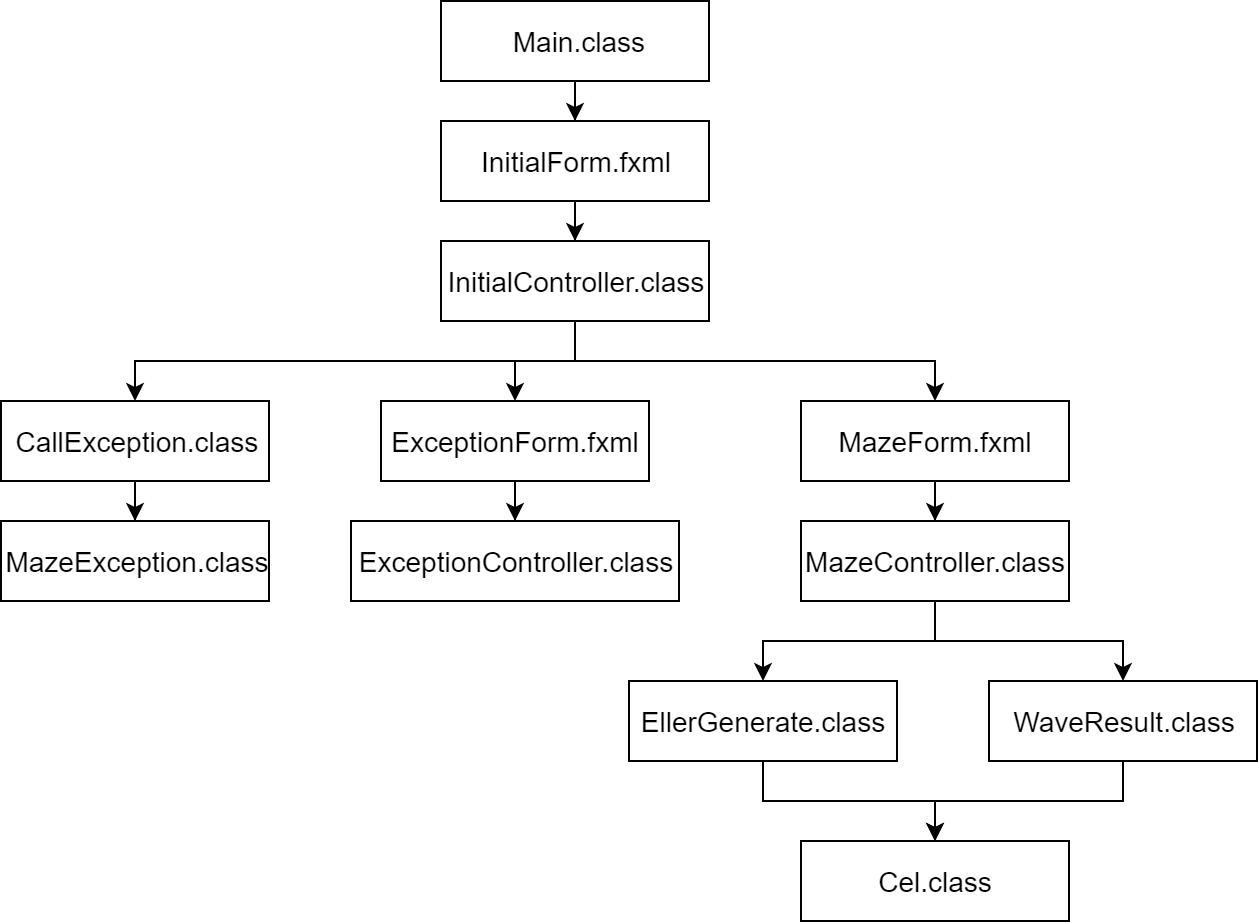


Рисунок 4.1.1 – Модель программы

Модель программы, изображённая на рис. 4.1.1, показывает очерёдность вызовов классов и файлов, отвечающих за интерфейс программы. С класса Main происходит вызов главной формы InitialForm, которая в свою очередь сразу вызывает класс, который является контроллером к ней. В случае самостоятельного ввода размер лабиринта происходит вызов класса CallException, и если ввод был некорректный, то происходит вызов ExceptionForm с контроллером, а если всё было введено правильно то запускается форма MazeForm. MazeController, в свою очередь, вызывает класс для генерации EllerGenerate и для прохождения WaveResult. Оба эти класса исспользуют Cel как класс-структуру.

4.2 Структура проекта

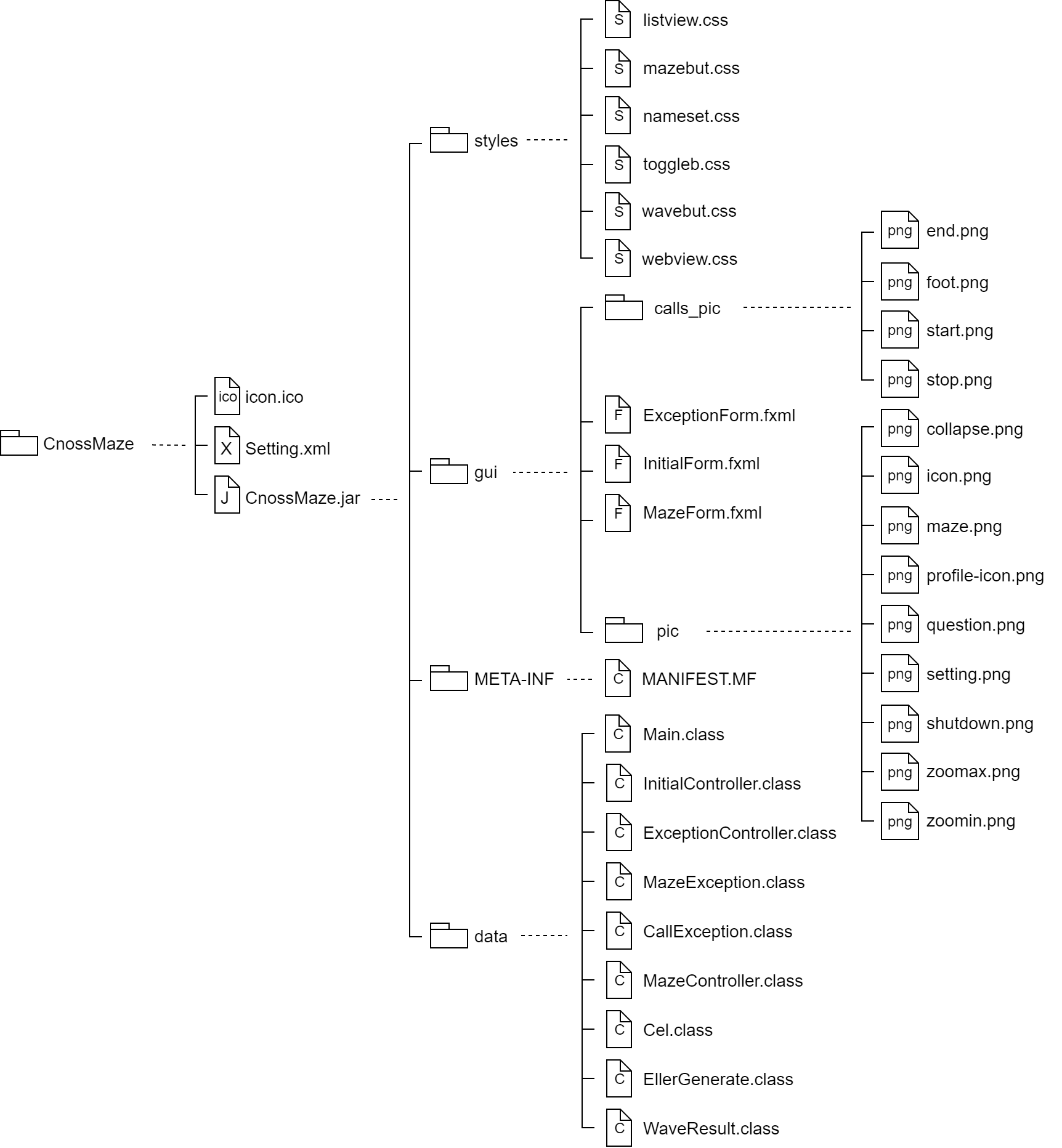


Рисунок 4.2.1 – Структура классов и файлов проекта

Описание классов и файлов программы приведено в таблице 4.2.1.

Таблица 4.2.1 – Описание классов и файлов проекта

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| Каталог CnossMaze | | |
| CnossMaze | .jar | Исполняемый файл |
| Setting | .xml | Файл разметки для сохранения настроек программы |
| icon | .ico | Иконка для ярлыка |
| Состав архива CnossMaze.jar | | |
| Пакет data | | |
| Main | .class | Вызов главной формы InitialForm |
| InitialController | .class | Обработка событий на форме InitialForm |
| ExceptionController | .class | Обработка событий на форме ExceptionForm |
| MazeException | .class | Класс по обработке исключений. Содержит код по обработке исключения |
| CallException | .class | Класс для вызова исключений |
| MazeController | .class | Обработка событий на форме MazeForm |
| Cel | .class | Класс выполняющий роль структуры, для хранения информации о ячейке |
| EllerGenerate | .class | Класс генерации лабиринта |
| WaveResult | .class | Класс прохождения лабиринта |
| Пакет gui | | |
| InitialForm | .fxml | Структура интерфейса главной формы |
| MazeForm | .fxml | Структура интерфейса вызываемой формы для генерации и прохождения лабиринта |
| ExceptionForm | .fxml | Структура интерфейса формы для отображения исключения |
| pic | package | Изображения и иконки для элементов |
| calls\_pic | package | Вызываемые иконки при генерации лабиринта |

Продолжение таблицы 4.2.1

|  |  |  |
| --- | --- | --- |
| Пакет styles | | |
| listview | .css | Файл описания внешнего вида списка координат |
| mazebut | .css | Файл описания внешнего вида кнопок на главной форме |
| nameset | .css | Файл описания внешнего вида переключателей на главной форме в разделе «Лабиринт» |
| togglebut | .css | Файл описания внешнего вида переключателя на главной форме в разделе «Настройки» |
| wavebut | .css | Файл описания внешнего вида кнопок на вторичной форме |
| webview | .css | Файл описания внешнего вида веб-контейнера, содержащего лабиринт |
| Пакет META-INF | | |
| MANIFEST | .mf | Манифест. Содержит информацию о программе и создаётся автоматически |

Функции и их описание предоставлено на таблице 4.2.2

Таблица 4.2.2 – Функции основных классов и их описание

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Доступ | Описание |
| Класс EllerGenerate | | | |
| driver | void | public | Основная функция класса, из неё вызываются все методы |
| getMaze | int[][] | public | Возвращает лабиринт, где: -1 граница, 0 пустая клетка |
| makeSet | Cel[] | private | Группирует ячейки для создания новых наборов |
| makeRightWalls | Cel[] | private | Создаёт границы справо |
| merge | Cel[] | private | Объединяет множества |
| isContainsInList | boolean | private | Проверка на принадлежность ячейки множеству |

Продолжение таблицы 4.2.2

|  |  |  |  |
| --- | --- | --- | --- |
| isNotDone | boolean | private | Проверяет элемент множества на наличие границы снизу |
| makeDown | Cel[] | private | Создаёт границы снизу |
| end | Cel[] | private | Функция завершения алгоритма |
| findPos | int | private | Ищет элемент в множестве |
| genNextRow | Cel[] | private | Убирает границы и принадлежность ячеек к множествам |
| printMaze | void | private | Отображает границы в массиве |
| Класс WaveResult | | | |
| generateWave | void | public | Основная функция класса, из неё вызываются все методы |
| getResultList | List<Cel> | public | Возвращает список координат |
| getMaze | int[][] | public | Возвращает лабиринт, где: -1 граница, 0 пустая клетка, 1 путь выхода |
| motionWave | void | private | Распространение волны |
| returnWave | void | private | Восстановление волны |

4.3 Основные алгоритмы программы

Основной алгоритм программы изображён на рис. 4.3.1 и описан в таб. 4.3.1:

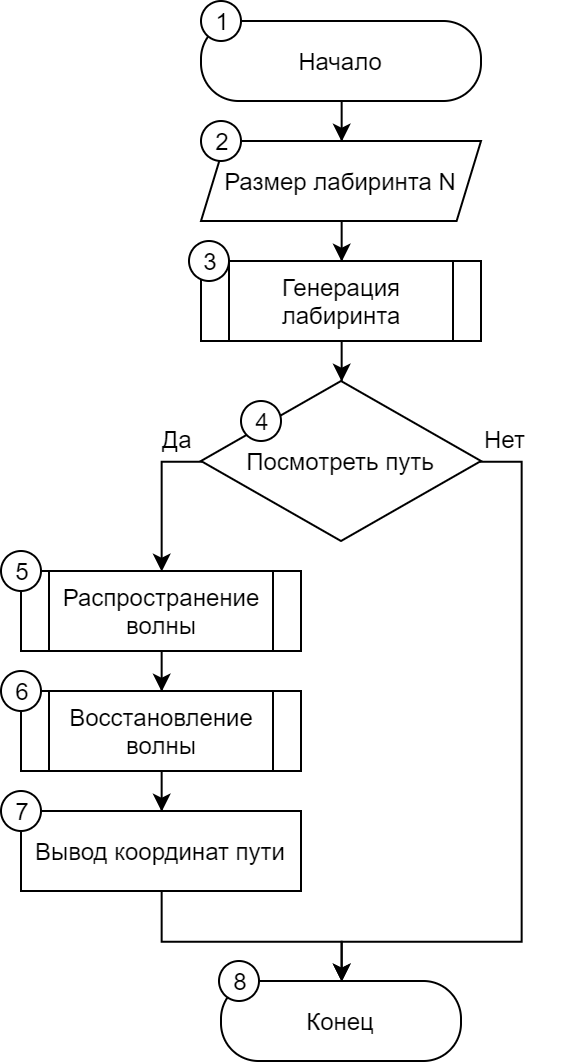


Рисунок 4.3.1 – Алгоритм основной программы

Таблица 4.3.1 – Приложение к основному алгоритму программы

|  |  |
| --- | --- |
| **№** | **Назначение** |
| 2 | Ввод конфигурации лабиринта |
| 3 | Алгоритм генерации лабиринта |
| 4 | Запрос просмотра координат выхода из лабиринта |
| 5 | Алгоритм распространения волны |
| 6 | Алгоритм восстановления волны |
| 7 | Вывод результата |

Алгоритм Эйлера создает «совершенные» лабиринты, имеющие только один путь между любыми двумя ячейками, по одной строке за раз. Основной алгоритм генерации лабиринта представлен в на рис. 4.3.2 и описан в таб. 4.3.2.

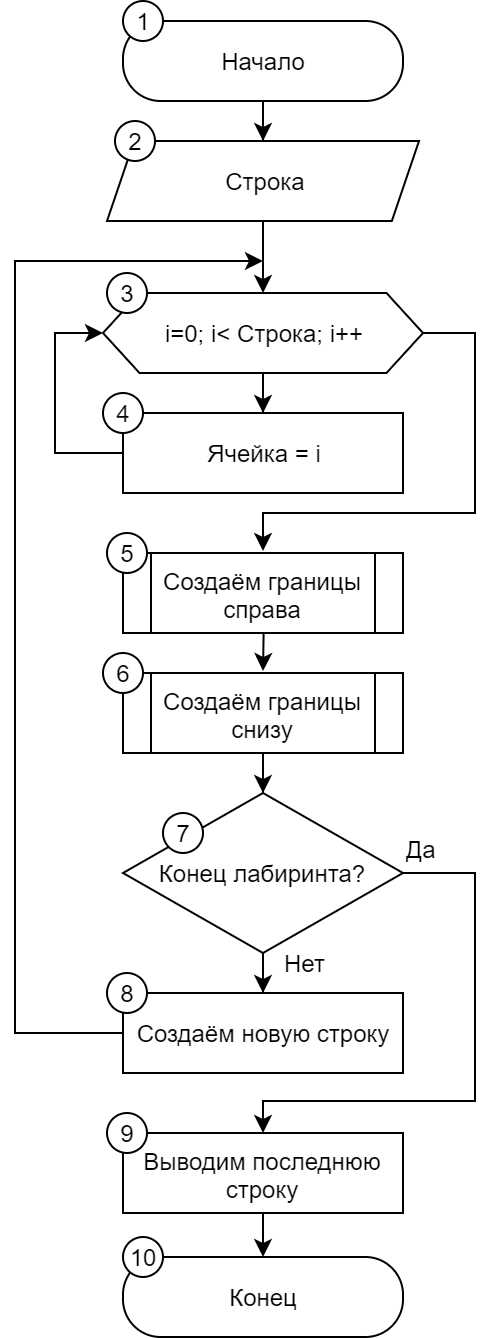


Рисунок 4.3.2 – Основной алгоритм генерации лабиринта

Таблица 4.3.2 – Приложение к основному алгоритму генерации лабиринта

|  |  |
| --- | --- |
| **№** | **Назначение** |
| 2 | Создаём первую строку. Ни одна ячейка не будет являться частью ни одного множества. |
| 3 | Цикл. Охватывает каждую ячейку строки |
| 4 | Присвоим ячейкам, не входящим в множество, свое уникальное множество. |
| 5 | Алгоритм создания границ справа |
| 6 | Алгоритм создания границ снизу |
| 7 | Если хотим закончить лабиринт |
| 8 | Создаём новую строку |
| 9 | Выводим строку и завершаем лабиринт |

Алгоритм создания границ справа представлен в на рис. 4.3.3 и описан в таб. 4.3.3.

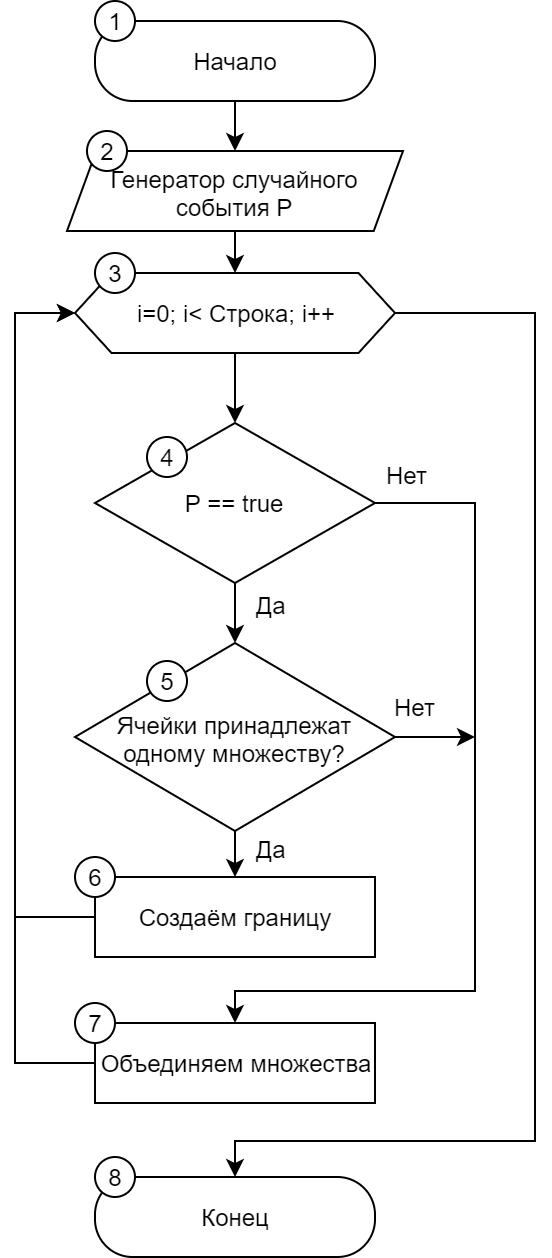


Рисунок 4.3.3 – Алгоритм создания границ справа

Таблица 4.3.3 – Приложение к алгоритму создания границ справа

|  |  |
| --- | --- |
| **№** | **Назначение** |
| 2 | Инициализируем генератор случайного события |
| 3 | Цикл. Охватывает каждую ячейку строки |
| 4 | Случайное событие true |
| 5 | Если текущая ячейка и ячейка справа принадлежат одному множеству |
| 6 | Создаём границу между ними |
| 7 | Объединяем два множества в которых находится текущая ячейка и ячейка справа |

Алгоритм создания границ снизу представлен в на рис. 4.3.4 и описан в таб. 4.3.4.

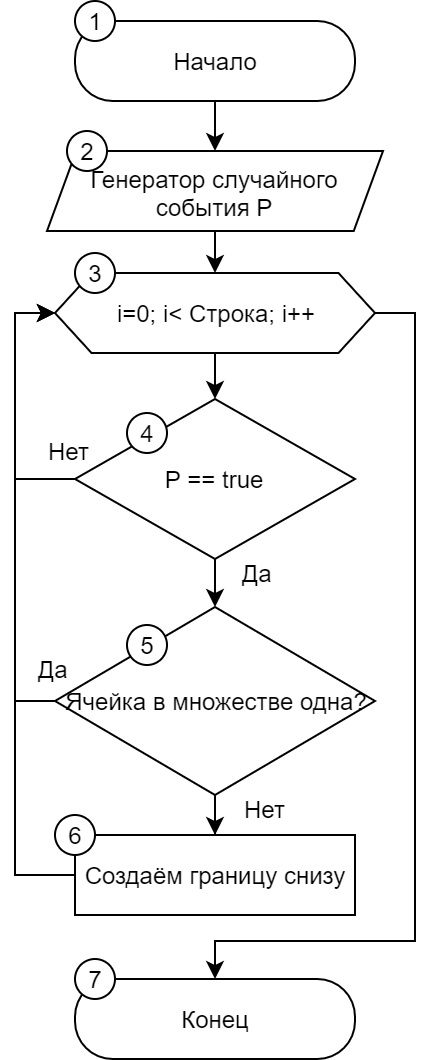


Рисунок 4.3.4 – Алгоритм создания границ снизу

Таблица 4.3.4 – Приложение к алгоритму создания границ справа

|  |  |
| --- | --- |
| **№** | **Назначение** |
| 2 | Инициализируем генератор случайного события |
| 3 | Цикл. Охватывает каждую ячейку строки |
| 4 | Случайное событие true |
| 5 | Если текущая ячейка одна в своём множестве |
| 6 | Создаём границу снизу |

Волновой позволяет построить путь между двумя элементами в любом лабиринте. Сам процесс делится на два этапа.

1. Распространения волны. Из начального элемента распространяется в 4-х направлениях волна. Элемент, в который пришла волна образует фронт волны.

Каждый элемент первого фронта волны является источником вторичной волны. Элементы второго фронта волны генерируют волну третьего фронта и т.д. Процесс продолжается до тех пор, пока не будет достигнут конечный элемент.

Алгоритм распространения волны изображён на рис.4.3.5 и описан в таб. 4.3.5.

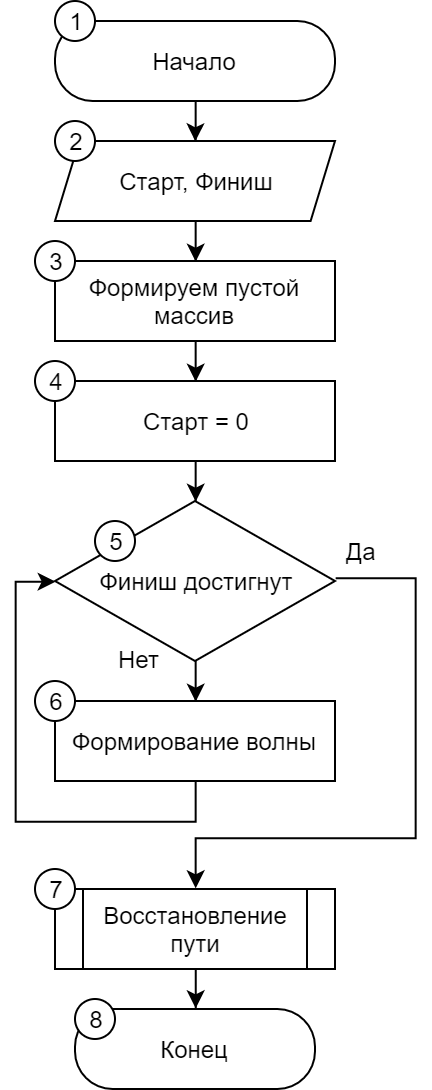


Рисунок 4.4.5 – Алгоритм распространения волны

Таблица 4.4.5 – Приложение к алгоритму распространения волны

|  |  |
| --- | --- |
| **№** | **Назначение** |
| 2 | Инициализация клеток начала и конца пути |
| 3 | Формируем пустой массив размером в массив размера карты лабиринта (пустая карта без препятствий) |
| 4 | Стартовую ячейку помечаем как 0 |
| 5 | Распространение волн происходит до тех пор, пока финиш не будет достигнут |
| 6 | Каждая волна распространяется на четыре стороны и охватывает по одной линии клеток каждый раз. Все соседние клетки заполнятся числом d=d+1 |

2. Восстановление волны. Строится сама трасса. Её построение осуществляется от конечного элемента к начальному.

Алгоритм восстановления волны представлен на рис. 4.4.6 и описан в таб. 4.4.6.

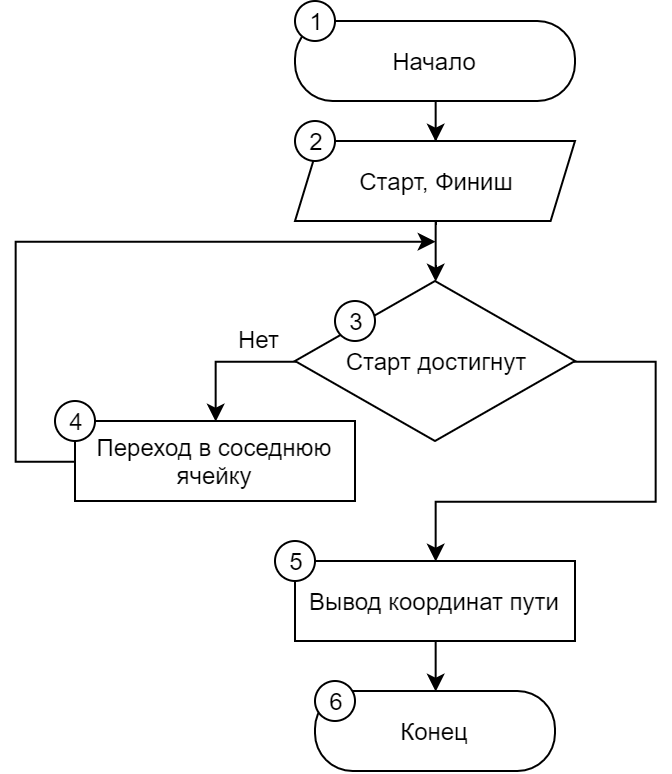


Рисунок 4.4.6 – Алгоритм восстановления волны

Таблица 4.4.5 – Приложение к алгоритму распространения волны

|  |  |
| --- | --- |
| **№** | **Назначение** |
| 2 | Инициализация клеток начала и конца пути |
| 3 | Идём начиная от финиша до старта. При достижение клетки старта цикл останавливается |
| 4 | Движение по лабиринту осуществляется в соответствии в расставленными приоритетами при распространении волны. Начиная от финиша, индекс в соседней клетке должен быть меньше чем в текущей на 1 |
| 5 | Выводим из стека координаты пути |
| 6 | Инициализация клеток начала и конца пути |

4.5 Общие требования к программе

Программа реализует генерацию лабиринта и поиск выхода в нём на языке программирования Java. Интерфейс программы реализован как отдельное меню, которое открывает панели с контентом на прозрачной форме. Меню содержит в себе элементы «О программе», «Лабиринт», «Настройки» и кнопку выхода. В «настройках» можно свернуть программу и выбрать тип запуска программы по умолчанию. В панели интерфейса «Лабиринт» можно выбрать размер лабиринта или ввести самому, после чего сгенерировать его нажав на соответствующую кнопку.

При нажатии на кнопку «Сгенерировать» вызывается новое окно, которое содержит сгенерированный лабиринт и кнопку «Пройти». Последняя в свою очередь проходит лабиринт и выдаёт список координат.

Программа имеет следующие системные требования, приведенные в таблице 4.5.1

Таблица 4.5.1 – Системные требования программы

|  |  |
| --- | --- |
| **Процессор:** | 1 ГГц |
| **ОЗУ:** | 128 Мбайт |
| **Место на жестком диске:** | 150 Кбайт |
| **Экран:** | 800x600 и выше |
| Установленная версия Java**:** | 1.8+ |
| Операционная система: | Windows XP, Vista, Windows 7, 8, 8.1, 10 |

5 ПРИЛОЖЕНИЕ

5.1 Интерфейс программы



Рисунок 5.1 – Окно «О программе»

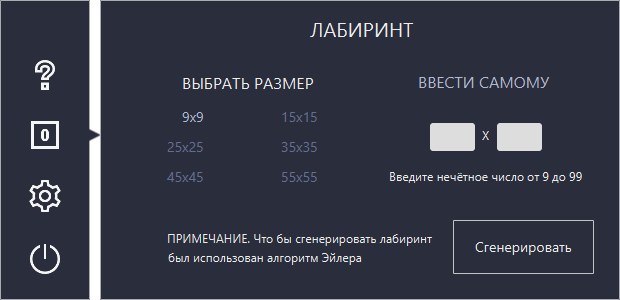


Рисунок 5.2 – Окно «Лабиринт»

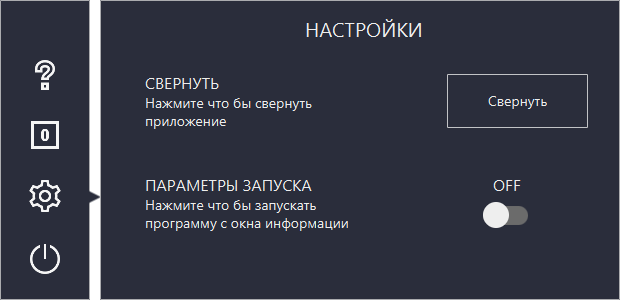


Рисунок 5.3 – Окно «Настройки»

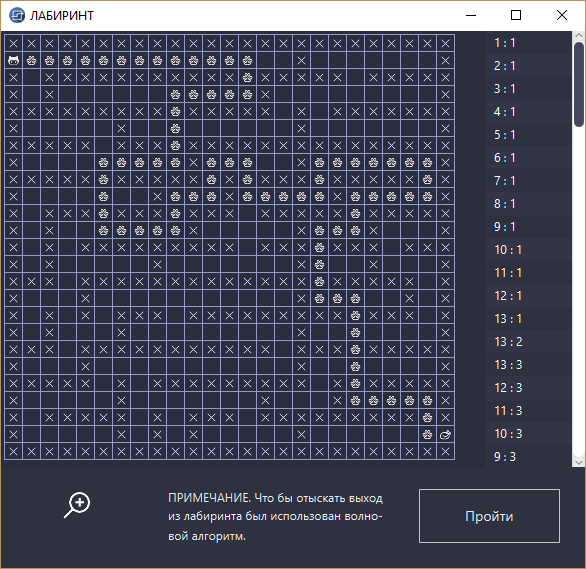


Рисунок 5.3 – Результат работы алгоритмов

5.2 Программный код

Листинг 5.2.1 – Исходный код класса EllerGenerate

package data;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
  
/\*\*  
 \* **@author:** Schischko A.A.  
 \* **@version:** 1.0  
 \* **@date:** 01.11.2017  
 \* Класс EllerGenerate предназначен для генерации лабиринта  
 \* он принимает в себе значение size и возращает готовый массив.  
 \* Вызываемые методы:  
 \* EllerGenerate(int s) - конструктор, принимает размер  
 \* drive() - генерация лабиринта  
 \* getMaze() - возвращает лабиринт, где:  
 \* -1: граница  
 \* 0: пустая клетка  
 \*/  
  
public class EllerGenerate {  
 private static Random randomizer = new Random();  
 private static int size;  
 private static int[][] maze;  
  
 EllerGenerate(int s)  
 {  
 size = s; size--;  
 }  
  
 /\*\*  
 \* Группируем негруппированные ячейки для создания новых наборов  
 \* **@param** row - ряд  
 \* **@return** ряд с набором  
 \*/  
 private Cel[] makeSet(Cel[] row) {  
 for(int index = 0; index < row.length; ) {  
 Cel cell = row[index++];  
 if(cell.set == null) {  
 List<Cel> list = new ArrayList<Cel>();  
 list.add(cell);  
 cell.set=list;  
 }  
 }  
 return row;  
 }  
  
 /\*\*  
 \* Функция создания границ справо  
 \* Случайно решаем добавлять границу или нет. Если текущая ячейка и ячейка справа  
 \* принадлежат одному множеству, то создаём границу между ними.  
 \* Если мы решили не добавлять границу, то объединяем два множества (merge), в которых  
 \* находится текущая ячейка и ячейка справа  
 \* **@param** row - ряд  
 \* **@return** ряд с границами справо  
 \*/  
 private Cel[] makeRightWalls(Cel[] row) {  
 for(int i = 1; i < row.length; i++) {  
 if(isContainsInList(row[i-1].set,row[i])) {  
 row[i-1].right=true;  
 continue;  
 }  
 if(randomizer.nextBoolean())  
 row[i-1].right=true;  
 else  
 row=merge(row,i);  
 }  
 return row;  
 }  
  
 /\*\*  
 \* Функция поглощения  
 \* Объединяем два можества  
 \* **@param** row - множество  
 \* **@param** i - индекс элемента  
 \* **@return** объединённое множество  
 \*/  
 private Cel[] merge(Cel[] row,int i) {  
 List<Cel> currentList = row[i-1].set;  
 List<Cel> nextList = row[i].set;  
 for(Cel j : nextList)  
 {  
 currentList.add(j);  
 j.set=currentList;  
 }  
 return row;  
 }  
  
 /\*\*  
 \* Функция проверки принадлежности ячейки множеству  
 \* **@param** set - проверяемое множество  
 \* **@param** cell - проверяемая ячейка  
 \* **@return** логический ответ  
 \*/  
 private boolean isContainsInList(List<Cel> set,Cel cell) {  
 for(Cel i : set) {  
 if(i==cell)  
 return true;  
 }  
 return false;  
 }  
  
 /\*\*  
 \* Вспомогательная функция  
 \* Проверяем первый элемент множества на наличие границы снизу  
 \* **@param** set - проверяемое множество  
 \* **@return** логический ответ  
 \*/  
 private boolean isNotDone(List<Cel> set){  
 boolean rslt=true;  
 rslt=rslt&&set.get(0).down;  
 return rslt;  
 }  
  
 /\*\*  
 \* Функция создания границ снизу  
 \* Создаём нижнюю границу у всех ячеек. После убираем их  
 \* случайным образом до тех пор, пока первый элемент  
 \* множества не останется без границы. Таким образом,  
 \* если ячейка в своём множестве одна, то она не будет иметь  
 \* границу снизу.  
 \* **@param** row - ряд  
 \* **@return** ряд с нижними границами  
 \*/  
 private Cel[] makeDown(Cel[] row){  
 for(int i=0;i<row.length;i++){  
 for(Cel x:row[i].set) x.down = true;  
 while(isNotDone(row[i].set)){  
 do{  
 row[i].set.get(randomizer.nextInt(row[i].set.size())).down=false;  
 }while(randomizer.nextBoolean());  
 }  
 }  
 return row;  
 }  
  
 /\*\*  
 \* Основная функция для генерации лабиринта.  
 \* Из неё вызываются все методы и строится  
 \* лабиринт  
 \*/  
 public void driver(){  
 maze=new int[size+1][size+1];  
 Cel[] cur=new Cel[size/2];  
 for(int i=0;i<size/2;i++)  
 cur[i]=new Cel(0,i);  
 for(int i=0;i<size/2;i++){  
 cur=makeSet(cur);  
 cur=makeRightWalls(cur);  
 cur=makeDown(cur);  
 if(i==size/2-1)  
 cur=end(cur);  
 printMaze(cur,i);  
 if(i!=size/2-1)  
 cur=genNextRow(cur);  
 }  
 for(int i=0;i<=size;i++)  
 maze[i][0]=maze[0][i]=maze[i][size]=maze[size][i]=-1;  
 for(int i=2;i<=size;i+=2)  
 for(int j=2;j<=size;j+=2)  
 maze[i][j]=-1;  
 }  
  
 /\*\*  
 \* Функция завершения алгоритма  
 \* Удаляем границы справо и объединяем множества  
 \* **@param** row - ряд  
 \* **@return** - завершающий ряд лабиринта  
 \*/  
 private Cel[] end(Cel[] row)  
 {  
 for(int i = 1; i < row.length; i++)  
 {  
 if(findPos(row[i-1].set,row[i]) == -1)  
 {  
 row[i-1].right=false;  
 row=merge(row,i);  
 }  
 }  
 return row;  
 }  
  
 /\*\*  
 \* Вспомогательная функция  
 \* Ищем элемент в множестве  
 \* **@param** set - множество  
 \* **@param** x - искомая ячейка  
 \* **@return** позиция  
 \*/  
 private int findPos(List<Cel> set, Cel x){  
 Cel[] tmpArray = new Cel[set.size()];  
 tmpArray = set.toArray(tmpArray);  
 for(int i=0;i<tmpArray.length;i++)  
 if(tmpArray[i]==x)  
 return i;  
 return -1;  
 }  
  
 /\*\*  
 \* Функция продолжения алгоритма  
 \* Убираем границы у ячеек и их принадлежность  
 \* к множествам  
 \* **@param** pre - ряд  
 \* **@return** ряд без множеств и границ  
 \*/  
 private Cel[] genNextRow(Cel[] pre){  
 for(int i = 0; i < pre.length;i++ ) {  
 pre[i].right=false;  
 pre[i].x++;  
 if(pre[i].down) {  
 pre[i].set.remove(findPos(pre[i].set, pre[i]));  
 pre[i].set=null;  
 pre[i].down=false;  
 }  
 }  
 return pre;  
 }  
  
 /\*\*  
 \* Функция отображения границ  
 \* Отображаем все сгенерированные границы  
 \* **@param** row - ряд  
 \* **@param** rowPos - позиция  
 \*/  
 private void printMaze(Cel[] row,int rowPos){  
 rowPos=2\*rowPos+1;  
 for(int i=0;i<row.length;i++){  
 if(row[i].right)  
 maze[rowPos][2\*i+2]=-1;  
 if(row[i].down)  
 maze[rowPos+1][2\*i+1]=-1;  
 }  
 }  
  
 /\*\*  
 \* Возвращаем сгенерированный лабиринт, в котором  
 \* -1: граница  
 \* 0: пустая клетка  
 \* **@return** лабиринт  
 \*/  
 public int[][] getMaze()  
 {  
 return maze;  
 }  
}

Листинг 5.2.2 – Исходный код класса WaveResult

package data;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\*\*  
 \* @author: Schischko A.A.  
 \* @version: 1.0  
 \* @date: 01.11.2017  
 \* Класс WaveResult реализует волновой алгоритм для поиска выхода из лабиринта  
 \* Вызываемые методы:  
 \* Wave(int [][] matrix, int x1, int y1, int x2, int y2) - поиск пути, принимает  
 \* массив с лабиринтом (-1: граница, 0: пустая клетка)  
 \* и координаты начала и конца  
 \* getResultList() - возвращает список с координатами пути  
 \* getMaze() - возвращает лабиринт, где 1 - выход из него  
 \*/  
public class WaveResult {  
 private int[][] wave;  
 private List<Cel> waveBase = new ArrayList<>();  
 private String[] waveString;  
  
 /\*\*  
 \* Функция распространения волны.  
 \* Из начального элемента распространяется в 4-х направлениях  
 \* волна.Элемент в который пришла волна образует фронт волны.  
 \* Каждый элемент первого фронта волны является источником  
 \* вторичной волны. Элементы второго фронта волны генерируют  
 \* волну третьего фронта и т.д. Процесс продолжается до тех пор  
 \* пока не будет достигнут конечный элемент.  
 \* @param x1,y1 - начальная точка  
 \* @param x2,y2 - конечная точка  
 \*/  
 private void motionWave(int x1, int y1, int x2, int y2)  
 {  
 waveBase.add(new Cel(x1, y1));  
 wave[x1][y1]=1;  
 while (wave[x2][y2]==0)  
 {  
 int x, y;  
 /\*\*  
 \* Сохраняем координаты прошлого фронта волны  
 \* т.к. новый распространяется в зависимости от них  
 \*/  
 List<Cel> step = new ArrayList<>();  
 step.addAll(waveBase);  
 waveBase.clear();  
 for(Cel entry : step)  
 {  
 x=entry.x;  
 y=entry.y;  
 if(wave[x][y-1] == 0) {  
 wave[x][y-1] = wave[x][y]+1;  
 waveBase.add(new Cel(x, y-1));  
 }  
 if(wave[x][y+1] == 0) {  
 wave[x][y+1] = wave[x][y]+1;  
 waveBase.add(new Cel(x, y+1));  
 }  
 if(wave[x-1][y] == 0) {  
 wave[x-1][y] = wave[x][y]+1;  
 waveBase.add(new Cel(x-1, y));  
 }  
 if(wave[x+1][y] == 0) {  
 wave[x+1][y] = wave[x][y]+1;  
 waveBase.add(new Cel(x+1, y));  
 }  
 }  
 }  
 waveBase.clear();  
 }  
  
 /\*\* Функция восстановления волны  
 \* Строится сама трасса. Её построение осуществляется  
 \* от конечного элемента к начальному.  
 \* @param x1,y1 - начальная точка  
 \* @param x2,y2 - конечная точка  
 \*/  
 private void returnWave(int x1, int y1, int x2, int y2)  
 {  
 int step=wave[x2][y2];  
 waveBase.add(new Cel(x2, y2));  
 /\*\*  
 \* Каждый следующий элемент должен быть на 1 меньше  
 \* текущего. Таким образом алгоритм приводит  
 \* нас к началу.  
 \*/  
 while(wave[x1][y1]!=0)  
 {  
 if(wave[x2][y2-1] == step-1)  
 {  
 waveBase.add(new Cel(x2, y2-1));  
 y2--;  
 step=wave[x2][y2];  
 wave[x2][y2]=0;  
 }  
 else if (wave[x2][y2+1] == step-1)  
 {  
 waveBase.add(new Cel(x2, y2+1));  
 y2++;  
 step=wave[x2][y2];  
 wave[x2][y2]=0;  
 }  
 else if (wave[x2-1][y2] == step-1)  
 {  
 waveBase.add(new Cel(x2-1, y2));  
 x2--;  
 step=wave[x2][y2];  
 wave[x2][y2]=0;  
 }  
 else if (wave[x2+1][y2] == step-1)  
 {  
 waveBase.add(new Cel(x2+1, y2));  
 x2++;  
 step=wave[x2][y2];  
 wave[x2][y2]=0;  
 }  
 }  
 }  
  
 /\*\*  
 \* Основная функция класса. Из неё вызываются все методы  
 \* @param matrix - входной лабиринт  
 \* @param x1,y1 - начальная точка  
 \* @param x2,y2 - конечная точка  
 \*/  
 void generateWave(int [][] matrix, int x1, int y1, int x2, int y2) {  
 int mx = matrix[0].length, my = matrix.length;  
 wave = new int[mx][my];  
 for (int i = 0; i<matrix[0].length; i++)  
 {  
 for (int j = 0; j<matrix.length; j++)  
 {  
 wave[i][j] = matrix[i][j];  
 }  
 }  
 motionWave(x1, y1, x2, y2);  
 returnWave(x1, y1, x2, y2);  
 waveString = new String[waveBase.size()];  
 for(int i=0; i<waveBase.size(); i++)  
 {  
 waveString[i]=waveBase.get(i).toString();  
 }  
 wave=matrix;  
 for(Cel entry : waveBase)  
 {  
 wave[entry.x][entry.y] = 1;  
 }  
 }  
  
 /\*\*  
 \* Возвращаем список координат  
 \* @return список  
 \*/  
 public List<Cel> getResultList()  
 {  
 return this.waveBase;  
 }  
  
 /\*\*  
 \* Возвращаем лабиринт, в котором:  
 \* -1: граница  
 \* 0: пустая клетка  
 \* 1: путь выхода  
 \* @return лабиринт с построенной трассой  
 \*/  
 public int[][] getMaze()  
 {  
 return wave;  
 }  
}

Листинг 5.2.2 – Исходный код класса Cel

package data;  
  
import java.util.List;  
  
/\*\*  
 \* Класс Cel выполняет роль конструктора  
 \* Принимает в себе координаты x и y, содержит информацию  
 \* о границах клетки и имеет лист содержащий информацию о себе  
 \* для создания множеств  
 \*/  
public class Cel{  
 public boolean right,down;  
 public List<Cel> set;  
 public int x,y;  
  
 Cel(int a,int b){  
 x=a;  
 y=b;  
 right=false;  
 down=true;  
 set=null;  
 }  
  
 @Override  
 public String toString()  
 {  
 return String.format("%d : %d", this.x, this.y);  
 }  
}

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Neocomputer. Eller’s Alghorithm [Электронный ресурс] / Электр. журнал. – режим доступа: <http://www.neocomputer.org/projects/eller.html> - (Дата обращения: 06.10.2017).

2. Know. Алгоритм Ли [Электронный ресурс] / Электр. журнал. – режим доступа: https://www.know.cf/enciclopedia/RU/Алгоритм\_Ли - (Дата обращения: 03.10.2017).

3. А.Р. Фахрутдинов ИСПОЛЬЗОВАНИЕ ПОИСКА ПУТИ В АВТОМАТИЗАЦИИ (см. с. 126) / А.Р. Фахрутдинов и др. - Томск, 2014. – 431 с.