|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  |
|  | | | |
|  | |  |  |
| МИНОБРНАУКИ РОССИИ | | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"Московский государственный университет информационных технологий, радиотехники и электроники"**  **МИРЭА** | | | |
| Институт информационных технологий (ИТ) | | | |
| Кафедра информатики и информационных систем (ИИС) | | | |

|  |  |
| --- | --- |
| **КУРСОВАЯ РАБОТА** | |
| **по дисциплине** | |
| **«ОБЪЕКТНО-ОРИЕНТИРОВАНОЕ ПРОГРАММИРОВАНИЕ»** | |
| **Тема курсовой работы: «Разработка программного обеспечения, реализующее создание и прохождение лабиринта посредством алгоритма Hunt and kill»** | |
| Студент группы ИСБО-01-13 | *Бабаханян А.А.* |
| Руководитель курсовой работы | *Хлебников А.А.* |
| Рецензент | *Хлебников А.А.* |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_2015 г. |  |
|  |  |  |
| «Допущен к защите» | «\_\_»\_\_\_\_\_\_\_2015 г. |  |

Москва 2015

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования"Московский государственный технический университет радиотехники,электроники и автоматики"МГТУ МИРЭА | |
| Институт информационных технологий (ИТ) | |
| Кафедра информатики и информационных систем (ИИС) | |
|  | **Утверждаю** |
|  | Заведующий  кафедрой Соловьев Игорь Владимирович |
|  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_2015 г. |
| **ЗАДАНИЕ** | |
| **на выполнение курсовой работы** | |
| **по** **дисциплине** «Объектно-ориентированное программирование» | |

Студент Бабаханян Артур Аветович Группа ИСБО-01-13

1. **Тема**: Разработка программного обеспечения, реализующее создание и прохождение лабиринта посредством алгоритма «Hunt and kill»
2. **Исходные данные:** - JAVA JDK 1.8

- описание алгоритма «Hunt and kill» для создания лабиринта.

- описание алгоритма прохождения лабиринта.

- сторонние библиотеки

* 1. **Перечень вопросов, подлежащих разработке, и обязательного графического материала:**
  + Реализация классов, обеспечивающих создания лабиринта.
  + Реализация классов нахождения путей в лабиринте.
  + Реализация классов для визуализации лабиринта.
  + Разработка модуля Java Extensible Applications.
  + Разработка окружения тестирования.

|  |  |  |  |
| --- | --- | --- | --- |
| 1. **Срок представления к защите курсовой работы:** **до** «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_\_ г. | | | |
|  | | | |
| Задание на курсовой  проект (работу) выдал | «\_\_\_»\_\_\_\_\_\_\_\_\_ 201\_\_г. |  | *Хлебников А. А.* |
| Задание на курсовой  проект (работу) получил | «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 201\_\_ г. |  | *Бабаханян А.А.* |

УДК 004.9

ББК 32.972

В 12

*Руководитель и рецензент:* старший преподаватель кафедры ИИС МИРЭА Чехарин Е.Е.

*Студент:* Бабаханян А.А.

*Тема:* «Разработка программного обеспечения, реализующее создание и прохождение лабиринта при помощи алгоритма Hunt and kill»

Курсовая работа включает в себя описание разрабатываемой программы, задачей которой является создание лабиринта, с помощью алгоритма Hunt and Kill, а также поиск прохождения созданного лабиринта.

The course project includes description of developed program, which purpose of this program is creation of maze, using Hunt and Kill algorithm, also search for the passage of the created maze.

**Реферат**

Данная курсовая работа содержит 17 листов, 14 иллюстраций, 5 использованных источников литературы.

В курсовой работе будет выполнено поэтапное проектирование программы, которая по предложенному заданию будет создавать и проходить лабиринт.

Основными задачами явились: изучение и реализация алгоритма Hunt and Kill и алгоритма Ли.

The course work will be carried out step by step the design of the program, which task will be creation and path finding of the maze.

The main objectives were: to study and implementation of the algorithm Hunt and Kill and Lee algorithm.

# Оглавление

[Оглавление 5](#_Toc438997335)

[Список основных сокращений 6](#_Toc438997336)

[Словарь терминов 7](#_Toc438997337)

[Введение 8](#_Toc438997338)

[**1.Алгоритм создания лабиринта 9**](#_Toc438997339)

[**1.1.Алгоритм Hunt and Kill 9**](#_Toc438997340)

[**1.2.Работа алгоритма 10**](#_Toc438997341)

[**2.Алгоритм прохождения лабиринта 10**](#_Toc438997342)

[**2.1.Алгоритм Ли 10**](#_Toc438997343)

[**2.2.Описание и работа алгоритма 10**](#_Toc438997344)

[**2.3.Вариации алгоритма 12**](#_Toc438997345)

[**3.Реализация 12**](#_Toc438997346)

[**3.1.Создание лабиринта 12**](#_Toc438997347)

[**3.2.Прохождение лабиринта 14**](#_Toc438997348)

[**4.Заключение 16**](#_Toc438997349)

[**Список использованных источников 17**](#_Toc438997350)

# Список основных сокращений

**HK** – Hunt and Kill

**ДРП** - дискретное рабочее поле

**Int** – integer (целочисленный тип данных)

# Словарь терминов

**Алгоритм** — набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий, при любом наборе исходных данных.

**Планарный граф** — граф, который может быть изображён на плоскости без пересечения ребер. Иначе говоря, граф планарен, если он изоморфен некоторому плоскому графу, то есть графу, изображённому на плоскости так, что его вершины — это точки плоскости, а рёбра — непересекающиеся кривые на ней. Области, на которые граф разбивает плоскость, называются его гранями. Неограниченная часть плоскости — тоже грань, так называемая внешняя грань.

**Поиск в ширину** (англ. breadth-first search, BFS) — метод обхода графа и поиска пути в графе. Поиск в ширину является одним из неинформированных алгоритмов поиска.

**ДРП** – это прямоугольник, разбитый на квадратные ячейки одинакового размера.

**Окрестность фон Неймана** (англ. von Neumann neighborhood) — совокупность четырёх клеток на квадратном паркете, имеющих общую сторону с данной клеткой. Окрестность получила своё название в честь Джона фон Неймана, использовавшего её в своих клеточных автоматах, включая универсальный конструктор.

**Окрестность Мура** (англ. Moore neighborhood) — в двумерном случае — совокупность восьми клеток на квадратном паркете, имеющих общую вершину с данной клеткой. Окрестность получила своё название в честь одного из пионеров теории клеточных автоматов Эдварда Мура.

**Класс** — разновидность абстрактного типа данных в объектно-ориентированном программировании (ООП), характеризуемый способом своего построения.

**Метод** — это функция или процедура, принадлежащая какому-то классу или объекту.

# Введение

Данный курсовой проект состоит из 4 разделов:

* Алгоритм создания лабиринта
* Алгоритм прохождения лабиринта
* Реализация
* Заключение

В разделе алгоритм создания лабиринта рассказывается об алгоритме Hunt and Kill, который будет использован в ходе выполнения курсовой работы.

В разделе алгоритм прохождения лабиринта об алгоритме Ли, который будет использован в ходе выполнения курсовой работы.

В разделе реализация рассказывается, как были реализованы алгоритм Hunt and Kill и алгоритм Ли при выполнения задания курсовой работы

В разделе Заключение раскрываются результаты проделанной работы.

## 1.Алгоритм создания лабиринта

## 1.1.Алгоритм Hunt and Kill

Hunt and Kill – алгоритм создания лабиринта, свое название он берт из двух методов, с помощью которых и создается лабиринт:

* Kill – из выбранной клетки переходят в случайно выбранную соседнюю клетку, создавая между ними ход и переходя из нее в следующую, в посещенную клетку больше нельзя переходить, и так далее, пока из клетки можно перейти в соседнюю, не посещенную клетку.

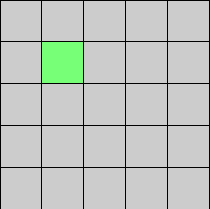


Рис.1.1 Выбор первой клетки для метода kill

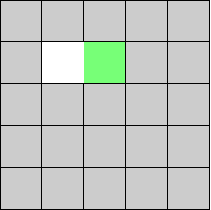


Рис.1.2 Переход в случайно выбранную клетку

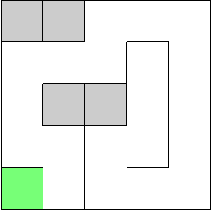


Рис.1.3 Конец метода kill

* Hunt – в массиве клеток лабиринта ищется еще не посещенная клетка, сосед которой уже посещен, между ними создается ход.

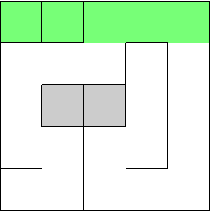


Рис.1.4 Поиск не посещенной клетки – метод hunt

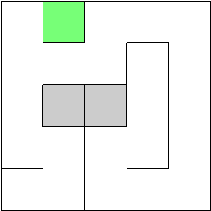


Рис.1.5 Создание хода с найденной клеткой

## 1.2.Работа алгоритма

Работа алгоритма включает в себя четыре этапа:

1. Выбирается случайная клетка и на ней используется метод kill
2. Метод hunt
3. На найденной клетке используется метод kill
4. 2-ой и 3-ий пункт повторяются, пока все клетки не будут посещены

## 2.Алгоритм прохождения лабиринта

## 2.1.Алгоритм Ли

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.

## 2.2.Описание и работа алгоритма

Алгоритм работает на дискретном рабочем поле, 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.

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.

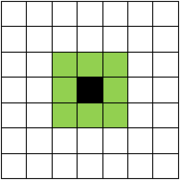


Рис.2.1 Окрестность Мура

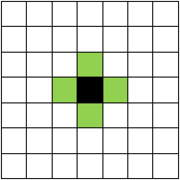


Рис.2.2 Окрестность фон Неймана

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, меченная 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 путьмежду парой заданных ячеек. 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.



Рис.2.3 Результат работы волнового алгоритма (ортогональный)



Рис.2.4 Результат работы волнового алгоритма (ортогонально-диагональный)

Достоинства алгоритма Ли - простота, надёжность, 100% самый короткий путь. Недостатки - большой объём требуемой памяти и не самая высокая скорость нахождения пути. В некоторых условиях, нахождение пути может достигать по времени до 1/10 секунды. Это, конечно, приемлемо для трассировки плат, пошаговых стратегий и логических игрушек, но с трудом подойдёт для динамических игр.

## 2.3.Вариации алгоритма

* Двойная волна - распространение волны начинается как от конечной, так и от начальной точки, a маршрут составляется из двух участков - от точки встречи волн до старта и до финиша. Теоретически, может повысить скорость поиска в 3-4 раза.
* В случае острой нехватки памяти, например, для трассировщика плат, применяться усечённое кодирование волны. Первая волна имеет номер 1, вторая - 2, третья - 2, четвёртая - 1, и так далее. На кодировку одного элемента потребуется два бита (числа 0/3 будут описывать проходимое/непроходимое поле). При поиске маршрута ищутся соседние ячейки памяти в том же порядке (... 1 1 2 2 1 1 2 2 1 1 2 2...). Не применим при диагональных перемещениях.

## 3.Реализация

## 3.1.Создание лабиринта

Для реализации алгоритма HK был создан класс MazeBlueprint. Лабиринт будет представлять собой двумерный массив, элементы которого – целочисленные числа, являющиеся суммой чисел, каждое из которых отвечает за стенку одой из сторон:

* 1 = стенка слева
* 2 = стенка сверху
* 4 = стенка справа
* 8 = стенка снизу

Для удобства работы с массивом лабиринта был создан метод fillMaze, который благодаря двойному циклу for приравнивает все элементы массива 0.

Метод kill был реализован с помощью внешнего цикла while, условиями которого является наличие соседнего элемента массива равного 0, и четырех вложенных операторов if. Условием операторов if является комбинация из значений двух целочисленных типов данных, значения которых определяются с помощью генератора псевдослучайных чисел Random, оба числа генерируются в диапазоне [-1; 1], и проверка соседнего по x или y, в зависимости от сгенерированных чисел, элемента на равенство нулю. Если условие оператора if выполняется, то значение элемента массива уменьшатся на число, соответствующее числу стенки, которую нужно убрать, а следующая клетка равняется сумме стенок минус стенка, со стороны предыдущего элемента.

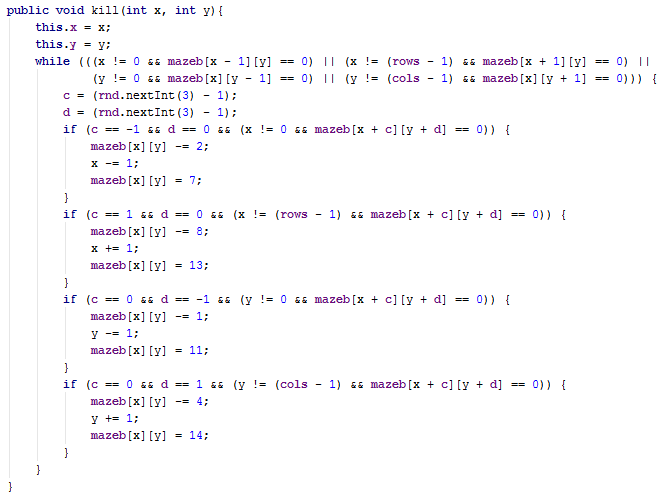


Рис.3.1 Метод kill

Для упрощения работы с алгоритмом, был создан метод firstKill, который выбирает случайный элемент массива и на нем использует метод kill, что соответствует первому пункту работы алгоритма HK.

Метод hunt также реализован с помощью цикла while, который выполняется, пока int fullness не равен числу элементов массива. Fullness обнуляется в начале цикла, а в конце изменяется с помощью двойного цикла for, который проверяет все элементы лабиринта, если элемент не равен нулю, то fullness увеличивается на единицу, таким образом, пока все элементы не будут заполнены, цикл будет продолжатся. В середине цикла находится двойной цикл for проверяющий элементы лабиринта, внутри него находится оператор if, который проверяет, является ли соседи проверяемого элемента заполненными. Внутри оператора if находятся еще четыре оператора if, которые проверяют соседей сверху, снизу, справа и слева, в зависимости от того, с какой стороны находится ненулевой элемент, проверяемый элемент приравнивается сумме стенок минус стенка со стороны ненулевого соседа, а из самого соседа вычитается число стенки со стороны элемента. После на элементе используется метод kill и применяется оператор перехода break, который входит из цикла for, так как метод hunt всегда после нахождении нужного элемента применяется с начала массива.

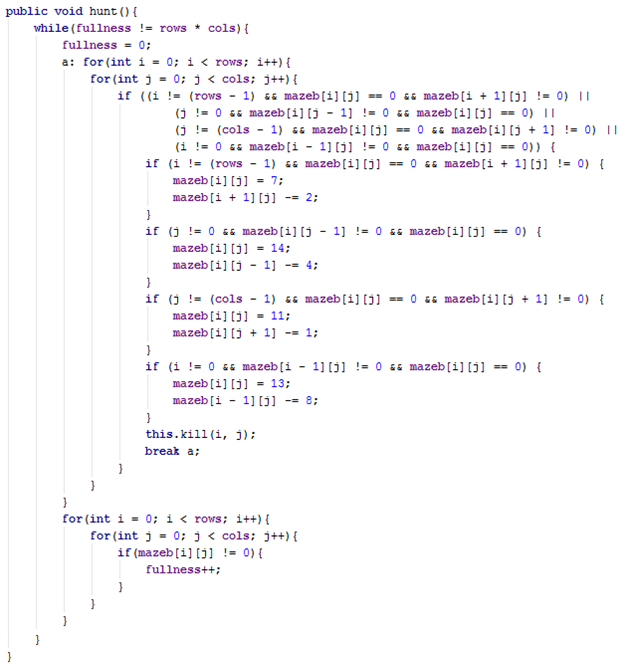


Рис.3.2 Метод hunt

## 3.2. Прохождение лабиринта

Для нахождения пути в лабиринте с помощью алгоритма волны нужно реализовать инициализацию, распространение волны и восстановление пути. Для этого понадобятся три двумерных массива размера лабиринта, в первом будет содержать элементы лабиринта, второй будет заполняться волнами и третий, в котором будет отмечать, что элемент уже заполнен волной. Для удобства реализации они были объединены в один трехмерный массив map размера длина лабиринта x ширина лабиринта x 3.

Для реализации распространения волны был использован цикл while, который будет выполняться пока элемент, отвечающий за выход из лабиринта, не будет заполнен. Элемент начала лабиринта был приравнен единице, а int w, отвечающего за уровень волны, двум. Внутри цикла двойной цикл for, который проверяет элементы лабиринта. В цикле for находится четыре оператора if, которые проверяют соседние элементы проверяемого элемента, если соседний элемент на единицу меньше уровня волны и между элементами нет стенки, то проверяемый элемент приравнивается уровню волны. После того, как двойной цикл for заканчивается, уровень волны увеличивается на единицу.

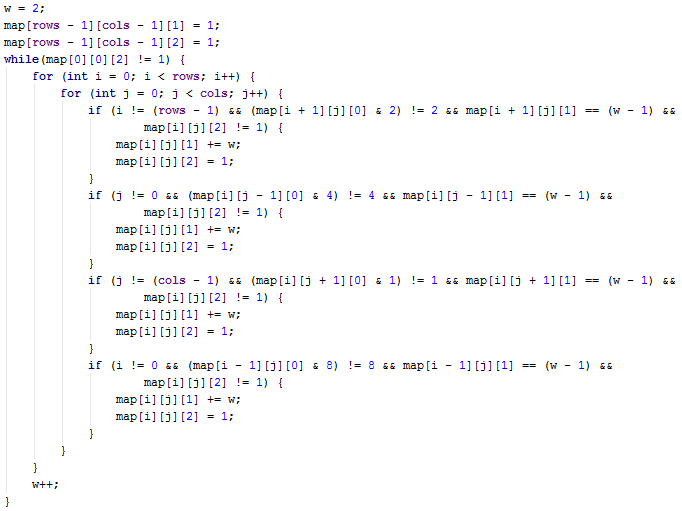


Рис.3.3 Реализация распространения волны

Восстановление пути также было реализовано с помощью цикла while, который будет работать пока int bw, равный волне элемента выхода из лабиринта, не равен 1. Внутри цикла четыре оператора if, которые проверяют соседние элементы. Если соседний элемент меньше bw на единицу и между элементами нет стенки, то координаты элемента запоминаются, bw уменьшается на единицу, и теперь будут проверяться соседние элементы этого элемента. Координаты элемента начала и конца пути сохраняются отдельно.

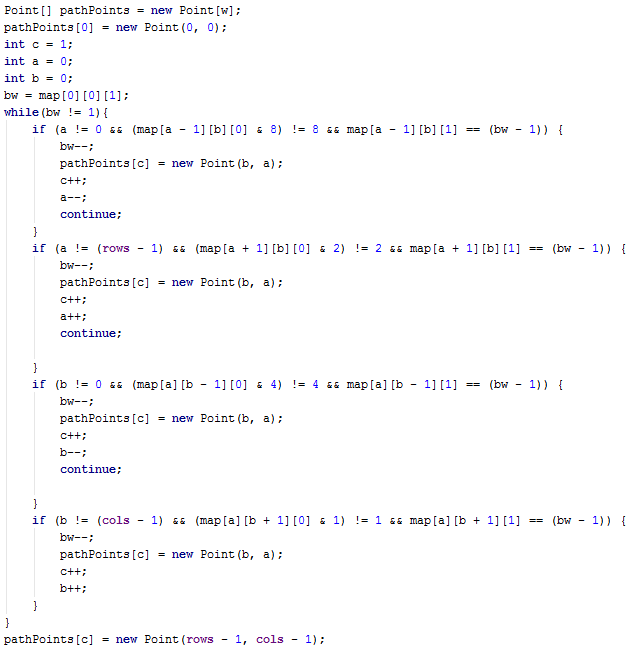


Рис.3.4 Реализация восстановления пути

## 4.Заключение

В ходе выполнения курсовой работы была достигнута цель, суть которой заключается в создании программного обеспечения исходным требованиям. В результате работы, созданное приложение выполняет поставленные перед ней задачи в создании лабиринта и его прохождения.

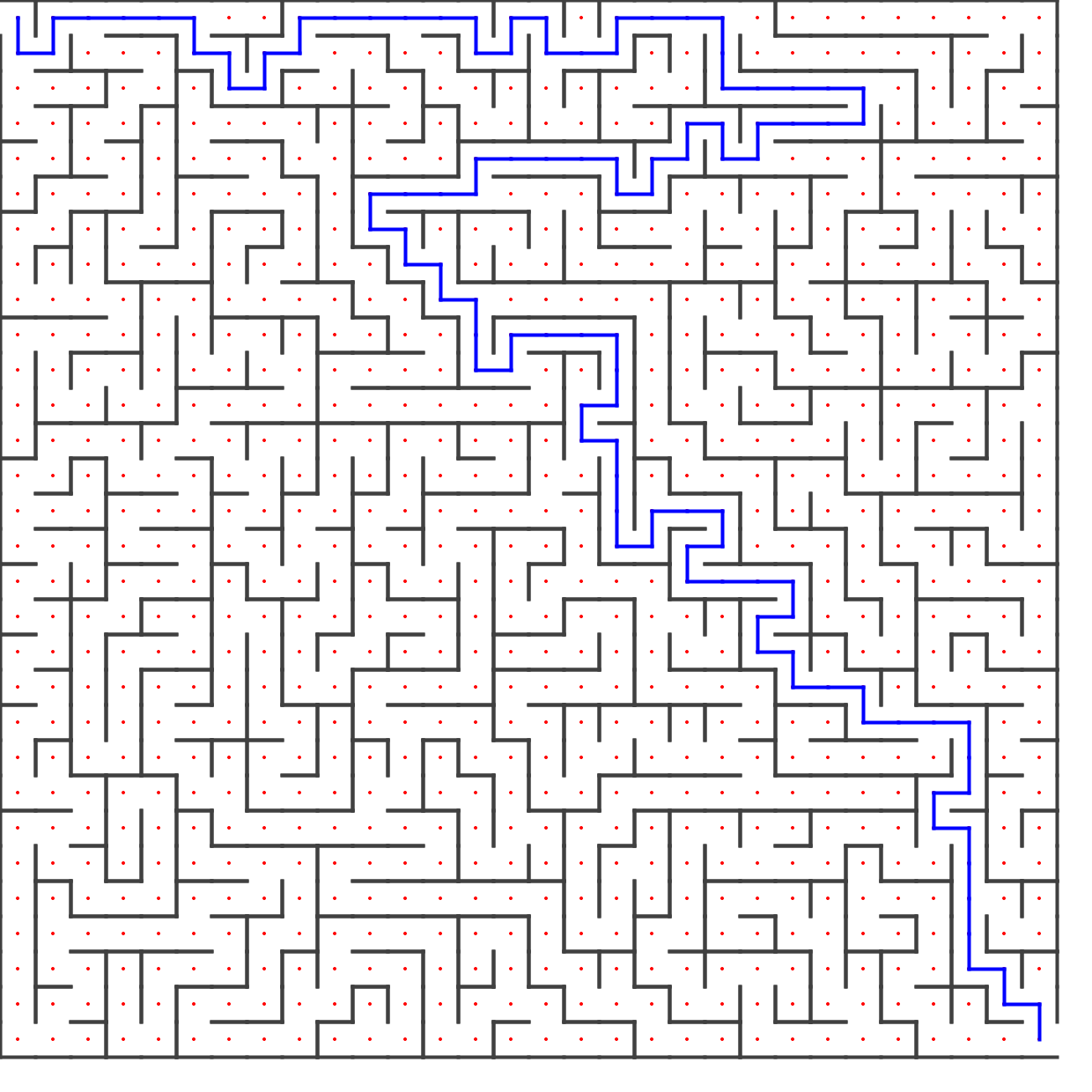


Рис.4 Пример создания и прохождения лабиринта 30x30

## Список использованных источников

1. Электронная энциклопедия Википедия// Алгоритм Ли

URL:[https://ru.wikipedia.org/wiki/Алгоритм\_Ли](https://ru.wikipedia.org/wiki/Алгоритм_Ли%20) (дата обращения - 23.12.2015)

1. Электронная энциклопедия Википедия// Алгоритмы генерации лабиринтов

URL: <https://en.wikipedia.org/wiki/Maze_generation_algorithm> (дата обращения - 24.12.2015)

1. Статьи по программированию Suvitruf// Обход препятствий: волновой алгоритм (Алгоритм Ли) URL: [http://suvitruf.ru/2012/05/13/1176](http://suvitruf.ru/2012/05/13/1176%20) (дата обращения - 23.12.2015)
2. Блог Джамис Бака// Генерация лабиринтов: алгоритм Hunt and Kill

URL: <http://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm> (дата обращения - 23.12.2015)

1. Статьи по программированию CodeNet// Волновой алгоритм - Построение кратчайшего маршрута URL: [http://www.codenet.ru/progr/alg/way.php](http://www.codenet.ru/progr/alg/way.php%20) (дата обращения - 24.12.2015)