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

|  |  |
| --- | --- |
| **КУРСОВАЯ РАБОТА** | |
| **по дисциплине** | |
| **«Объектно-ориентированное программирование»** | |
| 1. **Тема курсовой работы «**Разработка программного обеспечения, реализующее создание и прохождение лабиринта посредством алгоритма «Бинарное дерево» | |
| Студент группы ИСБО-01-13 | *Овчаров Ю.С.* |
| Руководитель курсовой работы | *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"МГТУМИРЭА | |
| Институт информационных технологий (ИТ) | |
| Кафедра информатики и информационных систем (ИИС) | |
|  | **Утверждаю** |
|  | Заведующий  кафедрой Соловьев Игорь Владимирович |
|  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_2015 г. |
| **ЗАДАНИЕ** | |
| **на выполнение курсовой работы** | |
| **подисциплине** «Объектно-ориентированное программирование» | |

Студент Овчаров Юрий Сергеевич Группа ИСБО-01-13

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

- описание алгоритма «Бинарное дерево» 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.
  + Разработкамодуля 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. |  | *Овчаров Ю.С.* |

УДК 004.4

ББК 32.973-018.2

*Руководитель:* преподаватель кафедры ИИС МИРЭА Хлебников А. А.

*Студент: Овчаров Ю.С.*

*Тема:***«**Разработка программного обеспечения, реализующее создание и прохождение лабиринта посредством алгоритма «Бинарное дерево»

В ходе реализации задания на курсовую работу осуществляется изучение таких технологий и инструментов как JDK1.8, документации к ним, характеристик, сопутствующих понятий и свойств, сравнительный анализ различных технологий и инструментов с использованием наиболее актуальных из них на момент выполнения работы.

Была создана и разработана документация работы, включающая в себя: утвержденное задание на курсовую работу; расчетно-пояснительную записку (РПЗ), исходный код разработанного модуля расширения.

This work focused on study of technologies and tools such as JDK 1.8, documents to them, characteristics, related concepts and properties, a comparative analysis of the various technologies and tools, using the most relevant ones at the time of performance.

The project documentation, including approved task for course work; settlement and an explanatory note (RPG); the source code of the developed expansion module.

Координаты автора:

МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

Кафедра Информатики и Информационных систем (ИИС)

**Реферат**

Данная курсовая работа содержит 20 листов. Целью данной курсовой работы является изучение таких технологий и инструментов как JDK, необходимых для реализации задания, документации к ним, характеристик, сопутствующих понятий и свойств, сравнительный анализ различных технологий и инструментов с использованием наиболее актуальных из них на момент выполнения работы.

Основным задачами являлась реализация классов, обеспечивающих создания лабиринта, реализация классов нахождения путей в лабиринте,

реализация классов для визуализации лабиринта, разработка модуля JavaExtensibleApplications,разработка окружения тестирования.

The objectiveof this course work is to study these techniques and instruments as the JDK, needed for the implementation of tasks, documents to them characteristics related concepts and properties, a comparative analysis of the various technologies and tools with the most urgent of them at the time of execution operation.

The main tasks are the implementation classes that create a maze, finding ways to implement the class in a maze,

implementation classes for the visualization of the maze, the development of the module Java Extensible Applications, test development environment.

**Оглавление:**

Список сокращений………………………………………………………………6

Введение…………………………………………………………………………...7

Платформа, среда разработки, система контроля версий, сборщик проекта…8

Платформа JDK 1.8……………………………………………………………….8

Среда разработки IntellijIdea……………………………………………………..9

Описание алгоритма «Бинарное дерево»………………………………………11

Алгоритм нахождения кратчайшего пути……………………………………..12

Программная реализация………………………………………………………..14

Заключение……………………………………………………………………….20

**Список сокращений:**

JDK - JavaDevelopmentKit -комплект разработчика приложений на языке Java.

API – application programming interface - интерфейс программирования приложений, интерфейс прикладного программирования.

ООП- объектно-ориентированное программирование

ДРП - дискретное рабочее поле

**Введение.**

Целью данной работы является разработка программного обеспечения, реализующего создание и прохождение лабиринта при помощи алгоритма «Бинарное дерево».

Лабиринты широко применяются в науке и технике. Психологи с их помощью изучают поведение людей и животных в повторяющихся или экстремальных ситуациях.

Кибернетикам лабиринты помогают конструировать ЭВМ, в частности роботов, которые способны к самообучению. Такие эксперименты первым провел американский математик Клод Шеннон (род. 1916 г.): кибернетические мыши ученого по определенным алгоритмам могли выбираться из самых запутанных лабиринтов. По принципу лабиринта изготавливают глушители в двигателях внутреннего сгорания, заполняют части деталей под высоким давлением.

В ходе работы будут описаны различные функциональные части разрабатываемого программного продукта, технологии и инструменты, использованные в них.

В результате работы планируется получить:

* Рабочий модуль BinaryTreeMaze
* Изображение лабиринта, по заданному алгоритму
* Изображение найденного пути в данном лабиринте
* Окружение тестирования

**Платформа, среда разработки, система контроля версий, сборщик проекта.**

## Платформа JDK 1.8.

Руководствуясь заданием на курсовую работу, для разработки модуля расширения была использована платформа JDK версии 1.8, являющейся последней на данный момент.

Java Development Kit (сокращенноJDK) — бесплатно распространяемый компанией Oracle Corporation (ранееSun Microsystems) комплект разработчика приложений на языке Java, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную системуJava (JRE). В состав JDK не входит интегрированная среда разработки на Java, поэтому разработчик, использующий только JDK, вынужден использовать внешний текстовый редактор и компилировать свои программы, используя утилиты командной строки.

Все современные интегрированные среды разработки приложений на Java, такие, как JDeveloper, NetBeans IDE, SunJavaStudioCreator, IntelliJ IDEA, BorlandJBuilder, Eclipse, опираются на сервисы, предоставляемые JDK. Большинство из них для компиляции Java-программ используют компилятор из комплекта JDK. Поэтому эти среды разработки либо включают в комплект поставки одну из версий JDK, либо требуют для своей работы предварительной инсталляции JDK на машине разработчика[[1]](#footnote-2).

## 

## Среда разработки IntellijIdea.

Конечно, минимальный набор ПО для выполнения данной работы состоит из текстового редактора, JDKи командной строки, но куда проще и приятней работать в интегрированной среде разработки. В данной работе использовалась среда разработки IntellijIdea. Стоит отметить, что JetBrains, разработчик этой ИСР, предоставляет студентам возможность бесплатно пользоваться полными версиями своих продуктов на условиях образовательной лицензии (ClassroomLicense).

IntelliJ IDEA — интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Первая версия IntelliJ IDEA появилась в январе 2001 года и быстро приобрела популярность, как первая Java IDE с широким набором интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ. Дизайн среды ориентирован на продуктивность работы программистов, позволяя им сконцентрироваться на разработке функциональности, в то время как IntelliJ IDEA берёт на себя выполнение рутинных операций.

Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса.

Среди прочих возможностей, IntelliJ IDEA хорошо совместима со многими популярными свободными инструментами разработчиков, такими как CVS, Subversion, ApacheAnt, Maven и JUnit.

В феврале 2007 года разработчики IntelliJ анонсировали раннюю версию плагина для поддержки программирования на языке Ruby.

Начиная с версии 9.0, IntelliJ IDEA доступна в двух версиях: CommunityEdition и UltimateEdition. CommunityEdition является полностью свободной версией, доступной под лицензией Apache 2.0. В ней реализована полная поддержка Java SE, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями.

В версии UltimateEdition реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.[[2]](#footnote-3)

**Описание алгоритма «Бинарное дерево»**

Данный алгоритм способен генерировать лабиринты, не сохраняя при этом в памяти никаких состояний всего лабиринта или его частей. Он способен построить весь контур, имея в зоне видимости лишь текущее положение.

Для каждой клетки в сетке случайно вырезается проход либо на север, либо на запад. Можно перемешать это и выбирать между другими диагональными наборами: север/восток, юг/запад или юг/восток, но при этом набор должен оставаться таким на протяжении всей работы алгоритма.

Существует другой лаконичный атрибут этого алгоритма: если он применим к любой ячейке, то мы всегда будем вырезать конкретное направление и впоследствии не придётся хранить весь лабиринт в памяти. Когда нужно отобразить какую-то часть лабиринта, мы должны просто "перестроить" его. Это означает, что можно создавать бесконечно большие лабиринты в маленьком объёме памяти.

Однако, это ещё не всё. Побочный эффект этого алгоритма в том, что у него есть сильное диагональное смещение. Также, две из четырёх сторон лабиринта будут покрываться одним коридором. Но для некоторых приложений, это имеет место быть.

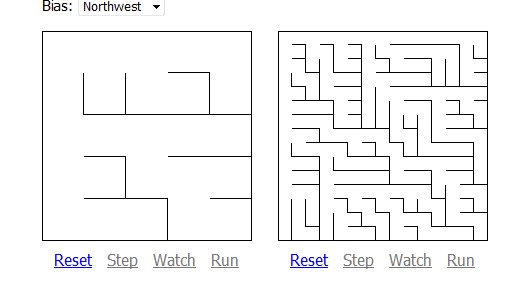


Рис 1.Пример лабиринта.

**Алгоритм нахождения кратчайшего пути**

Алгори́тм волново́й трассиро́вки (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. МуромЛи независимо открыл этот же алгоритм при формализации алгоритмов трассировки печатных плат в 1961 году.

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 непроходимости.

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 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.1 Результат работы волнового алгоритма (ортогональный путь)

**Программная реализация.**

*BinaryTree.*

Прежде всего был реализован класс ***Cell*** для описания сущности клетки нашего поля.

final class Cell {  
final intx;  
 final inty;  
booleanup;  
booleanleft;  
booleandown;  
booleanright;  
intdistance;

Здесь поля ***up, left, down, right*** соответствуют стенам вокруг клетки, true, если стена есть и false–наоборот.***X***и ***Y*** – координаты клетки.

Основную работу совершает метод ***generateMaze,*** который и реализует описанный выше алгоритм.

publicMaze generateMaze(introws, intcols) {

Рассмотрим основные блоки этого метода.

for (intj = 0; j < cols; j++) {  
for (inti = 0; i< rows; i++) {  
 Cell nc = new Cell(i, j);  
 if (j == 0) {  
nc.right= false;  
nc.left= false;  
nc.up= true;  
} else {  
nc.up= true;  
nc.down= true;  
nc.right= true;  
 if (getRandomBoolean() || i == 0) {  
nc.up= false;  
nc.left= true;  
cells[i][j - 1].down = false;  
} else {  
nc.left= false;  
cells[i - 1][j].right = false;  
}  
 }  
  
if (j == rows - 1) {  
nc.down= true;  
}  
  
if (i == cols - 1) {  
nc.right= true;  
}  
  
 cells[i][j] = nc;  
}  
}

В этом блоке совершается проход по матрице лабиринта, создаем объект класса ***Cell***с текущими координатами. Особенностью реализации алгоритма является то, что первая строка и столбец остаются пустыми, учитываем это в соответствующем блоке **IF**.

Далее реализуется основная идея алгоритма. Генерируем случайное логическое значение, если оно верно, то прорубаем проход вверх, иначе – влево, не забываем прорубать также проходы в соседних клетках.

Также основное контур лабиринта должен быть обрамлен стенами.

Mazemaze = new Maze(rows, cols);  
for (intx = 0; x < rows; x++) {  
for (inty = 0; y < cols; y++) {  
maze.data[x][y] = cells[x][y].getValue();  
}  
}

В этом блоке мы преобразуем готовую матрицу в объект класса ***Maze.***

***Поиск путей.***

Поиск пути реализован методом,

publicPoint[] findPath(Maze maze)

который принимает на вход объект класса Maze, полученный в предыдущем пункте. Этот метод является реализацией волнового алгоритма нахождения пути.

По аналогии рассмотрим основные части кода.

cols = maze.cols;  
rows = maze.rows;  
Cell[][] cells = new Cell[maze.rows][maze.cols];  
Set<Cell>cellSet = new HashSet<>();  
for (inty = 0; y <maze.rows; y++) {  
for (intx = 0; x <maze.cols; x++) {  
 Cell cell = new Cell(x, y, maze.data[x][y]);  
cells[x][y] = cell;  
cellSet.add(cell);  
}  
}  
  
Cell start = cells[0][0];  
start.distance= 0;  
Cell finish = cells[rows - 1][cols - 1];  
intwave = 0;

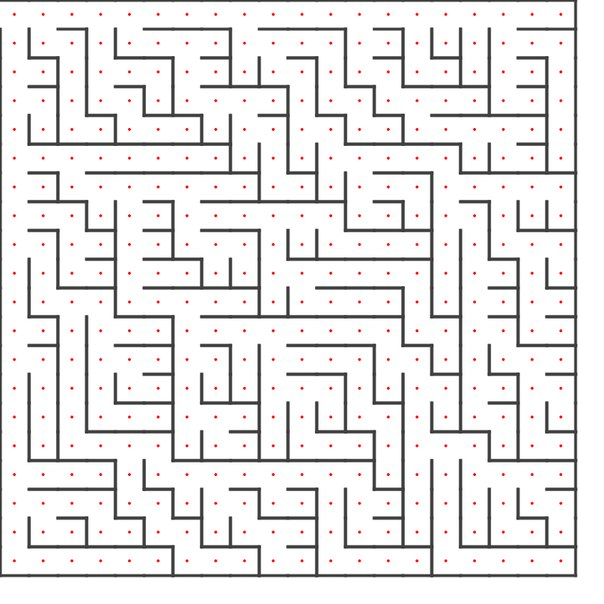
Кладем все клетки в список множества, где все элементы хранятся в единственном экземпляре. Устанавливаем начальные параметры.

do {  
for (Cell cell : cellSet) {  
if (cell.distance== wave) {  
 List<Cell> neighbors = cell.getNeighborCells(cellSet);  
 for (Cell neighbor : neighbors) {  
if (neighbor.distance== -1) {  
neighbor.distance= wave + 1;  
}  
 }  
 }  
 }  
 wave++;  
} while (finish.distance== -1);  
List<Cell> path = new ArrayList<>();  
path.add(finish);  
Cell currCell = finish;  
while (!path.contains(start)) {  
assert currCell != null;  
currCell = *getСlosest*(currCell.getNeighborCells(cellSet));  
path.add(currCell);  
}  
*logger*.debug("Построенпутьотвходаквыходудлиной {}", wave);

Начинаем двигаться от начала, помечая соседние клетки значениями, зависящими от текущего значения волны (wave), когда достигли финиша, собираем наш путь от конца к началу, двигаясь по ближайшим соседям.

Point[] pathPoints = new Point[path.size()];  
for (inti = 0; i<pathPoints.length; i++) {  
pathPoints[i] = new Point(path.get(i).x, path.get(i).y);  
}  
  
return pathPoints;

Возвращаем массив объектов класса ***Point***.



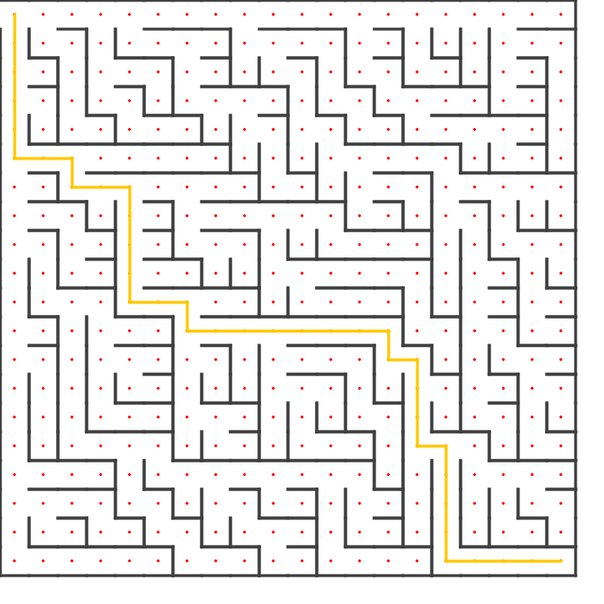


Рис 4 Результат работы алгоритма нахождения пути

**Заключение.**

В ходе данной курсовой работы были реализованы классы, обеспечивающие создание лабиринта, нахождение путей в лабиринте. Была рассмотрена реализация классов для визуализации лабиринта.

Также разработан модуль JDK и окружение тестирования. Всё, что планировалось во введении - успешно реализовано.

1. [JavaDevelopmentKit](https://ru.wikipedia.org/wiki/Java_Development_Kit)Материал из Википедии — свободной энциклопедии [↑](#footnote-ref-2)
2. [IntelliJIDEA](https://ru.wikipedia.org/wiki/IntelliJ_IDEA)Материал из Википедии — свободной энциклопедии [↑](#footnote-ref-3)