****

**实验报告**

**课程名称： 计算机综合实习**

**项目名称： java迷宫**

**专业班级： 数学与应用数学1902班**

**姓 名： 叶正荣 学 号： 190705212**

**完成时间： 2022.6.30 批阅时间：**

**指导教师： 王博 成 绩：**

**沈阳工业大学实验报告**

（适用计算机程序设计类）

专业班级： 学号： 姓名：

项目名称：

基于迷宫的系统设计

摘 要

迷宫（希腊语：λαβύρινθος，拉丁转写：labyrinthos）指的是充满复杂通道，很难找到从其内部到达入口或从入口到达中心的道路，道路复杂难辨，人进去不容易出来的建筑物。通常比喻复杂艰深的问题或[难以捉摸](https://baike.baidu.com/item/%E9%9A%BE%E4%BB%A5%E6%8D%89%E6%91%B8/3220651)的局面。单迷宫是只有一种走法的迷宫。对于单迷宫而言，有一种万能的破解方法，即沿着某一面墙壁走。在走的时候，左（右）手一直摸着左（右）边的墙壁，这种方法可能费时最长，也可能会使你走遍迷宫的每一个角落和每一条死路，但玩者绝不会永远困在里面。生成迷宫算法：**1.Recursive backtracker ( 递归回溯，也是深度优先算法），**2.Randomized Prim's algorithm（随机Prim算法，让我想起了最小生成树的Prim算法），3.Recursive division （递归分割算法）。路径寻找算法：BFS，DFS，A\*。

关键词：DFS；BFS；递归；非递归；栈

目录

[摘 要 2](#_Toc106805697)

[一． 数据结构基础 3](#_Toc106805698)

[1.1.1 常用数据结构概述 3](#_Toc106805699)

[1.2.1 数据结构—栈（如下图所示） 3](#_Toc106805700)

[1.3.1 Java中的栈实现 3](#_Toc106805701)

[1.4.1数据结构—队列（如下图所示） 4](#_Toc106805702)

[1.5.1Java中队列的实现 4](#_Toc106805703)

[1.6.1 相关算法 4](#_Toc106805704)

[二．迷宫生成算法 8](#_Toc106805705)

[2.1.1 Recursive backtracker ( 递归回溯，也是深度优先算法） 9](#_Toc106805706)

[2.2.1 随机Prim算法，让我想起了最小生成树的Prim算法 10](#_Toc106805707)

[三．迷宫寻路 13](#_Toc106805708)

[3.1.1 深度遍历寻路 13](#_Toc106805709)

[3.2.1 广度遍历寻路 15](#_Toc106805710)

[3.3.1 A\*算法 16](#_Toc106805711)

[四．综合迷宫实现 17](#_Toc106805712)

[功能介绍： 17](#_Toc106805713)

[效果展示： 17](#_Toc106805714)

# 数据结构基础

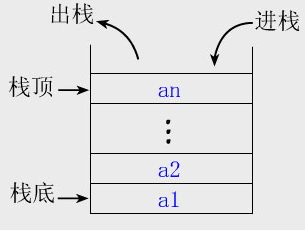
* + 1. 常用数据结构概述
* 寻路：栈（存储走过的路径）、二维数组
* 遍历：栈、二维数组（记录整个迷宫的墙信息，每个数组元素为一个结构体结点包含一个迷宫坐标点上右下左的墙信息）
* 寻优：队列（建立等高表）、栈（记录最优路径）

1.2.1 数据结构—栈（如下图所示）

操作特点：先进后出(Last In First Out, LIFO)

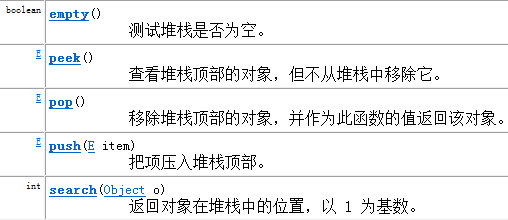
结构特点：只允许一端插入、查看和删除，不允许对另一端的操作

概念：栈顶（入栈、出栈）；栈底（固定，无操作）；空栈（栈中没有元素）



1.3.1 Java中的栈实现

Java中类 Stack<E>实现了栈的相关功能，继承自Vector<E>，构造方法：Stack()，表示创建一个空堆栈，常用方法如下图所示。



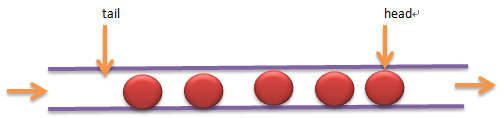
1.4.1数据结构—队列（如下图所示）

操作特性：先进先出(First In First Out, FIFO)。

结构特点：只允许一端入队，另一端查看和删除。

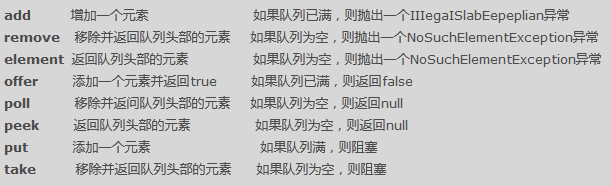
概念：队头（出队）；队尾（入队）；空队列（队列中没有元素）。

主要类型：顺序队列（队列、循环队列）、链式队列、两端队列



1.5.1Java中队列的实现

Java中的队列常用操作通过接口Queue来实现，它继承自Collection接口。LinkedList实现了Queue接口，实现了对队列的相关操作，常用操作如下图所示。



1.6.1 相关算法

算法—回溯

基本思想和步骤：回溯算法的基本思想就是深度优先遍历，基本步骤如下：

步骤一：先将根结点作为活动结点；

步骤二：向活动结点的子结点扩展，若其存在，子结点作为新的活结点，若不存在可扩展子结点，将当前活动结点设为不可扩展结点，其父节点作为当前的活动结点；

步骤三：重复步骤二，直至找到所求结点，输出路径，或者到没有可扩展结点时结束，输出null。如下图所示。

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

算法-广度优先遍历

基本思想和步骤：横向优先遍历，基本步骤如下：

步骤一：先将跟结点入队列；

步骤二：取队头元素，遍历队头元素的所有子结点，并将所有子结点加入队列；

步骤三：重复步骤二，直至找到所求结点，或者队列为空。如下图所示。

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

寻路过程中可以遵行的几种法则

* 右手法则：小车在搜过程中有两个以上的搜索方向时，优先选择向右转，其次是向前行进，最后才考虑向左转弯。
* 左手法则：小车在搜过程中有两个以上的搜索方向时，优先选择向左转，其次是向前行进，最后才考虑向右转弯。
* 向心法则：把整个迷宫地图分解为左下，右下，左上，右上四个均等的区域，在不同的区域中选择不同的转弯策略，使得小车始终向着迷宫的中心靠近，这样就可以以最快的速度接近终点，总体上说，遍历迷宫时此方法更科学有效。

# 二．迷宫生成算法

首先，为了方便后续处理，默认的迷宫元素表示为[x,y,w]：

1.我们的迷宫为常规的矩形，因此可以用二维表示一个迷宫单元, 每个迷宫单元表示为一个二维数组元素[x,y]。

3.每个迷宫单元包含左上右下四个属性, 用w表示，分别表示迷宫单元四个面的墙，墙不占据迷宫单元。

下面，我们一一介绍这三种算法：

* + 1. ****Recursive backtracker ( 递归回溯，也是深度优先算法）****

迷宫的初始状态是一张有众多单元格组成的[网格](https://so.csdn.net/so/search?q=%E7%BD%91%E6%A0%BC&spm=1001.2101.3001.7020)，单元格的初始状态是“四面有墙”，DFS的步骤如下:

1.将起点作为当前迷宫单元并标记为已访问

2.当还存在未标记的迷宫单元，进行循环

1.如果当前迷宫单元有未被访问过的的相邻的迷宫单元

1.随机选择一个未访问的相邻迷宫单元

2.将当前迷宫单元入栈

3.移除当前迷宫单元与相邻迷宫单元的墙

4.标记相邻迷宫单元并用它作为当前迷宫单元

2.如果当前迷宫单元不存在未访问的相邻迷宫单元，并且栈不空

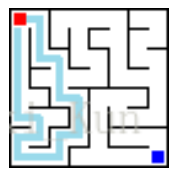
1.栈顶的迷宫单元出栈

2.令其成为当前迷宫单元

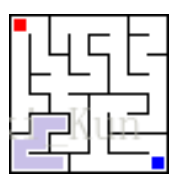
环路与孤立区域问题

有一些迷宫会出现两种情况

1、环路



2、孤立区域



深度优先搜索是完全可以避免环路与“孤立区域”的。图论中生成树的算法也可以采用深度优先搜索，这里产生的迷宫本质上也可以看作是一棵生成树。

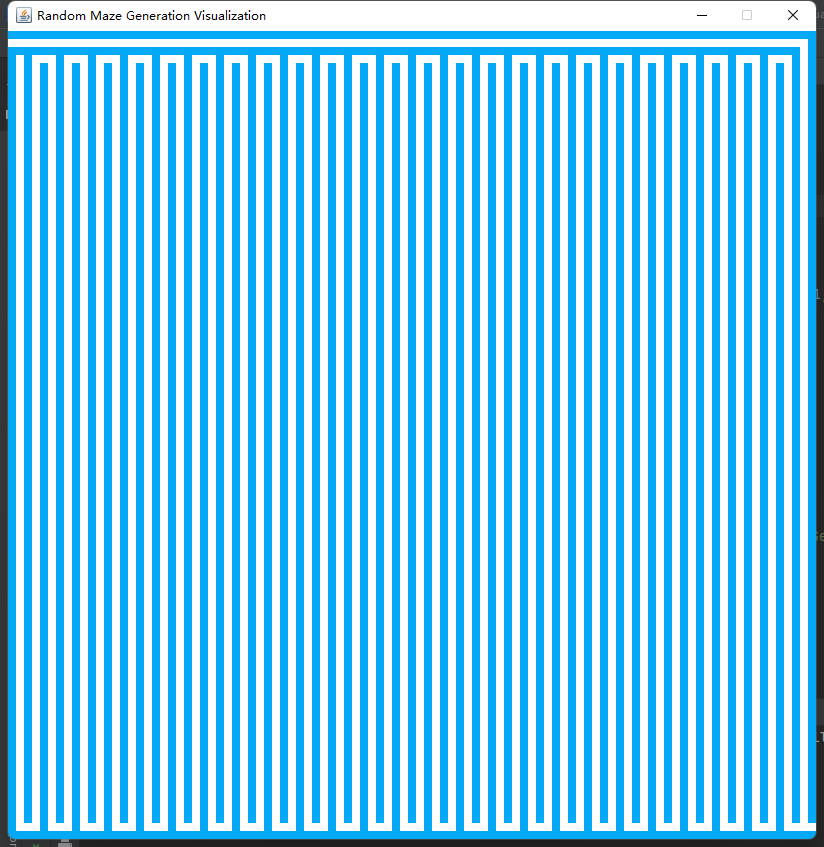
生成树有几条重要的性质

1、包含所有节点，迷宫中所有的单元格都会被搜索

2、树不存在环，所以迷宫也不会产生环路

3、树是连通图，因此用DFS产生的迷宫不会产生“孤立区域”

4、任意两节点都有通路，在迷宫中起点终点随意摆放都会有解

自我实现生成效果展示（代码详见代码包DfsMap包中）：

2.2.1 随机Prim算法，让我想起了最小生成树的Prim算法

随机Prim算法生成的迷宫岔路较多，整体上较为自然而又复杂。

1.让迷宫全是墙.

2.选一个单元格作为迷宫的通路，然后把它的邻墙放入列表

3.当列表里还有墙时

1.从列表里随机选一个墙，如果这面墙分隔的两个单元格只有一个单元格被访问过

1.那就从列表里移除这面墙，即把墙打通，让未访问的单元格成为迷宫的通路

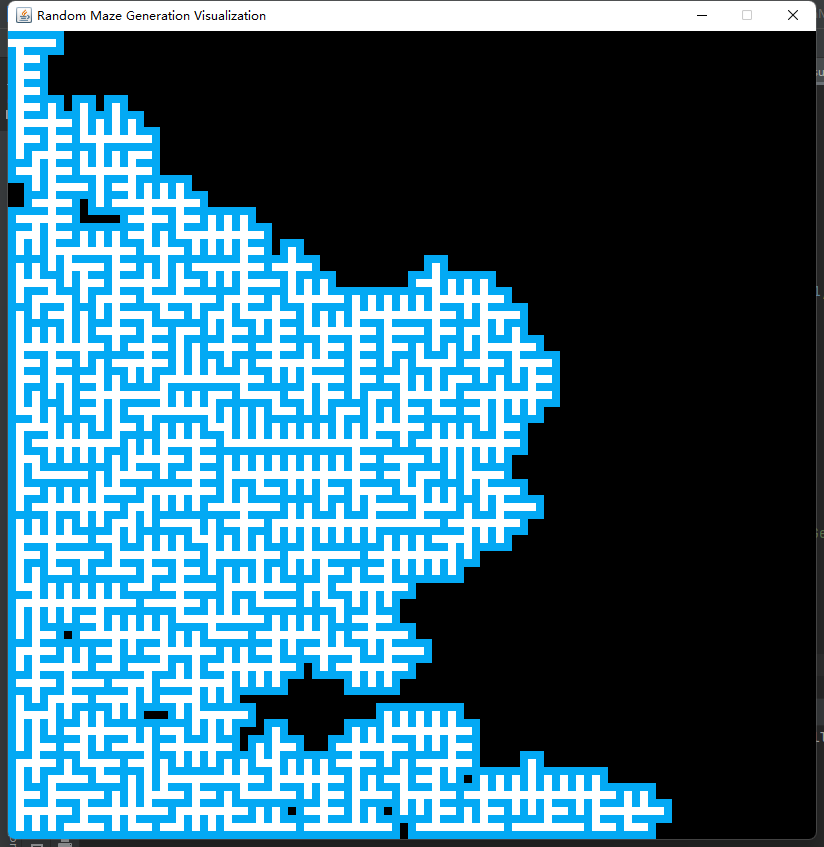
2.把这个格子的墙加入列表

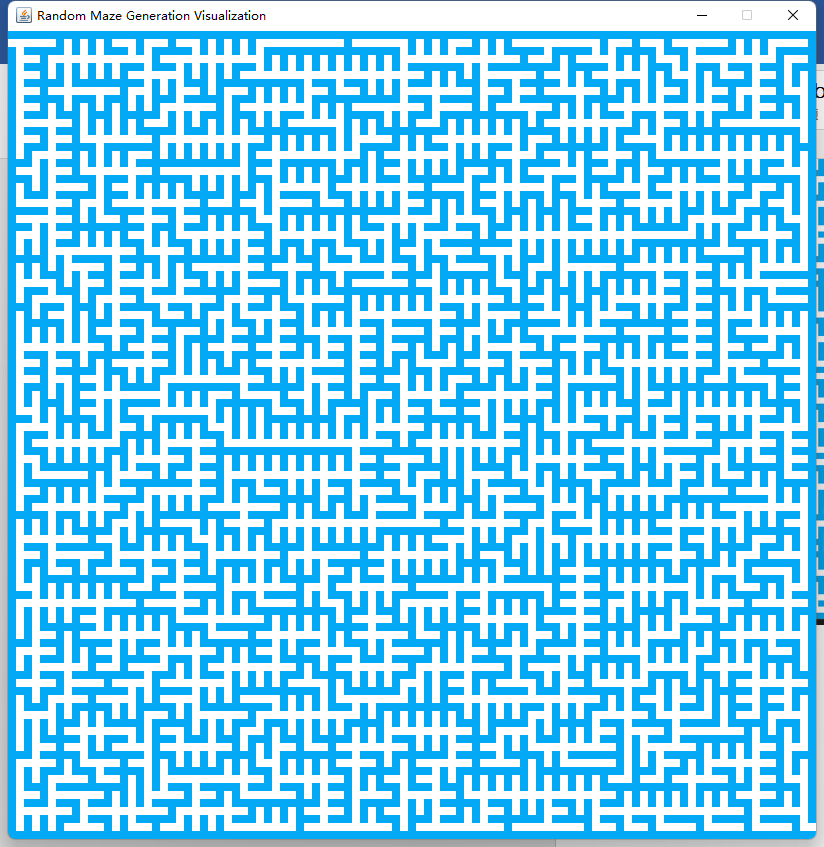
2.如果墙两面的单元格都已经被访问过，那就从列表里移除这面墙

在操作过程中，如果把墙放到列表中，比较复杂，维基里面提到了改进策略：

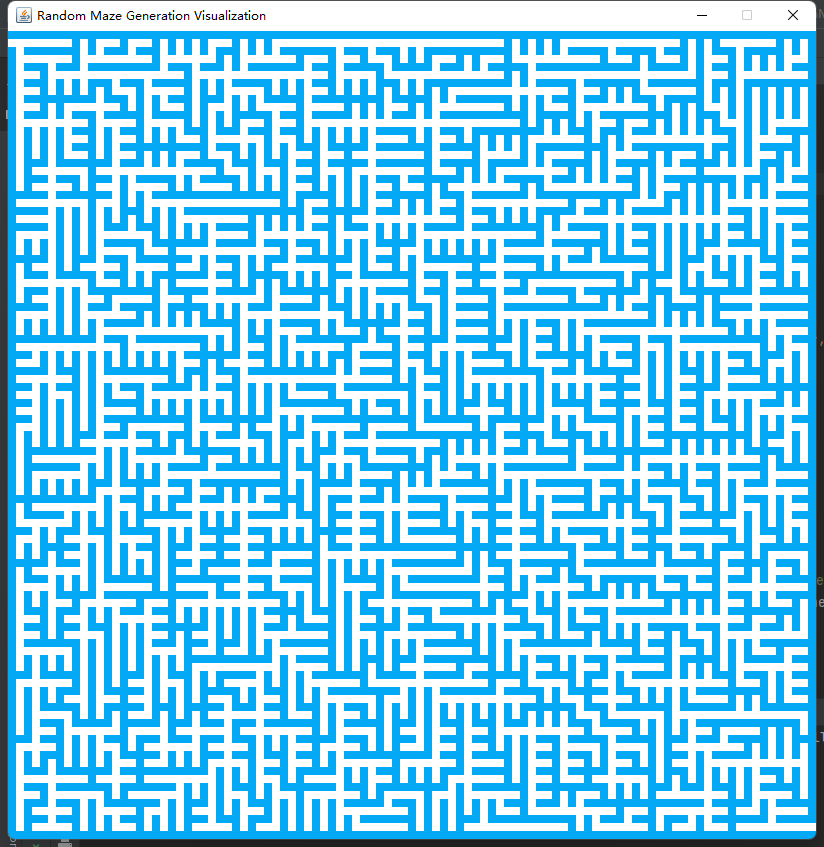
我们可以维护一个迷宫单元格的列表，而不是边的列表。在这个迷宫单元格列表里面存放了未访问的单元格，我们在单元格列表中随机挑选一个单元格，如果这个单元格有多面墙联系着已存在的迷宫通路，我们就随机选择一面墙打通。这会比基于边的版本分支稍微多一点。

相对于深度优先的算法，Prim随机算法不是优先选择最近选中的单元格，而是随机的从所有的列表中的单元格进行选择，新加入的单元格和旧加入的单元格同样概率会被选择，新加入的单元格没有有优先权。因此其分支更多，生成的迷宫更复杂，难度更大，也更自然。生成的迷宫如图所示（生成代码详见RadomMap包）：

生成中： 

生成结束：

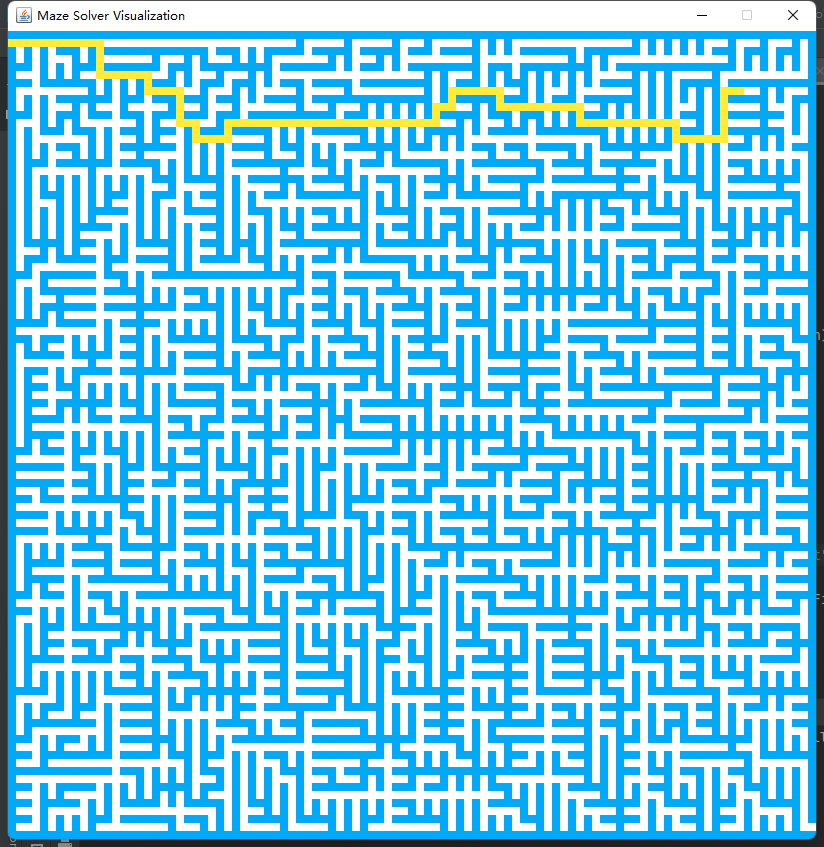
**2.3.1 Recursive division （递归分割算法）**

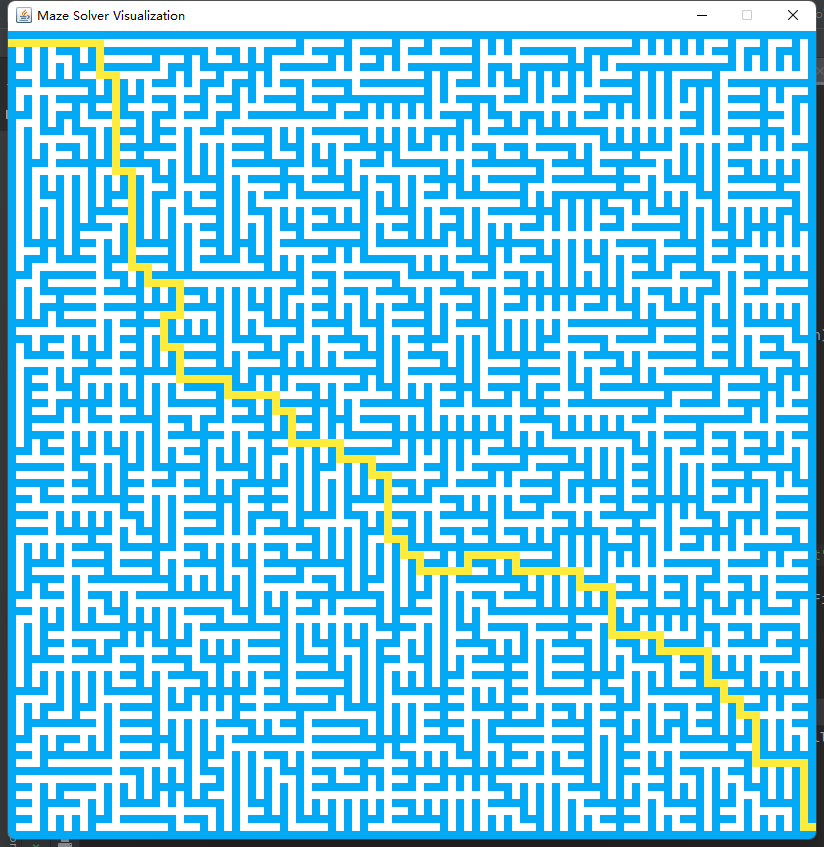
最后是递归分割，递归分割法生成的迷宫较为简单，有点像四叉树，直路多且不扭曲。通俗的说，就是把空间用十字分成四个子空间，然后在三面墙上挖洞（为了确保连通），之后对每个子空间继续做这件事直到空间不足以继续分割为止。此算法十分高效。效果图是如下（代码详见RadomMap3包）：

# 三．迷宫寻路

3.1.1 深度遍历寻路

深度优先搜索属于图算法的一种，英文缩写为DFS即Depth First Search.其过程简要来说是对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次.

顾名思义DFS就是从一个节点出发直到不能访问然后回转到上一层 也就是所说的 回溯+递归 实现方法就是从开始节点出发递归其四周只有满足要求就调用函数进行递归最后返回；所以我设置了两个数组dis，visit；dis存放步数，visit判断是否被访问；效果演示（代码详见DFS包）：



3.2.1 广度遍历寻路

广度优先搜索算法（又称广度优先搜索）是最简便的图的搜索算法之一。其别名又叫BFS，属于一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能位置，彻底地搜索整张图，直到找到结果为止。

如上所示我要做的就是搜索整张，定义一个二维数组visit， visit[x][y]判断坐标x,y是否被访问，初始化visit为0都没有被访问；定义一个结构体point里面的参数有x,y,dis;其中x,y表示坐标，dis表示出发点到该节点的步数；

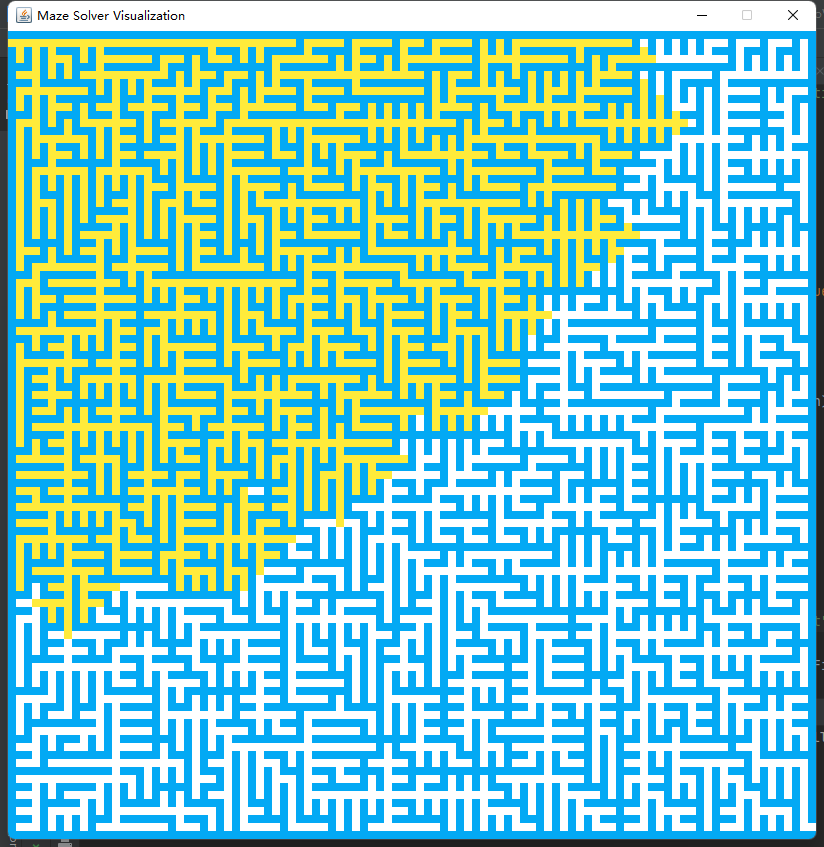
bfs函数操作：

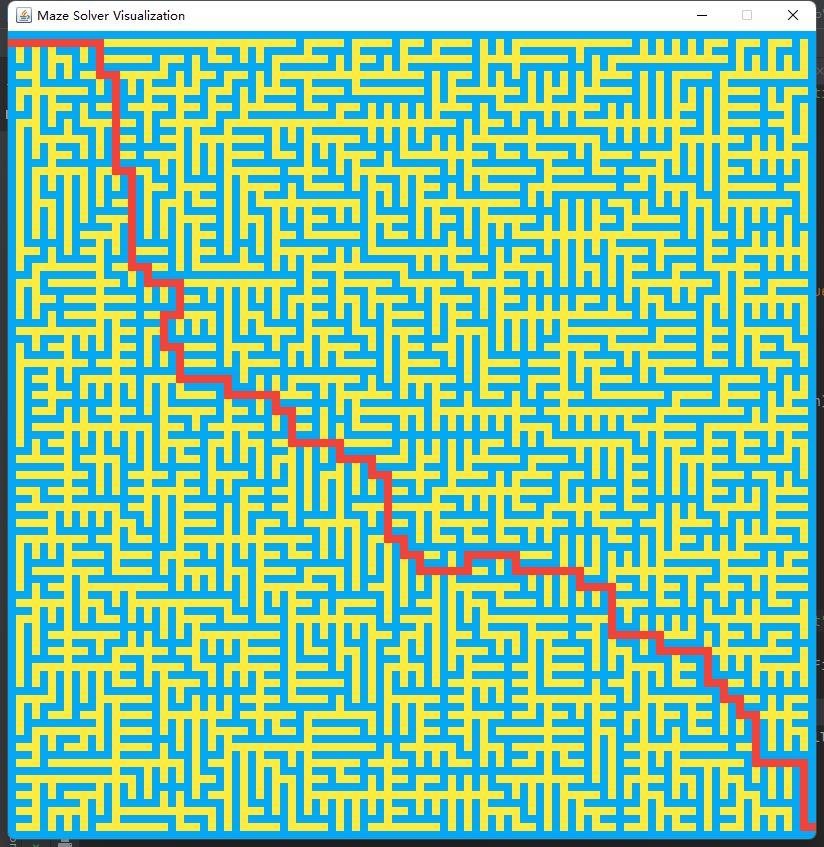
1，将节点添加到队列里面；

2，从队头取出节点将其访问状态设为1，判断其上下左右四个节点将符合要求的节点添加到队列中；

3，重复1，2操作直到从队列中取出的节点终点返回其dis；

效果演示（代码详见BFS包）：

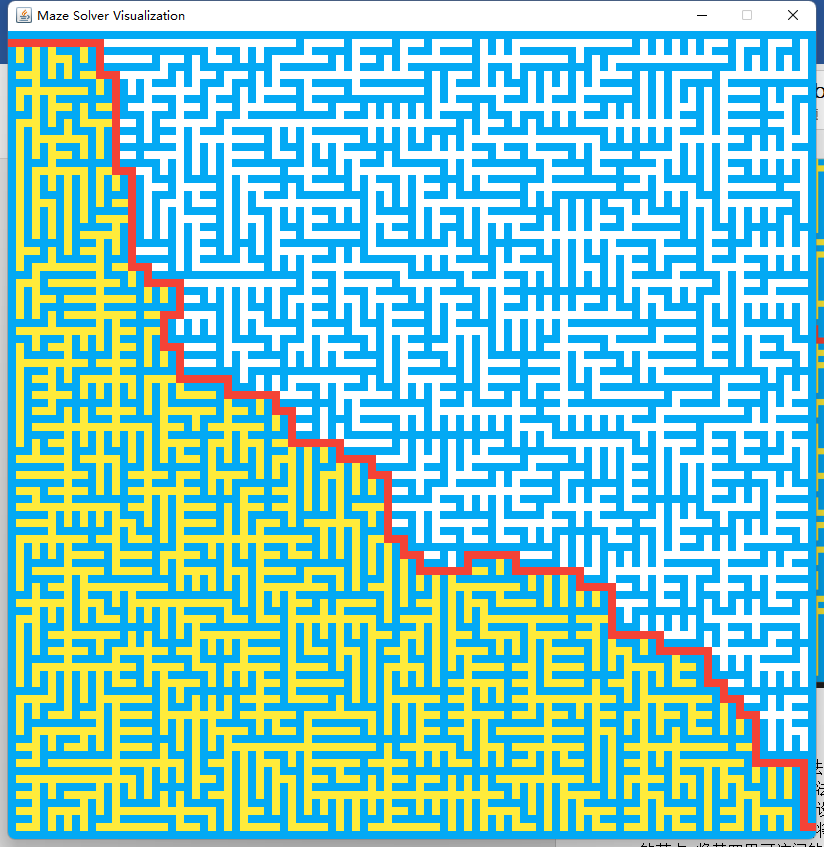




3.3.1 A\*算法

A算法，A\*（A-Star)算法是一种静态路网中求解最短路径最有效的直接搜索方法，也是解决许多搜索问题的有效算法。算法中的距离估算值与实际值越接近，最终搜索速度越快。

A\*算法就是对于每个节点设一个权值 F=G+H;G表示到开始节点的距离，H表示到终点的距离；然后从开始节点开始将可以访问的节点内添加到一个链表中 然后再从链表中取F最小的节点 将其四周可访问的节点添加到链表中 重复操作直到找到终点输出；

结果演示：

# 四．综合迷宫实现

功能介绍：

自动深成迷宫，大小宽度自定义。自动寻找出口。

效果展示：

初始化界面：

寻找路径后：

修改迷宫大小并寻找路径：