任务书

摘要

目录

[第1章 绪论 3](#_Toc483228350)

[1.1 研究背景及意义 3](#_Toc483228351)

[1.2 相关理论 3](#_Toc483228352)

[1.2.1 复杂有解迷宫生成算法 3](#_Toc483228353)

[1.2.2 路径规划算法 4](#_Toc483228354)

[1.3 主要内容 6](#_Toc483228355)

[1.4 结构安排 7](#_Toc483228356)

[第2章 系统需求 7](#_Toc483228357)

[2.1 需求概述 7](#_Toc483228358)

[2.1.1 需求分析目标 7](#_Toc483228359)

[2.1.2 需求概述 8](#_Toc483228360)

[2.1.3 系统结构 8](#_Toc483228361)

[2.2 游戏角色管理 9](#_Toc483228362)

[2.2.1 账号管理 9](#_Toc483228363)

[2.2.2 登录与注销 9](#_Toc483228364)

[2.3 迷宫生成 10](#_Toc483228365)

[2.3.1 自定义迷宫 10](#_Toc483228366)

[2.3.2 生成迷宫 10](#_Toc483228367)

[2.3.3 执行信息 11](#_Toc483228368)

[2.4 自动寻路 11](#_Toc483228369)

[2.4.1 算法选择 11](#_Toc483228370)

[2.4.2 自动寻路 12](#_Toc483228371)

[2.4.3 执行信息 12](#_Toc483228372)

[2.5 信息栏 12](#_Toc483228373)

[2.5.1 迷宫生成 13](#_Toc483228374)

[2.5.2 自动寻路 13](#_Toc483228375)

[2.6 速度设置 13](#_Toc483228376)

[2.7 其他非功能需求 13](#_Toc483228377)

[2.7.1 性能需求 13](#_Toc483228378)

[2.7.2 扩展需求 14](#_Toc483228379)

[2.7.3 界面和交互需求 14](#_Toc483228380)

[2.8 本章小结 14](#_Toc483228381)

[第3章 系统设计 15](#_Toc483228382)

[3.1 概要设计 15](#_Toc483228383)

[3.1.1 系统目标 15](#_Toc483228384)

[3.1.2 设计方案 15](#_Toc483228385)

[3.1.3 主要类图及关系 16](#_Toc483228386)

[3.2 游戏角色管理 16](#_Toc483228387)

[3.2.1 功能描述 16](#_Toc483228388)

[3.2.2 接口设计 17](#_Toc483228389)

[3.2.3 流程分析 17](#_Toc483228390)

[3.2.4 时序图 17](#_Toc483228391)

[3.3 迷宫生成 17](#_Toc483228392)

[3.3.1 功能描述 17](#_Toc483228393)

[3.3.2 接口设计 17](#_Toc483228394)

[3.3.3 流程分析 17](#_Toc483228395)

[3.3.4 时序图 17](#_Toc483228396)

[3.4 自动寻路 17](#_Toc483228397)

[3.4.1 功能描述 17](#_Toc483228398)

[3.4.2 接口设计 17](#_Toc483228399)

[3.4.3 流程分析 17](#_Toc483228400)

[3.4.4 时序图 17](#_Toc483228401)

[3.5 信息栏 17](#_Toc483228402)

[3.5.1 功能描述 17](#_Toc483228403)

[3.5.2 接口设计 17](#_Toc483228404)

[3.5.3 流程分析 18](#_Toc483228405)

[3.5.4 时序图 18](#_Toc483228406)

[3.6 速度设置 18](#_Toc483228407)

[3.6.1 功能描述 18](#_Toc483228408)

[3.6.2 接口设计 18](#_Toc483228409)

[3.6.3 流程分析 18](#_Toc483228410)

[3.6.4 时序图 18](#_Toc483228411)

[3.7 功能模块之间的交互 18](#_Toc483228412)

[3.7.1 交互关系 18](#_Toc483228413)

[3.7.2 设计难点 18](#_Toc483228414)

[3.8 本章小结 18](#_Toc483228415)

[第4章 算法设计 18](#_Toc483228416)

[4.1 迷宫自动生产算法 18](#_Toc483228417)

[4.1.1 递归回溯 18](#_Toc483228418)

[4.1.2 XXXXXX 18](#_Toc483228419)

[4.2 自动寻路算法 19](#_Toc483228420)

[4.2.1 宽度优先搜索 19](#_Toc483228421)

[4.2.2 深度优先搜索 19](#_Toc483228422)

[4.2.3 启发式搜索 19](#_Toc483228423)

[4.2.4 遗传算法 20](#_Toc483228424)

[4.3 本章小结 20](#_Toc483228425)

[第5章 系统实现 20](#_Toc483228426)

[5.1 运行环境 20](#_Toc483228427)

[5.2 数据分析与比较 20](#_Toc483228428)

[5.2.1 执行数据 20](#_Toc483228429)

[5.2.2 分析与比较 20](#_Toc483228430)

[5.3 寻路算法总结 20](#_Toc483228431)

[5.4 系统展示 20](#_Toc483228432)

[总结与展望 20](#_Toc483228433)

[致谢 20](#_Toc483228434)

[参考文献 20](#_Toc483228435)

# 绪论

## 研究背景及意义

2016年3月进行的围棋人机大战中， AlphaGo最终以4:1战胜了韩国九段名将李世石，引起了全世界的广泛关注，也让人工智能、机器人等概念进入了公众的视野。近年来，越来越多的机器人被应用到生产和生活中，家里的扫地机器人可以让你在结束一天的劳累工作后免去做家务的烦恼；高级轿车的自动驾驶技术可以让你有个免费安全的24小时司机。

现代社会高速发展，我们的日常生产和生活越来越离不开自动化和人工智能技术，越来越多的工场使用机器人等高科技设备来代替人工完成一些重复性高，精确度高，任务繁重的生产任务。创新工场的CEO李开复先生在2016年WISE大会上说：“在未来十年，世界上50%的工作，都会被人工智能所取代”。人工智能及自动化技术将在今后的人类社会中扮演至关重要的角色。绝大部分的机器人等智能设备在工作过程中，都要涉及到自动寻路这一环节，即机器人需要像人类一样在复杂的环境中实现从起始点到目标点的过程。机器人完成自动寻路需要依靠程序人员事先为其设计好的自动寻路程序，而程序如何在各种复杂的真实情况下快速、稳定地寻找出最佳路径就需要程序人员在众多已有的寻路算法中选择出最适用的算法，或者设计出更高效的寻路算法。

迷宫可以模拟复杂的工作环境，通过设计和实现一个可以自动生成复杂有解迷宫并自动寻路的系统，可以了解和掌握多种路径规划算法，比较各种算法之间的区别和应用范围，并根据实际情况对算法进行优化，从而在解决问题时提出更加高效、合理的路径规划算法，提高人工智能水平。

## 相关理论

路径规划问题指的是：在一个充满障碍物的环境中，如何避开障碍物，规划一条从起始点到达目标点的最优路径。路径规划问题涉及到迷宫的表示和自动生成、路径规划算法和搜索策略。迷宫自动生成算法负责如何生成一个复杂有解的迷宫并有效地表达出来；路径规划和搜索算法研究的是如何在已有迷宫的基础上求解出最优路径的过程。

### 复杂有解迷宫生成算法

常见的迷宫在程序中被描述为一个M\*N大小的矩阵，每个单元分别用0和1表示能否通过。迷宫自动生成算法就是通过一定的策略将初始全为0的M\*N矩阵变为一个任意两点都有路径可以到达的由0和1组成的M\*N矩阵。

递归回溯法是常见的生成复杂有解迷宫的算法。它的基本流程是：初始一个M\*N的矩阵，任意两个单元之间的四个方向的墙壁都是存在的。选择一个单元为开始单元，从当前单元四个方向的临界单元中选择一个合法的没有访问过的单元，并打通与它之间的墙壁，然后将当前单元压入栈中，并将打通的单元作为当前单元，接着继续寻找没有访问过的临界单元。如果当前单元已经没有未访问过的临界单元时，则从退回到栈顶单元作为当前单元，继续寻找它的临界单元是否可以访问。当栈为空（初始单元被退回）的时候，算法结束，M\*N的矩阵单元变为了一个有解的迷宫。

### 路径规划算法

路径规划算法是在之前生成的迷宫矩阵基础上，找到一条可以从初始点到达目标点的最优路径。常见的路径规划算法有宽度优先搜索（BFS）、深度优先搜索（DFS）、启发式搜索、遗传算法等。

1. 宽度优先搜索（Breadth First Search）

宽度优先搜索算法（又称宽度优先搜索）是最简便的图的搜索算法之一，这一算法也是很多重要的图的算法的原型。Dijkstra单源最短路径算法和Prim最小生成树算法都采用了和宽度优先搜索类似的思想。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。

宽度优先搜索算法的具体描述为：已知图G=(V,E)和一个源顶点s，宽度优先搜索以一种系统的方式探寻G的边，从而“发现”s所能到达的所有顶点，并计算s到所有这些顶点的距离(最少边数)，该算法同时能生成一棵根为s且包括所有可达顶点的宽度优先树。对从s可达的任意顶点v，宽度优先树中从s到v的路径对应于图G中从s到v的最短路径，即包含最小边数的路径。该算法对有向图和无向图同样适用。

之所以称之为宽度优先算法，是因为算法自始至终一直通过已找到和未找到顶点之间的边界向外扩展，就是说，算法首先搜索和s距离为k的所有顶点，然后再去搜索和S距离为k+l的其他顶点。

宽度优先搜索一般使用队列（queue）来实现，整个过程可以看做一个倒立的树形：

1、把根节点放到队列的末尾。

2、每次从队列的头部取出一个元素，查看这个元素所有的下一级元素，把它们放到队列的末尾。并把这个元素记为它下一级元素的前驱。

3、找到所要找的元素时结束程序。

4、如果遍历整个树还没有找到，结束程序

1. 深度优先搜索（Depth First Search）

深度优先搜索是一种在开发爬虫早期使用较多的方法。它的目的是要达到被搜索结构的叶结点(即那些不包含任何超链的HTML文件) 。在一个HTML文件中，当一个超链被选择后，被链接的HTML文件将执行深度优先搜索，即在搜索其余的超链结果之前必须先完整地搜索单独的一条链。深度优先搜索沿着HTML文件上的超链走到不能再深入为止，然后返回到某一个HTML文件，再继续选择该HTML文件中的其他超链。当不再有其他超链可选择时，说明搜索已经结束。

深度优先搜索算法是一种用于遍历或搜索树或图的算法。沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点v的所在边都己被探寻过，搜索将回溯到发现节点v的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。如果还存在未被发现的节点，则选择其中一个作为源节点并重复以上过程，整个进程反复进行直到所有节点都被访问为止，也属于盲目搜索。

深度优先搜索一般使用栈（stack）来实现，整个过程也可以想象成一个倒立的树形：

1、把根节点压入栈中。

2、每次从栈中弹出一个元素，搜索所有在它下一级的元素，把这些元素压入栈中。并把这个元素记为它下一级元素的前驱。

3、找到所要找的元素时结束程序。

4、如果遍历整个树还没有找到，结束程序。

1. 启发式搜索（Heuristically Search）

启发式搜索又称为有信息搜索(Informed Search)，它是利用问题拥有的启发信息来引导搜索，达到减少搜索范围、降低问题复杂度的目的，这种利用启发信息的搜索过程称为启发式搜索。启发式策略可以通过指导搜索向最有希望的方向前进，降低了复杂性。通过删除某些状态及其延伸，启发式算法可以消除组合爆炸，并得到令人能接受的解(通常并不一定是最佳解)。

然而，启发式策略是极易出错的。在解决问题的过程中启发仅仅是下一步将要采取措施的一个猜想，常常根据经验和直觉来判断。由于启发式搜索只有有限的信息(比如当前状态的描述)，要想预测进一步搜索过程中状态空间的具体行为则很难。一个启发式搜索可能得到一个次最佳解，也可能一无所获。这是启发式搜索固有的局限性。这种局限性不可能由所谓更好的启发式策略或更有效的搜索算法来消除。一般说来，启发信息越强，扩展的无用节点就越少。引入强的启发信息，有可能大大降低搜索工作量，但不能保证找到最小耗散值的解路径(最佳路径)。因此，在实际应用中，最好能引入降低搜索工作量的启发信息而不牺牲找到最佳路径的保证。

用于评价节点重要性的函数称为估价函数，其一般形式为：

f(x) = g(x) + h(x)

式中：g(x)为从初始节点到节点x付出的实际代价；h(x)为从节点x到目标节点的最优路径的估计代价。启发性信息主要体现在h(x)中，其形式要根据问题的特性来确定。

1. 遗传算法（Genetic Algorithm

遗传算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群（population）开始的，而一个种群则由经过基因（gene）编码的一定数目的个体(individual)组成。每个个体实际上是染色体(chromosome)带有特征的实体。染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现（即基因型）是某种基因组合，它决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此，在一开始需要实现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂，我们往往进行简化，如二进制编码，初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代（generation）演化产生出越来越好的近似解，在每一代，根据问题域中个体的适应度（fitness）大小选择（selection）个体，并借助于自然遗传学的遗传算子（genetic operators）进行组合交叉（crossover）和变异（mutation），产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境，末代种群中的最优个体经过解码（decoding），可以作为问题近似最优解。

## 主要内容

智能迷宫游戏系统是一个可以自动生成随机复杂有解迷宫，然后选择不同的路径规划算法寻找一条从起始点到目标点的最优路径，并且可以将迷宫生成过程和寻路过程动态地展示出来的系统。

本文的研究工作主要包括：输入迷宫的宽度、高度和起始点位置，然后生成随机、复杂、有解的迷宫；选择多种不同的路径算法寻找从起始点到目标点的最优路径并对不同算法进行分析和比较；动态可控地展示迷宫生成和路径规划过程；总结不同路径规划算法的优势和适用范围，给出根据具体情况的改进等。

## 结构安排

本文一共包括五个章节，详细叙述了论文中所涉及到的相关理论算法以及作者所做的分析和总结工作，并且阐述了相应的智能迷宫游戏系统的设计及实现，最后对人工智能路径规划的未来发展做出展望。

第一章，绪论。主要介绍本课题的研究背景及其意义，对涉及到的相关路径规划算法理论做了简单的介绍，并说明了本课题的研究工作内容及本文结构。

第二章，系统需求。这一章主要阐述智能迷宫游戏系统的系统需求，包括需求概述，系统结构，各个模块的功能需求图解和非功能需求。

第三章，系统设计。这一章主要阐述智能迷宫系统的系统设计，包括概要设计，各个功能模块的详细设计、接口设计、流程分析和图解，模块之间的交互设计等。

第四章，算法设计。这一章主要阐述本文所研究的迷宫生成算法和路径规划算法，包括数据结构设计、算法流程图解和具体实现等。

第五章，系统实现。这一章主要阐述智能迷宫游戏系统及论文研究的总体情况，包括系统运行环境、运行截图展示，不同路径规划算法的执行数据及其分析、比较和总结。

总结和展望对整篇论文进行了总结，提出了研究过程的不足之处，并对未来的人工智能研究进行展望。

# 系统需求

## 需求概述

### 需求分析目标

为了明确本系统的需求，合理地安排软件的开发计划与进度；保证软件开发人员有目的、有安排地编写程序；使测试人员的测试工作完整、有据可循。

进行智能迷宫游戏系统进行需求分析的主要目的：

1)对智能迷宫系统的功能做一个全面的分析和阐述，帮助作者判断该系统实现功能的正确性、一致性和完整性，促使作者在对该系统进行概要设计之前周密地、全面地思考系统的需求。

2)全面地了解和描述实现该系统所需要的全部信息，为系统的设计、实现和测试提供一个标准

需求分析的具体内容可以总结为以下四个方面：系统的功能需求，系统的主要接口，系统的非功能性需求和技术难点等。

### 需求概述

智能迷宫游戏系统是一个可以让用户能够生动、直观地了解、学习复杂有解迷宫自动生成算法和各种路径规划算法的学习工具程序，它内置了几种常用的自动寻路算法和复杂有解迷宫生成算法，用户可以自定义迷宫的大小、起始位置等参数，选择不同的算法来自动生成复杂有解迷宫，能看到选择不同的自动寻路算法让游戏角色从起点走到迷宫终点的动态过程，并且能比较不同算法在不同维度的差别，从而能够直观地了解和学习各种寻路算法。

本系统的主要功能包括迷宫角色管理、复杂有解迷宫的绘制和生成、自动寻路算法的实现及不同算法的多维度比较等。用户输入自定义的迷宫参数，选择迷宫生成算法，然后点击生成迷宫按钮，接着在迷宫主界面可以看到迷宫的动态生成过程，并能够在右侧输出栏看到算法的耗时情况等信息。生成完迷宫之后，用户可以选择不同的自动寻路算法，然后点击寻路按钮，可以看到迷宫主角从起到一步一步寻路到终点的动态过程，并且右侧信息输出栏会显示算法的耗时情况等信息。用户还可以选择两种不同的寻路算法点击比较按钮，可以看到所选算法的稍微快一点的动态寻路过程，两种算法寻路都完成后，可以在右侧看到两种算法的比较图表。

### 系统结构

图 2.1—1

用户层

界面交互

游戏角色管理模块

智能迷宫自动生成模块

自动寻路模块

信息栏模块

速度设置模块

智能迷宫游戏系统的结构如图 2.1—1所示，用户通过与界面进行交互，输入一些操作命令，进入下层的逻辑模块。界面交互主要负责给用户提供美观、友好的操作界面，使得用户不仅可以通过方便、简洁、易于理解的操作步骤完成想要达到的目的，而且可以直观、清晰地看到操作带来的反馈和迷宫的处理过程及结果。

游戏角色管理模块主要负责对智能迷宫游戏系统的账号管理，登录和注销功能。智能迷宫自动生成模块主要负责根据用户选择的不同复杂有解迷宫自动生成算法来自动创建迷宫，绘制迷宫等逻辑。自动寻路模块主要负责根据用户选择的不同自动寻路算法动态地完成游戏角色的自动寻路过程，其中还要负责控制游戏角色的移动，寻路的动态过程的可视化绘制等逻辑，并把寻路算法执行的信息输出到信息栏和比较图表模块。信息栏模块主要负责将寻路算法执行的信息有序、完整地输出到输出栏中，并且能够将不同寻路算法的数据以图表的形式显示出来。速度设置模块主要负责设置整个系统的单步执行速度，用户可以随时设置速度并立刻应用于当前的执行过程中。

## 游戏角色管理

游戏角色管理模块主要负责智能迷宫游戏系统的账号管理，账号登录和账号注销功能。该模块可以保证用户只有通过账号验证登录后才可以使用本系统。

### 账号管理

账号管理主要负责智能迷宫游戏系统的账号注册、密码修改等功能。用户在第一次打开本系统时需要登录账号才能使用，如果之前没有在该系统注册过账号，用户需要进行账号注册。用户需要输入符合要求的用户名、密码及重复密码，然后点击账号注册按钮来成功注册账号。成功注册账号后，会自动登录当前注册的账号。

用户登录过一个账号后，可以进入账户管理界面，进行账号信息修改。可以修改的信息包括用户的昵称、个性签名、密码等。修改密码时需要用旧密码做验证，验证通过后才可修改新的密码。成功修改密码后，用户需要用新密码重新登录一遍账号。

### 登录与注销

用户在第一次打开智能迷宫游戏的时候，需要进行账号登录验证。用户需要输入注册过的用户名、密码来验证身份，验证通过后才可以使用本系统。登录界面应该包括账号注册按钮，方便没有账号的用户进行注册操作。另外，登录界面还应该包扩自动登录选项，默认打开，即如果本次登录成功后，下次再打开系统时将默认登录本次的账号。

如果用户想要切换当前登录的账号或者退出当前账户时，可以通过账号注销功能，点击账号注销时，应该弹出二级窗口提醒用户是否确定退出注销当前账号，用户点击“确认”按钮后才注销账号并退回到登录界面。如果用户点击“取消”按钮，则取消注销操作。

## 迷宫生成

迷宫自动生成功能模块包括的主要功能有：用户自定义生成迷宫的大小、起始位置等信息，生成随机、复杂、有解的迷宫；将迷宫的动态生成过程动态地展现出来；将迷宫生成过程的算法执行信息发送给信息栏模块。

### 自定义迷宫

自定义迷宫功能允许用户输入预生成迷宫的高度、宽度、起始X坐标、起始Y坐标等信息。系统需要设计出人性化的设置界面：包含输入信息的合法性限制说明，键盘输入、鼠标点击上下箭头等多种输入方式，系统默认输入信息等。系统需要对用户输入的不合法信息进行处理，比如用户输入了超出宽度或者高度限制的起始位置坐标，系统需要自动将输入值改为限制的相应界值；用户输入了非奇数的宽度或者高度，系统需要自动将输入值改为最近似的合法奇数值。

另外，因为系统生成迷宫的过程是动态的，需要一定时间，如果在生成迷宫的过程中用户修改了迷宫的初始宽度、高度等信息，界面应该能够正确响应，并且系统应该将新的输入信息缓存，应用于下一次迷宫的生成，而不能影响当前正在生成的迷宫。

### 生成迷宫

当用户输入了预生成迷宫的初始信息后，点击生成迷宫按钮，开始生成迷宫。迷宫界面需要先根据用户输入的宽度和高度绘制一个相应大小的矩阵，初始时矩阵单元的背景颜色都是深灰色，表示不可通行。接着从用户设置的起始位置开始，系统逐步将当前算法执行结果对应的矩阵单元的背景颜色设置为白色，表示可以通行。整个迷宫生成过程是逐步、动态地展现出来的，系统会用绿色来表示算法递归的当前单元，用红色表示回溯时的当前单元。在整个迷宫生成期间，用户都可以设置单步的速度，即可以动态地控制迷宫的生成速度。

用户设置了不同的初始信息后，再次点击生成迷宫按钮后要清除上一次的迷宫，根据新的初始信息重新绘制新的迷宫，上一次的迷宫结果不能对新的迷宫生成过程有影响。即使没有改变迷宫初始信息，用户再次点击生成迷宫时，系统也应该清除上一次的迷宫，并生成相同大小、初始位置，但路径与上次不同的复杂有解迷宫，即有随机性。

如果用户在迷宫的生成过程中点击了生成迷宫或者其他功能模块的互斥按钮，系统应该通过弹出消息窗的方式提醒用户，等待本次生成迷宫过程完成后再进行操作，并无视相应的程序处理，以免出现不可控制的错误。

### 执行信息

在迷宫的动态生成过程中，迷宫生成模块应该将算法执行的每一步信息发送给信息栏模块，包括算法开始和结束提示、当前执行单元的位置信息、试探方向、是递归还是回溯和堆栈的深度等等。

发送的执行信息应该与用户看到的迷宫生成过程保持同步，不能出现用户看到的单元状态和输出信息不一致的情况。用户再次点击“生成迷宫”按钮时，程序应该清除上一次的执行信息。

## 自动寻路

自动寻路功能模块包含的主要功能有：用户可以选择不同的算法，迷宫路径规划，将寻路过程逐步地、动态地展现出来，将算法执行信息同步地发送给信息栏模块。

### 算法选择

本系统应该提供深度优先搜索、宽度优先搜索、启发式搜索、遗传算法等多种路径规划算法供用户选择。所有选择框应该为单选框，选中某一个算法时，其他算法选择框都应该是非选中状态，并且必选有一个算法是选中状态。默认状态下，系统选中深度优先搜索供路径规划使用。

如果在自动寻路过程用户切换了不同的算法，并不能影响本次的寻路过程，而是对下一次的寻路生效。

### 自动寻路

用户点击“自动寻路”按钮后，系统开始自动寻路。系统会从迷宫左上角的点开始，逐步将寻路算法执行结果对应的矩阵单元背景颜色设置为黄色，当算法寻路到迷宫右下角的终点时，会逐步回退重现最终路径，系统应该从终点开始逐步地将最终路径对应的矩阵单元的背景颜色设置绿色，表示系统最终寻找到的路径。最后要清除无效的试探信息，即把黄色的单元格设置为白色。整个过程是动态的，逐步的，用户可以随时设置寻路的单步速度以便于直观地观察算法的执行过程。

如果用户事先没有生成迷宫而直接点击“自动寻路”按钮，系统应该弹出窗口提示用户需要先生成迷宫。如果在自动寻路的过程中再次点击“自动寻路”按钮，系统应该弹出窗口提示用户当前正在寻路中，请等待本次寻路完成后再试。

一次寻路完成后，用户选中另一种路径规划算法或者继续使用当前的算法，然后点击自动寻路时，迷宫界面应该清除上一次的寻路结果，根据当前选中的算法开始新的寻路过程，并且上一次的寻路结果不能对本次的寻路造成任何影响。

### 执行信息

在路径规划的动态过程中，自动寻路模块应该将算法执行的每一步信息发送给信息栏模块，包括算法开始和结束提示、当前寻路结果单元的位置信息、试探方向、是递归还是回溯和堆栈的深度等等。

发送的执行信息应该与用户看到的自动寻路过程保持同步，不能出现用户看到的单元状态和输出信息不一致的情况。用户再次点击“自动寻路”按钮时，程序应该清除上一次的执行信息。

## 信息栏

信息栏模块包含的主要功能有：处理迷宫生成过程发来的信息，处理自动寻路过程发来的信息。信息栏界面应该以类似命令行模式的方式输出信息，当输出信息过多时，界面应该自动生成滚动条并滚动到最底部。另外，输出的算法信息应该简单易懂，与算法执行过程保持同步，便于用户理解算法执行过程。

### 迷宫生成

在迷宫生成模块开始生成迷宫时，信息栏界面需要清空上一次的输出信息，并输出算法开始执行的提示信息。在单步递归生成迷宫的过程中，信息栏界面需要输出单步算法执行的单元格位置、本次随机方向等信息。在回溯程序栈的过程中，信息栏界面需要指出回溯过程及其位置，算法执行深度等信息。生成迷宫算法执行完成后，信息栏界面需要输出算法完成的提示信息。

### 自动寻路

在自动寻路模块开始寻路时，信息栏模块需要清除上一次的执行信息，并输出当前选择的路径规划算法和算法开始的提示信息。在自动寻路的单步执行过程中，信息栏界面需要输出本次执行结果对应的矩阵单元的位置信息。在自动寻路回溯确认最终路径的过程中，信息栏界面需要输出最终路径对应的矩阵单元的位置信息。当自动寻路完成时，信息栏界面需要输出算法结束的提示信息。

## 速度设置

速度设置模块主要负责设置整个系统的单步执行速度。该模块应该由一个可以拖动的滑竿组成，让用户设置速度更加便捷。速度范围应该保证当用户不想重视过程时很快的完成，当用户需要仔细观察每一步时又可以有充足时间可以理解和反应。

系统默认的单步速度为50ms，保证用户的初次体验效。另外，用户可以随时地设置单步速度并立即应用到当前的过程中，做到想慢就慢，想快就快。

## 其他非功能需求

智能迷宫游戏系统作为一个学习工具软件，除了必要的功能需求以外，还需要满足一些性能、扩展性、界面和交互性上的非功能需求。

### 性能需求

智能迷宫游戏系统需要保证在迷宫自动生成和自动寻路的动态过程，程序不能出现任何的卡顿，崩溃等性能问题。当用户实时改变单步速度时，程序生成的动态过程要在下一帧时立刻响应最新的速度，不允许出现延迟。当用户连续生成迷宫、自动寻路时，程序应该保证连续稳定地正常运行。对于任何可能的非法操作和输入，系统都应该有处理措施，避免产生不可控制的错误。

### 扩展需求

随着路径规划算法的不断演进和优化，越来越多的路径规划算法将会被提出，本系统应该可以很便捷地添加新的寻路算法并将它的执行过程直观地展现出来，而不需要对程序结构做非常大的变动。

### 界面和交互需求

作为一个工具性游戏软件，智能迷宫游戏系统需要有简洁、友好、易操作的界面，多种直观易懂的颜色与合理便捷的输入输出组成的良好交互以保证用户可以通过熟悉地、舒适地方式进行输入，直观易懂地观察系统输出。

界面主要包含三个部分，上方是操作栏窗口，它又细分为三个部分：迷宫生成部分，自动寻路部分和速度设置部分。其中迷宫生成部分包含迷宫初始信息输入框和生成迷宫按钮；自动寻路部分包括左侧的算法单选框和右侧的自动寻路按钮；速度设置部分包含一个可以滑动的滑竿。下方左侧是整个迷宫游戏的窗口，该系统的主要窗口，负责显示迷宫和寻路。下方右侧是信息输出栏。

## 本章小结

本章主要对智能迷宫游戏系统的需求做出了分析，包括需求分析的目标，系统的总体结构，游戏角色管理、迷宫生成、自动寻路、信息栏和速度设置五个功能模块，性能、扩展性、界面和交互性等非功能需求。明确了本系统的需求，方便作者更加合理地安排软件的开发计划与进度；保证作者有目的、有安排地编写程序；使得系统的测试工作更加完整、有据可循。

# 系统设计

## 概要设计

优秀的概要设计可以指导本系统的详细设计，明确本系统开发的任务和需求，帮助作者清楚软件开发的流程，正确地理解需求分析中说明的需求完整地实现，而且可以确保后期的软件自测试工作有据可循。

### 系统目标

智能迷宫游戏系统需要实现三个主要目标：

1.用户可以自定义迷宫的大小、起始位置等初始信息，然后点击生成迷宫按钮，接着系统动态地将迷宫从初始状态一步一步变为有解的迷宫，并且将算法的执行信息实时显示到信息栏中。

2.用户可以选择不同自动路径规划算法，点击开始寻路后，随后系统按用户选择的算法开始寻路，将算法的执行过程动态、逐步地显示出来，并将算法的执行信息实时输出到信息栏中。

3.在迷宫生成过程和自动寻路过程中，用户可以实时设置系统的单步运行速度，并立刻运用到下一帧中。

### 设计方案

作者在设计本系统时采用了面向对象的设计思想，作为一个迷宫游戏系统，需要有登录类来处理用户的登录验证、账号注册和管理等功能，储存账号的用户名、密码等信息；需要有迷宫类，包含迷宫生成、自动寻路等方法，储存迷宫的大小、单元情况等信息。为了显示迷宫生成过程和自动寻路过程的算法动态执行信息需要有信息栏类，包含添加信息、重置信息等方法。为了处理用户的操作，需要有一个处理界面交互的类，包含用户输入自定义迷宫的初始信息，路径规划算法选择和速度设置等，为了方便管理，降低系统耦合性，将主界面类按功能模块划分为三个子类，分别是迷宫标签类，寻路标签类和速度设置类。

### 主要类图及关系

UserManager类：

登录

Maze类：

生成迷宫

自动寻路

MazeLabel类

自定义迷宫

RouteLabel类

选择算法

SpeedLabel类

设置速度

InfoWindow类：

输出信息

图 3.1—1

如图3.1—1所示，系统的主要有六个类：UserManager类主要负责系统的登录验证、账号注册、注销及管理功能；MazeLabel类主要负责对用户输入迷宫的大小、起始位置等初始信息进行处理和缓存；RouteLabel类主要负责用户进行路径规划算法的选择；SpeedLabel类主要负责用户进行系统单步速度的设置功能；Maze类是本系统最重要的类，它负责处理初始迷宫的绘制、迷宫的动态生成、动态自动寻路等处理过程及迷宫信息的保存。它也是本系统的设计难点所在，作者需要在此类中设计随机复杂有解迷宫的生成算法和多种路径规划算法；InfoWindow类主要负责将Maze类迷宫生成过程和自动寻路过程的算法执行信息显示出来供用户查看

## 游戏角色管理

### 功能描述

游戏角色管理功能主要负责账号验证登录、注册、注销和管理等功能。用户在第一次打开系统时，会弹出登录界面，用户在输入正确的用户名和密码后，点击登录按钮。系统验证成功之后，才能进入到系统主功能界面。登录界面上包含记住密码选项框，默认选中；包含注册按钮，方便没有账号的用户直接跳转到注册界面。成功登录后，用户可以修改账户密码等信息，也可以通过“注销”按钮退出当前已经登录的账号。

### 接口设计

class UserManager

void login ();

int register();

void logout();

void modifyPsw();

void modifyInfo();

图 3.2—1

如图3.2—1所示，在UserManager类中，login(String userName, String password, bool save )方法用来处理用户登录功能，userName参数是用户登录时输入的用户名，password参数是用户输入的用户密码，save参数是用户是否选择记住当前登录的账号；register(String userName, String password String repass)方法用来处理用户注册功能，参数userName是用户注册时输入的用户名，password参数是密码，repass是重复密码，返回值是本次成功注册账号的唯一ID；logout(int userID)方法用来处理用户注销功能，userID是注销账号的唯一ID；modifyPsw (String oldPass, String newPass)函数用来处理用户修改密码的操作，oldPass是验证用户的旧密码，newPass是用户想要修改的新密码；modifyInfo(int key, String content)方法用来处理用户修改账户信息，key参数是用来标识本次修改的内容类型，content参数是本次修改的内容。

### 数据结构

struct UserData{

int ID;

String userName;

String password;

String instruction;

}

图—2

如图3.2—2所示，游戏角色管理模块使用UserData数据结构来存储用户的账户信息，其中ID是区别每个账号的唯一标识；userName是账户的用户名，由8-16位的字母或数字组成；password是账户密码，由6-12位字母或数字组成；instruction是账户的介绍信息，可以为空，最大64个字符。

### 时序图

UserManager

4: modify()

1:打开系统

modifyWindow

logoutWindow

3:修改信息

2: login()

loginWindow

5:注销账号

6: logout()

7: show()

游戏角色管理模块的时序图如图3.2—3所示，用户第一次打开系统后，输入正确的用户名和密码，点击登录界面的“登录”按钮调用UserManager类的login()方法来验证用户。用户通过点击修改信息界面的“保存”按钮调用UserManager类的modify()方法修改账户信息。当用户想要退出当前的账号或者切换账号时，通过点击注销界面的“注销”按钮调用UserManager的logout()方法，成功注销后返回登录界面。

图 —3

## 迷宫生成

### 功能描述

迷宫生成模块主要负责初始迷宫的绘制、迷宫的动态逐步生成、发送算法执行信息给信息栏模块。用户在迷宫标签界面输入自定义的迷宫宽度、高度、起始位置X坐标和起始位置Y坐标等信息，然后点击生成迷宫按钮，迷宫生成模块先根据新的迷宫宽度高度绘制一个新的初始化的迷宫，接着执行迷宫生成算法逐步、动态地生成随机复杂有解迷宫，并将算法每一步的执行信息发送给信息栏模块显示出来。在生成迷宫的过程中，迷宫生成模块要能实时响应速度设置模块设置的最新速度，还要对非法的互斥操作进行处理，防止出现不可控制的错误。

### 接口设计

class Maze

void createMaze();

void resetMaze();

void setWidth();

void setHeight();

void setStartX();

void setStartY();

void setIsOncreating();

void setIsCreated();

void onStep();

void onReturn();

图 —

Maze类中关于迷宫生成功能的相关接口如图3.3—1所示，createMaze()方法负责当用户第一次点击生成迷宫后，绘制初始迷宫并逐步执行迷宫生成算法；resetMaze()方法用于当用户再一次点击生成迷宫时，清理上一次迷宫的信息，销毁相关界面等；setWidth(int w)方法用于用户在迷宫标签界面设置迷宫的宽度，参数w是用户新设置的宽度；setHeight(int h)方法用于用户在迷宫标签界面设置迷宫的高度，参数h是用户新设置的高度；setStartX(int x)方法用于用户设置迷宫的起始点X坐标，参数x是用户新设置的X坐标；setStartY(int y)方法用于用户设置迷宫的起始点Y坐标，参数x是用户新设置的Y坐标；setIsOncreating(bool b)方法用于设置系统当前是否正在处于产生迷宫的过程中，参数b表示是否处于正在生成迷宫的过程中，当b为true时，系统会屏蔽用户非法输入的与生成迷宫过程互斥的操作；setIsCreated(bool b)方法用于设置系统是否已经生成了一个迷宫，防止用户在没有生成迷宫时就点击“自动寻路”按钮而发生无法预知的错误。onStep(int x, int y)方法用于迷宫生成算法每一步执行完成后调用，设置迷宫单元的背景颜色，参数x表示单元格所在的Y坐标，参数y表示单元格所在位置的Y坐标；onReturn(int dir, int x , int y)方法用于迷宫生成算法过程中的回溯步骤，dir为回退的方向，1，2，3和4分别表示上下左右，参数x和y分别表示回溯单元格所在位置的X和Y坐标。

### 数据结构

class MyPoint

int x;

int y;

bool state;

bool visited;

图 —

MyPoint类主要用于存储迷宫单元的信息，其关于迷宫生成模块的主要数据结构信息如图3.3—2所示，包括位置和一些状态标识信息。对于迷宫生成功能来说，变量x存储迷宫单元所在的X坐标；变量y存储Y坐标；变量state用于标记在生成迷宫结果中本单元是否可以通过，true表示可以通过，false表示不可通过，初始状态的迷宫即为state全为false的矩阵；变量visited用于标记在生成迷宫算法的执行过程中当前单元是否已经访问过，true表示访问过，false表示没有访问过，初始时全为false。

## 自动寻路

### 功能描述

当用户生成一个迷宫后，选择一个路径规划算法，然后点击开始寻路，自动寻路模块会根据用户选择的算法逐步动态地寻找路径，并将算法每一步的运行结果发送给信息栏模块显示出来，还会将最终结果用明显的颜色表示出来。在自动寻路过程中，自动寻路模块需要实时响应速度设置模块最新设置的速度，并处理与自动寻路过程互斥的各种非法操作，防止系统出现错误。

### 接口设计

class Maze

void setAlgo();

void findPath();

void resetPath();

void setIsFindindPath();

void onFindStep();

void onFindStepBack();

void onFindReturn();

图 —

Maze类中关于自动生成模块的主要接口如图3.4—1所示，setAlgo(AlgoType index)用于处理用户选择路径规划算法，参数index类型为AlgoType的算法种类枚举类型，值为用户想要设置的算法类型；findPath()为用户点击自动寻路按钮时调用的函数，系统根据当前用户选择的算法类型执行对应的算法；resetPath()为用户已经进行过一次自动寻路后，再次点击自动寻路后调用的函数，系统将上一次的寻路结果清除，重置迷宫单元格的相关标识信息；setIsFindindPath(bool b)方法用来设置系统当前是否正在进行自动寻路过程，防止寻路过程中用户进行一些与此过程互斥的非法操作；onFindStep(int dir, int x, int y)方法是自动寻路过程中路径规划算法每一步执行时调用的函数，参数dir表示本次寻路的方向，1-4分别表示上下左右，参数x和y分别表示当前寻路单元格所在位置的X和Y坐标；onFindStepBack(int dir, int x, int y)方法是路径规划算法执行回退时调用的函数，参数dir表示本次回退的方向，1-4分别表示上下左右，参数x和y分别表示被回退的单元格所在位置的X和Y坐标；onFindReturn(int x, int y)方法是路径规划算法执行结束倒推最终路径时调用的函数，参数x和y分别表示被倒推的单元格所在位置的X和Y坐标。

### 数据结构

class MyPoint

int h;

int g;

int openOrClose;

int lastDir;

bool isPath;

MyPoint \*lastPoint;

bool operator<(const MyPoint& n) const ;

图 —

MyPoint类中关于自动寻路功能的数据结构如上图所示，变量h用于存储启发式搜索时单元格在估值函数中的h()值，变量g存储g()的值；变量openOrClose用于存储单元格是否在open和close队列中，0表示都不在，1表示在open队列中，2表示在close队列中；变量lastDir用来存储当前单元格的父路径所在的方向，1-4分别表示上下左右；指针lastPoint用来指向当前单元格在最终路径上的父单元格；重载“<”为了方便在启发式搜索中对单元格进行排序。

## 信息栏

### 功能描述

信息栏模块主要负责将迷宫生成过程和自动寻路过程的算法执行信息显示出来，新的过程开始时，信息栏窗口需要清空上一次的执行信息。当输出的信息过多时，信息栏窗口需要自动生成滚动条并优先显示最新生成的信息。

### 接口设计

class InfoWindow

void addMazeStep();

void mazeReturn();

void addInfo();

void clearInfo();

void addFindSearch();

void addFindStep();

String\* info;

图 —

InfoWindow类主要负责显示系统运行过程的算法执行信息，其主要接口和数据结构如图所示：addMazeStep(int level, int dir, int x, int y)方法用于在迷宫生成过程中迷宫生成算法每执行一步结束后将执行信息显示到信息栏窗口时调用，参数level表示递归的深度，参数dir表示本次递归随机的方向，1-4分别表示上下左右，参数x和y分别表示本次递归单元格所在位置的X和Y坐标；mazeReturn(int level, int x, int y)方法用于迷宫生成算法回溯时将执行信息显示到信息栏窗口时调用，参数level表示算法的深度，参数x和y分别表示本次回退单元格所在位置的X和Y坐标；addInfo (String info)方法是系统需要显示某些特殊信息到信息栏时调用的函数，如算法开始、结束等，参数info为需要显示的信息；clearInfo()方法用于清除信息栏时调用，每一次新的算法执行开始前都需要清空信息栏；addFindSearch(int x, int y)方法是在自动寻路过程中显示算法每一步执行结果时调用的函数，参数x和y分别表示当前单元格所在位置的X和Y坐标；addFindStep(int x, int y)方法是当自动寻路算法回退时调用的函数，参数x和y分别表示回退的单元格所在位置的X和Y坐标。

信息栏模块只需要一个String类型的info数组就可以存储当前信息栏中输出的信息，注意清空时要清空对应数组。

## 速度设置

### 功能描述

功能设置模块主要负责用户设置系统单步运行的速度，在迷宫生成和自动寻路过程中，用户实时设置的最新单步速度要在下一帧立刻生效，不能出现延迟。

### 接口设计

class Speed

void setSpeed();

int speed;

图 —

Speed类主要负责用户设置系统的单步运行速度，其主要接口和数据结构如图3.6—1所示：setSpeed(int speed)方法用于设置系统的单步运行速度，参数speed表示想要设置的速度值。变量speed用来存储系统的单步运行速度，初始值为50ms，有效范围为10~500ms。

## 功能模块之间的交互

智能迷宫游戏系统的迷宫生成和自动寻路两大功能需要各个功能模块之间相互配合完成，它们之间有互斥也有同步。

### 迷宫生成

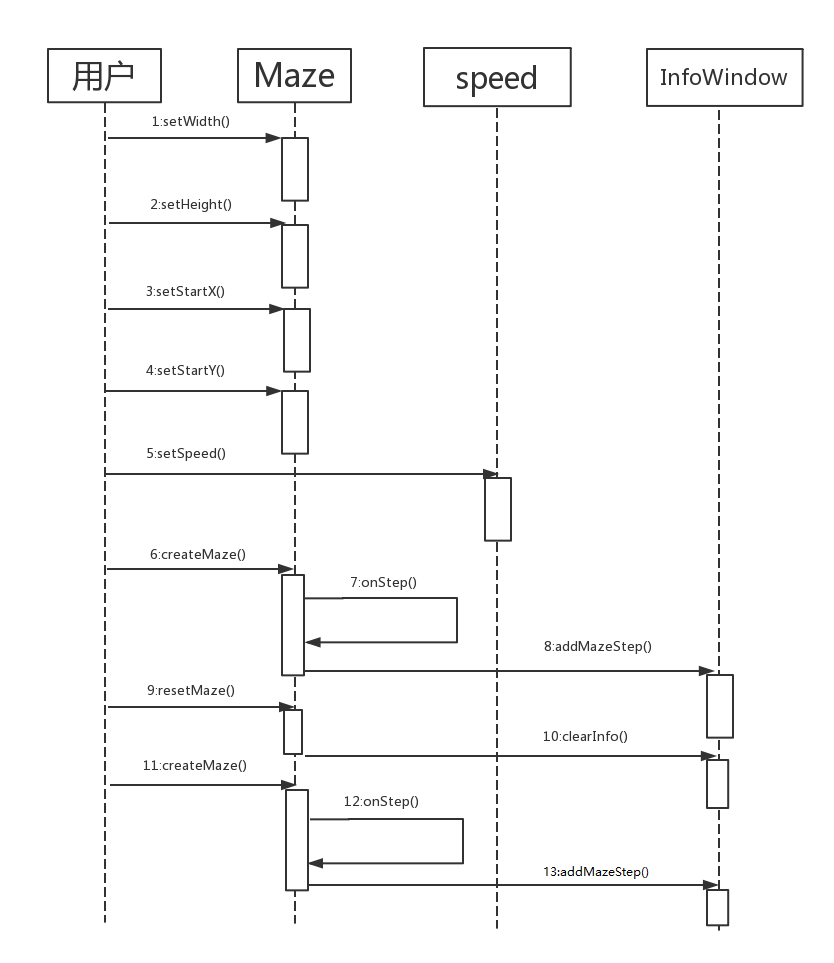


图3.7—1

智能迷宫游戏系统迷宫生成功能的时序图如图3.7—1所示，用户首先通过迷宫标签界面调用Maze类的setWidth()，setHeight()，setStartX()，setStartY()等方法设置想要生成的迷宫的宽度、高度和起始位置等信息，然后通过速度设置界面调用Speed类的setSpeed()方法设置自己想要的速度，接着点击“生成迷宫”按钮调用Maze类的createMaze()方法生成初始迷宫，在createMaze()方法中，生成迷宫算法每执行一步运算就会调用自身的onStep()方法绘制本次打通的单元格，并调用InfoWindow类的addMazeStep()方法将本次算法的执行信息显示到信息栏窗口。

再一次生成迷宫时，用户点击迷宫标签界面的“生成迷宫”按钮，调用Maze类的resetMaze()方法清除上一次生成的迷宫信息，然后调用InfoWindow类的clearInfo()方法清除上一次的算法执行信息，接着和第一次生成迷宫的过程一样，调用Maze类的createMaze()方法进入后续流程，生成一个与上次不同的迷宫。

### 自动寻路

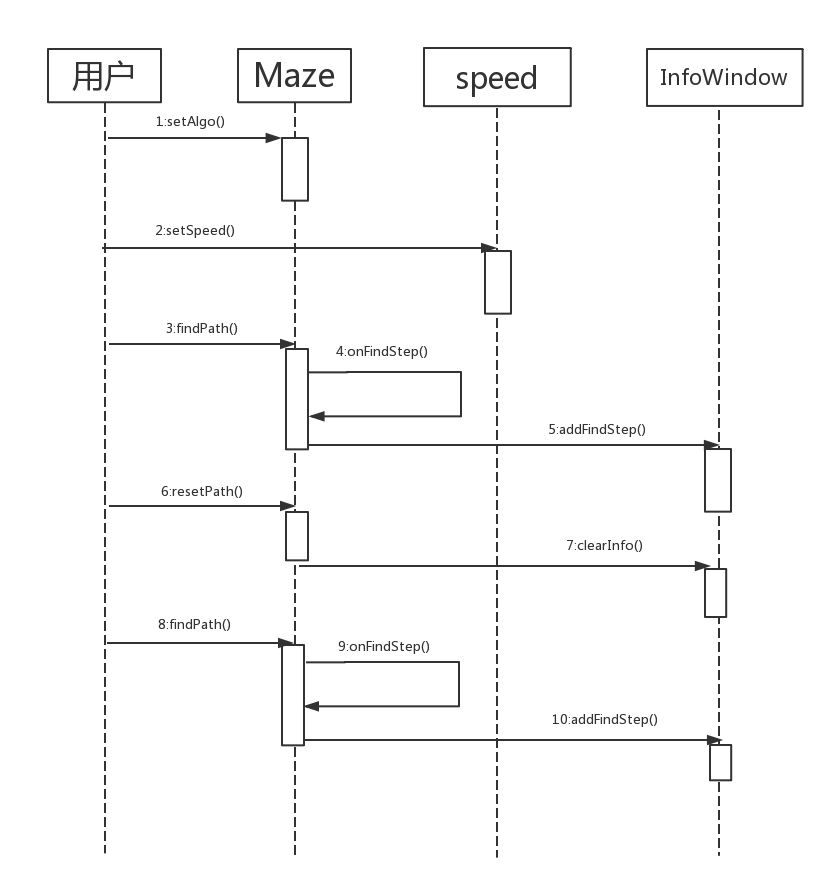


图3.7—2

智能迷宫游戏系统自动寻路功能的时序图如图3.7—2所示，用户需要先生成一个迷宫，然后在算法选择标签界面调用Maze类的setAlgo()方法选择一个路径规划算法，然后通过速度标签界面调用Speed类的setSpeed()方法设置系统的单步运行速度，接着点击“自动寻路”按钮调用Maze类的findPath()方法开始自动寻路，在findPath()方法中，Maze类的路径规划算法每执行一步就会调用自身的onFindStep()方法改变本步算法的执行单元格背景颜色，并调用InfoWindow类的addFindStep()方法将算法的执行信息显示到信息栏窗口中。

用户再次点击“自动寻路”按钮时，先调用Maze类的resetPath()方法清除上一次的寻路结果，并调用InfoWindow类的clearInfo()方法清除上一次输出的算法执行信息，接着调用Maze类的findPath()进行本次的自动寻路，后续流程和第一次自动寻路的流程一样。

## 本章小结

本章主要介绍了智能迷宫游戏系统的系统设计，包括系统目标、设计方案和主要类图及关系，并按照功能模块介绍了功能描述、接口设计、数据结构和时序图等详细设计情况。最后将本系统的最核心两个功能：迷宫生成和自动寻路功能通过时序图进行了详细介绍，图中包括各个模块之间的调用、互斥和同步关系。

# 算法设计

## 迷宫生成算法

总的来说，迷宫生成算法有三大类：深度优先，宽度优先和非图论型。对一般情况而言，迷宫生成算法由上面的三类算法衍生出来的，比如说，经典的Janmis Buck地下城算法就是深度优先的回溯随机版本，而经典的Prim算法就是宽度优先的深度随机版。因此，作者选择了分别对应这三类算法的三个经典算法来进行研究分析。它们是深度优先的递归回溯、随机Prim和递归分割。

为了方便后续的算法描述，先在此介绍总体的前提条件：

1. 迷宫是一个由M\*N个单元格组成的矩阵
2. 两个路单元格之间有墙壁阻挡
3. 墙壁也是占用一个单元格

### 深度优先递归回溯

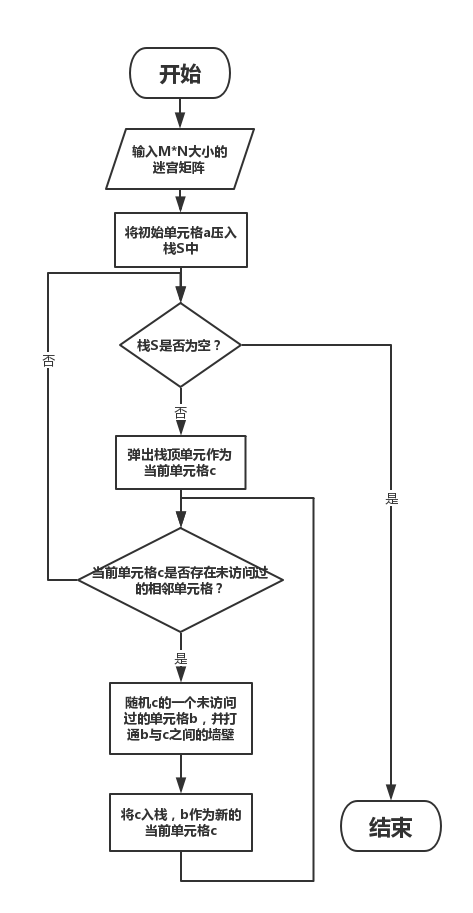
深度优先递归回溯算法是一个最常用、最直接、较高效的迷宫生成算法。它的一般思路是将起始单元格作为当前单元格，从当前单元格的四个相邻单元格随机一个没有访问过的单元格，打通它们之间的墙壁，然后将当前单元格入栈，将随机到的新单元格作为当前单元格继续重复随机步骤，直到当前单元格的所有相邻单元格都已经被访问过，则从栈顶弹出一个单元作为单元格，继续重复随机步骤，直到栈为空为止。它的算法流程图如图4.1—1所示。

图 —

### 随机普利姆

普里姆算法（Prim算法），图论中的一种算法，可在加权连通图里搜索最小生成树。意即由此算法搜索到的边子集所构成的树中，不但包括了连通图里的所有顶点，且其所有边的权值之和亦为最小。但当我们将Prim算法应用于迷宫生成时，情况有些不同，它的解释及实现过程是：

1. 初始一个单元格作为通路，然后把它的墙壁放入列表中
2. 当列表中还有墙壁时：
   1. 从列表里随机选一个墙壁，如果它对面的单元格不是通路：

2.1.1把墙壁打通，把对面的单元格作为迷宫通路

2.1.2把新单元格的墙壁加入列表中

* 1. 如果对面的单元格已经是通路，从列表中移除这个墙壁

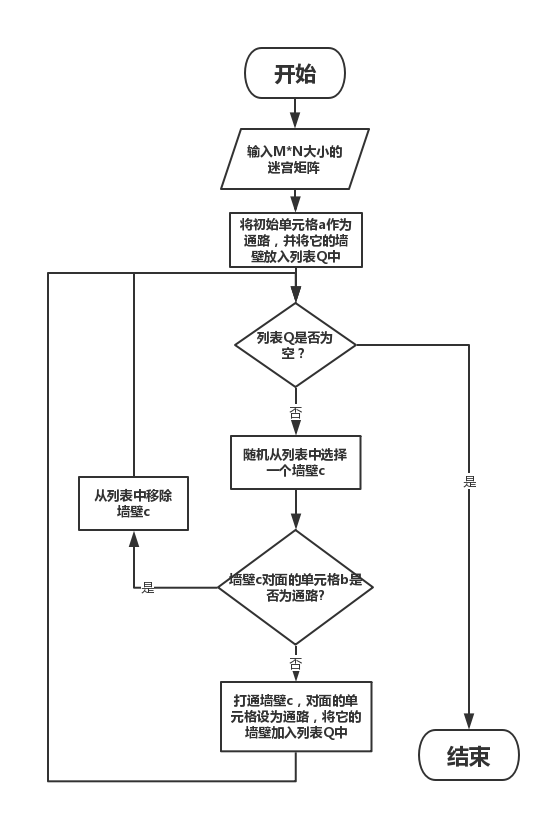
简单研究算法实现过程我们可以发现，Prim算法就是不断地从所有可以是通路的位置中随意选一个挖洞，直到没有可能为通路的位置。整个实现过程还是相当于随意为路线附权值的Prim算法。随机Prim迷宫生成算法的算法流程图如图4.1—2所示。

图 —

### 递归分割

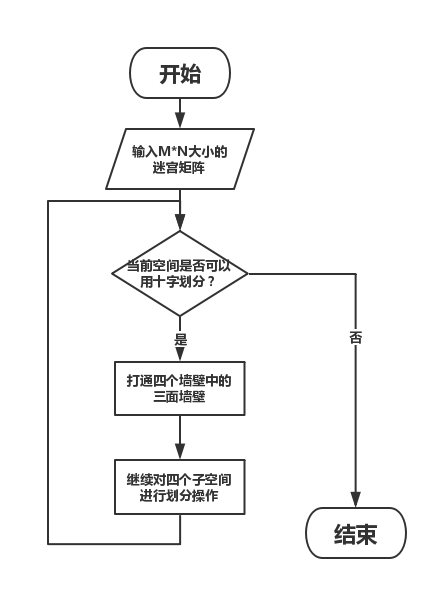
递归分割算法生成的迷宫比较简单，会包含很多的直路，曲路很少，还可以生成很多个“小房间”，此算法效率非常高。其核心思路是，把矩阵空间不断用十字划分，为四个子空间，然后打通其中的三个墙壁（为了确保联通），然后继续对每个子空间重复划分和打通，直到空间不足以分割为止。它的流程图如图4.1—3所示。

图 —

## 路径规划算法

路径规划是指，在具有障碍物的环境中，按照一定的评价标准，寻找一条从起始状态到目标状态的无碰撞路径。通常路径规划算法被分为全局路径规划（环境完全已知）和局部路径规划（环境未知或者部分未知，通过感知实时获取环境信息）。另外根据环境类型又可分为全局静态路径规划、全局动态路径规划、局部静态路径规划和局部动态路径规划。本系统所涉及的算法都是全局静态路径规划算法，它们是宽度优先搜索。深度优先搜索、启发式搜索和遗传算法。

### 宽度优先搜索

1. 数据结构表示

宽度优先搜索算法（BFS）中使用的主要数据结构为一个队列结构，里面保存的是当前带扩展节点，通过队列的先进先出（FIFO）性质来实现宽度优先过程。在代码中通过一个节点数组que和一个队首下标front以及一个队尾下标tail来实现。

同时还有一个二维数组dis用来记录到达每个节点是从入口到这个节点所走过的距离，初始值-1表示当前节点尚未被访问过。还有一个二维数组pre用于记录搜索过程中每个到达节点的前驱节点用于算法中记录路径。

2. 算法描述

在宽度优先搜索算法中，一开始所有节点dis值设为-1，表示所有节点未被访问过，并清空队列。然后将入口节点dis值设为1，并将其压入队列中。

接下来从队首取出一个节点，对于这个节点进行扩展操作，具体是对于此节点的上下左右四个节点进行查看，如果其中有一个是终点，那么算法终止，返回找到路径。对于上下左右四个节点中在地图内可以通过并且没有被访问过的节点，标记这个节点的dis为当前被扩展节点的dis值+1，并记录此节点前驱为当前被扩展节点。同时将此节点压入队列中。如果队列为空，则一直执行上述扩展过程，直至到达终点或队列为空。

如果最终队列为空且中间没有到达过终点，则说明此迷宫无解，返回无解即可。对于有解的情况，通过从终点对于每个点的前驱进行查找就可以找到一条穿越迷宫的路径。

从直观上理解，宽度优先搜索算法和深度优先搜索算法的不同，在于深度优先搜索算法是深度越大的结点越先得到扩展，如果在搜索中把算法改为按结点的层次进行搜索，本层的结点没有搜索处理完时，不能对下层结点进行处理，即深度越小的结点越先得到扩展，也就是说先产生的结点先得以扩展处理，这种搜索算法就是宽度优先搜索法。

宽度优先搜索算法（BFS）是一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果，其找到的是最优解。

宽度优先搜索算法流程图如图4-6所示。

### 深度优先搜索

1. 数据结构表示

深度优先搜索算法（DFS）中使用的主要数据结构为一个栈结构，里面保存的是当前带扩展结点，通过栈的后进先出（LIFO）性质来实现深度优先过程。在代码中通过一个结点数组stk和一个栈顶下标top来实现。

同时还有一个二维数组dis用来记录到达每个节点时从入口到这个节点所走过的距离，初始值-1表示当前节点尚未被访问过。还有一个二维数组pre用来记录搜素过程中每个到达节点的前驱节点用于算法中记录路径。

2. 算法描述

在深度优先搜索算法中，一开始将所有节点dis值设为-1，表示所有节点未被访问过，并将栈清空。然后将入口节点dis值设为1，将其压入栈中。

接下来从栈顶取出一个节点，对于这个节点进行扩展操作，具体是对于此节点的上下左右四个节点进行查看，如果其中有一个是终点，那么算法终止，返回找到路径。对于上下左右四个节点中在地图内可以通过并且没有被访问过的节点，标记这个节点的dis为当前被扩展节点的dis值+1，并记录此节点前驱为当前被扩展节点。同时将此节点压入栈中。如果栈非空，则一直执行上述扩展过程，直至到达终点或者栈为空。

如果最终栈为空且中间没有到达过终点，则说明迷宫无解，返回无解即可。对于有解的情况，通过从终点对于每个点的前驱进行查找就可以找到一条穿越迷宫的路径。

深度优先搜索算法（DFS）是一种盲目的搜索算法，搜索过程中可能很多次的搜索到目标点，其找到的是一个可行解，不一定是最优解。

深度优先搜索算法流程图如图4-4所示。

### 启发式搜索

1. 数据结构表示

A\*算法执行过程中，每次需要从带扩展节点列表中找到一个代价最小的节点进行扩展，可以使用一个链表结构，每次线性查找代价最小的节点，但是这样效率很低。我们可以使用一个优先队列来保存待扩展节点，在优先队列中每次出队的是优先级最高的元素，在本问题中就是代价最小的节点，同时入队和出队的时间复杂度均为O(logn)，从而大幅度提高效率。优先队列的具体实现可以使用二叉堆（heap）、左偏树、斐波那契堆等等实现，在这里由于不需要合并，所以使用普通的二叉堆（heap）即可，作者使用了C++ STL库中提供的priority\_queue模板实现，其内部就是二叉堆（heap）。

同时还有一个二维数组g用来记录到达每个节点是从入口到这个节点所走过的距离，-1表示当前节点尚未被访问过。还有一个二维数组pre用于记录搜索过程中每个到达节点的前驱节点用于记录路径。

2. 算法描述

A\*算法执行过程中需要一个对每个节点的估价函数f(x)=g(x)+h(x)，其中g(x)表示从出发点到达这个节点实际走过的距离，h(x)为当前节点到达终止节点的估计值，其中这个估计值必须保证小于真实值，作者使用了当前节点与终点的曼哈顿距离作为h(x)的估计值。在优先队列中对于节点的比较标准即为对于节点x而言，f(x)越小表示节点x优先级越高，越先出队，每次扩展时都是f最小的点出队。

接下来的过程与广搜类似，一开始所有节点dis值（即g(x)值）设为-1，表示所有节点未被访问过，并将队列清空。然后将入口节点dis值设为1，并将源点压入优先队列中。

接着从优先队列顶部取出一个节点，对于这个节点进行扩展操作，具体是对于此节点的上下左右四个节点进行查看，如果其中有一个是终点，那么算法终止，返回找到路径。对于上下左右四个节点中在地图内可以通过并且没有被访问过的节点，标记这个节点的dis值为当前扩展节点的dis值+1，并记录此节点前驱为当前扩展节点。同时将此节点压入优先队列中。如果优先队列非空，则一直执行上述扩展过程，直至到达终点或者优先队列为空为止。

如果最终优先队列为空且中间没有到达过终点，则说明迷宫无解，返回无解即可。对于有解的情况，通过从终点对于每个点的前驱进行查找就可以找到一个穿越迷宫的路径。

A\*算法找到的是最优解。A\*算法流程图如图4-9所示。

### 遗传算法

1. 数据结构表示

遗传算法中需要的数据结构主要有以下几个，两个字符串数组一个用于保存当前种群的基因序列，另一个用于保存当前种群杂交产生的子代的基因序列；一个前缀数组用于保存当前种群的每个个体的基因估价值的前缀和，用于在轮转法中使得估价较高的亲本被选中的概率较高，被淘汰概率较低。除此之外还有一个vis数组用作访问标记，标记当前节点是否被访问过。

2. 算法描述

遗传算法的过程主要是模拟自然界中一个种群生物的遗传变异过程，通过若干的遗传和变异过程，一直到选出一个问题的解为止。首先对于迷宫问题进行编码，使得解空间中的每个元素都对应一个基因序列，相当于种群中的一个个体。在这里为了保证完备性，我们定义基因长度为迷宫的长度\*宽度，序列中每个元素取值为0,1,2,3，表示相比上一个节点，下一个节点的方向，如果当前中每一个节点走到了墙或者过去走到过的节点，那么久直接跳过当前基因元素，看下一个，这样每一个基因序列就对应了一个走法，我们要找的是一个能够到达终点的走法序列。

具体算法流程是我们首先随机生成若干序列，也就是初始种群，然后开始模拟自然界中优胜劣汰的过程，对种群进行淘汰，对于每个基因序列定义一个估价函数，在本问题中可以使用路径上距离终点最近点的曼哈顿距离作为估价函数，距离越小估价值越高。如果发现当前种群中某个个体基因序列中存在到达终点的点，那么这个个体就是一个解，算法终止即可。否则则进行下面的算法流程。

这样通过记录每个基因序列估价的前缀和，然后生成一个随机数r，通过二分查找找到一个最小的前缀和满足sum[x]>r，则认为x个体被选中，没有被淘汰，这样可以保证估价越高的个体，被选中的概率越高，同时由于二分查找的使用，每次选取的时间复杂度为O(logn)，有较高的时间效率。我们通过上述方法选出一定数量的个体之后，将种群中其他个体淘汰，模拟自然界优胜劣汰的过程。

接下来在剩余种群中模拟杂交过程，即我们随机选取种群中两个个体作为亲本，然后对这两个亲本随机选取的一位后面的全部序列进行交换，这样就等于两个亲本杂交产生了两个子代个体，使得种群的基因序列发生了变化，更有利于产生问题的解。

但是如果只有杂交过程，种群内不会产生新的基因序列，就像自然界不会产生新的性状一样，这样就可能得不到问题的解，所以我们还有模拟变异过程，即对于当前种群中的每一个个体按一定变异率发生变异，变异过程就是在这个个体的基因序列中随机选取一位进行改变，这样就会产生一个变异。

上面的淘汰、杂交、变异三个过程进行一次，就等于种群繁衍了一代，对整个种群反复模拟上述过程，直达其中某个个体能够成为问题的解为止，就可以得到问题的解。当然与蒙特卡洛算法类似，可以设置一个时间上限，若达到这个时间上限还没有得出解，则可以认为问题无解。

遗传算法求得的问题的解只是一个可行解，非最优解，但可以通过改进估价函数，同时在一定时间内进行反复遗传变异过程，并记录当前最优解的方法，最终在有限时间内获得一个比较优秀的解。遗传算法流程图如图4-11所示。

## 本章小结

本章对三类经典迷宫生成算法——深度优先递归回溯、随机普利姆和递归分割算法进行了详细描述，并给出算法流程图。详细阐述了本系统采用的四种路径规划算法，给出了算法的数据结构、实现描述和流程图。

# 系统实现

## 实现环境

本文使用的开发环境为：

系统：Window10专业版 64位

处理器：Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz

内存：8.00 GB

开发语言：C++

开发工具：Qt Creator 4.2.1

## 数据分析与比较

### 执行数据

### 分析与比较

## 算法总结

### 迷宫生成算法

对于

### 自动寻路算法

## 系统展示

# 总结与展望

# 致谢

# 参考文献