Nico School Maze 程序报告

组员：赵智宇(12110814)，孙全超(12110811)

1. 程序使用说明

我们提供了 java 工程文件包和编译后的.class 文件包。前者可以放入 IDEA 等开发环境中打开，后者可以在命令行中进行执行。

1.1 编译（已完成）

将工程文件包“NicoSchoolMaze”目录放于桌面，然后在命令行中依次执行

① cd C:\Users\<用户名>\Desktop\NicoSchoolMaze\src

② javac -cp .;algs4.jar -d . *.java

然后在从工程文件包中将“src”目录拷贝至桌面，将其中软件包内的所有源代码文件(.java)删除，只留下编译文件(.class)，然后将目录重命名为“NicoClass”即可。

我们已提供“NicoClass”编译文件包，故测试中可以省去此步骤。

1.2 运行程序

将编译文件包“NicoClass”放于桌面，然后在命令行中依次执行

① cd C:\Users\<用户名>\Desktop\NicoClass

② java -cp .;algs4.jar Main <*Parameter*>

其中<*Parameter*>是程序运行方式参数，可选“gui”和“terminal”分别令程序在 GUI 界面运行和在终端运行（默认：“gui”）。即有以下两种选择：

java -cp .;algs4.jar Main gui

java -cp .;algs4.jar Main terminal

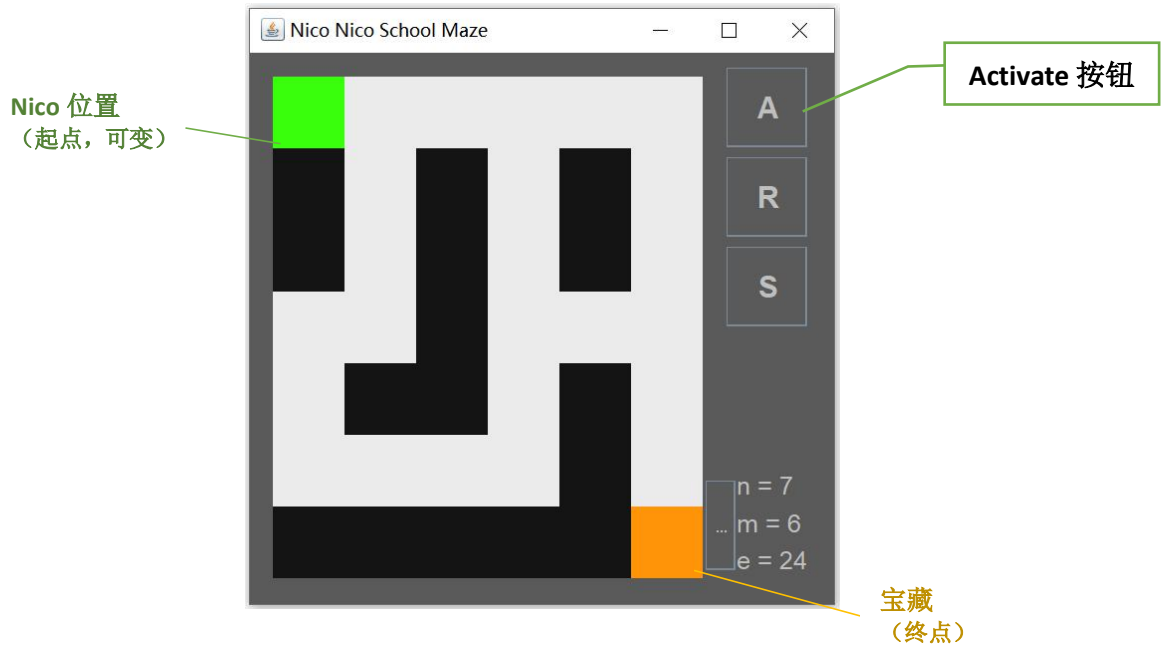
然后程序会等待输入数据以进行测试。

1.2.1 Terminal 模式运行

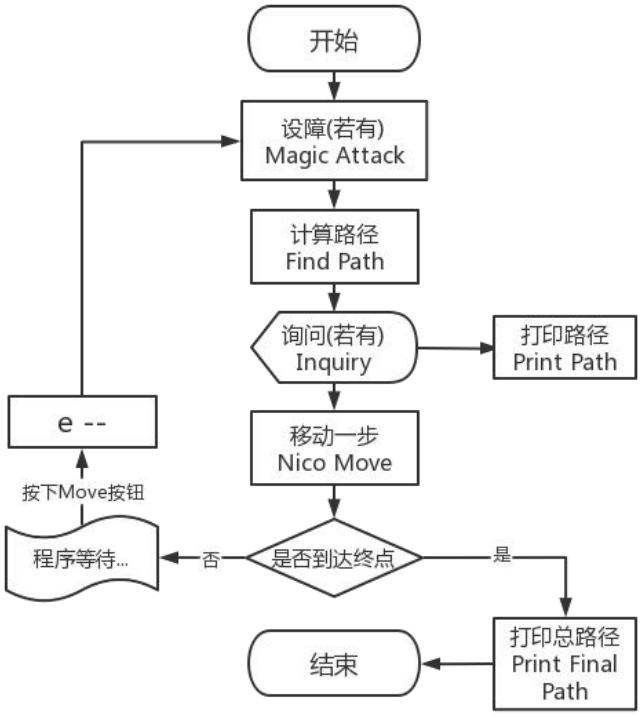
在命令行输入数据后，程序会按照输出要求直接在终端打印输出结果。

1.2.2 GUI 模式运行

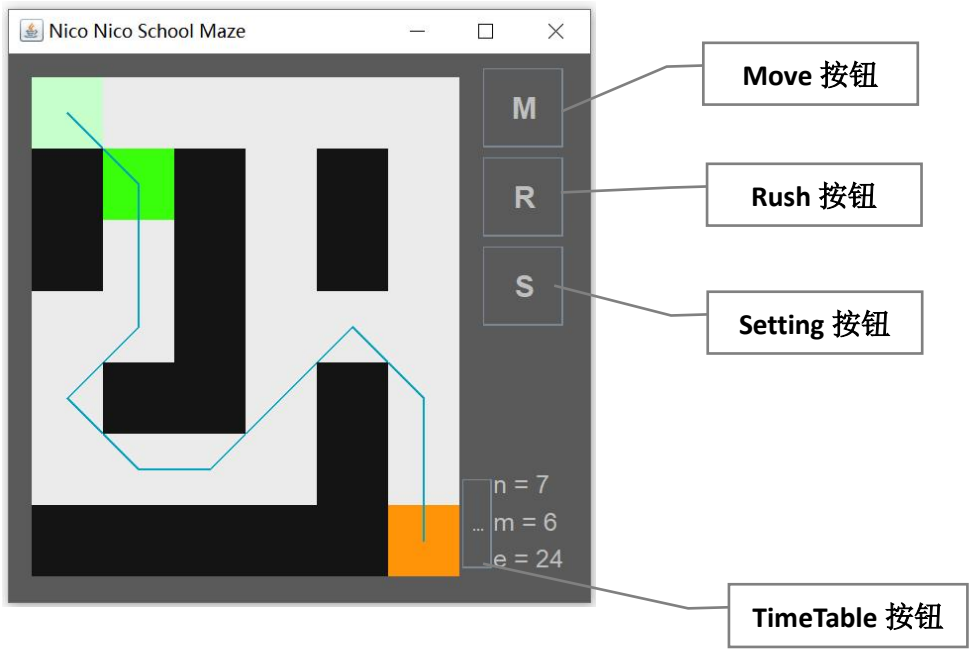
在命令行输入数据后，程序会打开一个窗口，如下图所示。



点击 **Activate 按钮**后，程序将按照以下流程图运行。

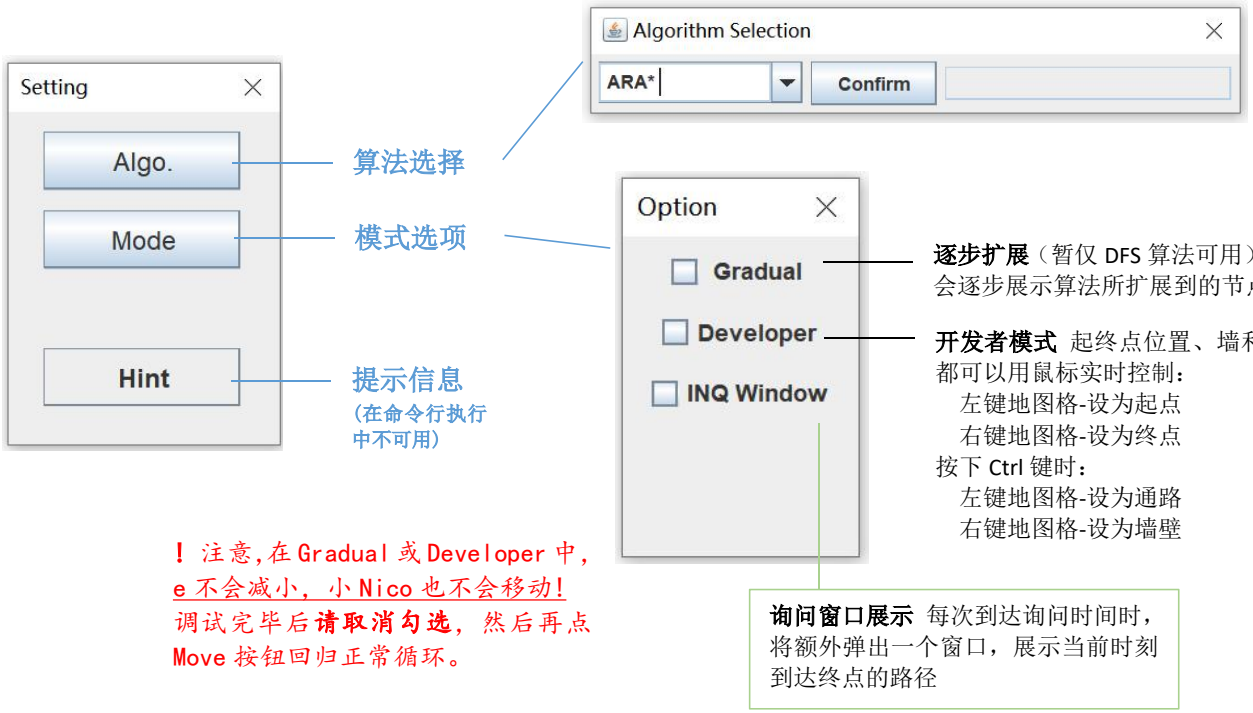


即程序会按照**初始 e 参数**来执行一次设障—计算—询问—移动循环，然后挂起等待，同时 **Activate 按钮**转变成 **Move 按钮**，如下图。



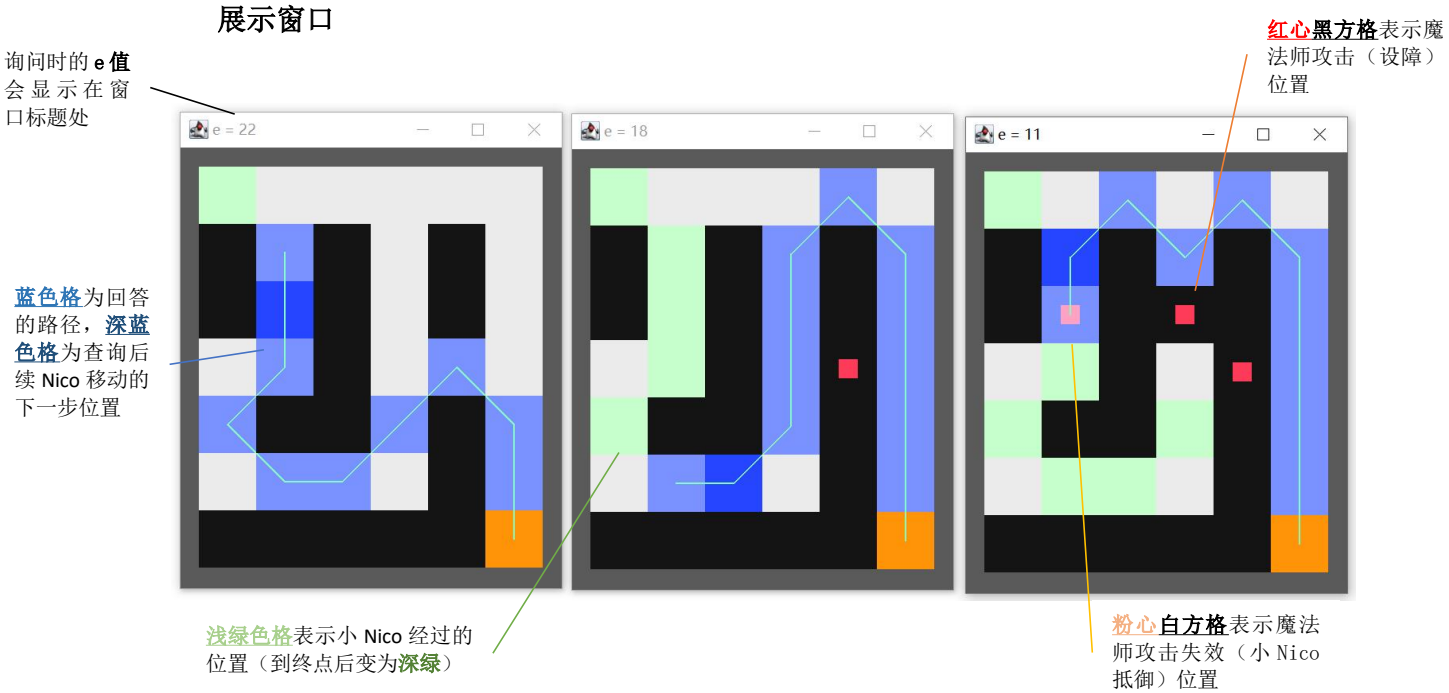
Move 按钮：令程序执行下一轮循环（见流程图）。
Rush 按钮：直接跳跃到下一个询问时间。
Setting 按钮：打开设置界面。
TimeTable 按钮：依次打开设障时间表和询问时间表界面，以供核查。

设置界面



程序 GUI 会实时给出当前位置计算出的到达终点的路径。而到达询问时间时，给出的路径会特别标色以作区分。也可以在模式选项里打开“INQ Window”来弹出额外的展示窗口。

展示窗口



1.3 可解迷宫数据随机生成器*

我们还提供了一个可以随机生成所要求输入格式的迷宫初始数据的生成器，即“MazeDataGenerator”工程文件包，在开发环境中运行“Generator”主类即可随机生成迷宫数据（1≤n＜100，1≤m＜100），并打印在控制台。同时数据也会保存在在工程目录下的"./resources/data/Test.txt"文件中供用。

2. 寻路算法分析

我们在实现GUI和Terminal两种程序模式的过程中所采用的算法改良方式不完全相同。下面将详细介绍。

2.1 Terminal 模式中的寻路算法

设输入地图大小 m*n，初始膨胀权值为 e，施加魔法 M 次，咨询 C 次

ARAStarMethod.java 作为实现 ARA 方法文件，在用终端实现时，在参数传入之后，调用 ARA()方法，实现并返回相关的数据。

2.1.1 ARA()方法的具体实现

建立链表 RealWay 记录 Nico 实际走过的 Point，先使用 AStar()方法解出一条路径（AStar()方法将在下面详细介绍），将路径保存给链表 result。在循环体 While 中进行优化迭代路径（时间复杂度 O(e)），在 While 中每循环一次，会使 Nico 向前走一步，详细步骤：

（1）将当前位置 start 加入到 RealWay 中。

（2）添加障碍，扫描 Magic 施加的 e（复杂度 O(M)），查看当前是否应当施加魔法，如不需要，则进入下一步。如需要施加魔法，判断魔法位置是否为当前 Nico 的位置，如是则跳入下一步。如不是，则将魔法位置设为不可到达，并判断魔法位置是否位于链表 close，open，inCon 里（时间复杂度 O(m*n)）,如果位于链表中则清空三个链表以及上一次的路径清空，AStar()重新规划出一条新的路径存入 result 里。

（3）判断是否咨询路径（时间复杂度 O(C)）,如果是，则将 result 和上一次规划的路径相比较，并将比较短的路径保存为 lastAnswer 并打印结果。

（4）设置新起点，将上次的起点的父节点设为下一步的起点，并将 cost 改为 1，下一步的父节点设为 null，cost 改为 0。下一步节点设为 start，完成 Nico 的移动并将 e-1。同时将 inCon 里的节点全部添加到 open 里。

（5）随着 Nico 抵达终点 While 循环主体结束，将最后一次 start 加入到 RealWay 中，打印 Nico 实际走过的路径。

2.1.2 AStar()方法的具体实现

（1）将 start 节点加入到链表 open 中，进入 While 循环主体，找出链表 open 里的 f 值（e*切比雪夫距离[1]+该节点的 cost 值）最小的节点存为 test（复杂度 O(m*n)），并将其加入到链表 close 中，对 test 周围八个依次取出判断。

（2）对每一个 test 邻居节点，如果该坐标不可到达或不在地图范围里或已经加入到了 close 中则跳过，判断下一个邻居节点。

（3）如果邻居节点为 target 终点，则将终点和它的父节点，父父节点……有序加入到 result 中，并返回 result 跳出 AStar()方法。

（4）如果该邻居节点不在 open 里面则将其加入到 open 里，并将 test 设为该邻居节点的父节点。如果已经在 open 里，则判断该邻居节点走其当前父节点的 cost 和走 test 的 cost，如果是后者，将其父节点设为 test，并将其加入到 inCon 里（如果 inCon 里已经有了，则不重复添加）

（5）持续循环，直到返回出一条可走的路径。AStar()方法结束。

---

[1] 切比雪夫距离（Chebyshev distance）是指两个点各坐标数值差绝对值的最大值。如在二维坐标系中，d = Max{|x1-x2|, |y1-y2|}. 一个点周围 8 格到它的切比雪夫距离都是 1.

综合以上，**时间复杂度**为 O(e*m*n*M+e*C)。

2.2 GUI 模式中的寻路算法

在 GUI 实现过程中，棋盘组件（迷宫格）BlockComponent 类继承了 javax.swing 中的 JComponent 类，存储在二维数组 Maze = BlockComponent[n][m]中，投射到窗口界面上实现可视化。由于每次点击按钮后才会刷新棋盘状态，为了更好的实现 ARA*算法，将使用 ARAStar.java 配合 GUI 实现相关功能。根据流程图，在每次点击之后，会先进行当前步骤的魔法攻击（设障），然后再进入 ARA*算法来计算路径。路径得出后返回给 GUI 循环，然后进行后续的询问、移动等步骤。

2.2.1 寻路算法 ARAStar. java 类的具体实现

使用类 FindPath.java 进行指挥（作为连接 GUI 和 ARA*的桥梁）。在其中建立二维整数数组 maze = int[n][m]来转存 Maze 的迷宫格状态（State = 0:通路；1:墙壁；2:起点；3:终点）。然后将 ARAStar 类实例化为 araStar 对象，将保存的 open，close，maze 等数据在实例化时传入 araStar 并运行。

（1） 初始化。将传入的 open 和 close 存于 araStar.open 和 araStar.close 中（不为 null）。将传入的 maze 存入 araStar.map 中。

（2） 调用 AStar 算法规划出一条路径存入 result 中，AStar()的实现与上文类似。

3. 样例数据的测试结果

3.1 样例 1

Input：

```
7 6 24
0 0 0 0 0 0
1 0 1 0 1 0
1 0 1 0 1 0
0 0 1 0 0 0
0 1 1 0 1 0
0 0 0 0 1 0
1 1 1 1 1 0
3
19 3 4
17 2 3
12 2 1
3
22
18
11
```

Output：

```
10
2 1 3 1 4 0 5 1 5 2 4 3 3 4 4 5 5 5 6 5
12
5 2 4 3 3 3 2 3 1 3 0 4 1 5 2 5 3 5 4 5 5 5 6 5
10
1 1 0 2 1 3 0 4 1 5 2 5 3 5 4 5 5 5 6 5
23
0 0 1 1 2 1 3 1 4 0 5 1 5 2 4 3 5 2 5 1 4 0 3 1 2 1 1 1 0 2 1 3 0 4 1 5 2 5 3 5 4 5 5 5 6 5
```

3.2 样例 2

Input：

```
9 9 12
0 1 0 0 0 0 0 0 0
0 0 1 1 0 1 0 1 1
0 0 0 0 0 1 0 0 0
1 1 1 1 0 1 1 1 0
0 0 0 1 0 0 0 1 0
0 1 0 1 1 1 0 1 0
0 1 0 0 0 1 0 1 0
0 1 0 1 1 1 0 1 0
0 1 0 0 0 0 0 0 0
5
10 2 3
8 6 0
4 8 2
2 7 2
1 5 2
3
11
6
2
```

Output：

```
10
1 1 2 2 2 3 3 4 4 5 5 6 6 6 7 6 8 7 8 8
9
1 4 2 4 3 4 4 5 5 6 6 6 7 6 8 7 8 8
5
5 6 6 6 7 6 8 7 8 8
15
0 0 1 1 2 2 1 1 0 2 0 3 1 4 2 4 3 4 4 5 5 6 6 6 7 6 8 7 8 8
```