Stage3 - MazeBug Report

19335286 郑有为

实验要求

- 定义一个继承 Bug 类的 MazeBug 类，使虫子的行走方向只有东南西北四个方向，且在碰到迷宫出口（红石头）时，虫子会自动停下来，并显示探索花费的总步数。
- 修改 MazeBug 的方法，使探索为深度搜索过程。
- 增加方向的概率估计，在行走正确路径时，对四个方向的选择次数进行统计，从而控制随机选择时选择某个方向的概率。

运行说明

本过程涉及girdworld.jar的打包，使用命令行操作。

- **第一步：打包 jar**
  - 用 MazeBug 文件夹下的 WorldFrame.java 和" WorldFrameResources.properties 替换 GridWorldCode/framework/info/gridworld/gui 下面的对应的文件
  - 将我们编写好的 MazeBug.java 加入到 framework/info/gridworld/maze/ 文件夹中
    - 注意 gridworld 文件夹本来是没有 maze 文件夹的，需要自己创建。
    - 同时确保 MazeBug.java（**编写完成的代码**） 文件包含 package 信息，即文件第一行：

      ```
      1 package info.gridworld.maze;
      ```

    - 这时候我们没有必要把 Runner 文件放进 jar 里
  - 在 GridWorldCode 文件夹下打开命令行，输入 `ant clean` 清空之前的生成结果，在使用 `ant` 生成 gridworld.jar 文件，新生成的 gridworld.jar 文件保存在 GridWorldCode/dist/GridWorldCode/ 文件夹里， 我们将他替换掉 GridWorldCode/ 下的 gridworld.jar（编译运行时会使用该 jar 文件）
- **第二步：安放Runner文件**
  - 将 MazeBugRunner 放在 projects/maze 文件夹下，在这里，我们删去了它的 package 信息，即删掉：

    ```
    1 package info.gridworld.maze;
    ```

    因为不将它打包进 jar 里，路径也不对，没必要。

- **第三步：编译运行**
  - 在 GridWorldCode 文件夹下打开命令行，执行：

```
javac -classpath .:gridworld.jar ./projects/maze/MazeBugRunner.java
java -classpath .:gridworld.jar:./projects/maze MazeBugRunner
```

- **第四步：修改代码**
  - 若修改了 MazeBug.java 需要重新完成第一步、第三步骤。

算法说明

数据结构定义

数据定义在原代码下做了部分修改，以适应后续的算法。以下是数据定义以及注解：

```
// next: 下一个移动到的位置，在 move 方法中，Bug 会移动到 next 位置
public Location next;

// isEnd: 判断是否到达了终点，每次 canMove 会检查四周是否有出口（红石头）
//        如果找到则将 isEnd 置为 true
public boolean isEnd = false;

/** crossLoaction: DFS核心栈结构，每一个栈单元是一个 Location 的链表
 *                 即 ArrayList<Location>
 *
 * 由于 DFS 需要标记或记录来实现路径回溯
 * 我们在探索一个新的位置时，会打包一个"位置信息"入栈
 * - 在前进时，我们查看栈顶的"位置信息"选择下一个移动位置
 * - 若没有可移动的位置，回溯，我们弹出栈顶的"位置信息"，并返回上一个位置
 *
 * 一个 "位置信息" ArrayList<Location> 的内容定义如下：
 * - ArrayList<Location>[0] : 保存的是该位置的上一个位置
 *                 例如当前位置是(1,0)，从(0,0)移动而来
 *                 则当前栈顶的ArrayList<Location>[0] 是 (0,0)
 * - ArrayList<Location>[i] : i > 0 保存的是该位置下一步的还可以走的位置
 *                 每当我们从栈顶选择一个向前走，就从栈顶删除该Location
 *                 以此来实现一个标记的过程。
 **/
public Stack<ArrayList<Location>> crossLocation = new
Stack<ArrayList<Location>>();

// stepCount: 计算总探索步数（包括回溯带来的花销）
// pathLength：计算从起点到终点的最短路径长度（不含错误探索和回溯）
public Integer stepCount = 0;
public Integer pathLength = 0;

// 以下是为启发式搜索提供的数据结构

// FourMoves: 0 - NORTH; 1 - EAST; 2 - SOUTH; 3 - WEST;
// 统计正确路径上每个方向的步数
private Integer[] fourMoves = new Integer[4];

// State: 0 - GO AHEAD, 1 - GO BACK
// 当前状态是 往前探索新路径 / 进行回溯
public static int GO_AHEAD = 0;
```

```
public static int GO_BACK = 1;
private Integer state = GO_AHEAD;

private Random r = new Random();

boolean hasShown = false;
```

DFS部分

- 算法思路：
    1. 将 MazeBug 起点打包入栈
    2. 查看栈顶元素，是否到达终点（`isEnd`）
        1. 若到达终点：结束并输出步数
    3. 判断是否可以移动（`canMove`）
        1. 若可以移动：**前移**
            1. 更新状态（`state = GO_AHEAD`）
            2. 从栈顶选择一个前移位置（`selectNext(1)`）
            3. 从栈顶删除迁移的位置（`headRecord.remove(next)`）
            4. 移动（`move`）
            5. 更新步数（`stepCount++, pathLength++`）
            6. 打包移动后节点的"位置信息"，入栈
        2. 若不可移动：**回溯**
            1. 更新状态（`state = GO_BACK`）
            2. 弹出栈顶元素，并取出上一步的位置
            3. 移动返回上一个位置（`move`）
            4. 更新步数（`stepCount++, pathLength--`）
- 打包"位置信息"：

```
// now 是上一个位置
ArrayList<Location> newRecord = new ArrayList<Location>();
newRecord.add(now);
for(Location loc : getValid(getLocation())){
    newRecord.add(loc);
}
crossLocation.push(newRecord);
```

启式策略部分

- 启发式搜索主要实现于 `selectNext` 方法，即根据统计结果来从可行位置中随机选择下一步的位置。
- 每一方向的步数记录在数组 fourMoves 中，并在move方法中更新：

```
if(state == GO_AHEAD){
    fourMoves[getDirection() / 90]++;
}
else{
    fourMoves[getDirection() / 90]--;
    if(fourMoves[getDirection() / 90] <= 0){
        fourMoves[getDirection() / 90] = 0;
    }
}
```

- 选择每一个可行方向的概率：该方向的步数 / 可行方向的总步数。
  - 举例：设东西南北四个方向的步数统计分别是e, w, s, n，若当前可选路径由东、西、南，则它们被选择的概率分别是$\frac{e}{e+w+s}$，$\frac{w}{e+w+s}$，$\frac{s}{e+w+s}$。
- 算法伪码：

```
let record = 栈顶链表，ArrayList<Location>

let times[] = {
    if i 对应的 Location in record, times[i] = fourMoves[i]
    else time[i] = 0
}，长度为4的整形数组

let positions[] = 每个方向对应的位置在record中的索引值，长度为4的整形数组

let sum = 可行位置的已走总步数，即对 times 数组求和的结果

let ran = 随机数 in [ 0, sum )

if ran in [ 0, times[0] )
    next = record.get(positions[0])
else if ran in [ times[0], times[0] + times[1] )
    next = record.get(positions[1])
else if ran in [ times[0] + times[1], times[0] + times[1] + times[2] )
    next = record.get(positions[2])
else if ran in [ times[0] + times[1] + times[2], sum )
    next = record.get(positions[3])

```

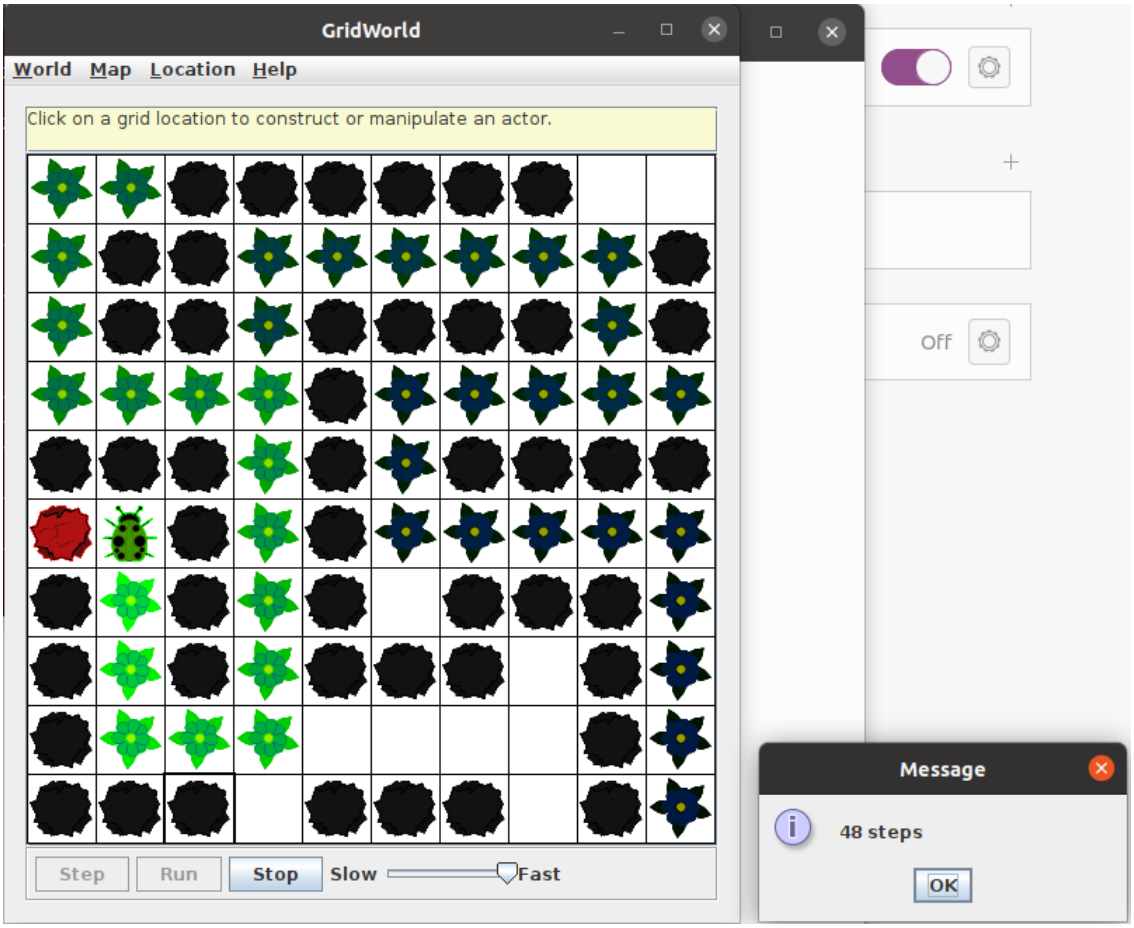
运行效果

在MazeBug文件夹中有若干个测试文件，可以通过GridWorld界左上角的Map打开，打开后直接生成迷宫。

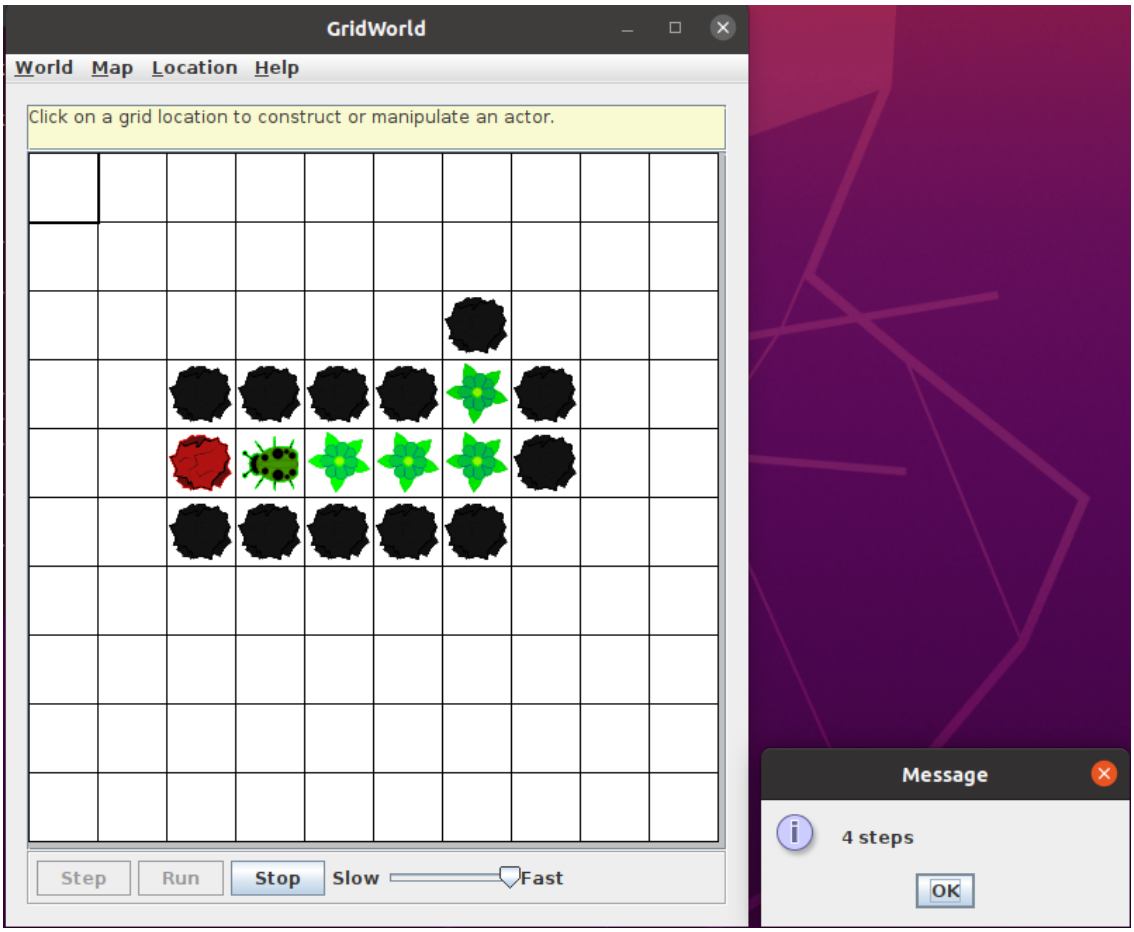
由于MazeBug每次选取路径具有一定的随机性，每次到达重点的步数是不同的。迷宫越大，因回溯而增加的步数也就越明显，以下是所有测试文件的一次测试。结果，不保证每次都能得到这样的解。（例如对于普通测试，由于随机性，测试结果可能小于500步，也可能大于1000步）

以下是测试截图：

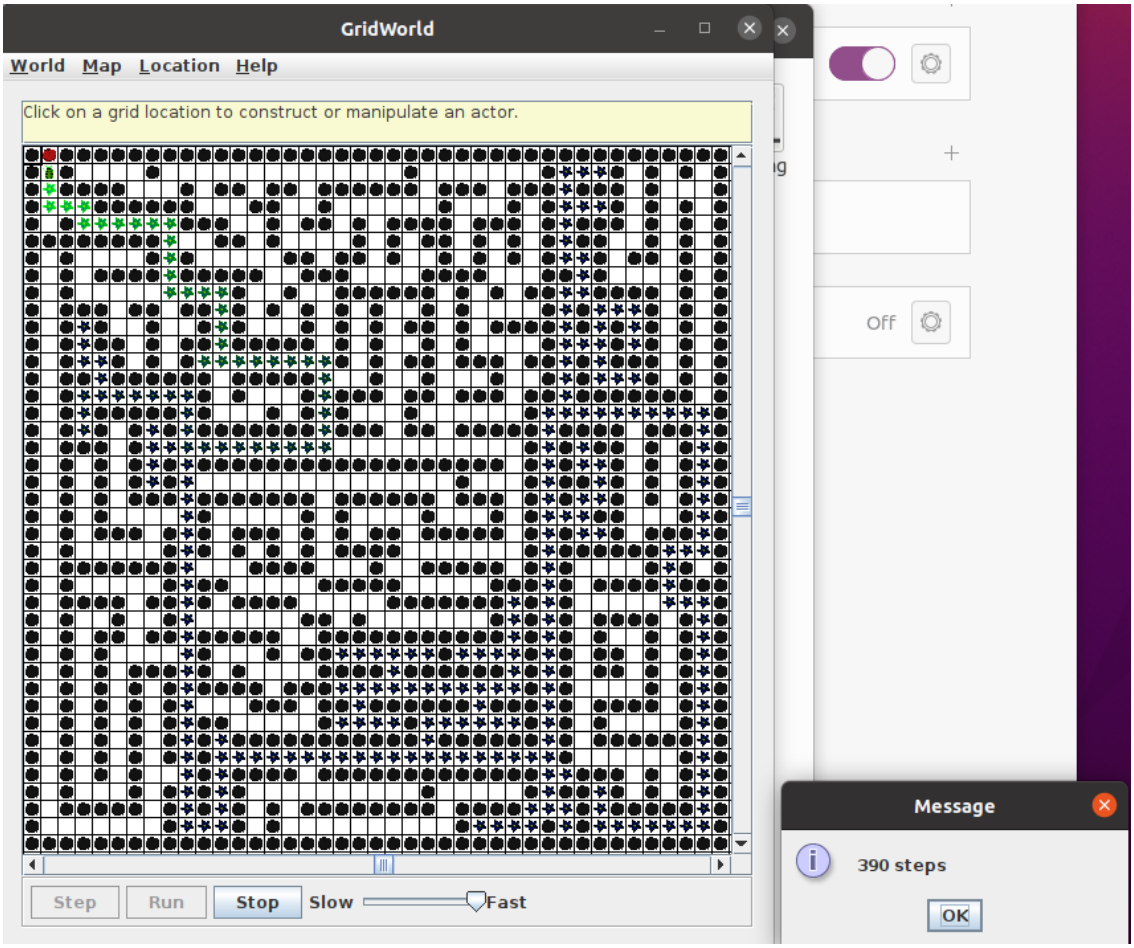
- **简单测试：总步数（含回溯）48步。**



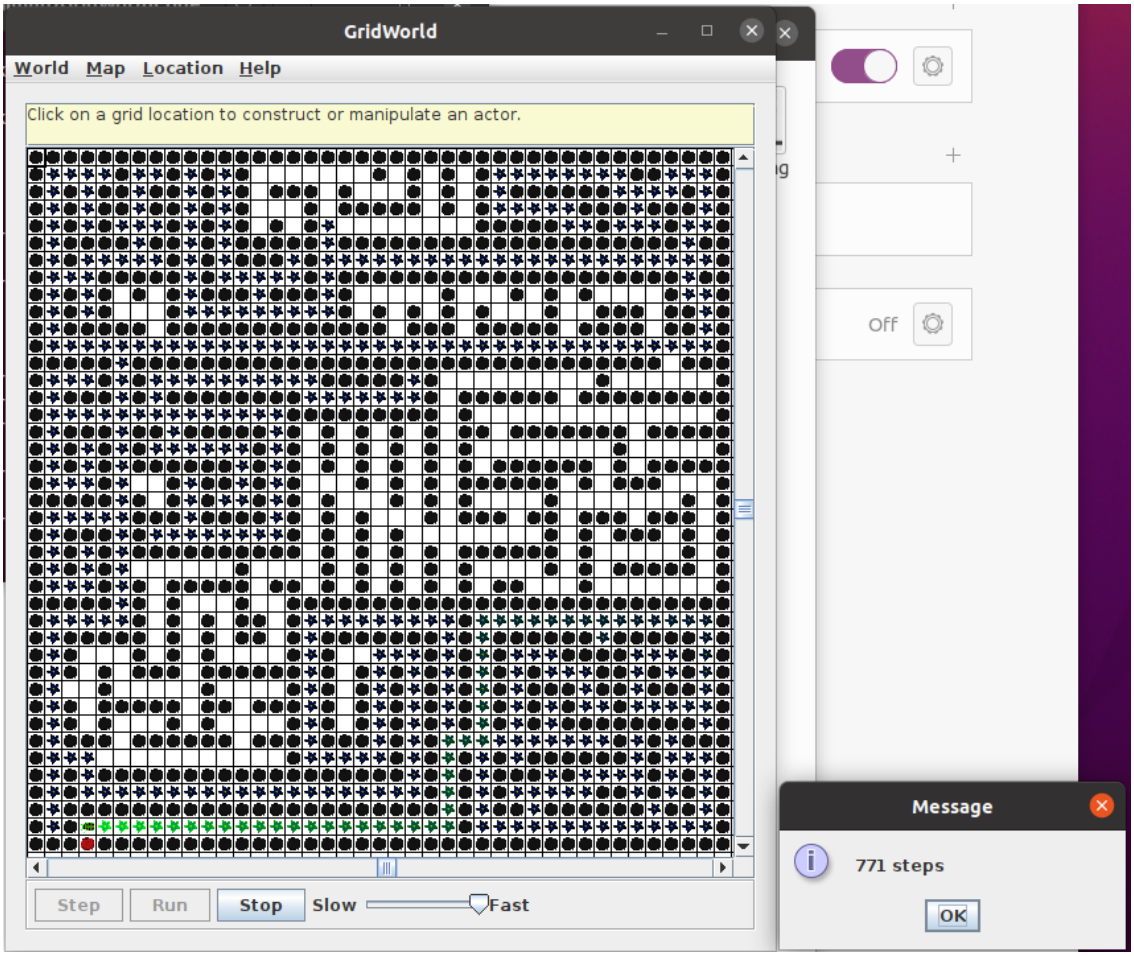
- **直线测试：**



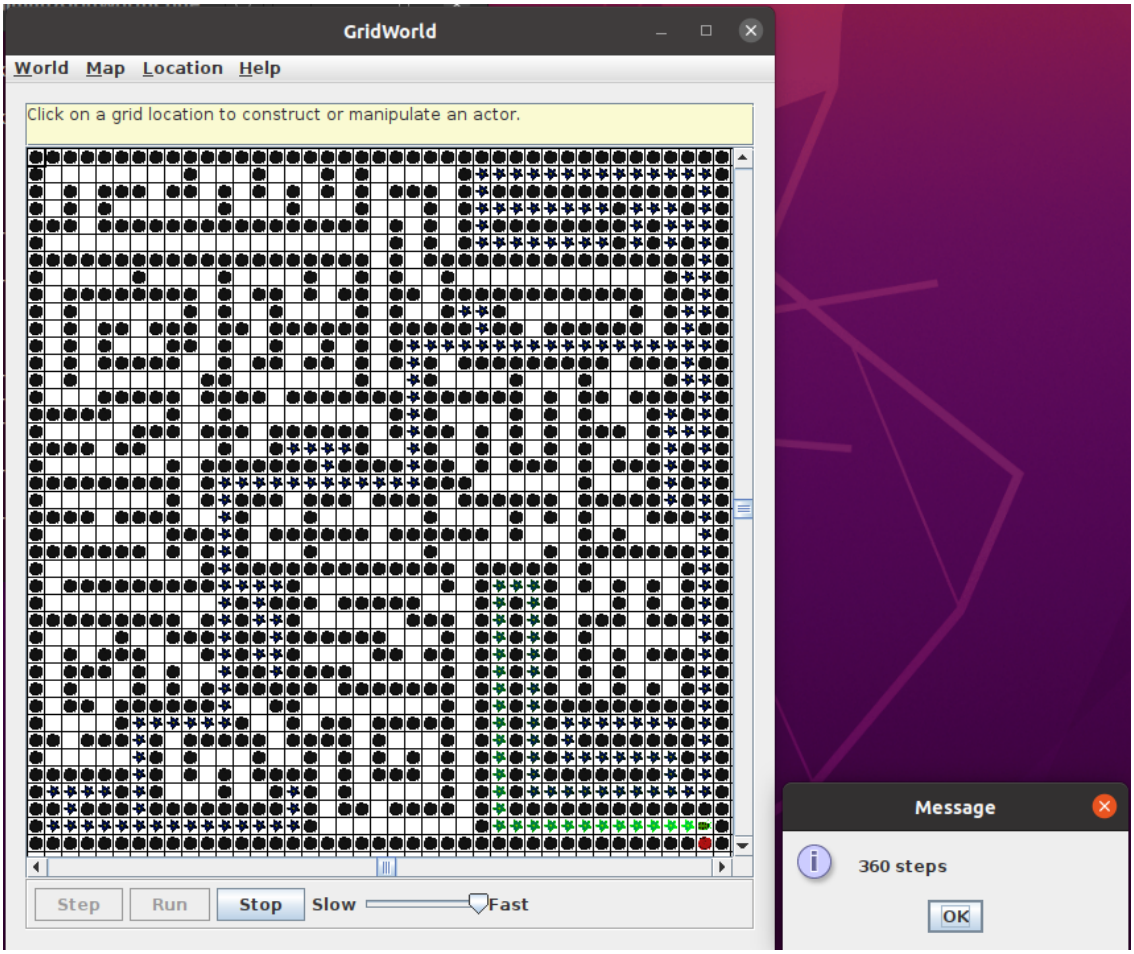
- **普通迷宫测试 - 1：总步数（含回溯）390步。**



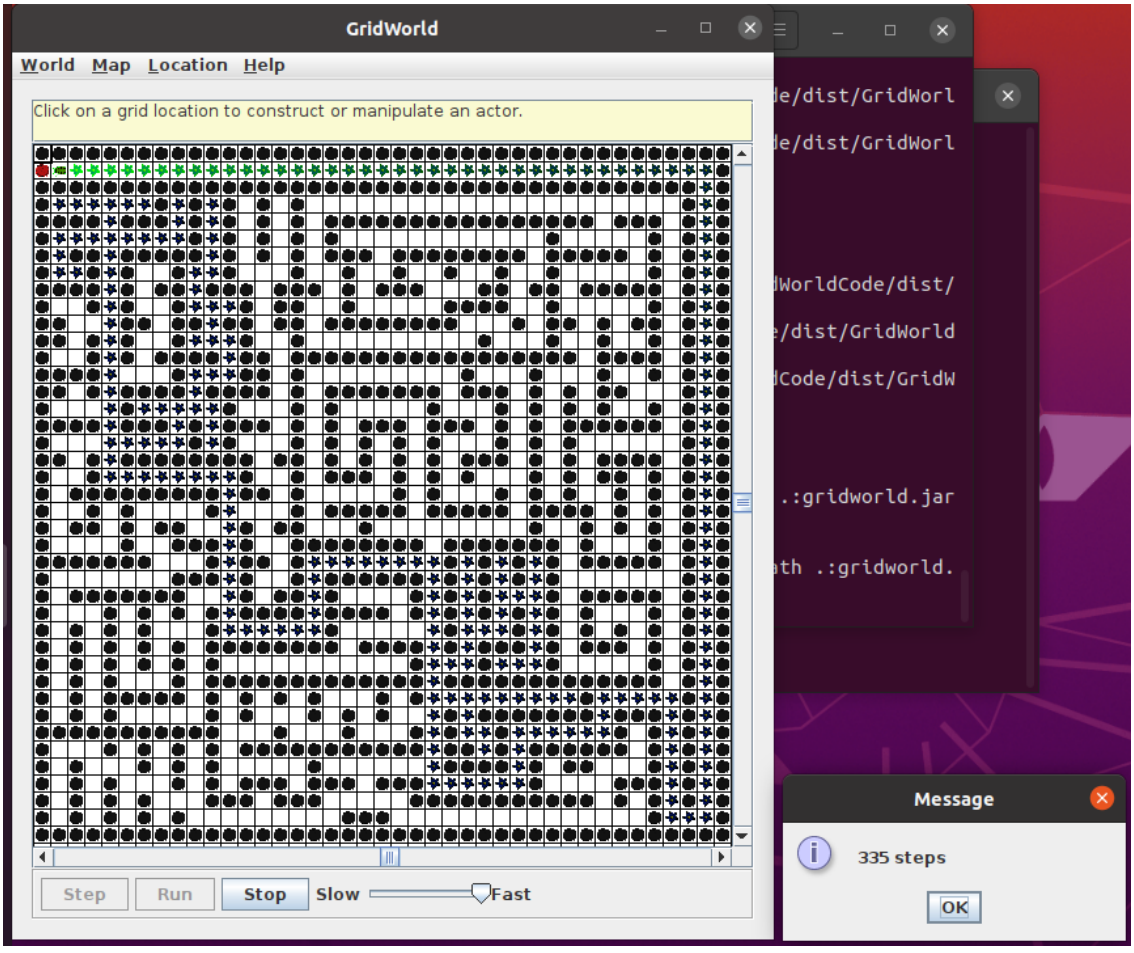
- **普通迷宫测试 - 2：总步数（含回溯）771步。**



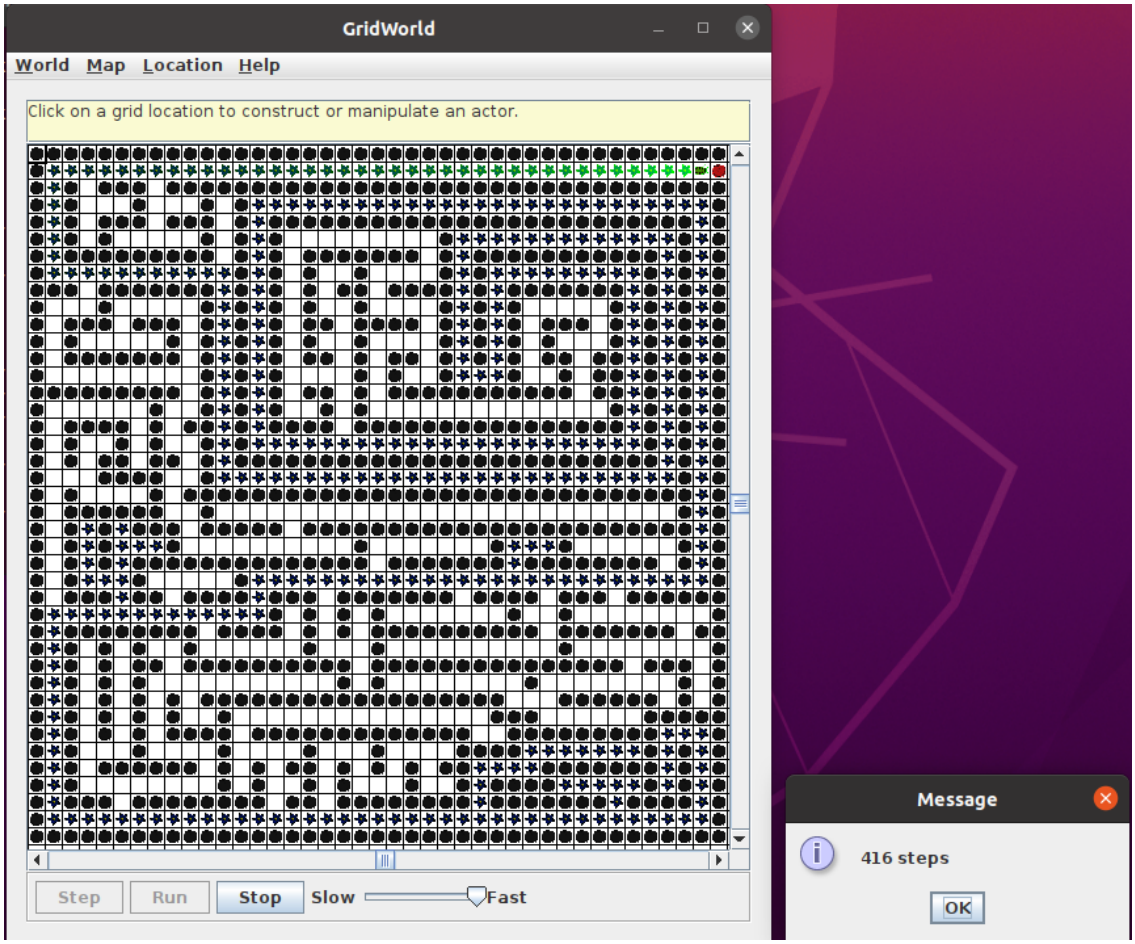
- **普通迷宫测试 - 3：总步数（含回溯）360步。**



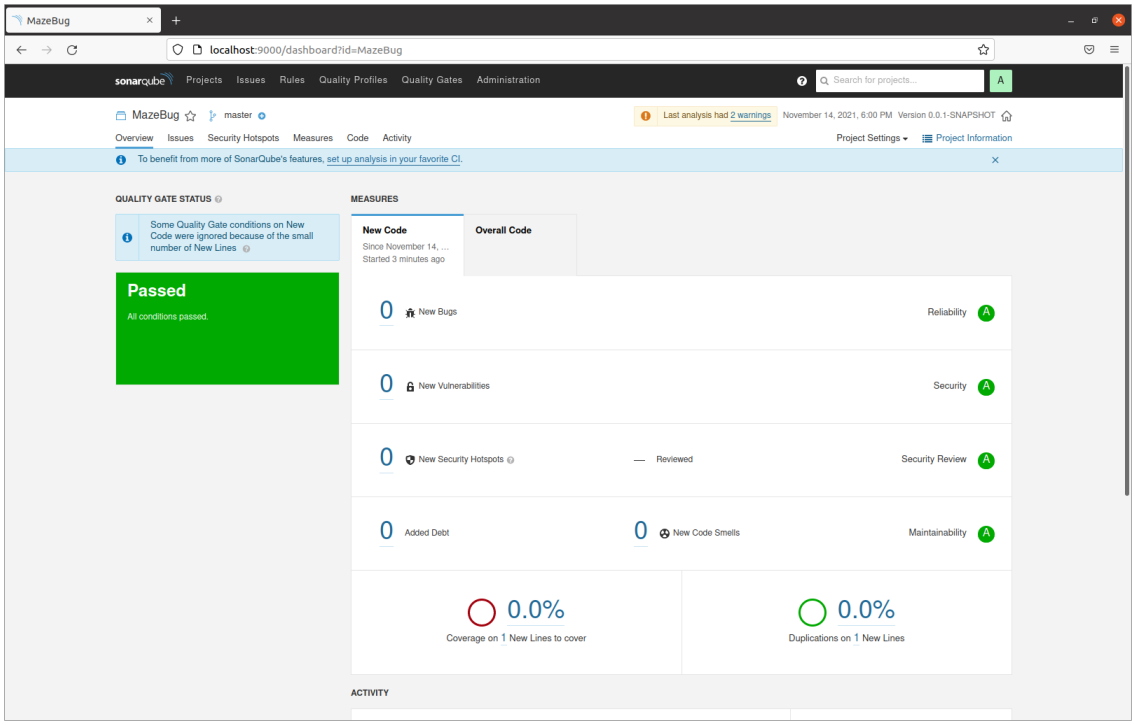
- **普通迷宫测试 - 4：总步数（含回溯）335步。**



- **普通迷宫测试 - 5：总步数（含回溯）416步。**



- **sonar: PASSED**



问题解决

- 问题1：打包程序后运行代码报错找不到
  - 找不到到类的问题我怀疑，如果jar打包了info.gridworld.info.MazeBugRunner类，可能会与project里的 MazBugRunner类冲突。在这个问题上浪费了很多时间，最后决定不要把MazeBugRunner打包进去。我去查看 WorldFrame 代码，需要打包 MazeBug的 原因是 GUI需要该类来创建迷宫上面的 MazeBug，但其实是不需要Runner程序也打包进去的。
- 问题2：GUI的菜单栏也没有显示 Map

- 文件也替换了，ant也显示执行成功，但是运行时就是不出现 Map 一栏，但MazeBug可以正常创建并正常执行，我想了很久。考虑到可能的问题包括我们编译运行用的是 /GridWorldCode/gridworld.jar， 而ant执行结果在 dist 文件夹中，即 /GridWorldCode/gridworld.jar 不会被覆盖，因此若我们的编译-classpath路径是 /GridWorldCode/gridworld.jar，则需要手动用新生成的 /GridWorldCode/dist/GridWorldCode/gridworld.jar 的替换原本的 /GridWorldCode/gridworld.jar。
- 最后，**需要 `ant clean` 先清空原来的结果**，再 `ant` 重新编译打包。
