OO 博客作业

——15061125 赵泽

一、第九次作业

1、程序主体

第九次作业是在一个 80*80 的地图上，模拟 100 台出租车的抢单接客情景。第九次作业在第七次的基础上，增加了道路开闭操作。基础操作仍然采用的是每台出租车一个线程，每次通过用刷新周期减去当前时刻到上一刷新周期的时间的差值，来进行 sleep 等待，让每台出租车接近“同一个钟”的状态。若一个叫车指令发出尚未达 3 秒，则每 100ms 扫描一次周围 4*4 格子中有没有符合条件的出租车。若达到 3 秒，则从叫车的点开始向周围进行广度优先遍历，找到第一个 mark 最大的且可以接客的出租车，完成摊派。通过逆序，将出租车到达叫车点的路径存储到出租车任务队列中。出租车接到客人后，从出租车所在点进行广度优先遍历，仍然通过逆序得到前往终点的路径。新增的操作我添加了一个新的正则匹配项，通过外部输入增加删除边的指令完成地图的更新。

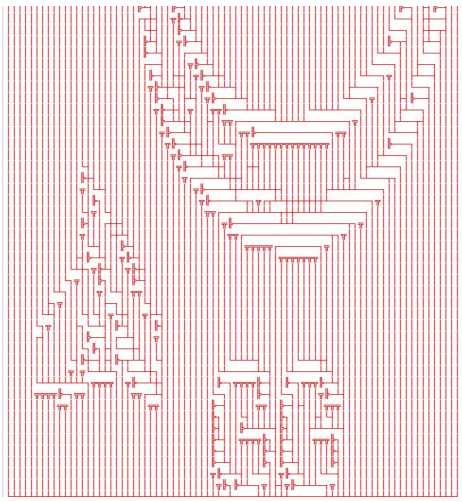
各类大致说明：

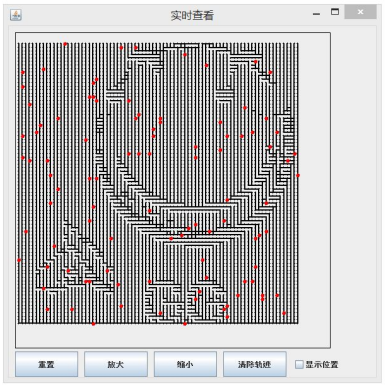
Main 主类：程序 main 函数所在类，此类无专属方法，主要起到调用其他各类，及在其他各类间传递信息的用处。

Controller 类：调度器，可完成查询出租车状态、查询某状态所有出租车、某单的扫描、某单的摊派等任务。

Driver 类：司机类，每台出租车一个线程。用于模拟每一台出租车的运动，写法类似于状态机，按照要求相应的进行状态转换。

Map 类：地图类，用于存储输入的地图，并判断是否合法（是否出界及是否为连通图）。并且在控制台输出我自行设计的按要求绘制的地图，效果如图：左侧为我画的静态图，右侧为 GUI 实际图。





People 类：模拟用户下单。不断读入外部输入，将查询或叫车请求发送给调度器类。

Random 类：辅助类，可提供两种操作，①传入一个 n，返回一个范围在[1,n]的随机数。此处使用的是 Math 中自带的随机数方法。②补全时间，可补全时间有 100ms，200ms 和 1s，在操作之前设置一次系统时间，操作完毕要睡眠时进行差值计算，计算结果即为睡眠时间。

Readin 类：读取外部地图文件，判断合法性后将地图发给地图类。

Syntagm 类：用于存储不同的结构体，可存储的大概有如下几类。

①存储叫车请求（出发点，目的地，时间）

②存储可抢单的出租车（位置，出租车号）

③存储广度优先遍历中已遍历过结点（位置，权值）

④存储司机在广度优先遍历过程中的中间位置（位置）

⑤存储查询指令的查询内容（查询内容及编号，查询结果存储路径）

2、程序结构度量

Main 主类：共 110 行，无属性与方法，全部为各类实例化及类间元素的调度。

Controller 类：共 460 行，八个属性，是一个方法，其中广度优先算法占用约 110 行，指令摊派处理方法占用 200 行。

Driver 类：共 470 行，十四个属性，一个构造方法，十六个方法。其中广度优先算法占用约 130 行。出租车状态变化及运动方法占用约 250 行。

Map 类：共 190 行，三个属性，一个构造方法，十个方法。其中绘制地图占用约 40 行，初始化设置地图占用约 100 行。

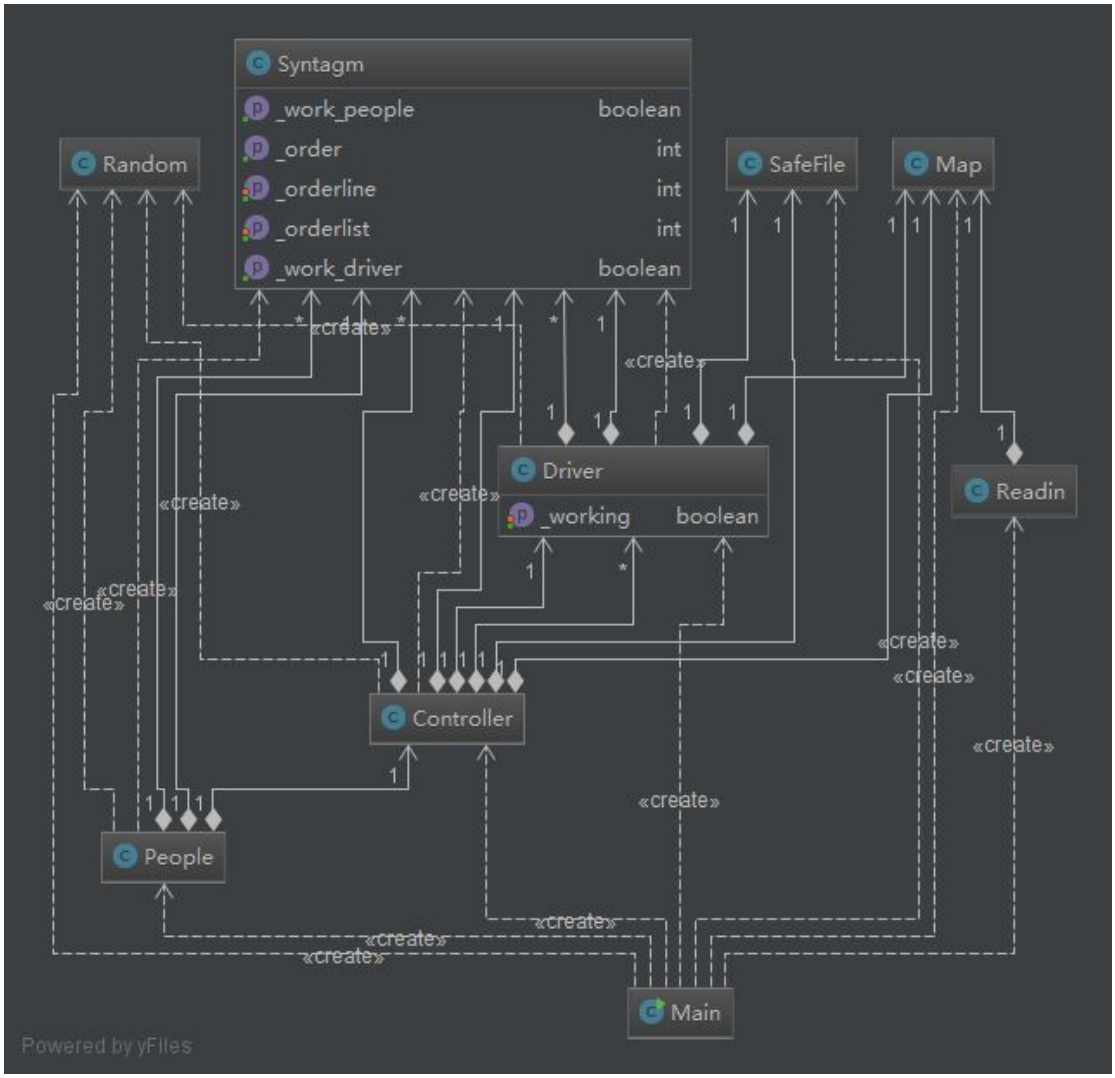
People 类：共 160 行，七个属性，四个方法。其中 run 方法占用约 130 行。

Random 类：共 35 行，两个属性，五个方法。其中非方法均不超过 5 行。

Readin 类：共 122 行，两个属性，三个方法。其中从文件读入占用约 95 行。

Syntagm 类：共 120 行，十六个属性，五个构造方法，二十三个方法。方法大多是一行的 get 或 set。

作业类图：



优点：方法细化，结构清晰

缺点：线程延迟较为严重

3、分析自己程序的 bug

当指令较多的时候，某些线程可能因为流量变化不统一导致出现死循环的情况，不断计算当前路径但是不进行抢单接单指令，导致出现问题。这应该是随机运行或者接单运行时候的 BFS 算法实现不完善，导致出现的错误。

4、自己发现别人 bug 的策略

我通过观察与操作，让一些可以接单的出租车去接一些需要跑的时间较长的单，来观察他最短路径的算法是否正确。和别人讨论我才知道，还可以将所有出租车一开始初始化到 0,0 点，然后在附近添加指令，观察对方的算法延迟导致的错误。新的道路开闭操作，我是通过让某辆车前往远处一个点，在她行进过程中关闭必经路线，观察是否会重新规划路线来测试的。

二、第十次作业

1、程序主体

第十次作业是在一个 80*80 的地图上，模拟 100 台出租车的抢单接客情景。第十次作业在第九次的基础上，增加了道路中的红绿灯。基础操作我仍然采用的是每台出租车一个线程，每次通过用刷新周期减去当前时刻到上一刷新周期的时间的差值，来进行 sleep 等待，让每台出租车接近“同一个钟”的状态。若一个叫车指令发出尚未达 3 秒，则每 100ms 扫描一次周围 4*4 格子中有没有符合条件的出租车。若达到 3 秒，则从叫车的点开始向周围进行广度优先遍历，找到第一个 mark 最大的且可以接客的出租车，完成摊派。通过逆序，将出租车到达叫车点的路径存储到出租车任务队列中。出租车接到客人后，从出租车所在点进行广度优先遍历，仍然通过逆序得到前往终点的路径。新增的操作我添加了一个红绿灯类，来模拟红绿灯周期性的变化，并且存储相应的红绿灯地图。

各类大致说明：

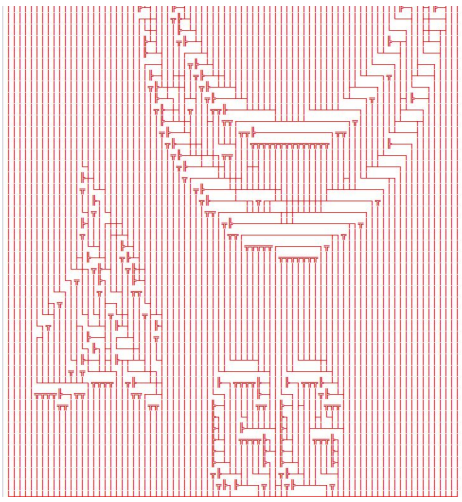
Main 主类：程序 main 函数所在类，此类无专属方法，主要起到调用其他各类，及在其他各类间传递信息的用处。

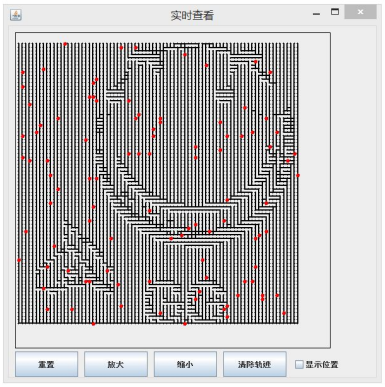
Controller 类：调度器，可完成查询出租车状态、查询某状态所有出租车、某单

的扫描、某单的摊派等任务。

Driver 类：司机类，每台出租车一个线程。用于模拟每一台出租车的运动，写法类似于状态机，按照要求相应的进行状态转换。

Map 类：地图类，用于存储输入的地图，并判断是否合法（是否出界及是否为连通图）。并且在控制台输出我自行设计的按要求绘制的地图，效果如图：左侧为我画的静态图，右侧为 GUI 实际图。





People 类：模拟用户下单。不断读入外部输入，将查询或叫车请求发送给调度器类。

Light 类：模拟红绿灯周期性变化，初始时从外部读入，之后不断地随开始时随机得到的周期进行红绿变化。

Random 类：辅助类，可提供两种操作，①传入一个 n，返回一个范围在[1,n]的随机数。此处使用的是 Math 中自带的随机数方法。②补全时间，可补全时间有 100ms，200ms 和 1s，在操作之前设置一次系统时间，操作完毕要睡眠时进行差值计算，计算结果即为睡眠时间。

Readin 类：读取外部地图文件，判断合法性后将地图发给地图类。

Syntagm 类：用于存储不同的结构体，可存储的大概有如下几类。

①存储叫车请求（出发点，目的地，时间）

②存储可抢单的出租车（位置，出租车号）

③存储广度优先遍历中已遍历过结点（位置，权值）

④存储司机在广度优先遍历过程中的中间位置（位置）

⑤存储查询指令的查询内容（查询内容及编号，查询结果存储路径）

2、程序结构度量

Main 主类：共 110 行，无属性与方法，全部为各类实例化及类间元素的调度。

Controller 类：共 460 行，八个属性，是一个方法，其中广度优先算法占用约 110 行，指令摊派处理方法占用 200 行。

Driver 类：共 470 行，十四个属性，一个构造方法，十六个方法。其中广度优先算法占用约 130 行。出租车状态变化及运动方法占用约 250 行。

Map 类：共 190 行，三个属性，一个构造方法，十个方法。其中绘制地图占用约 40 行，初始化设置地图占用约 100 行。

Light 类：共 150 行，十个属性，三个方法。其中 run 方法占约 50 行。

People 类：共 160 行，七个属性，四个方法。其中 run 方法占用约 130 行。

Random 类：共 35 行，两个属性，五个方法。其中非方法均不超过 5 行。

Readin 类：共 122 行，两个属性，三个方法。其中从文件读入占用约 95 行。

Syntagm 类：共 120 行，十六个属性，五个构造方法，二十三个方法。方法大多是一行的 get 或 set。

3、分析自己程序的 bug

当不同行同时输入两条相同指令时，不判断 same，分析可能是由于正则匹配时每次匹配一行后发送给相应类过程中，Array List 的 remove 操作耗时过长，导致出现了两行间的延迟现象。

4、自己发现别人 bug 的策略

通过更换不同的特殊地图，检测对方在不同的特殊情况下程序的运行情况。不同的地图可以引出不同的细节欠考虑。比如全连通图和一字长蛇图，可以验证对方的道路选择和道路检索算法优劣性。全横和全竖图可以验证对方随机路径选择的随机性和可靠性。

三、第十一次作业

1、程序主体

第十一次作业是在一个 80*80 的地图上，模拟 100 台出租车的抢单接客情景。第十一次作业在第十次的基础上，增加了可以经过已删除边的超级出租车。基础操作我仍然采用的是每台出租车一个线程，每次通过用刷新周期减去当前时刻到上一刷新周期的时间的差值，来进行 sleep 等待，让每台出租车接近“同一个钟”的状态。若一个叫车指令发出尚未达 3 秒，则每 100ms 扫描一次周围 4*4 格子中有没有符合条件的出租车。若达到 3 秒，则从叫车的点开始向周围进行广度优先遍历，找到第一个 mark 最大的且可以接客的出租车，完成摊派。通过逆序，将出租车到达叫车点的路径存储到出租车任务队列中。出租车接到客人后，从出租车所在点进行广度优先遍历，仍然通过逆序得到前往终点的路径。新增的操作我添加了一个超级出租车类，来模拟超级出租车，通过对父类方法的简单重写，完成了子类的超级出租车。

各类大致说明：

Main 主类：程序 main 函数所在类，此类无专属方法，主要起到调用其他各类，及在其他各类间传递信息的用处。

Controller 类：调度器，可完成查询出租车状态、查询某状态所有出租车、某单的扫描、某单的摊派等任务。

Driver 类：司机类，用于模拟每一台出租车的运动，写法类似于状态机，按照要求相应的进行状态转换。

Driver_VIP 类：超级出租车类，用于在普通出租车基础上，增加断路可通过能力。

Map 类：地图类，用于存储输入的地图，并判断是否合法（是否出界及是否为连通图）。并且在控制台输出我自行设计的按要求绘制的地图。

People 类：模拟用户下单。不断读入外部输入，将查询或叫车请求发送给调度器类。

Light 类：模拟红绿灯周期性变化，初始时从外部读入，之后不断地随开始时随机得到的周期进行红绿变化。

Random 类：辅助类，可提供两种操作，①传入一个 n，返回一个范围在[1,n]的

随机数。此处使用的是 Math 中自带的随机数方法。②补全时间，可补全时间有 100ms，200ms 和 1s，在操作之前设置一次系统时间，操作完毕要睡眠时进行差值计算，计算结果即为睡眠时间。

Readin 类：读取外部地图文件，判断合法性后将地图发给地图类。

Syntagm 类：用于存储不同的结构体，可存储的大概有如下几类。

①存储叫车请求（出发点，目的地，时间）

②存储可抢单的出租车（位置，出租车号）

③存储广度优先遍历中已遍历过结点（位置，权值）

④存储司机在广度优先遍历过程中的中间位置（位置）

⑤存储查询指令的查询内容（查询内容及编号，查询结果存储路径）

2、程序结构度量

Main 主类：共 110 行，无属性与方法，全部为各类实例化及类间元素的调度。

Controller 类：共 460 行，八个属性，是一个方法，其中广度优先算法占用约 110 行，指令摊派处理方法占用 200 行。

Driver 类：共 470 行，十四个属性，一个构造方法，十六个方法。其中广度优先算法占用约 130 行。出租车状态变化及运动方法占用约 250 行。

Driver_VIP 类：共 500 行，一个属性，六个方法。run 方法占用约 150 行。

Map 类：共 190 行，三个属性，一个构造方法，十个方法。其中绘制地图占用约 40 行，初始化设置地图占用约 100 行。

Light 类：共 150 行，十个属性，三个方法。其中 run 方法占约 50 行。

People 类：共 160 行，七个属性，四个方法。其中 run 方法占用约 130 行。

Random 类：共 35 行，两个属性，五个方法。其中非方法均不超过 5 行。

Readin 类：共 122 行，两个属性，三个方法。其中从文件读入占用约 95 行。

Syntagm 类：共 120 行，十六个属性，五个构造方法，二十三个方法。方法大多是一行的 get 或 set。

3、分析自己程序的 bug

多条最短路径可选择的时候，未进行随机道路选择。这个 bug 的原因是因为我仅找到一条符合路径的最短边就不继续寻找。我觉得在像全连通图这样的特殊图中，如果找到所有的合理道路并随机选择，耗时可能超乎想象。

4、自己发现别人 bug 的策略

依然是更换不同的特殊地图，检测对方在不同的特殊情况下程序的运行情况。阅读对方 JSF 中说明，构造符合前置条件但违背后置条件的例子。设置比较特殊的地图，使得某点到某点间存在 A、B 两条最短路径，安排多辆车完成两点间接单任务，观察是否前车所在路径会对后车产生影响。

四、规格化设计

人类历史长河中，规格化、标准化一直是人们所追求的。

人类从原始的自然人开始，在与自然的生存搏斗中为了交流感情和传达信息的需要，逐步出现了原始的语言、符号、记号、象形文字和数字。从第一次人类社会的农业、畜牧业分工中，由于物资交换的需要，要求公平交换、等价交换的的原则，决定度、量、衡单位和器具标准统一，逐步从用人体的特定部位或自然物到标准化的器物。

当人类社会第二次产业大分工，即农业、手工业分化时，为了提高生产率，对工具和技术规范化就成了迫切要求，从遗世的青铜器、铁器上可以看到那时科学技术和标准化水平的发展。宋代毕昇发明的活字印刷术，运用了标准件、互换性、分解组合、重复利用等标准化原则，更是古代标准化里程碑。

进入以机器生产、社会化大生产为基础的近代标准化阶段。科学技术适应工业的发展，为标准化提供了大量生产实践经验，也为之提供了系统实验手段，摆脱了凭直观和零散的形式对现象的表述和总结经验的阶段，从而使标准化活动进入了定量地以实验数据科学阶段，并开始通过民主协商的方式在广阔的领域推行工业标准化体系，作为提高生产率的途径。

工业现代进程中，由于生产和管理高度现代化、专业化、综合化、这就使现代产品或工程、服务具有明确的系统性和社会化，一项产品或工程、过程和服务、过程和服务，往往涉及几十个行业和几万个组织及许多门的科学技术，如美国的"阿波罗计划"、"曼哈顿计划"，从而使标准化活动更具有现代化特征。

规格化设计也是如此，通过满足规格的设定来撰写相应程序，达到满足规格的效果，加强了编码的交互能力，加强了封装性和用户体验。

五、好的例子

1）

```
/* @ REQUIRES: gui.repOK() && map.repOK() && light.repOK()
 *                && 0<=name<100
 * @ MODIFIES:  this  源自父类的属性
 * @ EFFECTS:   初始化所有的属性
 *              初始化Driver中的gui(用于后期和gui协调显示位置之类的操作)
 *              随机产生一个合法坐标(line,list)
 *              1<=line<=80
 *              1<=list<=80
 *              location和last location均初始化为line*100+list
 *              出租车状态均初始化为2
 *              创建每个出租车的工作列表worklist
 *              其余数组均初始化为0或真
 *              在gui上显示点初始位置
 */
```

2）

```
public void addmap(int line,int list,int value){
    /* @ REQUIRES:  1<=line<=80
     *          &&  1<=list<=80
     *          &&  0<=value<=4
     *          &&  road!=null
     *          &&  repOK()
     * @ MODIFIES:road[][] road_vip[][]
     * @ EFFECTS:
     *          初次使用: 初始化road和road_vip两个地图
     *          之后使用: 增删地图操作，修改(line,list)所在位置的右侧连线和下方连线的属性
     *              若右侧已有连线而要添加右侧连线==>提示已有该连线
     *              若下方已有连线而要添加下方连线==>提示已有连线
     *
     *              若右侧没有连线而要删除右侧连线==>提示不存在该连线
     *              若下方没有连线而要删除下方连线==>提示不存在该连线
     *
     *              否则==>添加或删除相应的连线，同步显示在gui上，增加操作会同步在road_vip上，但删除操作不会
     */
```

3）

```
/* @ REQUIRES:  Readin.MAX=80   this中各项非Null repOK()
 * @ MODIFIES:  weight_map_old[][]  weight_map_new[][]
 * @ EFFECTS:   刷新流量地图，将之前200ms流量地图赋给weight_map_old,将weight_map_new置零
 */
```

六、不好的例子

1）

```
public int getmapweight(int line,int list,int way){
    /* @ REQUIRES:  1<=line<=80
     *           &&  1<=list<=80
     *           &&  0<=way<=1
     *           &&  weight_map_old!=null
     *           &&  地图中(line,list)与way相对应的点之间存在连线
     *           (way=1时取得是相连的下方的连线
     *            way=0时取得是相连的右侧的连线)
     *           &&  repOK()
     * @ MODIFIES: none
     * @ EFFECTS:  通过与下方set map weight方法同样的数字处理算法
     *              返回(line,list)所相连的边的流量
     */
```

描述不清晰，应该清晰写出可能的返回值

2）

```
public void run(){
    /* @ REQUIRES: this属性全部非null &&  repOK()
     * @ MODIFIES: this
     * @ EFFECTS: 从控制台读入指令,共有三大类，五种指令请求
     *           1-叫车请求
     *           2-查询某出租车状态
     *           3-查询某状态所有出租车
     *           4-打开一条通路
     *           5-关闭一条通路
     *
     *           读入指令后，将其中合法的部分存入三个ArrayList中
     *
     *           100ms一处理合法请求(抛出给相应类使用)
     */
```

没有写明修改结果

3）

```
Syntagm(int line,int list,int orderline,int orderlist,int time){//存用户请求
    /* @ REQUIRES:  1<=line<=80
     *           &&  1<=list<=80
     *           &&  1<=orderline<=80
     *           &&  1<=orderlist<=80
     *           &&  0<=time
     *           &&  repOK()
     * @ MODIFIES: this
     * @ EFFECTS: 创建一个结构体，存储用户请求包含的信息
     */
```

没写明具体修改内容

4）

```
/* @ REQUIRES: String path
 *          && map!=null && map.repOK()
 *          && light!=null &&   light.repOK()
 *          &&  repOK()
 * @ MODIFIES: light
 * @ EFFECTS: read from the file and save it to the light map
 * return value
 *
 * 0    correct
 * -1   detail error
 * -2   file error
 * */
```

返回值条件不明确

5）

```
public boolean read_light(String path){
    /* @ REQUIRES: String path &&  repOK()
     * @ MODIFIES: none
     * @ EFFECTS: tell the main if the light file read succeed
     *                   succeed==>return true
     *                   failed ==>return false
     */
```

返回条件不明确

七、bug 与过程规格质量关系

过程规格不明确导致出现细节 bug，大体为正比关系