付出时间与努力，终可收获！

暨2015级软件工程中级实训总结报告

15331046 陈志扬 教务一班

从第四周到第十一周的实训今天终于宣告结束，虽然中间空了几周，但还是占了很多个周末。付出时间与努力，终可收获！谈到收获，我想可在报告中简单对每个阶段做些总结，下面我就简要谈谈我对每个阶段的思考。

第一阶段总结：

（1）vi编辑器的使用：vi分为三种模式，分别是command mode，insert mode，last line mode。

我们在终端输入vi+文件名进入vi编辑器时，是command mode，在命令行模式下可以进行的操作有很多，最为常用当然是插入：i，a，o，按这些字母将进入插入模式，此外还有移动光标操作，删除文字等等。若要从其他模式退出来，我们按ESC键即可回到命令行模式。另外，我们写文件当然要保存了，首先要确保在命令行模式下，我们按下:号(冒号),即可进入last line mode，一般输入wq即可保存并退出。若只需保存只需按w，若只需退出则按q!（注意感叹号)。

vi还有很多命令，在此就简单review上面几点。说实话，vim很强大，各种插件很多，但是原生vi对新手来说真的是一种折磨。我们当然可以使用gedit或者sublime

text 来代替。通过这次 vi 的学习，我逐渐了解了 vi，接下来还是需要多使用 vim，特别要体验它的插件。

（2）Java 语言的学习及手动配置：首先简单介绍一下我在虚拟机上配置 Java 环境吧。根据教程在 Sun 公司下载 JDK，然后安装 JDK(涉及 chmod 解决可执行权限问题，执行 bin 文件等)，配置环境变量 JAVA_HOME,PATH,CLASSPATH，source 生效，重启系统。在整个过程中，当然是配置环境变量最困难了，因为/etc/profile 是 readonly，又是因为尝试使用 vi 去添加环境变量，vi 真的难用，所以一开始一直失败，弄了好久才配置好。

接下来，学习 Java 语法。第一感觉，和 C++很像，基本语法相差无几。和 C++不同的是，Java 是纯粹面向对象的程序设计语言，处处体现着面对对象编程的思想,因此 Java 并没有 C++中的指针操作，不能再类外定义全局变量，只能在某个类中定义一种公用静态变量来实现全局变量的功能；不再支持头文件等等。

（3）Ant 配置和学习：关于 Ant 配置和学习，由于云桌面已经帮我们配置好，所以我是在虚拟机上配置的(云桌面有权限，不能修改 profile，但好像可以修改 bashrc 来代替)。按照提供的教程，在官网上下载后解压到指定目录，设置系统环境变量(嗯，我又用到 vi 了，就是这样不断熟悉 vi 的)，重登系统，直接 ant 和 ant -version 测试是否安装

成功。

环境配置：vi /etc/profile(虚拟机上)

```
export ANT_HOME=/usr/local/ant
export PATH=$PATH:$ANT_HOME/bin
```

检查测试：

命令行：

```
ant
```

显示：

```
Build build.xml does not exist!
Build failed
```

命令行：

```
ant -version
```

显示：

```
Apache Ant version XXX compiled ...
```

配置成功了！

接下来就是学习 Ant 了。几大关键元素 project、target、property、task；对应的属性的作用；Ant 的常用任务 copy、delete、move、mkdir、echo；最重要的：利用 Ant 构建和部署 Java 工程(利用 javac 任务编译 Java 程序，利用 java 任务运行 Java 程序，jar 任务生成 jar 文件等等)。

Ant 是一个非常强大的工具，可实现项目的自动构建和部署等功能，类似于我们之前用过的 makefile，通过这次简单地使用，对项目管理有一定的帮助，值得我们深入研究。

(4) Junit 的学习：由于要求使用 junit-4.9，所以要先把 junit-4.9.jar 解压放到同个项目文件夹里(这个非常重

要，如果没这个，要配置好 eclipse 的 junit)，编写 HelloWorldTest.java 进行简单的单元测试，使用以下两条命令运行：

```
javac -classpath .:../../junit-4.9.jar HelloWorldTest.java
java -classpath .:../../junit-4.9.jar -ea org.junit.runner.JUnitCore HelloWorldTest
```

在运行需要引用 jar 包的程序时，需要使用上述命令行编译运行，其中-classpath 可简写为-cp，若有多个 jar 包可用:号（冒号）隔开。

（5）SonarQube 是一个用于代码质量管理的开源平台，用于管理源代码的质量。下面简单谈谈 SonarQube 的配置（注意拼写正确的路径名）

1.已安装 Java 环境

2.安装 sonar 和 sonar-runner：将 zip 包解压。

3.设置 SONAR_HOME,SONAR_RUNNER_HOME 环境变量，并将 SONAR_RUNNER_HOME 加入 PATH。

具体操作：

`sudo vi /etc/profile`

(云桌面 `gedit ~/.bashrc`)

添加下面几条命令

`export SONAR_HOME=.../sonar-3.7.4/bin/Linux-x86-64` (...表示路径名)

`export SONAR_RUNNER_HOME=.../sonar-runner-2.4` (云桌面我们是用sonar-scanner,所以这里我们要把sonar-runner改成sonar-scanner)

`export PATH=$SONAR_RUUNER_HOME/BIN:$PATH`

最后保存退出，在终端输入 `source /etc/profile` (云桌面上 `source ~/.bashrc` ;重启系统。

4.添加**数据库**(略)

5.启动服务

shell里输入

`cd $SONAR_HOME`

`./sonar.sh start` 启动服务

`./sonar.sh stop` 停止服务

`./sonar.sh restart` 重启服务

使用SonarQube Runner分析源代码

创建sonar-project.properties配置文件，进入含该文件的目录下输入sonar-runner，打开localhost:9000查看即可。

注意每次使用完sonar记得关闭，进入启动目录，`./sonar.sh stop`

Sonar 的使用方法如下：

在 shell 里面键入 cd $SONAR_HOME，可以直接进入启动目录。在 shell 里面键入
./sonar.sh start 启动服务
./sonar.sh stop 停止服务
./sonar.sh restart 重启服务

```
sonar.sh start
Starting sonar...
Started sonar.
```

访问 http:\\localhost:9000，如果显示 SonarQube 的项目管理界面，表示安装成功。

但要注意在含有”sonar-project.properties”的目录下运行 sonar，输入命令 sonar-runner:

```
retoruto@ubuntu:~/Desktop/shixun/testSonar$ sonar-runner
```

确认代码无误运行 sonar 之后:

```
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
Total time: 33.315s
Final Memory: 14M/113M
```

sonar-runner 成功后主要检查几个方面：critical、major、minor、duplication、注释等等，然后逐一修改到满足要求即可，一般要求注释要超过 10%，总评要高于

60%，这个修改的过程也是极其艰难的，每一次在 sonar 方面都花了很长时间，其实这也是好的，代码规范的问题程序员有责！

第二阶段总结：

（1）各个 Bug 的代码编写：这次的实训和初级实训最大的不同之处在于：中级实训只需要我们实现逻辑代码，整个框架都已经给出，所以相对而言也比较简单，我们在实现逻辑代码的过程中只需要理解每个 api 接口的作用，然后灵活运用 api 实现我们的代码。考察的是理解代码的能力以及能正确使用已实现的 api 接口。

另外，需注意的一点是，各个 bug 之间的关系以及 bug 和 actor 的继承关系，例如 CircleBug、Dancing 和 Bug 类的继承关系：

```
public class CircleBug extends Bug
```

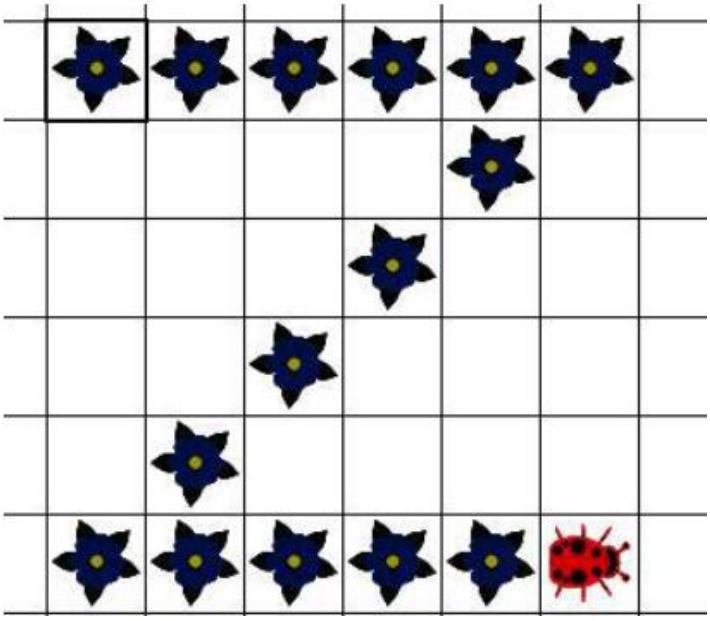
```
public class DancingBug extends Bug
```

各个 bug 之间也只有 act 函数不同，通过修改 act 函数可对每个 bug 定义不同的行为，例如 ZBug 的 act 函数如下：

```
public void act()
{
    if (steps < total && canMove())
    {
        move();
        steps++;
        if (steps % sideLength == 0 && steps / sideLength == 1) {
            setDirection(SW);
        }
        if (steps % sideLength == 0 && steps / sideLength == 2) {
            setDirection(EAST);
        }
    }

}
```

运行结果如下：



（2）Jumper 类的设计：Jumper 的设计需要参考 part2 的 Runner 类的写法，根据问题以及自己对 actor 的定义来编写，之前已提交过设计文档和测试文档，这里不做赘述。请 TA 谅解！

（3）Critter 类的设计：各个 Critter 类的设计思路和 part2 各个 bug 的设计思路大体相同，都是在继承 Critter 类然后定义 makeMove、getActors、getMoveLocations 这些函数，定义不同，实现不同的 Critter。

各个 Critter 类的不同之处在于：

① 遇到不同的 actor 如何处理，例如 RockHound 遇到 Rock 是把 Rock 清除掉，而不是换方向。

② 每个 Critter 有不同的方向，例如 Crab 只有左右方向，而其他的有八个方向。

③ 速度，例如 QuickCrab 每次移动两格，而其他是一格。

（4）Grid 类的设计：继承 AbstractGrid，实现不同功能的

Grid，主要问题是：

① 可用三种不同的方法：ArrayList、HashMap、TreeMap 来实现每个 Grid，区别好 HashMap 和 TreeMap 的用法，ArrayList 使用较为简单。

② 编写 UnBoundedGrid 可参考设计文档中，大部分代码也已给出。

第三阶段总结：

（1）ImageProcessing：用二进制流读取图像，要注意各个 api 的调用方法，好在这些函数的用法很容易查阅到。每一种颜色的写法都是类似的，只要定义好一种颜色，其他两种颜色照搬即可。

```
public Image showChanelR(Image image) {
    colorFilter red = new colorFilter(0);
    return Toolkit.getDefaultToolkit().createImage(
        new FilterImageSource(image.getSource(), red));
}
```

各个颜色的处理，圈起来的是灰度：

```
class colorFilter extends RGBImageFilter {
    private int colorNum;

    public colorFilter(int num) {
        colorNum = num;
    }
    public int filterRGB(int x, int y, int rgb) {
        if (colorNum == 0) {
            return rgb & 0xffff0000;
        }
        else if (colorNum == 1) {
            return rgb & 0xff00ff00;
        }
        else if (colorNum == 2) {
            return rgb & 0xff0000ff;
        }
        else {
            int gray = (int)( ((rgb & 0x00ff0000) >> 16) * 0.299
                + ((rgb & 0x0000ff00) >> 8) * 0.587
                + (rgb & 0x000000ff) * 0.114 );
            return (rgb & 0xff000000) + (gray << 16) + (gray << 8) + gray;
        }
    }
}
```

（2）ImageIO：实训要求“读”不能使用 api，所以只能通过位

图信息来处理图像，这里简单截图来说明一下：

```
// 保存位图数据位置的地址偏移，也就是起始地址
offset = (int)( (bmpHead[13] & 0xff) << 24 | (bmpHead[12] & 0xff) << 16
    | (bmpHead[11] & 0xff) << 8 | (bmpHead[10] & 0xff) );
offset -= 54;

// 保存位图宽度(以像素个数表示)
widthOfbmpInfo = (int)( (bmpInfo[7] & 0xff) << 24 | (bmpInfo[6] & 0xff) << 16
    | (bmpInfo[5] & 0xff) << 8 | (bmpInfo[4] & 0xff) );

// 保存位图高度(以像素个数表示)
heightOfbmpInfo = (int)( (bmpInfo[11] & 0xff) << 24 | (bmpInfo[10] & 0xff) << 16
    | (bmpInfo[9] & 0xff) << 8 | (bmpInfo[8] & 0xff) );

// 保存图像大小。这是原始(:en:raw)位图数据的大小，不要与文件大小混淆。
sizeOfbmpInfo = (int)( (bmpInfo[23] & 0xff) << 24 | (bmpInfo[22] & 0xff) << 16
    | (bmpInfo[21] & 0xff) << 8 | (bmpInfo[20] & 0xff) );

// 保存每个像素的位数，它是图像的颜色深度。常用值是1、4、8(灰阶)和24(彩色)。
bitCount = (int)( (bmpInfo[15] & 0xff) << 8 | (bmpInfo[14] & 0xff) );
```

```
// 读取所有RGB数据
file.read(totalByte, 0, pixelSize);
pixelArray = new int[widthOfbmpInfo * heightOfbmpInfo];

int index = 0;
for (int j = 0; j < heightOfbmpInfo; j++) {
    for (int i = 0; i < widthOfbmpInfo; i++) {
        // 第一个0xff << 24表示透明度
        pixelArray[widthOfbmpInfo * (heightOfbmpInfo - j - 1) + i] =
            (0xff << 24)
            | (totalByte[index + 2] & 0xff) << 16
            | (totalByte[index + 1] & 0xff) << 8
            | (totalByte[index] & 0xff);
        index += 3;
    }
    index += numOfEmptyByte;
}

image = Toolkit.getDefaultToolkit().createImage(new MemoryImageSource(
    widthOfbmpInfo, heightOfbmpInfo, pixelArray, 0, widthOfbmpInfo));
```

（3）MazeBug 的深度优先算法 DFS：

① 先将树的所有节点标记为“未访问”状态。

② 输出起始节点，将起始节点标记为“已访问”状态。

③ 将起始节点入栈。

④ 当栈非空时重复执行以下步骤：

1. 取当前栈顶节点

2. 如果当前栈顶节点是结束节点（迷宫出口），输出该节

点，结束搜索

3. 如果当前栈顶节点存在“未访问”状态的邻接节点，则选择一个未访问节点，置为“已访问”状态，并将它入栈，继续步骤 1

4. 如果当前栈顶节点不存在“未访问”状态的邻接节点，则将栈顶节点出栈，继续步骤 1.

（4）Jigsaw 的广度优先算法 BFS：

BFS：从图中某节点 V 出发，在访问了 V 之后依次访问 V 的各个未曾访问过的邻接点，然后分别从这些邻接点出发依次访问它们的邻接点，并使"先被访问的节点的邻接点"先于"后被访问的节点的邻接点"被访问，直至图中所有已被访问的节点的邻接点都被访问到。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。

BFS 算法步骤：

① 将起始节点放入一个 openList 中。

② 如果 openList 为空，则搜索失败，问题无解；否则重复以下步骤：

1. 访问 openList 中的第一个节点 v，若 v 为目标节点，则搜索成功，退出

2. 从 openList 中删除节点 v，放入 closeList 中

3. 将所有与 v 邻接且未曾被访问的节点放入 openList 中

BFS 部分代码：

```
// Write your code here.

// 用以存放某一节点的邻接节点
Vector<JigsawNode> followJNodes = new Vector<JigsawNode>();

// 重置求解完成标记为false
isCompleted = false;

// (1)将起始节点放入openList中
this.openList.addElement(this.beginJNode);

// (2) 如果openList为空，则搜索失败，问题无解;否则循环直到求解成功
while (!this.openList.isEmpty()) {

    // (2-1)访问openList的第一个节点N，置为当前节点currentJNode
    //      若currentJNode为目标节点，则搜索成功，设置完成标记isCompleted为true，计算解路径，退出。
    this.currentJNode = this.openList.elementAt(0);
    if (this.currentJNode.equals(this.endJNode)){
        isCompleted = true;
        this.calSolutionPath();
        break;
    }

    // (2-2)从openList中删除节点N,并将其放入closeList中，表示以访问节点
    this.openList.removeElementAt(0);
    this.closeList.addElement(this.currentJNode);
    searchedNodesNum++;
        // 记录并显示搜索过程
        pw.println("Searching.....Number of searched nodes:" + this.closeList.size() + "   Current state:" + this.currentJNode.toString());
        System.out.println("Searching.....Number of searched nodes:" + this.closeList.size() + "   Current state:" + this.currentJNode.toString());

    // (2-3)寻找所有与currentJNode邻接且未曾被访问的节点，将它们插入到openList中
    followJNodes = this.findFollowJNodes(this.currentJNode);
    while (!followJNodes.isEmpty()) {
        this.openList.addElement(followJNodes.elementAt(followJNodes.size() - 1));
        followJNodes.removeElementAt(followJNodes.size() - 1);
    }

}
```

（5）Jigsaw 的启发式搜索算法：

与盲目搜索不同，启发式搜索（如 A*算法）利用问题拥有的启发信息来引导搜索，动态地确定搜索节点的排序，以达到减少搜索范围，降低问题复杂度的目的。在 N-数码问题中，每搜索到每一个节点时，通过"估价函数"对该节点进行"评估"，然后优先访问"最优良"节点的邻接节点，能够大大减少求解的时间。

估价函数 f(n)用来估计节点 n 的重要性，表示为：从起始节点经过节点 n，到达目标节点的代价。f(n)越小，表示节点 n 越优良，应该优先访问它的邻接节点。Wiki 上给了下面三种估价方法：

1. 所有放错码的数码个数

2. 所有放错码的数码与正确位置的距离之和

3. 后续节点不正确的数码个数

可同时使用多个估价方法，通过适当调整权重能够加快搜索速度。

启发式搜索算法步骤：

一. 将起始节点放入一个 openList 中。

二. 如果 openList 为空，则搜索失败，问题无解；否则重复以下步骤：

1. 访问 openList 中的第一个节点 v，若 v 为目标节点，则搜索成功，退出；

2. 从 openList 中删除节点 v，放入 closeList 中；

3. 利用估价函数，将所有与 v 邻接且未曾被访问的节点进行估价，按照估价大小（小的在前）放入 openList 中

估价函数如下：

```
private void estimateValue(JigsawNode jNode) {
    // 后续节点不正确的数码个数
    int s = 0;
    int dimension = JigsawNode.getDimension();
    for(int index = 1 ; index < dimension * dimension; index++){
        // 后续节点不正确
        if(jNode.getNodesState()[index] + 1 != jNode.getNodesState()[index + 1]) {
            s++;
        }
    }
    // 放错位的数码个数
    int numOfMisplaced = 0;
    for (int index = 1; index < dimension * dimension; index++) {
        if (jNode.getNodesState()[index] != index) {
            numOfMisplaced++;
        }
    }
    // 曼哈顿距离
    int manhattanDistance = 0;
    for (int index = 1; index <= dimension * dimension; index++) {
        int num = jNode.getNodesState()[index];
        if (num != index && num != 0) {
            int x1 = (num - 1) % dimension;
            int y1 = (num - 1) / dimension;
            int x2 = (index - 1) % dimension;
            int y2 = (index - 1) / dimension;
            manhattanDistance += Math.abs(x1 - x2) + Math.abs(y1 - y2);
        }
    }
    jNode.setEstimatedValue(5 * s + 1 * numOfMisplaced + 12 * manhattanDistance);
}
```

心得体会：中级实训确实比初级实训简单，主要原因是框架代码都已编写好。阶段一遇到的困难主要在配置环境上和 vi 编辑器的初体验，手动配置环境真的要注意很多问题；另外，sonar 要改的问题也是很多，magic number、空格和 tab 键的替换、final class 等等。阶段二编写逻辑代码，一旦理解了各个函数的功能作用，就显得简单了，特别是 wiki 上这部分还有很多问题需要回答，进一步理解了各个 api 接口。

收获：

① 初探 Java 语言。

② 能熟悉使用 Eclipse。

③ 本实训涉及到的关于 Linux 环境下配置。

④ 提高了理解代码和利用 api 接口的能力。

最后，还是感叹一声：终于结束了！