大二实训总结

不知不觉为期六周的大二软件工程实训已经走向尾声，个人感觉而言比大一那会的实训简单多了。实训分成三个阶段：

阶段一是熟悉 Java 编程的一些常用工具：Vi, Ant、Junit 和 SonarQube 等等。

阶段二开始 GridWorld 项目的编程，设计和制造各种 Actor 的对象，将它们添加到一个网格中，并且根据一定的规则决定 Actor 的行为。

阶段三作为扩展，学习了在 java 二进制流读取 Bitmap 图像，深度优先搜索算法，广度优先搜索算法的运用。

下面就每个阶段的知识做个小小的总结，希望能给后来人做一点微小的贡献~

阶段一

这个阶段主要学习了 java 编程的一些常用工具，具体可参考 *Vi,Java,Ant 和 Junit 的自学报告*（好气啊居然没有备份）

Ant

Ant 是一个基于 Java 的生成工具. 既可以生成 jar 包，也能像 makefile 一样定义生成文件之间的依赖关系；然而，与使用特定于平台的 shell 命令来实现生成过程所不同的是，它使用跨平台的 Java 类。使用 Ant，您能够编写单个生成文件，这个生成文件在任何 Java 平台上都一致地操作（因为 Ant 本身也是使用 Java 语言来实现的）；这就是 Ant 最大的优势。

```
<?xml version="1.0"?>
<project name="targetStudy">
	<target name="targetA" if="ant.java.version">
		<echo message="Java Version: ${ant.java.ve
sion}"/>
	</target>
	<target name="targetB" depends="targetA" unless="a
migo">
		<description>
				a depend example!
		</description>
		<echo message="The base dir is: ${basedir}"
/>
	</target>
</project>
```

Ant 通过 xml 文件（默认是 build.xml）部署架构的，上面的代码截图是 xml 文件的写法，project 元素是 Ant 构件文件的根元素，Ant 构件文件至少应该包含一个 project 元素，否则会发生错误。在每个 project 元素下，可包含多个 target 元素。一般在 target 的标签里面会放我们需要执行的指令，比如上面的 echo 指令就是在终端输出信息。还可以创建，删除、移动、编译文件。利用这个特点，可以用 ant 来生成部署 java 工程

```
<?xml version="1.0"?>
<project name="javacTest"default="compile" basedir=".">
    <target name="clean">
       <delete dir="build"/>
    </target>

    <target name="compile" depends="clean">
       <mkdir dir="build/classes"/>
    <javac srcdir="src" destdir="build/classes"/>
    </target>
</project>
```

运行该 build.xml 文件，可在工程中看到新增了 build/classes 目录，并在该目录中生成了编译后的 HelloWorld.class 文件。

生成 jar 包：

```
<target name="jar" depends="run">
<jar destfile="helloworld.jar" basedir="build/classes">
           <manifest>
              <attribute name="Main-class" value="HelloWo
rld"/>
           </manifest>
      </jar>
</target>
```

Junit 单元测试

Junit 是 java 编程的一个单元测试工具，可以测试函数运行结果，方便调试。一般有两种运行方式，在命令行进行单元测试；在 Eclipse 下进行单元测试。

命令行参考以下例子：

```
----HelloWorld.java------
import java.util.*;

public class HelloWorld {
    String str;
    Public void hello()
    {
        str = "Hello World!";
    }
    Public String getStr()
    {
        Return str;
    }
}
-----HelloWorldTest.java----------
import static org.junit.Assert.*;
import org.junit.Test;

public class HelloWorldTest {
    public HelloWorld helloworld = new HelloWorld();
    @Test
    Public void testHello() {
        helloworld.hello();
        assertEquals("Hello World!", helloworld.getStr());
    }
}
```

使用如下命令运行：

```
@sser>javac –classpath .:junit-4.9.jar HelloWorldTest.java
@sser>java –classpath .:junit-4.9.jar –ea org.junit.runner.JUnitCore HelloWorldTest
```

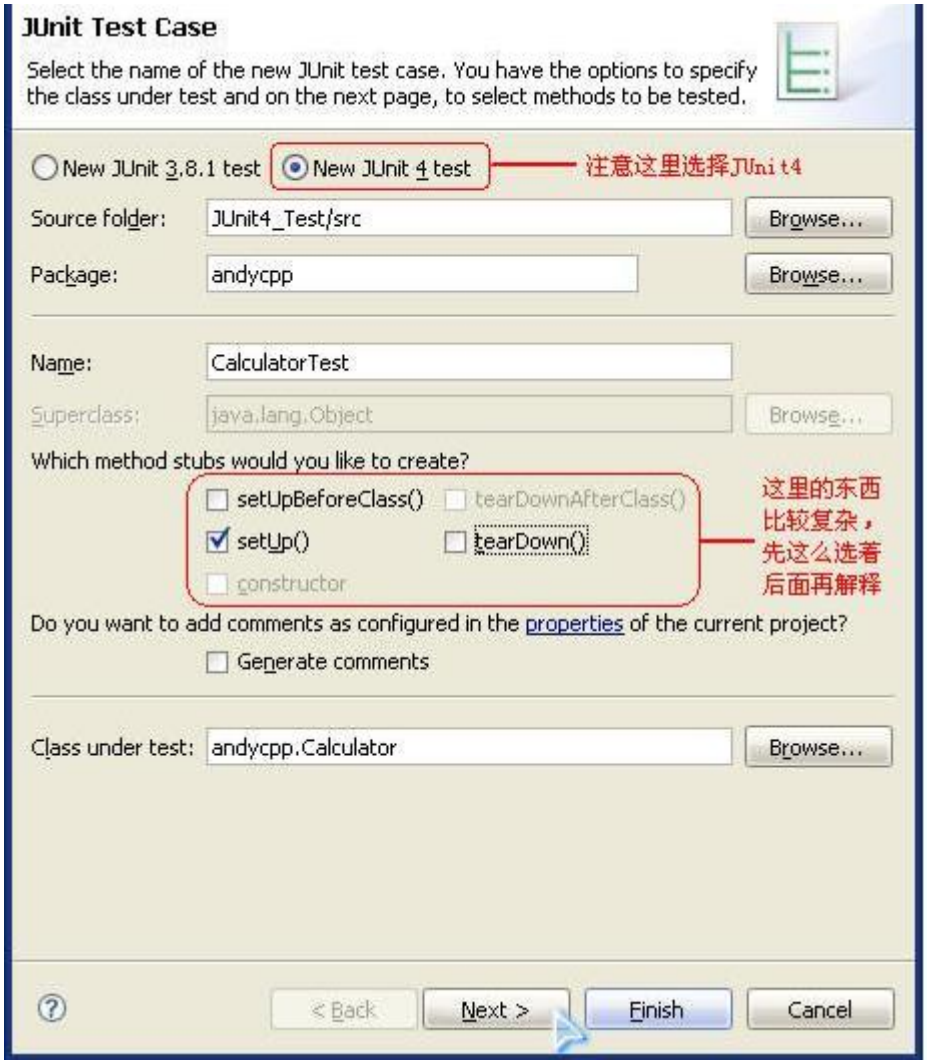
可得到如下输出结果：

```
JUnit version 4.9
.
Time 0.007
OK(1 test)
```

我们可以看到运行正确，这也证明了我们的环境配置正确。

当然了，像我这种手残党是用 Eclipse 编程的，从后面 part2 开始一直用的

Eclipse。接下来讲讲在 Eclipse 下添加单元测试:
一、将 JUnit4 单元测试包引入这个项目：在该项目上点右键，点 属性 properties
二、bulid path 里面，将 Junit 加入到 Library 中
三、回到 Eclipse 界面，在要测试的 java 文件上右击点 new a junit test case.



四、接下来会让你勾选要测试的 methods. 然后就会发现，Eclipse 已经自动生成了 Junit 的模版函数了，往里面加测试代码，OK!

SonarQube 的使用

2.3.1 在 shell 里面用 vi 或者 gedit 打开 /etc/profile `sudo vi /etc/profile`

2.3.2 在文件末端添加你解压缩的 sonar 文件夹的路径，如下图：

```
export SONAR_HOME=/home/greenhill/Documents/sonar-3.7.4/bin/linux-x86-32
export SONAR_RUNNER_HOME=/home/greenhill/Documents/sonar-runner-2.4
export PATH=$SONAR_RUNNER_HOME/bin:$PATH
```

注意：小编选择的系统是 linux 32bit，具体的系统配置大家要看看自己机子。

2.3.3 最后保存退出，在 shell 里面键入 `source /etc/profile`，重启系统。

2.4 添加数据库

如果使用 Sonar 默认的数据库 H2，则无需配置，如果需要使用其他数据库，包括 mysql，Oracle 等，可以自行上网查询。由于我们的是小项目，所以用 Sonar 自带的数据库 H2 完全可以了。

2.5 启动服务

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

唯一需要注意的是，由于云桌面没有 sudo 权限，这里的添加路径改成 gedit ~/.bashrc

已安装 SonarQube Runner 且环境变量已配置，即 sonar-runner 命令可在任意目录下执行

1.在项目源码的根目录下创建 sonar-project.properties 配置文件，内容如下：

src

sonar-project.
properties

```
#required metadata
#projectKey项目的唯一标识，不能重复
sonar.projectKey=10389179
sonar.projectName=10389179
sonar.projectVersion=1.0
sonar.sourceEncoding=UTF-8
sonar.modules=java-module

# Java module
java-module.sonar.projectName=Java Module
java-module.sonar.language=java
# .表示projectBaseDir指定的目录
java-module.sonar.sources=.
java-module.sonar.projectBaseDir=src
```

其中 projectKey 是项目的唯一标识，不能重复；

大家要修改的内容包括 sonar.projectKey，sonar.projectName，

java-module.sonar.projectBaseDir 三项；

阶段二

正式开始 GridWorld 项目的编程，首先是 Eclipse 导入 jar 文件，我自己是花了不少时间才弄清楚它的一些问题，一开始的时候各种找不到文件各种奇怪的编译错误。
（截图）
代码的编写并不难，基本上仿照样例上代码的编写，然后看看 wiki 上对每个 API 的描述，大概就可以了。如果还不会的吧，可以百度 GridWorld 自行参考:）这里不做多余的描述。

阶段三

• 图像读取
这部分任务是要求用 java 实现一个利用二进制流读取 Bitmap 图像，并且能够进行简单地处理和保存的软件。这里要求图像的读写不能用 java 自带的 API。所以只能根据位图文件的每个部分，进行信息读取，并赋值。

2.1 位图头

这部分是识别信息，典型的应用程序会首先普通读取这部分数据以确保的确是位图文件并且没有损坏。

字节 #0-1　保存位图文件的标识符，这两个字节的典型数据是BM。
字节 #2-5　使用一个dword保存位图文件大小。
字节 #6-9　是保留部分，留做以后的扩展使用,对实际的解码格式没有影响。
字节 #10-13　保存位图数据位置的地址偏移，也就是起始地址。

2.2 位图信息

这部分告诉应用程序图像的详细信息，在屏幕上显示图像将会使用这些信息，它从文件的第15个字节开

字节 #14-17　定义以下用来描述影像的区块（BitmapInfoHeader）的大小。它的值是：40 - Windows 3
字节 #18-21　保存位图宽度（以像素个数表示）。
字节 #22-25　保存位图高度（以像素个数表示）。
字节 #26-27　保存所用彩色位面的个数。不经常使用。
字节 #28-29　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）。
字节 #30-33　定义所用的压缩算法。允许的值是0、1、2、3、4、5。
0 - 没有压缩（也用BI_RGB表示）
1 - 行程长度编码 8位/像素（也用BI_RLE8表示）
2 - 行程长度编码4位/像素（也用BI_RLE4表示）
3 - Bit field（也用BI_BITFIELDS表示）
4 - JPEG图像（也用BI_JPEG表示）
5 - PNG图像（也用BI_PNG表示）
然而，由于大多数位图文件都是不压缩的，所以最常用的值是0。
字节 #34-37　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。
字节 #38-41　保存图像水平方向分辨率。
字节 #42-45　保存图像竖值方向分辨率。
字节 #46-49　保存所用颜色数目。
字节 #50-53　保存所用重要颜色数目。当每个颜色都重要时这个值与颜色数目相等。

ImplementImageIO.java

```
import java.awt.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;

import imagereader.*;

public class ImplementImageIO implements IImageIO {
    public Image myRead(String filePath){
        Image img;

        try{
            FileInputStream fis = new FileInputStream(filePath);
            //位图头
            byte bmpHead[] = new byte[14];
            fis.read(bmpHead,0,14);
            //位图信息
            byte bmpInfo[] = new byte[40];
            fis.read(bmpInfo,0,40);

            //位图宽度，进制转换对齐
            //字节18-21表示宽度
            int width = (( (int)bmpInfo[7]&0xff) << 24)| (((int)bmpInfo[6]&0xff) << 16)|
(((int)bmpInfo[5]&0xff) << 8)| (((int)bmpInfo[4]&0xff));
            //位图高度
            //字节22-25表示高度
            int height = (( (int)bmpInfo[11]&0xff) << 24)| (((int)bmpInfo[10]&0xff) << 16)|
                    (((int)bmpInfo[9]&0xff) << 8)| (((int)bmpInfo[8]&0xff));
            //位图大小
            //字节34-37表示图像大小
            int bitsize = (( (int)bmpInfo[23]&0xff) << 24)| (((int)bmpInfo[22]&0xff) << 16)|
                    (((int)bmpInfo[21]&0xff) << 8)| (((int)bmpInfo[20]&0xff));
            //字节28-29表示图像的颜色深度，1、4、8灰阶，24彩阶
            int bitcount = ((int)bmpInfo[15]&0xff) << 8|(int)bmpInfo[14] & 0xff;
```

```
 //如果是彩色
 if(bitcount == 24){
     //如果像素用字节不是4的倍数，会有空白填充
     int npad = ((bitsize / height) - width * 3);
     //填充4个空白相当于不用填充
     if(npad == 4){
         npad = 0;
     }
     int data[] = new int[height * width];
     //存放像素数据的数组
     byte RGB[] = new byte[bitsize];

     fis.read(RGB, 0, bitsize);


     int index = 0;
     for(int i = 0; i < height;i++){
         for(int j = 0; j < width;j++){

             int t = width * (height - i - 1) + j;
             //255表示透明度
             data[t] = (255&0xff)<< 24|
             (((int)RGB[index + 2] & 0xff) << 16)
             |(((int)RGB[index + 1] & 0xff) << 8)|
             (((int)RGB[index] & 0xff));
             //三色
             index += 3;
         }
         index += npad;
     }
```

```
                //使用RGB ColorModel的构造方法（w,h,pix,off,scan）
                //分别表示宽度，高度，像素数组，存储首个像素的便宜量，一行像素到下一行像素的距离
                img = Toolkit.getDefaultToolkit().createImage(new MemoryImageSource(
                        width,height,data,0,width));
            }
            else{
                img = (Image)null;
            }
            fis.close();
            return img;
        }catch(Exception e){
            e.printStackTrace();

        }

        return (Image)null;
    }
```

至于写的话可以用 java 的 API，这个就比较方便了

```
//使用java的API
public Image myWrite(Image image, String filePath){
    try{
        File imgfile = new File(filePath + ".jpg");
        //长，宽，图像类型
        BufferedImage bi = new BufferedImage(image.getWidth(null),image.getHeight(null),
                BufferedImage.TYPE_INT_RGB);

        Graphics2D g2 = bi.createGraphics();
        //图，横纵向坐标,转换类型
        g2.drawImage(image,0,0,null);
        //g2.dispose();
        ImageIO.write(bi, "jpg", imgfile);
        return image;
    }
    catch(Exception e){
        e.printStackTrace();
    }
    return image;
}
```

Junit 测试文件：

```java
public class ImplementImageIOTest {

    @Before
    public void setUp() throws Exception {
    }
    //测试是否能成功读取文件
    @Test
    public void testMyRead() throws IOException {
        FileInputStream in = new FileInputStream("/home/administrator/Desktop/"
                + "grid world/bmptest/1.bmp");
        BufferedImage goal = ImageIO.read(in);
        ImplementImageIO iio = new ImplementImageIO();
        Image image = iio.myRead("/home/administrator/Desktop/"
                + "grid world/bmptest/1.bmp");
        int width = image.getWidth(null);
        int height = image.getHeight(null);
        BufferedImage test = new BufferedImage(width,height,BufferedImage.TYPE_INT_BGR);
        test.getGraphics().drawImage(image, 0, 0, width, height, null);

        //读取图片的大小是否一致
        assertEquals(goal.getWidth(null),image.getWidth(null));
        assertEquals(goal.getHeight(null),image.getHeight(null));
        //读取图片的颜色RGB是否一致
        for (int i = 0; i < goal.getHeight(null);i++){
            for(int j = 0; j < goal.getWidth();j++){
                assertEquals(goal.getRGB(i, j),test.getRGB(i, j));
            }
        }

    }
```

ImplementImageProcessor.java

对图片进行简单的颜色处理，提取色彩通道就是把一个彩色图像的每个像素的 RGB 值分别提取出来。例如，提取 R 通道，则把原来图像每个像素的 RGB 的 R 值提取出来，并形成一个新的 RGB 值（R，0，0）。这样就能以图像的形式把一个彩色图像各个色彩通道显示出来了。
至于灰色图，wiki 上面的建议是：

2.2 彩色图像转换成灰度图像

将彩色图转换成灰度图，建议采用NTSC推荐的彩色图到灰度图的转换公式：

I = 0.299 * R + 0.587 * G + 0.114 * B，其中R,G,B分别为红、绿、蓝通道的颜色值。

然后将三个色彩通道的颜色值改为这个值即可。这样，原来的彩色图像就变成了灰度图像了。

代码如下：

```
    private Toolkit kit = Toolkit.getDefaultToolkit();
    //红色通道
    class RedSwapFilter extends RGBImageFilter {
        public RedSwapFilter() {
            canFilterIndexColorModel = true;
        }
        //像素单元由3个字节存储三色，一个字节存储透明度
        //顺序为透明度-红-绿-蓝
        //可以用颜色过滤器与色素相与得到对应的通道，过
        //红色 0xffff0000
        //绿色 0xff00ff00
        //蓝色 0xff0000ff
        public int filterRGB(int x,int y,int rgb){
            return (rgb & 0xffff0000);
        }

    }
    //绿色通道
    class GreenSwapFilter extends RGBImageFilter {
        public GreenSwapFilter() {
            canFilterIndexColorModel = true;
        }
        public int filterRGB(int x,int y,int rgb){
            return(rgb & 0xff00ff00);
        }
    }
```

```
//蓝色通道
class BlueSwapFilter extends RGBImageFilter {
    public BlueSwapFilter(){
        canFilterIndexColorModel = true;
    }
    public int filterRGB(int x,int y, int rgb){
        return(rgb & 0xff0000ff);
    }
}
//灰色通道
class GraySwapFilter extends RGBImageFilter {
    public GraySwapFilter(){
        canFilterIndexColorModel = true;
    }
    // I = 0.299 * R + 0.587 * G + 0.114 * B
    public int filterRGB(int x,int y, int rgb){
        int red = (rgb & 0x00ff0000) >> 16;
        int green = (rgb & 0x0000ff00) >> 8;
        int blue = (rgb & 0x000000ff);
        int gray = (int)((double)(0.299 * red) + (double)(0.587 * green) + (double)(0.114 * blue));
        return (rgb & 0xff000000) + (gray << 16) + (gray << 8) + gray;
    }
}
```

接下来只要调用 SwapFilter 就能显示各个色道的颜色了：

```
//显示红色通道
public Image showChanelR(Image sourceImage){
    RedSwapFilter red = new RedSwapFilter();
    return kit.createImage(new FilteredImageSource(sourceImage.getSource(),red));

}

//显示绿色通道
public Image showChanelG(Image sourceImage){
    GreenSwapFilter green = new GreenSwapFilter();
    return kit.createImage(new FilteredImageSource(sourceImage.getSource(),green));
}
//显示蓝色通道
public Image showChanelB(Image sourceImage){
    BlueSwapFilter blue = new BlueSwapFilter();
    return kit.createImage(new FilteredImageSource(sourceImage.getSource(),blue));
}
//显示灰色通道
public Image showGray(Image sourceImage){
    GraySwapFilter gray = new GraySwapFilter();
    return kit.createImage(new FilteredImageSource(sourceImage.getSource(),gray));
}
```

MazeBug. java

这次的任务是通过深度优先算法，让虫子 bug 走出迷宫。深度优先搜索算法（Depth-First-Search),是搜索算法的一种。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 的所有边都已被探寻过，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。

算法步骤如下：

（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。

1）和 2）的代码不用我们自己写，从 3）开始分析对应的关键代码：

3. 取起始节点

```
//将起始节点入栈
if (stepCount == 0) {
    ArrayList<Location> temp = new ArrayList<Location>();
    //进入节点的方向
    temp.add(getLocation());
    crossLocation.push(temp);
}
```

4. 栈非空时：

1）取当前栈顶节点

```
ArrayList<Location> valid = new ArrayList<Location>();
//取当前栈顶节点
ArrayList<Location> temp = crossLocation.peek();
```

2）如果栈顶节点是结束节点，输出并结束搜索。这里的终点用了一块红色的石头代替

```
//如果栈顶节点是结束节点（红色石头），结束搜索
if (a instanceof Rock && a.getColor().getRed() == Color.RED.getRed()) {
    isEnd = true;
    valid.add(loc1);
    moveTo(loc1);
}
```

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

```
Location loc1 = loc.getAdjacentLocation(i * Location.RIGHT);
//如果栈顶节点存在未访问的邻接节点，置为可访问的方向，入栈（加入方向树）
if ((temp.contains(loc1)== false) && gr.isValid(loc1)) {

    Actor a = gr.get(loc1);
    if ((a == null) || ( a instanceof Flower)) {
        valid.add(loc1);
    }
```

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，继续循环。

由于往回走也算在步数 count 里面，所以我的实现方法是仿照 move()写一个往回走的函数 back()，在 back 里面进行 pop()操作。

```
} else if (willMove) {
    move();
    //increase step count when move
    stepCount++;
}
else{
    back();
    stepCount++;
}
```

```java
public void back(){
    Grid<Actor> gr = getGrid();
    if (gr == null)
        return;
    crossLocation.pop();
    //取上一次方向树，取节点设为next
    ArrayList<Location> temp = crossLocation.peek();
    next = (Location) temp.get(0);
    Location loc = getLocation();
```

N-Puzzle

拼图游戏：

> （实验一）广度优先搜索算法，求解指定3*3拼图（8-数码问题）的最优解。
> 要求：填充此函数
>
> （Demo+实验二）启发式搜索。 演示：RunnerDemo使用此函数
> 要求：修改此函数
>
> （Demo+实验二）计算并修改节点的代价估值estimatedValue 演示：RunnerDemo使用此函数
> 要求：修改此函数

作为收尾的一次任务，相比前两个比较简单。广度优先搜索我这里是直接借用了启发式搜索的代码，然后把估值的因素去掉，变成盲排。然后估值函数estimatedValue参考wiki上面的方法：

1） 所有 放错位的数码 个数
2） 所有 放错位的数码与其正确位置的距离 之和
3） 后续节点不正确的数码个数
4） .....

可以同时使用多个估价方法，f(n) = a*f1(n) + b*f2(n) ...通过适当调整权重（a、b、...），能够加快搜索速度。