中山大學 SUN YAT-SEN UNIVERSITY

实训报告

作业名称	Part 2: Bug Variations					
院系	软件学院		班级	教务 2 班	姓名	李明宽
学号	11331173		完成日期	2013 年 07 月 03 日		
QQ	736459905		E-mail	limkuan@mail2.sysu.edu.cn		

Methods of the Bug Class

The Bug class provides three methods that specify how bugs move and turn.

public boolean canMove()

tests whether the bug can move forward into a location that is empty or contains a flower

public void move()

moves the bug forward, putting a flower into the location it previously occupied

public void turn()

turns the bug 45 degrees to the right without changing its location

Extending the Bug Class

Do You Know?

The source code for the BoxBug class can be found in the BoxBug directory。.

1. What is the role of the instance variable sideLength?
   Answer: The variable sideLength is defined as the side length of the square which the BoxBug draws. In other means, when the BoxBug run the sideLength step, it will turn.

2. What is the role of the instance variable steps?
   Answer: The variable steps is defined as a counter to record how many steps did a BoxBug have run in each side, if the step is equal to the sideLength, the bug will turn.

3. Why the turn method is called twice when steps become equal to sideLength?
   Answer: The method turn of class Bug can let the bug turn 45 degrees clockwise, but we need to draw a square, so each turn must be 90 degrees. This is the reason why we need call turn method twice when steps become equal to sideLength.

4. Why can the move method be called in the BoxBug class when there is no move method in the BoxBug code?

   Answer: Because class Bug is the superclass of the class BoxBug, and class BoxBug is inherited the class Bug, so if there is no move method in BoxBug code, we can use move method which is defined in the class Bug.

5. After a BoxBug is constructed, will the size of its square pattern always be the same? Why or why not?

   Answer: The size of its square pattern always be the same, because we can't change the variable sideLength after the Bug has been create.

6. Can the path a BoxBug travels ever change? Why or why not?

   Answer: The path of a BoxBug travels can be changed, because if the bug meets an actor like rock or another bug, it will turn to another direction and travel a different path.

7. When will the value of steps be zero?

   Answer: It has several conditions when the value of steps is zero.

   - ✓ First, when the bug is just created, the value of steps is initialized to zero.
   - ✓ Second, when the value of steps is equals to the value of sideLength, it means that the bug is finish a side of the square and the value of steps set to zero.
   - ✓ Third, when the bug meets a rock or another bug, or it meets the edge of the grid, it can't move and must turn to another direction, in this condition, the value of steps is also set to zero.

Runner Classes

In order to observe the behavior of one or more actors, a "runner" class is required. That class constructs an ActorWorld object, places actors into it, and shows the world. For the bug, this class is BugRunner. For the boxbug, it is BoxBugRunner. In each of these runner classes, the overloaded add method is used to place actors (instances of classes such as Bug, BoxBug, Rock) into the grid of the ActorWorld. The add method with an Actor parameter and a Location parameter places an actor at a specified location. The add method with an Actor parameter but no Location parameter places an actor at a random empty location. When you write your own classes that extend Bug, you also need to create a similar runner class.

Exercises

In the following exercises, write a new class that extends the Bug class. Override the act method to define the new behavior.

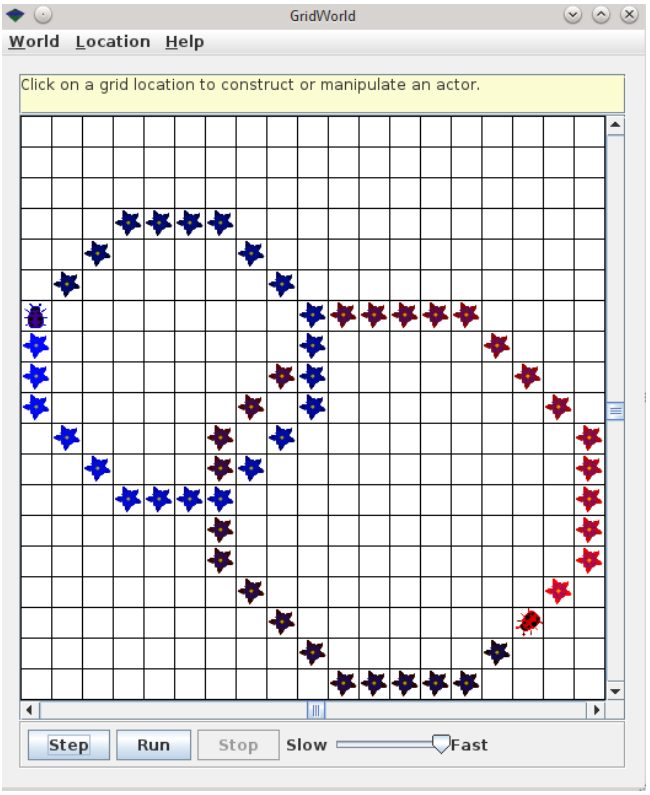
1. Write a class CircleBug that is identical to BoxBug, except that in the act method the turn method is called once instead of twice. How is its behavior different from a BoxBug?

   Answer: Just when the bug finish a side, turn only once, and turn clockwise 45 degrees.

   The code of CircleBug is below:

```
import info.gridworld.actor.*;

public class CircleBug extends Bug{
    private int steps;
    private int sideLength;

    public CircleBug(int length){
        steps = 0;
        sideLength = length;
    }

    public void act(){
        if(steps < sideLength && canMove()){
            move();
            steps++;
        }

        else {
            turn();
            steps = 0;
        }
    }
}
```

   And this is the result after CircleBug run in the GridWorld:



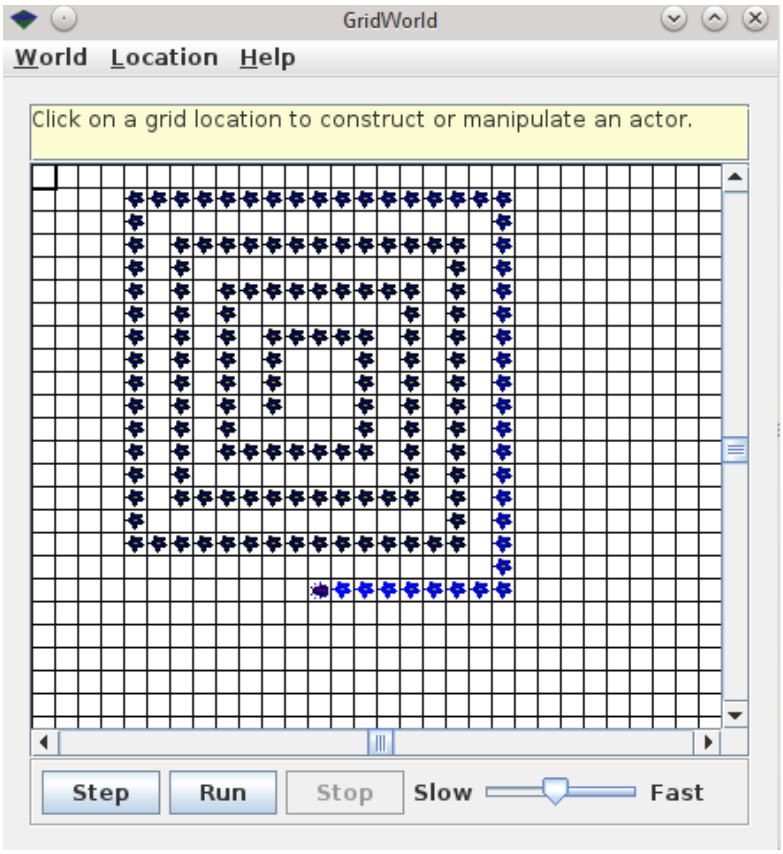
2. Write a class SpiralBug that drops flowers in a spiral pattern. Hint: Imitate BoxBug, but adjust the side length when the bug turns. You may want to change the world to an UnboundedGrid to see the spiral pattern more clearly.

Answer: When the Bug is finished a side, add one to sideLength, in this way, we will get a spiral pattern.

The code of SpiralBug is below:

```
import info.gridworld.actor.Bug;

public class SpiralBug extends Bug {
    private int steps;
    private int sideLength;

    public SpiralBug(int length) {
        steps = 0;
        sideLength = length;
    }

    public void act() {
        if (steps < sideLength && canMove()) {
            move();
            steps++;
        } else {
            turn();
            turn();
            sideLength++;
            steps = 0;
        }
    }
}
```

And this is the result after SpiralBug run in the GridWorld:



3. Write a class ZBug to implement bugs that move in a "Z" pattern, starting in the top left corner. After completing one "Z" pattern, a ZBug should stop moving. In any step, if a ZBug can't move and is still attempting to complete its "Z" pattern, the ZBug does not move and should not turn to start a new side. Supply the length of the "Z" as a parameter in the constructor. The following image shows a "Z" pattern of length 4. Hint: Notice that a ZBug needs to be facing east before beginning its "Z" pattern.

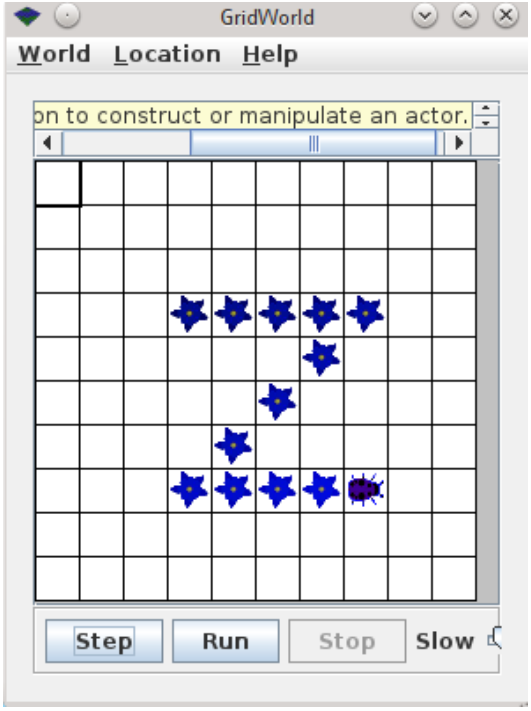
Answer: First, initialize the bug to the direction east, and have a counter, if the counter is greater than or equal 3, we don't call method move or turn in the function act, but if the counter is less than 3, let the bug draw a "Z". If the bug meets an actor, it will turn to east and initialize the counter, try to finish a new "Z". And when a "Z" has been finished, the bug will stop.

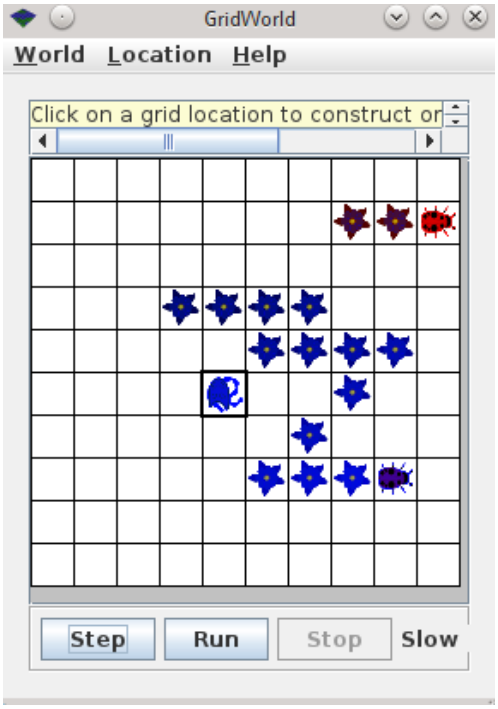
The code of ZBug is below:

```
import info.gridworld.actor.Bug;
import info.gridworld.grid.Location;

public class ZBug extends Bug {
    private int steps;
    private int turns;
    private int sideLength;

    public ZBug(int length) {
        steps = 0;
        turns = 0;
        sideLength = length;
        setDirection(Location.EAST);
    }

    public void act() {
        if (turns < 3 && steps < sideLength) {
            if (canMove()) {
                steps++;
                move();
            } else {
                setDirection(Location.EAST);
                turns = 0;
                steps = 0;
            }
        } else if (turns == 0) {
            setDirection(Location.SOUTHWEST);
            steps = 0;
            turns++;
        } else if (turns == 1) {
            setDirection(Location.EAST);
            steps = 0;
            turns++;
        } else if (turns == 2) {
            steps = 0;
            turns++;
        }
    }
}
```

And this is the result after ZBug run in the GridWorld:

If the bug can move all the time (left), if the bug faces edge or meets an actor (right):





4. Write a class DancingBug that "dances" by making different turns before each move. The DancingBug constructor has an integer array as parameter. The integer entries in the array represent how many times the bug turns before it moves. For example, an array entry of 5 represents a turn of 225 degrees (recall one turn is 45 degrees). When a dancing bug acts, it should turn the number of times given by the current array entry, then act like a Bug. In the next move, it should use the next entry in the array. After carrying out the last turn in the array, it should start again with the initial array value so that the dancing bug continually repeats the same turning pattern.

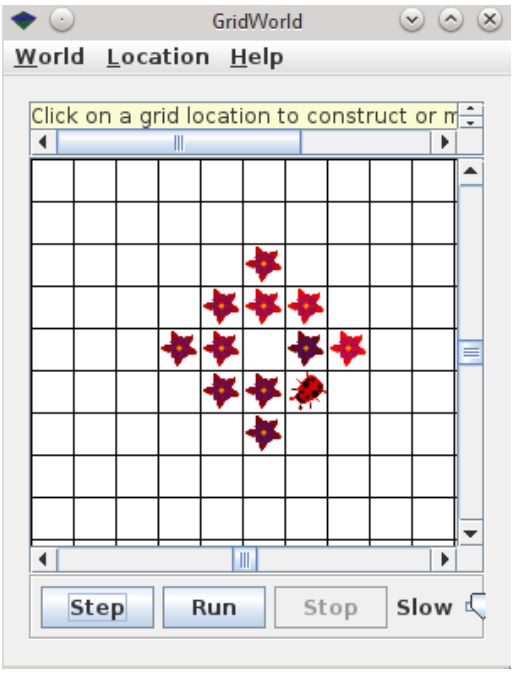
   The DancingBugRunner class should create an array and pass it as aparameter to the DancingBug constructor.

Answer: First, I use a string being the parameter of DancingBug, it contains each entry of the array and spilt by an ",", and the bug will turn n times before move, which n represents an entry of the array.

The code of DancingBug is below:

```
import info.gridworld.actor.Bug;

public class DancingBug extends Bug {
    private int[] turns;
    private int steps;

    public DancingBug(String array) {

        // change the string to integer array
        String[] strings = array.split(",");
        turns = new int[strings.length];

        for (int i = 0; i < strings.length; i++) {
            turns[i] = Integer.parseInt(strings[i]);
        }
        steps = 0;
    }

    public void act() {
        if (steps == turns.length) {
            steps = 0;
        }
        for (int i = 0; i < turns[steps]; i++)
            turn();
        steps++;
        if (canMove()) {
            move();
        } else {
            turn();
        }
    }
}
```

And this is the result after DancingBug run in the GridWorld (the array is [1, 2, 3, 4]):



5. Study the code for the BoxBugRunner class. Summarize the steps you would use to add another BoxBug actor to the grid.

   Answer: The steps to add a BoxBug to grid is below:

   First, if you don't have a world, new an ActorWorld:

   ```
   ActorWorld world = new ActorWorld();
   ```

   Second, new a BoxBug with the parameter of sideLength, and change color if necessary:

   ```
   BoxBug newBoxBug = new BoxBug(6);
   ```

   Third, new a location if you want to put the bug in the grid clearly:

   ```
   Location newLocation = new Location(7, 8);
   ```

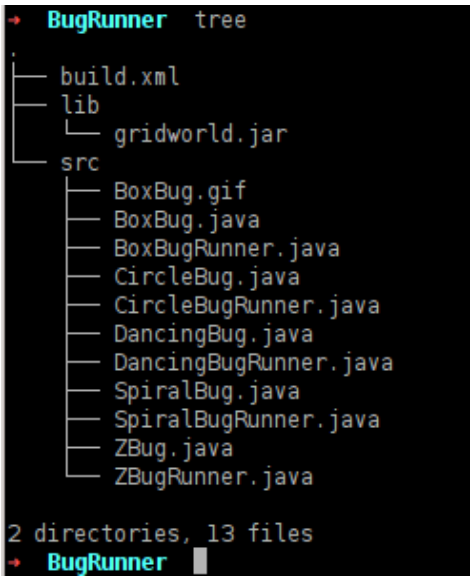
   Finally, add the bug to the world:

   ```
   world.add(newLocation, newBoxBug);
   ```

   In this way, we can add an Bug to the world.

Ant Build File

And the file tree of this project is below:



I use Ant Build File to build the project, we can run "ant run${projectName}" in termimal to build and run the proper project. The build.xml is below:

```
<?xml version="1.0"?>
<project name="BugRunner" default="compile" basedir=".">

  <path id="project.class.path">
    <fileset dir="${basedir}/lib">
      <include name="*.jar"/>
    </fileset>
    <pathelement location="${basedir}/bin" />
    <pathelement path="${java.class.path}" />
  </path>

  <target name="clean">
    <delete dir="bin"/>
    <delete dir="dist"/>
  </target>
```

```
  <target name="compile" depends="clean">
    <mkdir dir="${basedir}/bin"/>
    <javac srcdir="src" destdir="bin"
          fork="true">
      <classpath refid="project.class.path"/>
    </javac>
  </target>

  <target name="BoxBugJar" depends="clean, compile">
    <mkdir dir="dist" />
    <jar jarfile="dist/BoxBug.jar"
        basedir="bin">
      <manifest>
        <attribute name="Main-Class" value="BoxBugRunner"/>
        <attribute name="Class-Path"
value="${basedir}/lib/gridworld.jar"/>
      </manifest>
      <zipfileset excludes="META-INF/*.SF"
src="${basedir}/lib/gridworld.jar"/>
    </jar>
  </target>

  <target name="runBoxBug" depends="BoxBugJar">
    <java jar="${basedir}/dist/BoxBug.jar"
         fork="true">
    </java>
  </target>

  <target name="CircleBugJar" depends="clean, compile">
    <mkdir dir="dist" />
    <jar jarfile="dist/CircleBug.jar"
        basedir="bin">
      <manifest>
        <attribute name="Main-Class" value="CircleBugRunner"/>
        <attribute name="Class-Path"
value="${basedir}/lib/gridworld.jar"/>
      </manifest>
      <zipfileset excludes="META-INF/*.SF"
src="${basedir}/lib/gridworld.jar"/>
    </jar>
  </target>

  <target name="runCircleBug" depends="CircleBugJar">
    <java jar="${basedir}/dist/CircleBug.jar"
         fork="true">
    </java>
  </target>

  <target name="DancingBugJar" depends="clean, compile">
    <mkdir dir="dist" />
    <jar jarfile="dist/DancingBug.jar"
        basedir="bin">
      <manifest>
        <attribute name="Main-Class" value="DancingBugRunner"/>
        <attribute name="Class-Path"
value="${basedir}/lib/gridworld.jar"/>
      </manifest>
      <zipfileset excludes="META-INF/*.SF"
src="${basedir}/lib/gridworld.jar"/>
    </jar>
  </target>
```

```
  <target name="runDancingBug" depends="DancingBugJar">
    <java jar="${basedir}/dist/DancingBug.jar"
          fork="true">
    </java>
  </target>

    <target name="SpiralBugJar" depends="clean, compile">
    <mkdir dir="dist" />
    <jar jarfile="dist/SpiralBug.jar"
        basedir="bin">
      <manifest>
        <attribute name="Main-Class" value="SpiralBugRunner"/>
        <attribute name="Class-Path"
value="${basedir}/lib/gridworld.jar"/>
      </manifest>
      <zipfileset excludes="META-INF/*.SF"
src="${basedir}/lib/gridworld.jar"/>
    </jar>
  </target>

  <target name="runSpiralBug" depends="SpiralBugJar">
    <java jar="${basedir}/dist/SpiralBug.jar"
          fork="true">
    </java>
  </target>

    <target name="ZBugJar" depends="clean, compile">
    <mkdir dir="dist" />
    <jar jarfile="dist/ZBug.jar"
        basedir="bin">
      <manifest>
        <attribute name="Main-Class" value="ZBugRunner"/>
        <attribute name="Class-Path"
value="${basedir}/lib/gridworld.jar"/>
      </manifest>
      <zipfileset excludes="META-INF/*.SF"
src="${basedir}/lib/gridworld.jar"/>
    </jar>
  </target>

  <target name="runZBug" depends="ZBugJar">
    <java jar="${basedir}/dist/ZBug.jar"
          fork="true">
    </java>
  </target>
</project>
```