Blue Pelican GridWorld

Teacher Manual

AP Computer Science
Case Study

Table of contents

Chapter 2--*BoxBug* & *SpiralBug*

Modifying the methods of *Bug*

The *Bug* class is a very fundamental part of GridWorld. It should **not be modified**; rather, a new class is created **extending** the *Bug* class, and modifications are made in it by overriding the methods in the *Bug* superclass. One method that is very commonly overridden is the *act( )* method.

Cleaning up our *act( )*

Recall from the last chapter (Getting Started), the *Step* button on the graphical interface to GridWorld. Each time it is clicked (and also on each iteration of *Run*), the *act* method of each object in the *Grid* is called. Below is the source code for the *act* method of the *Bug* class:

```
public void act( )
{
        if( canMove( ) )
                move( );
        else
                turn( );
}
```

Notice how very simple this method is. It, in turn, uses three other methods of the *Bug* class:

- *canMove( )* … returns a *boolean* telling if it's safe to move in the direction set for this object.
- *move( )* …move one space to the nearest of this object's direction to horizontal, vertical, or at a 45 degree diagonal.
- *turn( )* …sets a new direction of 45 degrees clockwise from the current direction.

Notice that this code explains why when a *Bug* wants to move into the position of a *Rock*, another *Bug*, or is trying to move off the grid, it turns, instead. Also notice that with just a few changes, this is very fertile ground for **modifying the behavior** of the *Bug*.

BoxBug

The *Bug* class will now be extended to produce the *BoxBug* class. As its name suggests, *BoxBug* will travel in the shape of a box (square). The *BoxBug* will move along in its initial direction for a distance specified by the state variable (instance field) *sideLength*. It will then turn 90 clockwise and continue doing this unless it encounters an obstacle in which case it also turns 90 degrees clockwise and begins a new box.

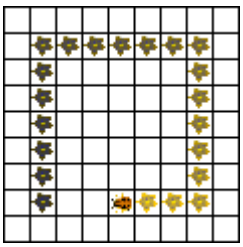

Fig 2-1. When testing the *BoxBug* class, the graphics should produce something like this for each *BoxBug* object on the Grid.

It has already been suggested that we will have an integer state variable called *sideLength* that determines the lengths of the sides of the square traced out by *BoxBug*. A good feature for this new class to have would be for its constructor to initialize *sideLength* as follows:

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

Notice that there is now evidence of a second state variable, *int steps*. For the sake of knowing when to turn 90 degrees, this variable keeps a tally of how many steps through which the *BoxBug* has progressed. Also, notice that this constructor specifies how *BoxBug* objects should be created:

```
BoxBug myBoxBug = new BoxBug( len );  //int len specifies side length
```

So far, the new *BoxBug* class appears as follows (notice ***extends Bug***):

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

public class BoxBug extends Bug
{
        //state variables
        private int sideLength;
        private int steps;

        //constructor
        public BoxBug(int length)
        {
                sideLength = length;
                steps = 0;
        }

        //…more code to come…
 }
```

Finally, and most important of all, a modified *act* method must be provided that overrides the *act* method of the *Bug* superclass. The requirements are that it keeps up with how far the *BoxBug* has moved and then turns it 90 degrees clockwise.

Project… *BoxBug*

As a project, complete the *BoxBug* class by providing code for the *act* method so that the behavior of *BoxBug* is as described: after turning 90 degrees be sure to reset *steps* to 0 so the count can start over. To test this class, see the next section titled, **Testing with a new *Runner* class.**

Testing with a new *Runner* class

(This discussion applies to testing a *BoxBug* class. A *Runner* class could be similarly created for any other modified type of *Bug*.)

Now that a *BoxBug* class has been created, how is it to be tested? First, create a new project: call it *BoxBug* and create the *BoxBug* class within it. The actual visual testing must be done with a *BoxBugRunner* class. This is **not an AP tested class**, but is necessary for the testing of *BoxBug* and to see it perform. Enter a second class into the project called *BoxBugRunner* as follows:

```
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;
import java.awt.Color;

public class BoxBugRunner
{
	public static void main( String args[] )
	{
		ActorWorld world = new ActorWorld( );
		BoxBug bug1 = new BoxBug(6);  //side of box = 6
		bug1.setColor(Color.ORANGE);

		BoxBug bug2 = new BoxBug(3);  //side of box = 3
		bug2.setColor(Color.GREEN);

		world.add (new Location(7, 8), bug1 );
		world.add (new Location(7, 5), bug2 );
		world.show( );
	}
}
```

Again this code is **not part of the AP test**. This is just a class we need to provide in order to test our *BoxBug* class with a graphical interface. One thing is; however, of importance if we wish to create other extensions of the *Bug* class. If for example, a spiral bug is created with a *SpiralBug* class, then the following two lines of code would replace the corresponding two lines in the *BoxBugRunner* class:

```
SpiralBug bug1 = new SpiralBug(6);
SpiralBug bug2 = new SpiralBug(6);
```

This new class could be called the *SpiralBugRunner* class.

It should be noted that this runner class (either *BoxBugRunner* or *SpiralBugRunner*) will not compile unless the class (*BoxBug* or *SpiralBug*), upon which it is dependent, has already been compiled.

Project… *SpiralBug*

As a project, create a *SpiralBug* class by providing code for the *act* method so that it moves in a spiral. A key feature is to use most of the *BoxBug* class and increase the value of *sideLength* at the end of each turn. To test this class, see the previous section titled, **Testing with a new *Runner* class.** When testing, set an unbounded grid.

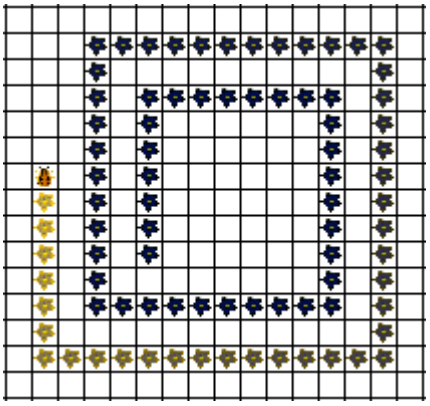

Fig 2-1. When testing the *SpiralBug* class, the graphics should produce something like this for each *SpiralBug* object on the grid

Project Key… *BoxBug*

The complete class for *BoxBug*:

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

public class BoxBug extends Bug
{
	//state variables
	private int sideLength;
	private int steps;

	//constructor
	public BoxBug(int length)
	{
		sideLength = length;
		steps = 0;
	}

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

The official code for this class from the College Board is in Appendix D. The code for the superclass, *Bug*, is also given in Appendix D.

Project Key… *SpiralBug*

The complete class for *SpiralBug*:

```
import info.gridworld.actor.Bug;
public class SpiralBug extends Bug
{
	//state variables
	private int sideLength;
	private int steps;
```

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

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

The complete class for *SpiralBugRunner*:

```
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;
import java.awt.Color;
public class SpiralBugRunner
{
        public static void main( String args[] )
        {
                ActorWorld world = new ActorWorld( );
                SpiralBug bug1 = new SpiralBug(6);  //side of box = 6
                bug1.setColor(Color.ORANGE);

                world.add (new Location(7, 8), bug1 );
                world.show( );
        }
}
```