Part 2

1. What is the role of the instance variable sideLength?

A: The sideLength decides the max number of steps that the boxbug can move in a direction.

2. What is the role of the instance variable steps?

A: It calculates the number of steps that the boxbug has run in a same direction , and decides whether the boxbug need to turn by being compared with the sideLength.

3. Why is the turn method called twice when steps becomes equal to sideLength?

A: Because the boxbug turns 45 degrees everytime and it need to turn 90 degrees for next side of a box.

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

A: Because the BoxBug class extends the Bug class, it inherits the Bug class and it can call the method that implemented in the Bug class.

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

A: Yes. Because we cannot change the sideLength once it is determined in the construction of the BoxBug class.

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

A: Yes. If there is a Rock or another Bug in the way of the BoxBug, it will change its direction and starts a new path.

7. When will the value of steps be zero?

A: When the BoxBug is constructed or the BoxBug turns 90 degrees.

Exercises

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?

A: The CircleBug turns 45 degrees when its steps equals to its sideLength, and it is like to draw a circle. The code is below:

CircleBug.java:

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

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;
    }
  }
}
```

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.

A: The code is below:

SpiralBugRunner.java:

```
import info.gridworld.actor.ActorWorld;

import info.gridworld.actor.Actor;

import info.gridworld.grid.Location;

import info.gridworld.grid.UnboundedGrid;

import java.awt.Color;
```

```java
public class SpiralBugRunner {

    public static void main(String[] args) {

        UnboundedGrid<Actor> grid = new UnboundedGrid<Actor>();

        ActorWorld world = new ActorWorld(grid);

        SpiralBug bob = new SpiralBug(2);

        world.add(new Location(5, 5), bob);

        world.show();

    }

}
```

SpiralBug.java:

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



public class SpiralBug extends Bug {

    private int steps;

    private int sideLength;



    public SpiralBug(int l) {

        steps = 0;

        sideLength = l;

    }



    public void act() {

        if (steps < sideLength && canMove()) {

            move();

            steps++;

        } else {
            sideLength++;

            turn();

            turn();

            steps = 0;

        }

    }

}
```

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.

A: The code is below:

ZBugRunner.java:

```
import info.gridworld.actor.ActorWorld;
import info.gridworld.actor.Actor;
import info.gridworld.grid.Location;
import info.gridworld.grid.UnboundedGrid;

import java.awt.Color;

public class ZbugRunner {
    public static void main(String[] args) {
        ActorWorld world = new ActorWorld();
        ZBug bob = new ZBug(7);
        world.add(new Location(0, 0), bob);
        world.show();
    }
}
```

Zbug.java:

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

import info.gridworld.grid.Location;


public class ZBug extends Bug {

    private int steps;

    private int sideLength;

    private int turnNum;


    public ZBug(int l) {

        steps = 0;

        turnNum = 0;

        sideLength = l;

        setDirection(Location.RIGHT);

    }


    public void act() {

        if (steps < sideLength && canMove()) {

            move();

            steps++;

        } else {

            if (this.getDirection() == Location.RIGHT && turnNum < 2) {

                this.setDirection(225);

                turnNum++;

                steps = 0;

            } else if (this.getDirection() == 225) {

                this.setDirection(Location.RIGHT);
                turnNum++;

                steps = 0;

            }

        }

    }

}
```

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.

DancingBugRunner.java:

```
import info.gridworld.actor.ActorWorld;

import info.gridworld.actor.Actor;

import info.gridworld.grid.Location;

import info.gridworld.grid.UnboundedGrid;


import java.awt.Color;


public class DancingBugRunner {

    public static void main(String[] args) {

        ActorWorld world = new ActorWorld();

        int[] nums = {2,5,6,3,9,10,2,0, 10};
```

```java
        DancingBug bob = new DancingBug(nums);

        world.add(new Location(5, 5), bob);

        world.show();

    }

}
```

DancingBug.java:

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

import info.gridworld.grid.Location;


public class DancingBug  extends Bug {

    private int[] nums;

    private int current;


    public DancingBug(int[] nums) {

        this.nums = nums.clone();

        current = 0;

    }


    public void act() {

        setDirection(getDirection() + Location.HALF_RIGHT*nums[current]);

        current = current + 1 > nums.length - 1 ? 0 : current+1;

        if (canMove()) {

            move();

        }

    }

}
```

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

A: 1. New a BoxBug object with a param that is its sideLength.

2.Call the method of ActionWorld ---- add(BoxBug) or add(Location, BoxBug). The first method create a random location for BoxBug.