Tasmiur Rabb

APCS – Period 2

Mr. Zamansky

Set 1 – Part 1

1. Does the bug always move to a new location? Explain.

No. A bug cannot move to a new location if the cell in front of its path is blocked by another species such as a rock or an actor, or if the cell does not exist.

2. In which direction does the bug move?

Forwards, in whatever direction it is facing.

3. What does the bug do if it does not move?

It turns 45 degrees clockwise.

4. What does a bug leave behind when it moves?

It leaves behind a flower.

5. What happens when the bug is at an edge of the grid? (Consider whether the bug is facing the edge as well as whether the bug is facing some other direction when answering this question.)

When a bug is facing the edge of the grid and is commanded by the user to act, it will turn 45 degrees clockwise, and turn 45 degrees clockwise again once commanded to act again. If the bug is told to move however, when facing the edge, the bug will disappear from the grid.

6. What happens when a bug has a rock in the location immediately in front of it?

The bug turns clockwise 45 degrees.

7. Does a flower move?

No, flowers are stationary.

8. What behavior does a flower have?

A flower slowly loses its color, degrading to a darker shade, eventually becoming gray.

9. Does a rock move or have any other behavior?

No rocks are stationary and do nothing, other than interrupt the bug’s path.

10. Can more than one actor (bug, flower, rock) be in the same location in the grid at the same time?

No, each cell in the grid houses only one actor.

Exercises- Part 1

1. Test the setDirection method with the following inputs and complete the table, giving the compass direction each input represents.

|  |  |
| --- | --- |
| Degrees | Compass Direction |
| 0 | North |
| 45 | Northeast |
| 90 | East |
| 135 | Southeast |
| 180 | South |
| 225 | Southwest |
| 270 | West |
| 315 | Northwest |
| 360 | North |

2. Move a bug to a different location using the moveTo method. In which directions can you move it? How far can you move it? What happens if you try to move the bug outside the grid?

A bug can be moved to any cell in the grid. The bug will not change direction, and one can move the bug as far as one wishes, given that the inputted cell is not outside the grid parameters. Attempting to move the bug to a location outside the grid results in an IllegalArgumentException.

3. Change the color of a bug, a flower, and a rock. Which method did you use?

setColor

4. Move a rock on top of a bug and move the rock again. What happened to the bug?

The bug disappeared when the rock was moved on it. The bug is no longer there after moving the rock.

Set 2 – Part 2

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

This instance variable determines the number of steps a BoxBug can move on each side of its box.

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

This instance variable monitors the number of steps a BoxBug has taken on the box’s side.

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

The turn method turns the bug only 45 degrees, but when steps becomes equal to sideLength, the BoxBug has reached the corner of the box and must turn the corner 90 degrees, calling the turn method twice.

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

The BoxBug class extends the Bug class, and the Bug class has the public move method.

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

Yes, a BoxBug’s constructor determines the side length of the box and cannot be changed by the user.

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

If another actor is in a BoxBug’s path, its path will be interrupted.

7. When will the value of steps be zero?

When steps equals sideLength, meaning the BoxBug has reached a corner of the box, the steps will be equal to zero, and the bug now must turn.

Exercises – Part 2

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 it its behavior different from a BoxBug?

**public** **class** CircleBug **extends** Bug{

**private** **int** steps;

**private** **int** sideLength;

**public** CircleBug(**int** n){

sideLength = n;

}

**public** **void** act(){

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

move();

steps++;

}

**else**{

turn();

steps = 0;

}

}

}

A CircleBug travels in an octagon.

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 Unbounded Grid to see the spiral pattern more clearly.

**public** **class** SpiralBug **extends** Bug{

**private** **int** sideLength;

**private** **int** steps;

**public** SpiralBug(**int** n){

sideLength = n;

steps = 0;

}

**public** **void** act(){

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

move();

steps++;

}

**else**{

turn();

turn();

steps = 0;

sideLength++;

}

}

}

3. Write a class ZBug to implement bugs that move in a “Z” pattern, starting in the top left corner. After completing one “Z” patterh, 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.

**public** **class** ZBug **extends** Bug{

**private** **int** steps;

**private** **int** zLength;

**private** **int** zSection;

**public** ZBug(**int** length){

setDirection(90);

steps = 0;

zSection = 1;

zLength = length;

}

**public** **void** act(){

**if** ((zSection <= 3) && (steps < zLength){

**if**(canMove()){

move();

steps++;

}

}

**else** **if** (zSection == 2){

setDirection(90);

steps = 0;

zSection++;

}

**else** **if** (zSection == 1){

setDirection(225);

steps = 0;

zSection++;

}

}

}

4. Write a class DancingBug that dances by making different turns before each move. The DancingBug constructor has an integer array as a 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 patter.

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

**public** **class** DancingBug **extends** Bug{

**private** **int**[] turnz;

**private** **int** current;

**public** DancingBug(**int**[] turns){

turnz = turns;

current = 0;

}

**public** **void** turn(**int** num){

**for**(**int** x = 1; x <= num; x++){

turn();

}

}

**public** **void** act(){

**if**(current == turnz.length){

current = 0;

turn(turnz[current]);

current++;

**super**.act();

}

}

}

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

a. Create a BoxBug with side length.

b. Add new BoxBug at a location, either randomly or specifically.

**int** fib (**int** n) {

**int** fib = 0;

**int** a = 1;

**for**(**int** i=0; i<n; i++) {

fib = fib + a;

a = fib;

}

**return** fib;

}