Software Methods and Tools

Assignment-6, Spring-2017

Submitted by:

Anusha Malineni

Student ID: 16233382

Initial Changes to Project

1) Changed the necessary private variables and methods to public.
2) Added startGame2() method in SnakeGame.java which is similar to startGame() method.

```
public void startGame2() {
        /*
         * Initialize everything.
         */
        this.random = new Random();
        this.snake = new LinkedList<>();
        this.directions = new LinkedList<>();
        this.logicTimer = new Clock(9.0f);
        this.isNewGame = true;

        //Set the timer to paused initially.
        logicTimer.setPaused(true);
}
```

1. What problems did you find in the code? For each problem, further explain what test case you used to find the problem.

The problem I found in the code is that the **fruitsEaten** score is not being updated when a fruit is eaten by a snake.

In order to test this problem I designed the test case **testFruitEatenScore()** and executing this test case helps us in finding that there is a bug in updating the fruitsEaten score.

For this test case, the expected result is 1 when the first fruit is eaten by the snake. In this I also set the position of fruit and snake in such a way that the fruit is eaten by snake.

Test Case Description	Expected Output	Actual Output	Result
Check for fruits eaten score	1	0	Fail

Below is the code for the test case:

```
public void testFruitEatenScore() {

        game.spawnFruit(); // position set fruit
        int expectedFruitScore = 1; // initially set to 1


        x = ++game.X_FruitTile;
        y = game.Y_FruitTile;

        snake_head = new Point(x, y);
        game.snake.add(snake_head);
        game.directions.add(Direction.North);
        game.updateGame();

System.out.println("Actual count of Fruits eaten by snake : " +game.getFruitsEaten());
assertEquals(expectedFruitScore, game.getFruitsEaten()); // Fruits Eaten Score should
be updated to 1 when the first fruit is eaten by snake
    }
```

2. Specifically explain the test case that you have created for each method. What is your input, and what is your expected output? What is your logic of testing this method?

In brief, I designed 5 test cases to test different scenarios in the SnakeGame. Following are the short descriptions of the test cases designed.

Test Case Description	Expected Output	Actual Output	Result
1) Check for score update when fruit is eaten	0	100	Pass
2) Check for fruits eaten score	1	0	Fail

3) Check for nextFruit Score	98	98	Pass
4) Check for Tiletype when fruit position is set	Fruit	Fruit	Pass
5) Check for SnakeBody during Collision	SnakeBody	SnakeBody	Pass

In the assignment description updateGame(), spawnFruit() and updateSnake() methods are asked to be tested.

To test updateGame(), test cases 1, 2, 3, 5 are designed. In these test cases , I checked for many scenarios which includes the functionality in updateGame() method. Some of the scenarios checked from updateGame() method are if the score is updated as per the rule i.e., when the fruit is eaten by the snake and similarly when the fruit is eaten, the fruitsEaten score should be updated simultaneously and the nextFruitScore should be decremented as soon as the next fruit appears on the screen.

To test spawnFruit(), test cases 1, 2, 3, 4 are designed. Here based on the point position of x and y , the location of fruit is set and the snake is moved to the direction of the fruit so that other values of score and fruitEaten can also be verified.

To test updateSnake(), test case 5 is designed. In this test case, the game is updated and the based on the point of x and y set to the location of fruit and snake, snake values are updated and then we can check for the collision here. We can check if the game is coming to end when collision occurs and we can also check for the collision value when collided with the wall or to itself.

Now, each test case is explained in detail below with expected outputs and actual outputs.

(a) Check for score update when fruit is eaten:

In this test case, I will check if the score is updated correctly when the fruit is hit by snake or not.

Below is the code for the test case:

```
public void testUpdateScore() {

        game.spawnFruit(); // position set fruit
        int initialScore = game.getScore(); // 0

        x = game.X_FruitTile;

        y = ++game.Y_FruitTile;

        snake_head = new Point(x, y);
        game.snake.add(snake_head);
        game.directions.add(Direction.North);
        game.updateGame();

        int actualScore = game.getScore();
        System.out.println("Expected Score of the game : " +initialScore);
        System.out.println("Actual Score of the game : " +game.getScore());
        assertNotEquals(initialScore, actualScore); //Score should be updated
        when the fruit is eaten by snake
}
```

In this test case the fruit is created and the snake is made to hit the fruit at the score of 100 only, then the actual score will also be 100 and the initial score of the game is 0.

As the initialScore and actualScore do not match with each other, we can say that the score is updated when the fruit is eaten and therefore the test case is a pass.

(b) Check for fruits eaten score:

In this test case, the fruitsEaten score is tested. That is when the fruit is eaten, the score of fruitsEaten should be incremented.

Below is the code for the test case:

```
public void testFruitEatenScore() {

        game.spawnFruit(); // position set fruit

        int expectedFruitScore = 1; // initially set to 1

        x = ++game.X_FruitTile;
        y = game.Y_FruitTile;


        snake_head = new Point(x, y);
        game.snake.add(snake_head);
        game.directions.add(Direction.North);
        game.updateGame();

        System.out.println("Actual count of Fruits eaten by snake : "
        +game.getFruitsEaten());
        assertEquals(expectedFruitScore, game.getFruitsEaten()); // Fruits Eaten
        Score should be updated to 1 when the first fruit is eaten by snake
}
```

In this test case, initially the fruitsEaten score is 0 as when the game starts, there will not be any fruits eaten by snake. As soon as the fruit is eaten by snake, its value should be incremented to 1.

But this is not happening in the game and this is where we find the problem also. Therefore, this test case is a failure and this helps in finding the bug in code.

(c) Check for nextFruit Score:

This test case will test if the nextFruitScore is decremented correctly or not.

Below is the code for the test case:

```
public void testNextFruitScore() {

        game.spawnFruit(); // position set fruit
        int expectedFruitScore = game.getNextFruitScore() - 2;
        x = ++game.X_FruitTile;
        y = game.Y_FruitTile;
        snake_head = new Point(x, y);
        game.snake.add(snake_head);
        game.directions.add(Direction.North);
        game.updateGame();
        game.updateGame();

        int actualFruitScore = game.getNextFruitScore();
        System.out.println("Expected score of the fruit when the game is updated
        : " +expectedFruitScore);
        System.out.println("Actual score of the fruit when the game is updated : "
        +actualFruitScore);

        assertEquals(expectedFruitScore, actualFruitScore);

}
```

In this test case, as the game is updated twice, the nextFruitsScore should be decremented by 2 from 100 i.e., it should be 98. We will check if the expected and actual score are equal or not. If they are equal, the values are updated correctly and functionality will be working as expected.

(d) Check for Tiletype when fruit position is set:

In this test case, we will test if the Tiletype returned is as expected or not.

Below is the code for the test case:

```
public void testSpawnFruit() {
        game.spawnFruit();
        x = game.X_FruitTile;
        y = game.Y_FruitTile;
        TileType type = game.board.getTile(x, y);
        System.out.println("Displays the tiletype : "+type);
        assertEquals(TileType.Fruit, type);
}
```

In this test case, using game.spawnFruit(), the tile type is set to fruit and we are returing the value from the boardpanel using getTile method. We verify the result with expected result i.e., TitleType.Fruit. If that matches then the test case is a pass.

(e) Check for SnakeBody during Collision:

In this test case, when the snake body is collided either with the wall or the snake head or when the snake is collided into the wall, the collision should return snakebody and the game should end.

Below is the code for the test case:

```
public void testUpdateSnake() {

        game.resetGame();
        game.directions.addLast(Direction.South);
        game.updateGame();

        TileType collision = game.updateSnake();
        System.out.println("Collision of the snake : "+collision);
        assertEquals(TileType.SnakeBody, collision);
}
```

In this test case, the expected value is snakebody and the actual value is also snake body. Therefore, the collision value is correct and the test case is a pass.

3. Include a screenshot of the result of running your test.

Below is the screen shot for the test suite of test cases.