TESTING VELOCITY RELATED FUNCTIONS IN BRICKLES

Christina Xuan Yu and Srinjay Paul

CS 583: Object Oriented Testing

Dr. Lawrence McCartney

10/03/2017

Testing Plan: We added positive and negative test cases to each test, and tested each test individually. After writing and successfully testing satisfactory amount of testcases (both components of velocity: direction and speed in each quadrant), we did a suite test to see if all the functions are working altogether. We also did some validation testing and suggested how the code can be made more concise. Following this if more testing is demanded then we would do integration testing to prove the robustness of the program when there is interaction between methods. Also, the Velocity class decides the speed and direction of a moving object, which is constantly changing during the game. Having the moving objects working normally is the key and fundamental to the game. Hence it is the most important class. Concisely, the main idea behind our test plan is:

- Run the junit to find obvious errors.
- Look through the code to have an understanding.

- Write tests for each method.
- Write testcases in each test.
- Do positive and negative testing.
- Look for validation errors.
- Do a suite test in the end.
- Look for logical errors throughout all this.

Positive Testcases: 8

Negative Tests: 5

Additional work: We did additional positive and negative tests and relevant optional work mentioned on the assignment page.

Division of work: Srinjay wrote 5 negative and 1 positive test, he formatted and annotated the document, wrote the method descriptions and wrote a part of the test plan section.

Christina wrote 7 positive tests in the VelocityTester.java. She wrote the method descriptions, the importance of the Velocity class and finalized the report.

Individual Tests: Here are the results to our testing:

1. Velocity Constructor Initialization: When initialized, it should set the fields xspeed, yspeed, speed and direction to zero.

Positive: We make sure the initialization is correct by throwing an exception in Velocity constructor if the initial values are not zeros.

```
public class VelocityTester {

    @Before
    public void setUp() throws Exception {
        Velocity v = new Velocity();
        // check for initial values
        assertEquals(v.getSpeedX(), 0);
        assertEquals(v.getSpeedY(), 0);
        assertEquals(v.getDirection(), 0);
    }
```

Throws exception when all values are not zero.

Problems | Javadoc | Declaration | Console | Error Log | JUnit

nished after 0.01 seconds

Runs: 5/5 (1 skipped) Errors: 0 Failures: 0

```
    /** Default Constructor initializes all attributes to zero */
    public Velocity() {
        xSpeed = 0;
        ySpeed = 0;
        speed = 0;
        direction = 0;
        if (this.xSpeed != 0 || this.ySpeed != 0 || this.speed != 0 || this.direction != 0) {
            throw new InvalidParameterException("Initial values must be 0.");
        }
    }
```

Problems | Javadoc | Declaration | Console | Error Log | JUnit

Finished after 0.01 seconds

Runs: 5/5 (1 skipped) Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.001 s) Failure Trace

2. Velocity Constructor: When a new speed and direction is passed, these values should get updated.

Positive & Negative: We tried positive tests with both direction and speed. We tried negative tests on speed. It takes character as input and can even be made to give a positive test result. There is no limit on speed.

```
22    public void velocitypass() {
23        //positive
24        Velocity v = new Velocity(50, 50);
25        assertEquals(v.getSpeedX(), 32);
26        assertEquals(v.getSpeedY(), 38);
27        assertEquals(v.getDirection(), 50);
```

A positive test for creating an instance of velocity constructor.

JUnit

nished after 0.023 seconds

Runs: 6/6 Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s) Failure

```
22    public void velocitypass() {
23        char A = 'a';
24        //negative
25        Velocity v = new Velocity(A, 50);
26        assertEquals(v.getSpeedX(), 63);
27        assertEquals(v.getSpeedY(), 73);
28        assertEquals(v.getDirection(), 50);
29
```

Takes a character value as speed.

JUnit

Finished after 0.034 seconds

Runs: 6/6 Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.001 s) Failure

No limit on speed.

```
public void velocitypass() {
    //negative
    Velocity v = new Velocity(500000000, 50);
    assertEquals(v.getSpeedX(), 324868523);
    assertEquals(v.getSpeedY(), 380079520);
    assertEquals(v.getDirection(), 50);
```

JUnit

nished after 0.032 seconds

Runs: 6/6 Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s) Failur

3. Reverse X: Reverses the direction and magnitude of X.

Positive: In the Velocity.reverseX(), 2 cases are computed separately when direction < 180 or direction >=180. We wrote the test case for each of them.

```
@Test
public void testReverseX() {
    // testing when direction is smaller than 180
    Velocity v = new Velocity(50, 50);
    v.reverseX();
    assertEquals(v.getSpeedX(), -32);
    assertEquals(v.getDirection(), 130);

    // testing when direction is greater than 180
    Velocity a = new Velocity(100, 300);
    a.reverseX();
    assertEquals(a.getSpeedX(), -45);
    assertEquals(a.getDirection(), 240);

}
```

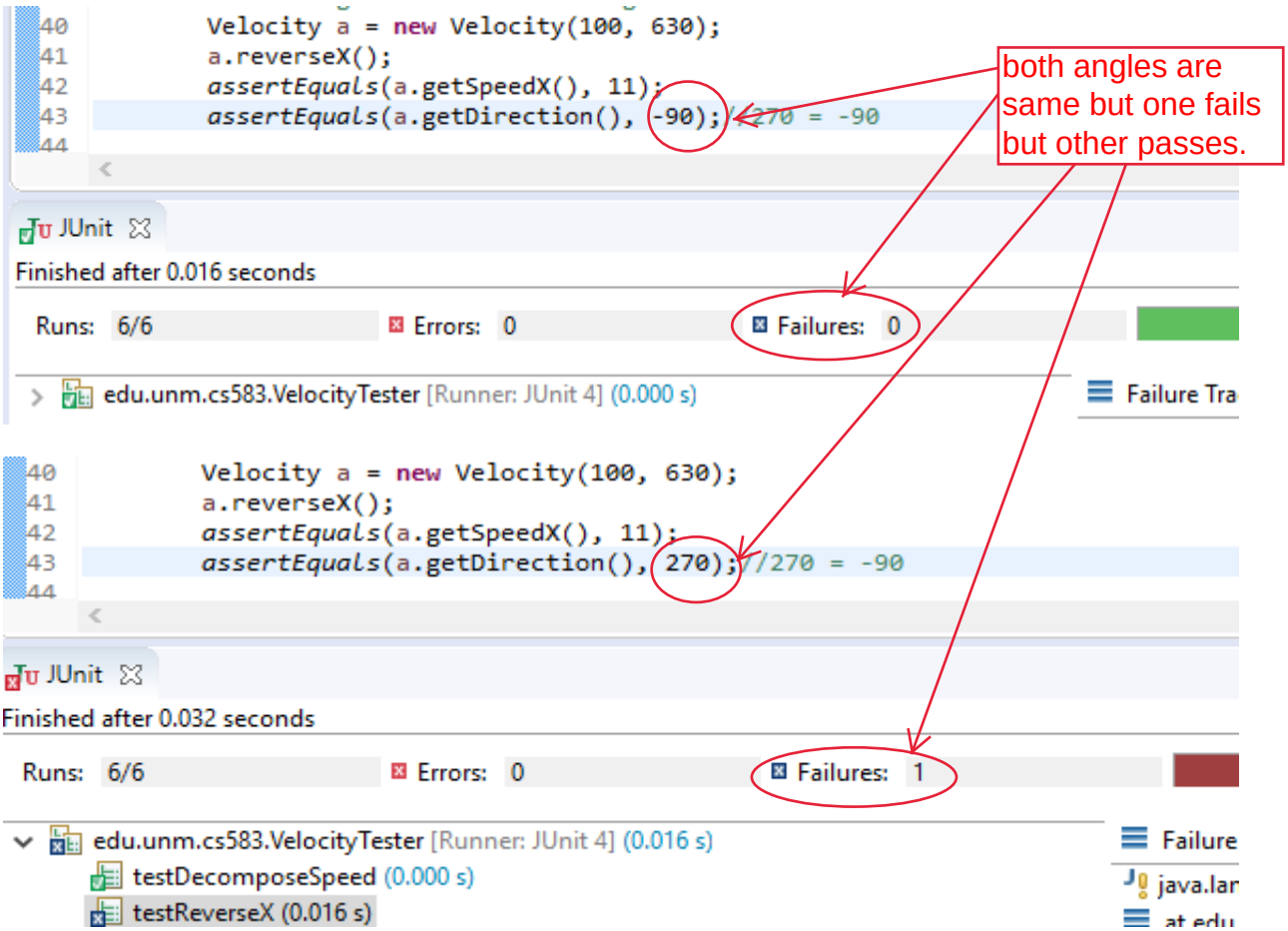
oblems Javadoc Declaration Console Error Log JUnit

ed after 0.01 seconds

s: 5/5 (1 skipped) Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s) Failure Trace

Negative: It returns the direction it calculates as it is. Direction can be -infinity to infinity. Hence there should be convention of returning the direction in the equivalent 0-360 format value. Here we see that -90 and 270 are same, but the test passes for one and fails for other.



4. Reverse Y: Reverses the direction and magnitude of Y.

Positive: We tested the Velocity.reverseY() to see if the ySpeed value is inverted, and direction is modified according to the formula in the code.

```
@Test
public void testReverseY() {
    Velocity v = new Velocity(50, 50);
    v.reverseY();
    assertEquals(v.getSpeedY(), -38);
    assertEquals(v.getDirection(), 310);
}
```

A positive test for reversing Y.

blems @ Javadoc Declaration Console Error Log JUnit

d after 0.01 seconds

: 5/5 (1 skipped) Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s) Failure Trace

5. Setters: Methods that can be used to set the speed and direction.

Positive: For positive test cases, Velocity.setSpeed() and Velocity.setDirection() are tested separately.

In testSetSpeed(), we created a new velocity, gave it initial values, and we tested if we get the correct value from it.

```
@Test
public void testSetSpeed() {
    Velocity v = new Velocity();
    v.setDirection(300);
    v.setSpeed(100);
    assertEquals(v.getSpeedX(), 45);
    assertEquals(v.getSpeedY(), -89);
//  System.out.println((int)(Math.sin(300 / 57.9) *100));
}
```

Positive test for setting speed and direction.

blems @ Javadoc Declaration Console Error Log JUnit

d after 0.01 seconds

: 5/5 (1 skipped) Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s) Failure Trace

In testDirection(), we use angles that test each quadrant of a circle(0-360 degrees) and also included the boundary cases. In regular cases, when direction is < 360, we call getDirection() and we should get the direction itself. In the boundary case, when direction is set to 360, we should get 360 mod 360 = 0.

```java
    @Test
    /** public void test() {
        fail("Not yet implemented"); **/
    public void testSetDirection() {
        Velocity out = new Velocity();
        out.setDirection(90);
        assertEquals("Result", 90, out.getDirection());
        System.out.println("The direction is "+out.getDirection());
        out.setDirection(180);
        assertEquals("Result", 180, out.getDirection());
        System.out.println("The direction is "+out.getDirection());
        out.setDirection(270);
        assertEquals("Result", 270, out.getDirection());
        System.out.println("The direction is "+out.getDirection());
        out.setDirection(360); // boundary case
        assertEquals("Result", 0, out.getDirection());
        System.out.println("The direction is "+out.getDirection());

    }
```

```
<terminated> VelocityTester [JUnit] /Lib
The direction is 90
The direction is 180
The direction is 270
The direction is 0
```

JUnit

Finished after 0.009 seconds

Runs: 5/5 Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s)

Failure Trace

Negative: Both methods to set speed and direction gave positive results when a character was used as speed or direction. Also, logically the speed cannot be zero or else the peg would stop. This constrain is not followed.

```
56      public void testSetters() {
57          Velocity v = new Velocity();
58          char B = 'a';
59          v.setDirection(B);
60          v.setSpeed(B);
61          assertEquals(v.getSpeedX(), -10);
62          assertEquals(v.getSpeedY(), 96);
63          assertEquals(v.getDirection(), 97);
64  //      System.out.println((int)(Math.sin(300 / 57.9) *100));
```

Speed and direction cannot be a character.

JUnit

nished after 0.022 seconds

Runs: 6/6 Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s) Failure Tr

```
56      public void testSetters() {
57          Velocity v = new Velocity();
58          int B = 0;
59          v.setDirection(B);
60          v.setSpeed(B);
61          assertEquals(v.getSpeedX(), -0);
62          assertEquals(v.getSpeedY(), 0);
63          assertEquals(v.getDirection(), 0);
64  //      System.out.println((int)(Math.sin(300 / 57.9) *100));
65      }
```

Speed cannot be zero or the peg will stop.

JUnit

Finished after 0.016 seconds

Runs: 6/6 Errors: 0 Failures: 0

edu.unm.cs583.VelocityTester [Runner: JUnit 4] (0.000 s) Failure Tra

6. DecomposeSpeed: Creates the X and Y components of the speed. It is called every time speed is set.

Positive: We wrote a test case to see if a given velocity with speed and direction values can be decomposed with correct xSpeed and ySpeed values.

```
@Test
public void testDecomposeSpeed() {
    Velocity v = new Velocity(50, 50);
    assertEquals(v. getSpeedX(), 32);
    assertEquals(v. getSpeedY(), 38);
}
```

A positive test for decomposing speed.