3S03 Assignment 3

Ethan Walsh - 400383566 - walshe11
Ahsan Muzammil - 400455916 - muzammia

April 8, 2025

Question 1

Class Objectives	Ensure that all methods and constructors meet expected behaviour. This includes usual operations such as getting and setting the direction, speed, and its components (speedX and speedY). Boundary values and unexpected inputs should also be tested in order to guarantee complete and robust test suite.
Inspection/Review Requirements	This class may require inspection in order to understand the interface implementations - for example, how the components (speedX and speedY) are calculated as well as how specific methods such as reverse are performed. This will give the test developers a better idea of what to target in their testing, taking advantage of any expected vulnerabilities that are found during exploration. include bit abt quality analysis here? code smells, ensuring things are maintainable etc.
Specification-Based Testing Objectives	Specification-Based Testing will target the behaviour outlined in the game's specification and high level requirements. Tests in this section should ensure expected behaviour from each method, such as reversing speed when hitting the player, a wall, or a brick. It should also confirm the functionality of all get and set methods within the class.
Implementation-Based Testing Objectives	Implementation-Based Testing Objectives will examine the inner workings of the specification by ensuring each individual function works as intended. These tests will be at a finer level of detail than those in the specification-based tier. Implementation-based tests will target individual methods usual input, boundary/edge cases, as well as unexpected inputs. An example of this would be testing reverseX by using input partitioning from the strong-robust quadrant.
Interaction-Based Testing Objectives	Interaction-Based testing will ensure that methods in the class work together when needed and do not interfere with each other. An example of this would be testing the setDirection method and ensuring speed remains the same but velocity changes. Another relevant example would be testing the interaction between the three methods reverse, reverseX, and reverseY to ensure they do not introduce any reversing redundancy that would lead to inconsistent behaviour.

Question 2

Class invariant:

- $\text{speed} \in \mathbb{Z} \land \text{speed} > 0$
- $\text{direction} \in [0, 360) \land \text{direction} \in \mathbb{Z}$ (direction is always a valid angle)
- $\text{speedX} = \text{speed} \cdot \cos(\text{direction})$
- $\text{speedY} = \text{speed} \cdot \sin(\text{direction})$

`public Velocity()`

Pre-condition:

- No input constraints (default constructor).

Post-condition:

- speed = 0, speedX = 0, speedY = 0.
- direction is initialized to a default value (e.g., 0).

`public Velocity(Speed speed, Direction direction)`

Pre-condition:

- $\text{speed} \in \mathbb{Z}$
- $\text{direction} \in \mathbb{Z} \wedge \text{direction} \in [0, 360)$

Post-condition:

- $\text{this.speed} = \text{speed}$
- $\text{this.direction} = \text{direction}$
- $\text{this.speedX} = \text{speed} \cdot \cos(\text{direction})$
- $\text{this.speedY} = \text{speed} \cdot \sin(\text{direction})$

`public Speed getSpeed()`

Pre-condition:

- $\text{speed} \in \mathbb{Z}$.

Post-condition:

- Returns speed.
- $\text{this.speed} = \frac{\text{speedX}}{\cos(\text{direction})}$.
- $\text{this.speed} = \frac{\text{speedY}}{\sin(\text{direction})}$.

`public Speed getSpeedX()`

Pre-condition:

- $\text{speedX} \in \mathbb{Z}$.

Post-condition:

- Returns speedX and $\text{this.speedX} = \text{speed} \cdot \cos(\text{direction})$.

`public Speed getSpeedY()`

Pre-condition:

- $\text{speedY} \in \mathbb{Z}$.

Post-condition:

- Returns speedY and $\text{this.speedY} = \text{speed} \cdot \sin(\text{direction})$.

`public Direction getDirection()`

Pre-condition:

- $\text{direction} \in \mathbb{Z} \land \text{direction} \in [0, 360)$

Post-condition:

- Returns direction, $\text{this.direction} = \text{direction}$, and $\text{direction} \in [0, 360) \land \text{direction} \in \mathbb{Z}$.

`public void setSpeed(Speed speed)`

Pre-condition:

- $\text{speed} \in \mathbb{Z}$

Post-condition:

- $\text{this.speed} = \text{speed} \land \text{speed} \geq 0$
- $\text{this.speedX} = \text{speed} \cdot \cos(\text{direction})$
- $\text{this.speedY} = \text{speed} \cdot \sin(\text{direction})$

`public void setDirection(Direction direction)`

Pre-condition:

- $\text{direction} \in \mathbb{Z} \land \text{direction} \in [0, 360)$

Post-condition:

- $\text{this.direction} = \text{direction}$
- $\text{this.speedX} = \text{speed} \cdot \cos(\text{direction})$
- $\text{this.speedY} = \text{speed} \cdot \sin(\text{direction})$

`public void reverse()`

Pre-condition:

- $\text{speedX} \in \mathbb{Z}$.
- $\text{speedY} \in \mathbb{Z}$.
- $\text{direction} \in \mathbb{Z} \land \text{direction} \in [0, 360)$

Post-condition:

- $\text{this.direction} = (\text{direction} + 180)\%360$.
- $\text{this.speedX} = \text{speed} \cdot \cos(\text{direction})$
- $\text{this.speedY} = \text{speed} \cdot \sin(\text{direction})$
- $\text{direction} \in [0, 360) \land \text{direction} \in \mathbb{Z}$

`public void reverseX()`

Pre-condition:

- $\text{speedX} \in \mathbb{Z}$.
- $\text{direction} \in \mathbb{Z} \land \text{direction} \in [0, 360)$

Post-condition:

- $\text{this.speedX} = -\text{speedX}$.
- $\text{this.direction} = (180 - \text{direction})\ \%\ 360$.

`public void reverseY()`

Pre-condition:

- speedY $\in \mathbb{Z}$.
- direction $\in \mathbb{Z} \wedge$ direction $\in [0, 360)$

Post-condition:

- this.speedY $= -$speedY.
- this.direction $=$ ($-$direction) % 360.

Question 3

The following test cases assume that the Java Math class (`https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html`) has been imported and can be used for sine and cosine. As recommended by Dr. Istvan, although it's generally considered poor practice to include multiple assert statements within a single test case, it is acceptable in this context due to the structure and nature of the Velocity class. Ideally, the puck's speed would be represented using a separate Vector class to handle the X and Y components of the velocity vector. However, for the purposes of this implementation, using multiple assertions per test case is sufficient to verify each parameter of the class when methods such as reverseX, getSpeed, or setDirection are invoked.

```java
    public class TestVelocity
    {
        // Input: speed = 5, direction = 90
        // Action: setDirection(180)
        // Expected: getDirection() returns 180
        @Test
        public void testSetDirection()
        {
            Velocity puck = new Velocity(5, 90);
            puck.setDirection(180);
            assertEquals(180, puck.getDirection());
        }

        // Input: speed = 5, direction = 90
        // Action: setSpeed(10)
        // Expected: getSpeed() returns 10
        @Test
        public void testSetSpeed()
        {
            Velocity puck = new Velocity(5, 90);
            puck.setSpeed(10);
            assertEquals(10, puck.getSpeed());
        }

        // Input: speed = 5, direction = 0 (x-axis right)
        // Action: reverse()
        // Expected: direction = 180, speedX = -5, speedY = 0
        @Test
        public void testReverseWithCurrentDirectionOnXAxisOnly()
        {
            Velocity puck = new Velocity(5, 0);
            puck.reverse();
            assertEquals(180, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(180)), puck.getSpeedX());
            assertEquals((int) (puck.getSpeed()*Math.sin(180)), puck.getSpeedY());
        }

        // Input: speed = 5, direction = 90 (y-axis up)
        // Action: reverse()
        // Expected: direction = 270, speedX = 0, speedY = -5
        @Test
        public void testReverseWithCurrentDirectionOnYAxisOnly()
        {
            Velocity puck = new Velocity(5, 90);
            puck.reverse();
            assertEquals(270, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(270)), puck.getSpeedX());
            assertEquals((int) (puck.getSpeed()*Math.sin(270)), puck.getSpeedY());
        }

        // Input: speed = 5, direction = 45 (northeast)
        // Action: reverse()
        // Expected: direction = 225, speedX = -3.53 = -4, speedY = -3.53 = -4 (int rounding)
        @Test
        public void testReverseWithCurrentDirectionOnDiagonalLessThan180()
        {
            Velocity puck = new Velocity(5, 45);
            puck.reverse();
            assertEquals(225, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(225)), puck.getSpeedX());
            assertEquals((int) (puck.getSpeed()*Math.sin(225)), puck.getSpeedY());
        }

        // Input: speed = 5, direction = 190
        // Action: reverse()
        // Expected: direction = 10, speedX = 4.92 = 5, speedY = 0.87 = 1 (int rounding)
        @Test
        public void testReverseWithCurrentDirectionOnDiagonalGreaterThan180()
        {
            Velocity puck = new Velocity(5, 190);
            puck.reverse();
            assertEquals(10, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(10)), puck.getSpeedX());
            assertEquals((int) (puck.getSpeed()*Math.sin(10)), puck.getSpeedY());
        }

        // Input: speed = 5, direction = 359
        // Action: reverse()
        // Expected: direction = 179, speedX = -5, speedY = 0.09 = 0 (int rounding)
        @Test
        public void testReverseWithCurrentDirectionEqualTo359()
        {
            Velocity puck = new Velocity(5, 359);
            puck.reverse();
            assertEquals(179, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(179)), puck.getSpeedX());
            assertEquals((int) (puck.getSpeed()*Math.sin(179)), puck.getSpeedY());
        }

        // Input: speed = 5, direction = 0
        // Action: reverseX()
        // Expected: direction = 180, speedX = -5
        @Test
        public void testReverseXWithCurrentDirectionOnXAxisOnly()
        {
            Velocity puck = new Velocity(5, 0);
            puck.reverseX();
            assertEquals(180, puck.getDirection());
            assertEquals(-5, puck.getSpeedX());
        }

        // Input: speed = 5, direction = 90
        // Action: reverseX()
        // Expected: direction = 90 (unchanged), speedX = 0
        @Test
        public void testReverseXWithCurrentDirectionOnYAxisOnly()
        {
            Velocity puck = new Velocity(5, 90);
            puck.reverseX();
            assertEquals(90, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(90)), puck.getSpeedX());
        }

        // Input: speed = 5, direction = 179
        // Action: reverseX()
        // Expected: direction = 1, speedX = 5
        @Test
        public void testReverseXWithCurrentDirectionLessThan180()
        {
            Velocity puck = new Velocity(5, 179);
            puck.reverseX();
            assertEquals(1, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(1)), puck.getSpeedX());
        }

        // Input: speed = 5, direction = 225
        // Action: reverseX()
        // Expected: direction = 315, speedX = 3.53 = 4 (int rounding)
        @Test
        public void testReverseXWithCurrentDirectionGreaterThan180()
        {
            Velocity puck = new Velocity(5, 225);
            puck.reverseX();
            assertEquals(315, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(315)), puck.getSpeedX());
        }

        // Input: speed = 5, direction = 359
        // Action: reverseX()
        // Expected: direction = 181, speedX = -4.99 = -5 (int rounding)
        @Test
        public void testReverseXWithCurrentDirectionEqualTo359()
        {
            Velocity puck = new Velocity(5, 359);
            puck.reverseX();
            assertEquals(181, puck.getDirection());
            assertEquals((int) (puck.getSpeed()*Math.cos(181)), puck.getSpeedX());
        }
    }
```