Testing Lab Report

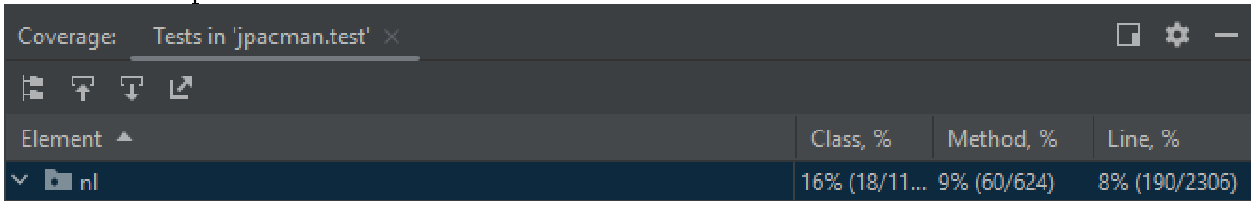
Fork: https://github.com/tchrjs/CS-472-Senior-Design-Project

Task 2.1

For this task, I have written 3 unit tests for the software project jpacman. The following tests are:

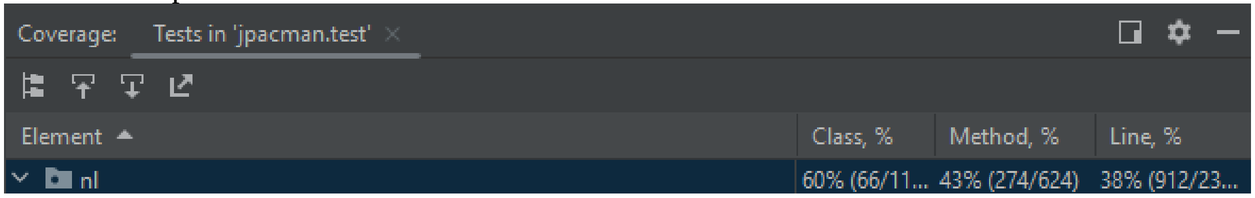
Name	Class	Method
Christian Toribio	PlayerCollisions	playerVersusPellet( )
Christian Toribio	GhostFactory	createPinky( )
Christian Toribio	Game	stop( )

Before unit implementation:



Element	Class, %	Method, %	Line, %
nl	16% (18/11...	9% (60/624)	8% (190/2306)

After unit implementation:



Element	Class, %	Method, %	Line, %
nl	60% (66/11...	43% (274/624)	38% (912/23...

Above is the test coverage results from before and after the implementation of the 3 unit tests I have listed. The class coverage increased from 16% to 60%. The method went from 9% to 43%.

Task 3

The coverage results from JaCoCo has some similarities to the ones I got from IntelliJ's results, but still different, nonetheless. The reason why the coverage was not the same is mostly like because both of them have their own methods of what should be considered. Because of JaCoCo's lower percentage in coverage, I assume that JaCoCo takes more things into account compared to IntelliJ's coverage.

I found that the source code visualization from JaCoCo helpful in analyzing the data compared to the one by IntelliJ. I would, however, say that I prefer IntelliJ's coverage window over JaCoCos. IntelliJ's window was easier to navigate through and had better organization overall. It is clear on what data is being show and gives a better look at the whole thing.

Task 2.1 Unit Testing Code

1. PlayerCollision - playerVersusPellet( )

```
public class PlayerCollisionsTest {

    /**
     * The player collision to test on.
     */
    private PlayerCollisions playerCollisions;

    /**
     * Default pellet value.
     */
    private static final int PELLET_VALUE = 10;

    /**
     * Creates player to test on.
     */
    private PacManSprites Store = new PacManSprites();
    PlayerFactory Factory = new PlayerFactory(new PacManSprites());
    Player ThePlayer = Factory.createPacMan();

    /**
     * Creates pellet to test on.
     */
    Pellet ThePellet = new Pellet(PELLET_VALUE, Store.getPelletSprite());

    /**
     * Collision test from player and pellets.
     */
    @BeforeEach
    void setup() {
        playerCollisions = new PlayerCollisions(new
DefaultPointCalculator());
        playerCollisions.collide(ThePlayer, ThePellet);
    }

    /**
     * Validates if player consumes pellet during collision.
     * Pellet is removed and player points increases by pellet value
(default by 10).
     */
    @Test
    void testPlayerVersusPellet() {
        assertThat(!ThePellet.hasSquare() && ThePlayer.getScore() ==
PELLET_VALUE);
    }

}
```

2. GhostFactory -.createPinky( )

```java
public class GhostFactoryTest {

    /**
     * The ghost under test.
     */
    private Ghost TheGhost;

    /**
     * Create a ghost to test on.
     */
    @BeforeEach
    void setUp() {
        PacManSprites ghostSprites = new PacManSprites();
        GhostFactory Factory = new GhostFactory(ghostSprites);
        TheGhost = Factory.createPinky();
    }

    /**
     * Validates if ghost created is Pinky.
     */
    @Test
    void testCreatePinky() {
        assertThat(TheGhost instanceof Pinky);
    }
}
```

3. Game - stop( )

```java
public class GameTest {

    /**
     * The game used to test stop on.
     */
    private Game TheGame;

    /**
     * Start the game.
     */
    @BeforeEach
    void setup() {
        Launcher TheLauncher = new Launcher();
        TheGame = TheLauncher.makeGame();
        TheGame.start();
    }

    /**
     * Pause the game
     */
    @Test
    void testStop() {
        TheGame.stop();
        assertThat(TheGame.isInProgress()).isEqualTo(false);
    }
```