CS 472 Unit Testing Lab Report

Ryan Bryant
https://github.com/Ruaaann/CS-472-2023-GROUP-2/tree/jpacman_tests

The following two tasks, task 2.1 and task 3, consist of unit testing done for the JPacman project in IntelliJ.

Task 2.1:

Coverage | Testing Lab [test]

Element	Class, %	Method, %	Line, %
nl	3% (2/55)	1% (5/312)	1% (14/1137)
tudelft	3% (2/55)	1% (5/312)	1% (14/1137)
jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	0% (0/13)	0% (0/78)	0% (0/345)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	0% (0/6)	0% (0/45)	0% (0/119)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

This is the original testing coverage of the JPacman project

After the PlayerTest was added, which tests the isAlive() method in the class, Player, the coverage was as follows in the next figure:

Coverage | Testing Lab [test]

Element	Class, %	Method, %	Line, %
nl	16% (9/55)	9% (30/312)	8% (95/1153)
tudelft	16% (9/55)	9% (30/312)	8% (95/1153)
jpacman	16% (9/55)	9% (30/312)	8% (95/1153)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	15% (2/13)	6% (5/78)	3% (13/350)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	83% (5/6)	44% (20/45)	52% (68/130)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

Coverage after PlayerTest

Three new unit tests were then implemented and the coverage was documented after each unit test was written and tested.
GhostFactoryTest tests the method `testInky()`
PointsTest tests the method `addPoints()`
PelletTest tests the Pellet constructor

The first unit test written was for the createInky() method from the Ghost class:

```java
package nl.tudelft.jpacman.npc.ghost;
import nl.tudelft.jpacman.npc.Ghost;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertNotNull;

new *
public class GhostFactoryTest {
    1 usage
    private static final PacManSprites sprites = new PacManSprites();

    1 usage
    private final GhostFactory factory = new GhostFactory(sprites);

    new *
    @Test
    void testCreateInky() {
        Ghost ghost = factory.createInky();
        // Ensure Inky has been created (object exists)
        assertNotNull(ghost);
    }
}
```

The coverage after unit test 1 is shown in the following figure (npc Class% increases by 40%:

Coverage | Testing Lab [test]

Element	Class, %	Method, %	Line, %
nl	23% (13/55)	11% (37/312)	9% (115/1159)
tudelft	23% (13/55)	11% (37/312)	9% (115/1159)
jpacman	23% (13/55)	11% (37/312)	9% (115/1159)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	15% (2/13)	6% (5/78)	3% (13/350)
npc	40% (4/10)	12% (6/47)	6% (17/243)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	83% (5/6)	46% (21/45)	54% (71/130)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

The second unit test written was for the addPoints() method from the Points class:

```java
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.level.Player;
import nl.tudelft.jpacman.sprite.PacManSprites;

import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
new *
public class PointsTest {

    1 usage
    private static final PacManSprites sprite = new PacManSprites();
    1 usage
    private final PlayerFactory factory = new PlayerFactory(sprite);
    2 usages
    private final Player temp_player = factory.createPacMan();
    new *
    @Test
    void testAddPoints(){
        // Give the player 250 points
        temp_player.addPoints(250);
        // Assert that the points have successfully been given to the player
        assertThat( actual: temp_player.getScore() == 250).isEqualTo( expected: true);
    }
}
```

The coverage after unit test 2 is shown in the following figure (Method% increases by 2%):

Coverage | Testing Lab [test]

Element	Class, %	Method, %	Line, %
nl	23% (13/55)	12% (39/312)	10% (118/1159)
tudelft	23% (13/55)	12% (39/312)	10% (118/1159)
jpacman	23% (13/55)	12% (39/312)	10% (118/1159)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	15% (2/13)	8% (7/78)	4% (16/350)
npc	40% (4/10)	12% (6/47)	6% (17/243)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	83% (5/6)	46% (21/45)	54% (71/130)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

The third unit test written was for the Pellet constructor from the Pellet class:

```java
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.sprite.PacManSprites;
import nl.tudelft.jpacman.sprite.Sprite;

import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertNotNull;
new *
public class PelletTest {
    1 usage
    private static final PacManSprites pac_sprites = new PacManSprites();
    1 usage
    private static final Sprite sprite = pac_sprites.getPelletSprite();
    2 usages
    private final Pellet pellet = new Pellet( points: 10, sprite);

    new *
    @Test
    void pelletTest(){
        int val = pellet.getValue();
        // Check that the pellet has the same value it was created with
        assertThat( actual: val == 10).isEqualTo( expected: true);
        // Ensure object exists
        assertNotNull(pellet);
    }
}
```

The coverage after unit test 3 is shown in the following figure (Level's Class% increased by 5%):

Coverage | Testing Lab [test]

Element	Class, %	Method, %	Line, %
nl	25% (14/55)	13% (42/312)	10% (124/1...
tudelft	25% (14/55)	13% (42/312)	10% (124/1...
jpacman	25% (14/55)	13% (42/312)	10% (124/1...
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	23% (3/13)	11% (9/78)	5% (21/351)
npc	40% (4/10)	12% (6/47)	6% (17/243)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	83% (5/6)	48% (22/45)	55% (72/130)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

Task 3:

Are the coverage results from `JaCoCo` similar to the ones you got from `IntelliJ` in the last task? Why so or why not?

The coverage results from JaCoCo are not super similar to the IntelliJ coverage results. JaCoCo generally shows higher levels of coverage than the IntelliJ results.

Did you find the source code visualization from `JaCoCo` on uncovered branches helpful?

I did find the source code visualization provided by JaCoCo to be helpful for uncovered branches. This lets you know the coverage of if statements and whether certain ones have been executed. This can be helpful to see where your unit tests are missing coverage and what direction to go in.

Which visualization did you prefer and why? `IntelliJ`'s coverage window or `JaCoCo`'s report?

IntelliJ's is nice to work with because it's in the IDE itself so you can change and edit code right where you are checking coverage, but JaCoCo's whole line highlighting is much more readable and superior to the small highlight IntelliJ does on the side. Overall I would probably prefer JaCoCo's visualization if I was able to have it open on a second monitor and reference it when needed. Color-coding is very quickly readable, which JaCoCo has.

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		67%		58%	73	155	103	344	21	69	4	12
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.ui		77%		47%	54	86	21	144	7	31	0	6
default		0%		0%	12	12	21	21	5	5	1	1
nl.tudelft.jpacman.board		86%		58%	44	93	2	110	0	40	0	7
nl.tudelft.jpacman.sprite		88%		62%	29	70	10	113	5	38	0	5
nl.tudelft.jpacman		69%		25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.points		60%		75%	1	11	5	21	0	9	0	2
nl.tudelft.jpacman.game		87%		60%	10	24	4	45	2	14	0	3
nl.tudelft.jpacman.npc		100%		n/a	0	4	0	8	0	4	0	1
Total	1,204 of 4,694	74%	290 of 637	54%	291	590	227	1,039	51	268	6	47

JaCoCo final coverage after unit tests implemented

<u>Following two tasks (4 and 5) will only be included in the canvas submission. They will not be present in this github submission.</u>