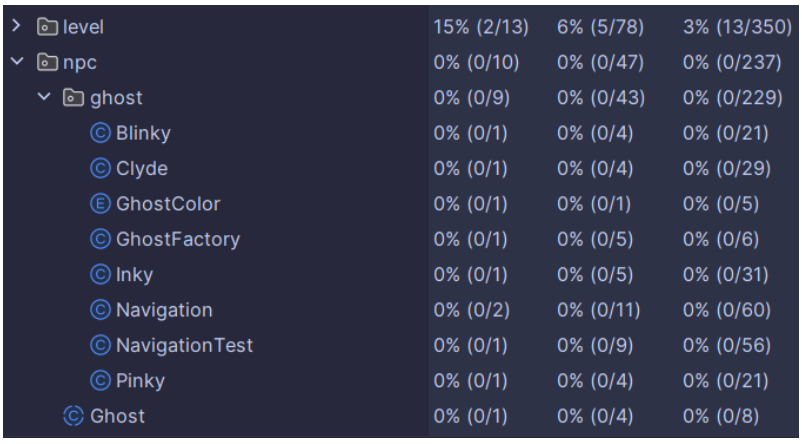
Unit Test Report

**Task 2.1**

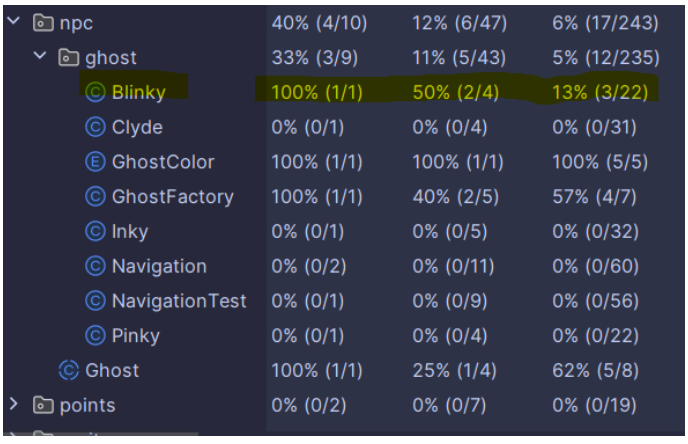
The first unit test that I wrote was for the `createBlinky()` method in the npc.ghost package. In this test, I verified that Blinky was being created successfully. I used the `PacManSprites()` method to initialize the PacManSprite. This test increased line coverage for the npc package by 13%.

```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 BlinkyTest {
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private final GhostFactory Factory = new GhostFactory(SPRITE_STORE);
    new *
    @Test
    void testCreateBlinky(){
        Ghost theGhost = Factory.createBlinky();
        assertNotNull(theGhost);
    }
}
```



| | | | |
|---|---|---|---|
| level | 15% (2/13) | 6% (5/78) | 3% (13/350) |
| npc | 0% (0/10) | 0% (0/47) | 0% (0/237) |
| ghost | 0% (0/9) | 0% (0/43) | 0% (0/229) |
| Blinky | 0% (0/1) | 0% (0/4) | 0% (0/21) |
| Clyde | 0% (0/1) | 0% (0/4) | 0% (0/29) |
| GhostColor | 0% (0/1) | 0% (0/1) | 0% (0/5) |
| GhostFactory | 0% (0/1) | 0% (0/5) | 0% (0/6) |
| Inky | 0% (0/1) | 0% (0/5) | 0% (0/31) |
| Navigation | 0% (0/2) | 0% (0/11) | 0% (0/60) |
| NavigationTest | 0% (0/1) | 0% (0/9) | 0% (0/56) |
| Pinky | 0% (0/1) | 0% (0/4) | 0% (0/21) |
| Ghost | 0% (0/1) | 0% (0/4) | 0% (0/8) |



| | | | |
|---|---|---|---|
| npc | 40% (4/10) | 12% (6/47) | 6% (17/243) |
| ghost | 33% (3/9) | 11% (5/43) | 5% (12/235) |
| Blinky | 100% (1/1) | 50% (2/4) | 13% (3/22) |
| Clyde | 0% (0/1) | 0% (0/4) | 0% (0/31) |
| GhostColor | 100% (1/1) | 100% (1/1) | 100% (5/5) |
| GhostFactory | 100% (1/1) | 40% (2/5) | 57% (4/7) |
| Inky | 0% (0/1) | 0% (0/5) | 0% (0/32) |
| Navigation | 0% (0/2) | 0% (0/11) | 0% (0/60) |
| NavigationTest | 0% (0/1) | 0% (0/9) | 0% (0/56) |
| Pinky | 0% (0/1) | 0% (0/4) | 0% (0/22) |
| Ghost | 100% (1/1) | 25% (1/4) | 62% (5/8) |
| points | 0% (0/2) | 0% (0/7) | 0% (0/19) |

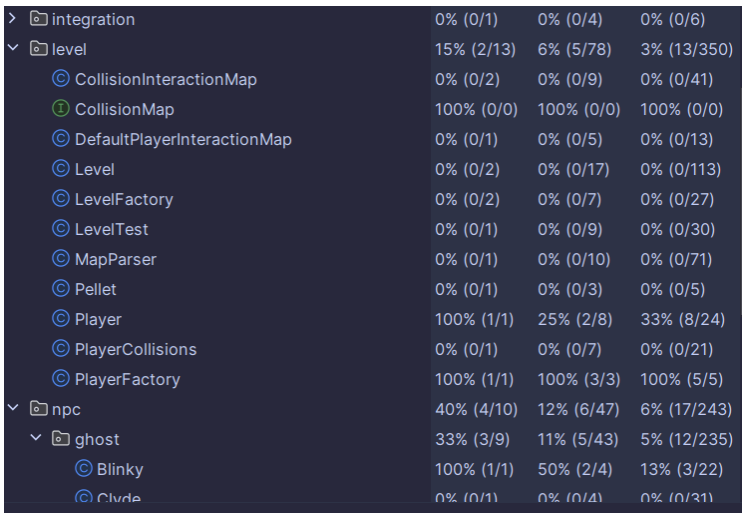
The second unit test is for the `getSprite()` method in the level package. This test verifies that the method returns the correct sprite that was set during the creation of the object pellet. Pellet requires a point and a sprite value, which I initialized before creating the object. I then tested that the method returns the same object that is equivalent to the object created. This test increased line coverage for the Pellet class to 83%.

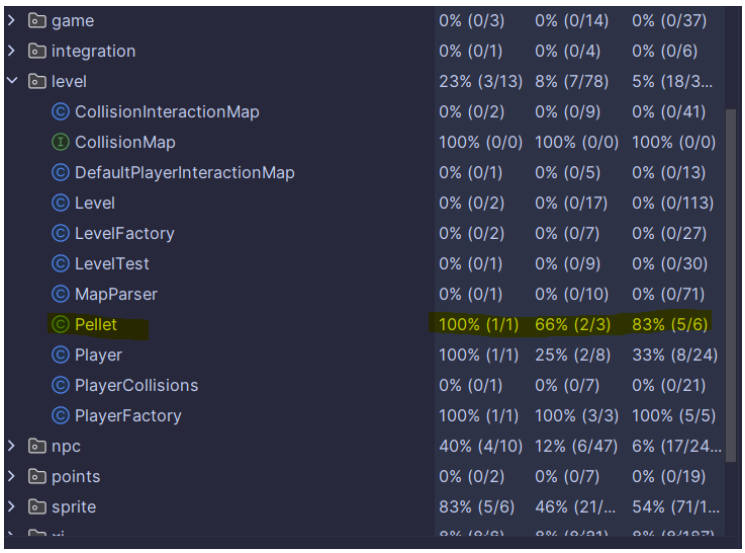
```java
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.sprite.Sprite;
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.AssertionsForClassTypes.assertThat;
import static org.mockito.Mockito.mock;

new *
public class PelletTest {
    1 usage
    private static final int POINTS = 10;
    2 usages
    private static final Sprite SPRITEY = mock(Sprite.class);
    1 usage
    private final Pellet pellet = new Pellet(POINTS, SPRITEY);
    new *
    @Test
    void testGetSprite(){
        assertThat(pellet.getSprite()).isEqualTo(SPRITEY);
    }
}
```



The last unit test covers the **`playerVersusPellet()`** method in the level package. In this test, we are verifying that the method updates the game state correctly when a player consumes a pellet. PointCalculator(), consumedAPellet(), and leaveSquare() all need to be checked for proper calls. Other objects were called as needed to instantiate the Player object, which was a necessary parameter for playerVersusPellet(). This test increased line coverage for the PlayerCollisions class to 21%

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

import nl.tudelft.jpacman.points.PointCalculator;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
new *
public class PlayerCollisionsTest {
    2 usages
    private final PointCalculator pointCalculator = mock(PointCalculator.class);
    1 usage
    private final PlayerCollisions playerCollisions = new PlayerCollisions(pointCalculator);
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private final PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    2 usages
    private final Player player = Factory.createPacMan();
    3 usages
    private final Pellet pellet  = mock(Pellet.class);
    new *
    @Test
    void testPlayerVersusPellet() {
        playerCollisions.playerVersusPellet(player, pellet);

        verify(pointCalculator).consumedAPellet(player, pellet);
        verify(pellet).leaveSquare();
    }
}
```

| | | | |
|---|---|---|---|
| integration | 0% (0/1) | 0% (0/4) | 0% (0/6) |
| level | 23% (3/13) | 8% (7/78) | 5% (18/35... |
| CollisionInteractionMap | 0% (0/2) | 0% (0/9) | 0% (0/41) |
| CollisionMap | 100% (0/0) | 100% (0/0) | 100% (0/0) |
| DefaultPlayerInteractionMap | 0% (0/1) | 0% (0/5) | 0% (0/13) |
| Level | 0% (0/2) | 0% (0/17) | 0% (0/113) |
| LevelFactory | 0% (0/2) | 0% (0/7) | 0% (0/27) |
| LevelTest | 0% (0/1) | 0% (0/9) | 0% (0/30) |
| MapParser | 0% (0/1) | 0% (0/10) | 0% (0/71) |
| Pellet | 100% (1/1) | 66% (2/3) | 83% (5/6) |
| Player | 100% (1/1) | 25% (2/8) | 33% (8/24) |
| PlayerCollisions | 0% (0/1) | 0% (0/7) | 0% (0/21) |
| PlayerFactory | 100% (1/1) | 100% (3/3) | 100% (5/5) |
| npc | 40% (4/10) | 12% (6/47) | 6% (17/24... |
| points | 0% (0/2) | 0% (0/7) | 0% (0/19) |

| | | | |
|---|---|---|---|
| integration | 0% (0/1) | 0% (0/4) | 0% (0/6) |
| level | 30% (4/13) | 11% (9/78) | 6% (24/35... |
| CollisionInteractionMap | 0% (0/2) | 0% (0/9) | 0% (0/41) |
| CollisionMap | 100% (0/0) | 100% (0/0) | 100% (0/0) |
| DefaultPlayerInteractionMap | 0% (0/1) | 0% (0/5) | 0% (0/13) |
| Level | 0% (0/2) | 0% (0/17) | 0% (0/113) |
| LevelFactory | 0% (0/2) | 0% (0/7) | 0% (0/27) |
| LevelTest | 0% (0/1) | 0% (0/9) | 0% (0/30) |
| MapParser | 0% (0/1) | 0% (0/10) | 0% (0/71) |
| Pellet | 100% (1/1) | 66% (2/3) | 83% (5/6) |
| Player | 100% (1/1) | 25% (2/8) | 33% (8/24) |
| PlayerCollisions | 100% (1/1) | 28% (2/7) | 21% (6/28) |
| PlayerFactory | 100% (1/1) | 100% (3/3) | 100% (5/5) |
| 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% (71/1... |

**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?

No, the coverage results were not similar. The branch coverage for the level package in JaCoCo is 58% and in IntelliJ, it's 30%. This could be due to differences in configuration, as one could exclude certain tests.

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

Yes, being able to see the source code was really helpful in identifying which sections have not been covered yet. This makes the process of writing tests a lot more efficient.

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

I prefer JaCoCo's report because it is easier to identify which branch has not been covered. It lists the methods and the specific lines as well.

**Task 4**

This method retrieves a random account data dictionary, creates an empty Account object, and sets its attributes. It then checks that they match the values in the dictionary.

```
73  # lines 34 - 35
74      def test_from_dict(self):
75          """ Test setting attributes from a dictionary """
76          data = ACCOUNT_DATA[self.rand] # get a random account
77          account = Account()
78          account.from_dict(data)
79
80          self.assertEqual(account.name, data["name"])
81          self.assertEqual(account.email, data["email"])
82          self.assertEqual(account.phone_number, data["phone_number"])
83          self.assertEqual(account.disabled, data["disabled"])
84          self.assertEqual(account.date_joined, data["date_joined"])
```

This method verifies that an account is being deleted correctly from the database. It checks that it no longer exists by ensuring that the number of records is equal to 0.

```
111 # lines 52 - 54
112     def test_delete_account(self):
113         """ Test deleting an account """
114         data = ACCOUNT_DATA[self.rand]  # get a random account
115         account = Account(**data)
116         account.create()
117         self.assertEqual(len(Account.all()), 1)
118
119         # Delete the account
120         account.delete()
121         self.assertEqual(len(Account.all()), 0)
122
```

This method checks that an account with a specific ID can be found correctly. There is also an edge case to ensure that finding an account with a non-existent ID returns None.

```
123     # lines 74 - 75
124         def test_find_account(self):
125             """ Test finding an account by ID """
126             data = ACCOUNT_DATA[self.rand]  # get a random account
127             account = Account(**data)
128             account.create()
129
130             found_account = Account.find(account.id)
131             self.assertIsNotNone(found_account)
132             self.assertEqual(found_account.id, account.id)
133
134         def test_find_account_not_found(self):
135             """ edge case """
136             found_account = Account.find(99999)
137             self.assertIsNone(found_account)
```

Results after tests:

```
----------------------------------------------------------------------
Ran 10 tests in 2.302s

FAILED (errors=1)
Name                    Stmts   Miss  Cover   Missing
-------------------------------------------------------
models\__init__.py          7      0   100%
models\account.py          40      0   100%
tests\test_account.py      84      0   100%
-------------------------------------------------------
TOTAL                     131      0   100%
```

## Task 5

First, I created `test_update_a_counter(self)` in order to verify the update functionality. I first created a counter that checks for `HTTP_201_CREATED`. Afterwards, I check that the initial value is 0 to confirm proper creation. Following that, I send a `PUT` request, which should update the counter. This should return `HTTP_200_OK` if executed correctly. Lastly, I retrieve the value of the updated value and verify that the `PUT` request incremented as it should.

```python
    def test_update_a_counter(self):  new *
        """It should update a counter"""
        result = self.client.post('/counters/tester')  # POST request for creation
        self.assertEqual(result.status_code, HTTP_201_CREATED)

        initial_value = COUNTERS['tester']
        self.assertEqual(initial_value, second: 0)  # check for proper creation

        result = self.client.put('/counters/tester')  # PUT request for update
        self.assertEqual(result.status_code, HTTP_200_OK)

        updated_value = COUNTERS['tester']
        self.assertEqual(updated_value, initial_value + 1)  # check for proper update

```

Here are my results after running nosetests:

```
(myenv37) PS C:\Users\Workstation\Desktop\cs 472\Lab2PleaseWork\tdd> nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates
- It should update a counter (FAILED)

======================================================================
FAIL: It should update a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Workstation\Desktop\cs 472\Lab2PleaseWork\tdd\tests\test_counter.py", line 73, in test_update_a_counter
    self.assertEqual(result.status_code, HTTP_200_OK)
AssertionError: 405 != 200
-------------------- >> begin captured logging << --------------------
test_counter: INFO: Request to create counter: tester
--------------------- >> end captured logging << ---------------------

Name             Stmts   Miss  Cover   Missing
------------------------------------------------
src\counter.py       7      7     0%   1-17
src\status.py        6      6     0%   2-7
------------------------------------------------
TOTAL               13     13     0%
```

There was a `405 != 200` error due to there being no endpoint.

I then implemented a new route `update_counter` by following the same structure as `create_counter`, but adjusted the method to use a `PUT` request and return code `200_OK`. It increments `COUNTER` if it exists, otherwise, it returns a `404` error

```python
@app.route('/counters/<name>', methods=['PUT'])  new *
def update_counter(name):
    """Update a counter"""
    app.logger.info(f"Request to update counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
        return {"Message": f"Counter {name} does not exist"}, HTTP_404_NOT_FOUND
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, HTTP_200_OK

```

Running `nosetests` again gives me no errors:

```
(myenv37) PS C:\Users\Workstation\Desktop\cs 472\Lab2PleaseWork\tdd> nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates
- It should update a counter

Name             Stmts   Miss  Cover   Missing
------------------------------------------------
src\counter.py       7      7     0%   1-17
src\status.py        6      6     0%   2-7
------------------------------------------------
TOTAL               13     13     0%
----------------------------------------------------------------------
Ran 3 tests in 0.455s

OK
```

Next, I wrote the test case for reading a counter. I made a `POST` request to create it, then a `GET` request to read the value. It should return `HTTP_200_OK` if successful. Afterwards, I check that the value is correct.

```
86      def test_read_a_counter(self):  new *
87          """It should read a counter"""
88          result = self.client.post('/counters/baz')  # POST request for creation
89          self.assertEqual(result.status_code, HTTP_201_CREATED)
90
91          result = self.client.get('/counters/baz')  # GET request for read
92          self.assertEqual(result.status_code, HTTP_200_OK)
93
94          data = result.get_json()
95          self.assertEqual(data['baz'], second: 0)  # check for correct value
96
```

Running `nosetests` gives me a `405 != 200` error again. I create the route for it and all my results come out green.

```
(myenv37) PS C:\Users\Workstation\Desktop\cs 472\Lab2PleaseWork\tdd> nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates
- It should read a counter
- It should update a counter
```