Unit Testing Report

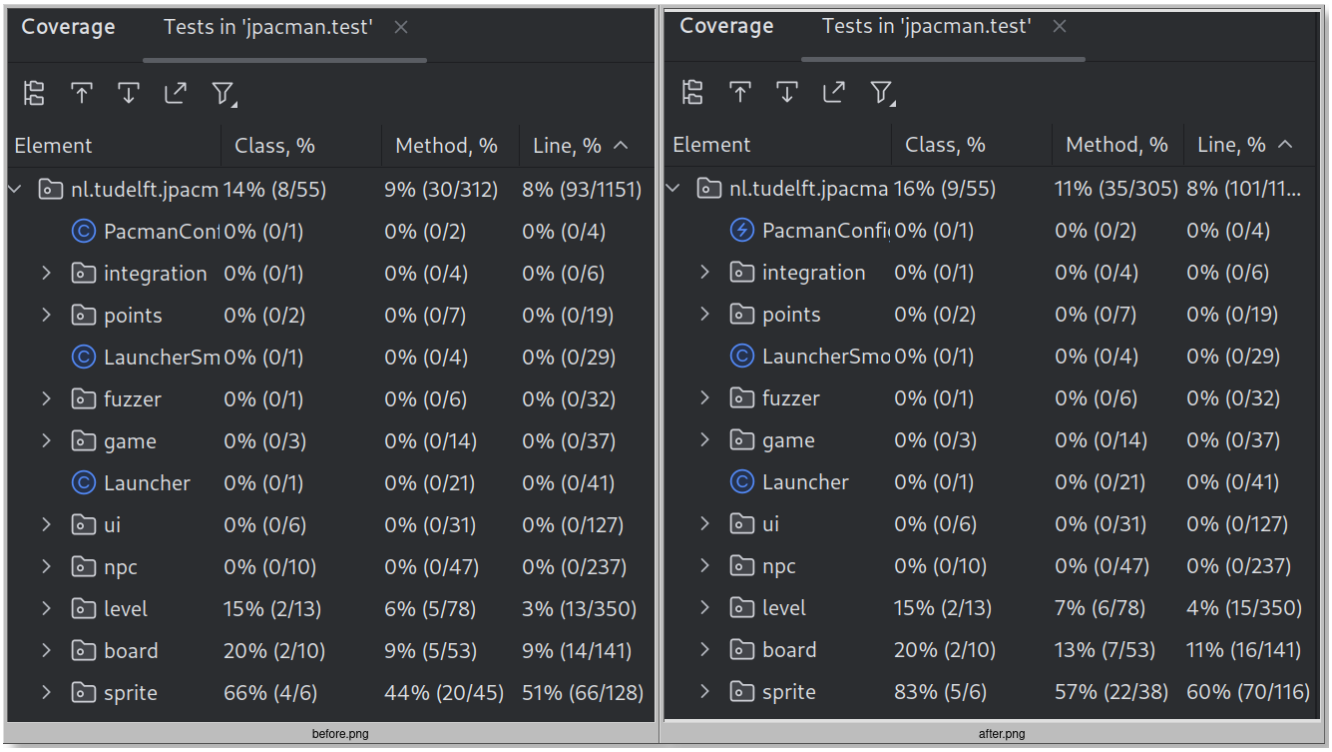
Introduction

In my fork of jpacman, I created unit tests for:

`src/main/java/nl/tudelft/jpacman/Board/Direction (Checking correct delta when player moves)`
`src/main/java/nl/tudelft/jpacman/level/Player.isAlive`
`src/main/java/nl/tudelft/jpacman/level/Player.getSprite`
`src/main/java/nl/tudelft/jpacman/sprite/ImageSprite.getWidth`
`src/main/java/nl/tudelft/jpacman/sprite/ImageSprite.getHeight`

jpacman Coverage

The test coverage both before and after I implemented the unit tests are shown below:



This increase in coverage is due to the following unit tests:

Listing 1: src/main/java/nl.tudelft.jpacman/Board/Direction

```
@Test
void testWest() {
    Direction west = Direction.valueOf("WEST");
    assertThat(west.getDeltaX()).isEqualTo(-1);
}

@Test
void testSouth() {
    Direction south = Direction.valueOf("SOUTH");
    assertThat(south.getDeltaY()).isEqualTo(1);
}

@Test
void testEast() {
```

```
    Direction east = Direction.valueOf("EAST");
    assertThat(east.getDeltaX()).isEqualTo(1);
}
```

Listing 2: src/main/java/nl.tudelft.jpacman/level/Player.getSprite

```
/**
  * Test that pacman is displaying the correct sprite depending on direction
  */
@Test
void testPlayerGetSprite() {
    Direction playerDirection = ThePlayer.getDirection();
    Sprite expectedSprite = SPRITE_STORE.getPacmanSprites().get(playerDirection);
    assertThat(ThePlayer.getSprite()).isEqualTo(expectedSprite);

    // For full coverage, we test when not alive as well
    ThePlayer.setAlive(false);

    assertThat(ThePlayer.getSprite()).isEqualTo(SPRITE_STORE.getPacManDeathAnimation());
}
```

Listing 3: src/main/java/nl/tudelft/jpacman/sprite/Sprite.getWidth/getHeight

```
package nl.tudelft.jpacman.sprite;

import static org.assertj.core.api.Assertions.assertThat;

import org.junit.jupiter.api.Test;

import java.io.IOException;

/**
 * A simple test class for testing getWidth and getHeight for sprites
 *
 * @author Ethan Hunt
 */
public class SpriteTest {

    final private SpriteStore spritestore = new SpriteStore();
    // attempt to load the pacman sprite
    final private Sprite sprite;

    {
        try {
            sprite = spritestore.loadSprite("/sprite/pacman.png");
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    @Test
    public void testGetSpriteWidth() {
        assertThat(sprite.getWidth()).isEqualTo(64);
    }

    @Test
    public void testGetSpriteHeight() {
```

```
        assertThat(sprite.getHeight()).isEqualTo(64);
    }

}
```

JaCoCo Report

The coverage results from the JaCoCo report are largely the same as the information provided by IntelliJ with some very minor differences. This could be simply due to differences in what is considered "covered" by each respective service.
That said, I found the information from the JaCoCo report much better since it also takes branch coverage into consideration. Likewise, IntelliJ runs very slow on my computer, whereas the html is very snappy in comparison, which makes looking at coverage much easier.
The JaCoCo report from a top level is shown below.

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		66%		55%	77	155	108	344	22	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		86%		59%	30	70	11	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	1,226 of 4,694	73%	297 of 637	53%	296	590	233	1,039	52	268	6	47

Working with Python Test Coverage

For this task, I achieved 100% code coverage with the following tests.

```
"""
Test Cases TestAccountModel
"""
import json
from random import randrange
from unittest import TestCase
from models import db, app
from models.account import Account, DataValidationError

ACCOUNT_DATA = {}

class TestAccountModel(TestCase):
    """Test Account Model"""

    @classmethod
    def setUpClass(cls):
        """ Load data needed by tests """
        db.create_all()  # make our sqlalchemy tables
        global ACCOUNT_DATA
        with open('tests/fixtures/account_data.json') as json_data:
            ACCOUNT_DATA = json.load(json_data)

    @classmethod
    def tearDownClass(cls):
        """Disconnext from database"""
```

```
        db.session.close()

    def setUp(self):
        """Truncate the tables"""
        self.rand = randrange(0, len(ACCOUNT_DATA))
        db.session.query(Account).delete()
        db.session.commit()

    def tearDown(self):
        """Remove the session"""
        db.session.remove()

    ######################################################################
    #  T E S T   C A S E S
    ######################################################################

    def test_create_all_accounts(self):
        """ Test creating multiple Accounts """
        for data in ACCOUNT_DATA:
            account = Account(**data)
            account.create()
        self.assertEqual(len(Account.all()), len(ACCOUNT_DATA))

    def test_create_an_account(self):
        """ Test Account creation using known data """
        data = ACCOUNT_DATA[self.rand] # get a random account
        account = Account(**data)
        account.create()
        self.assertEqual(len(Account.all()), 1)

    def test_repr(self):
        """ Test the representation of an account """
        account = Account()
        account.name = "Foo"
        self.assertEqual(str(account), "<Account 'Foo'>")

    def test_to_dict(self):
        """ Test account to dict """
        data = ACCOUNT_DATA[self.rand] # get a random account
        account = Account(**data)
        result = account.to_dict()
        self.assertEqual(account.name, result["name"])
        self.assertEqual(account.email, result["email"])
        self.assertEqual(account.phone_number, result["phone_number"])
        self.assertEqual(account.disabled, result["disabled"])
        self.assertEqual(account.date_joined, result["date_joined"])

    def test_from_dict(self):
        """ Test account from dict """
        dict = {
            "name": "Albert Einstein",
            "email": "albert@einstein.com",
            "phone_number": "123-456-7890",
            "disabled": "false",
            "date_joined": "02/03/2024"
        }
```

```
        account = Account()

        account.from_dict(dict)

        self.assertEqual(account.name, dict["name"])
        self.assertEqual(account.email, dict["email"])
        self.assertEqual(account.phone_number, dict["phone_number"])
        self.assertEqual(account.disabled, dict["disabled"])
        self.assertEqual(account.date_joined, dict["date_joined"])

    def test_update_an_account(self):
        """ Test Account update using known data """
        data = ACCOUNT_DATA[self.rand]
        account = Account(**data)

        # With no id
        account.id = None
        with self.assertRaises(DataValidationError):
            account.update()

        account.id = 1231
        account.email = "test@example.com"
        account.update()

        self.assertEqual(account.email, "test@example.com")

    def test_delete_an_account(self):
        """ Test Account deletion """
        data = ACCOUNT_DATA[self.rand]
        account = Account(**data)
        account.create()

        account.delete()
        self.assertEqual(len(Account.all()), 0)

    def test_find_account(self):
        data = ACCOUNT_DATA[self.rand]
        account = Account(**data)
        account.create()
        account.id = 1135

        self.assertEqual(Account.find(1), account)
```

After implementing the above and running `nosetests`, the following is shown.

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test Account deletion
- find account
- Test account from dict
- Test the representation of an account
- Test account to dict
- Test Account update using known data

Name                 Stmts   Miss  Cover   Missing
------------------------------------------------
```

```
models/__init__.py       7      0   100%
models/account.py       40      0   100%
----------------------------------------------------
TOTAL                   47      0   100%
----------------------------------------------------------------------
Ran 8 tests in 0.529s

OK
```

Test Driven Development

Following the red/green/refactor phases, I implemented the following test cases.

```python
def test_update_a_counter(self):
    """It should update a counter"""
    # Lets test updating a counter which doesn't exist
    result = self.client.put('/counters/baz')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)

    result = self.client.post('/counters/baz')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    # Check counter value (should be 0 for newly created)
    self.assertEqual(result.json["baz"], 0)

    result = self.client.put('/counters/baz')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    # Check that counter++
    self.assertEqual(result.json["baz"], 1)

def test_get_a_counter(self):
    """It should get a counter"""
    result = self.client.get('/counters/nonexistant')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)

    result = self.client.post('/counters/foobar')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    result = self.client.get('/counters/foobar')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
```

Here I was in the red phase and got the following output after running `nosetests`

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

======================================================================
FAIL: It should get a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/ethan/Documents/CS_472/testing/tdd/tests/test_counter.py", line 59,
in test_get_a_counter
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
AssertionError: 405 != 404

======================================================================
```

```
FAIL: It should update a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/ethan/Documents/CS_472/testing/tdd/tests/test_counter.py", line 44,
in test_update_a_counter
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
AssertionError: 405 != 404

Name             Stmts   Miss  Cover   Missing
------------------------------------------------
src/counter.py      11      0   100%
src/status.py        6      0   100%
------------------------------------------------
TOTAL               17      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.073s

FAILED (failures=2)
```

This shows a status returned of `HTTP_405_METHOD_NOT_ALLOWED` which is expected since I hadn't yet implemented the corresponding routes for PUT or GET yet.

After this, I created the following in counter.py.

```python
@app.route('/counters/<name>', methods=['PUT'])
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} doesn't exist"}, status.HTTP_404_NOT_FOUND
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK

@app.route('/counters/<name>', methods=['GET'])
def get_counter(name):
    """Get a counter"""
    app.logger.info(f"Request to get counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
        return {"Message":f"Counter {name} doesn't exist"}, status.HTTP_404_NOT_FOUND
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

Finally, running `nosetests` gave me this output, signalling that all test cases passed.

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

Name             Stmts   Miss  Cover   Missing
------------------------------------------------
src/counter.py      24      0   100%
src/status.py        6      0   100%
------------------------------------------------
TOTAL               30      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.072s
```

OK

Ethan Hunt
Github Fork: https://github.com/RedPenguin88/jpacman