

TASK 1:

Initial Test Coverage: the coverage is not good enough.

TASK 2 & 2.1:

Before Coverage:

Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.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)

After isAlive() unit test:

Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (93/1151)
> 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	66% (4/6)	44% (20/45)	51% (66/128)
> 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)

After first test:

Element ▾	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	16% (9/55)	9% (31/312)	8% (94/115...
⚙ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)
🕒 LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
🕒 Launcher	0% (0/1)	0% (0/21)	0% (0/41)
> 📁 ui	0% (0/6)	0% (0/31)	0% (0/127)
> 📁 sprite	66% (4/6)	44% (20/45)	51% (66/12...
> 📁 points	0% (0/2)	0% (0/7)	0% (0/19)
> 📁 npc	0% (0/10)	0% (0/47)	0% (0/237)
> 📁 level	15% (2/13)	6% (5/78)	3% (13/350)
> 📁 integration	0% (0/1)	0% (0/4)	0% (0/6)
> 📁 game	0% (0/3)	0% (0/14)	0% (0/37)
> 📁 fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> 📁 board	30% (3/10)	11% (6/53)	10% (15/142)

```
package nl.tudelft.jpacman.board;

import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;

new *
public class WithinBordersTest {
    1 usage
    int x = -1;
    1 usage
    int y = -1;
    no usages
    int z = 23;
    no usages
    int w = 20;

    1 usage
    Board testBoard = mock(Board.class);

    new *
    @Test
    void testBorders() {
        assertThat(testBoard.withinBorders(x,y)).isEqualTo(expected: false);
    }
}
```

For testing the method `withinBorders()` in the `board` package, the goal was to set parameters where the method would return false. In this case, the parameters should not be negative, so I tested negative values. The test method would assert that the return value of `withinBorders()` would return false.

After second test:

Element ▾	Class, %	Method, %	Line, %
<ul style="list-style-type: none"> nl.tudelft.jpacman <ul style="list-style-type: none"> PacmanConfigurationException LauncherSmokeTest Launcher ui sprite points npc level integration game fuzzer board 	20% (11/55) 0% (0/1) 0% (0/1) 0% (0/1) 0% (0/6) 66% (4/6) 50% (1/2) 0% (0/10) 23% (3/13) 0% (0/1) 0% (0/3) 0% (0/1) 30% (3/10)	11% (37/312) 0% (0/2) 0% (0/4) 0% (0/21) 0% (0/31) 46% (21/45) 14% (1/7) 0% (0/47) 11% (9/78) 0% (0/4) 0% (0/14) 0% (0/6) 11% (6/53)	9% (105/115...) 0% (0/4) 0% (0/29) 0% (0/41) 0% (0/127) 52% (67/128) 10% (2/20) 0% (0/237) 5% (21/351) 0% (0/6) 0% (0/37) 0% (0/32) 10% (15/142)

```
package nl.tudelft.jpacman.points;
import nl.tudelft.jpacman.level.Pellet;
import nl.tudelft.jpacman.level.Player;
import nl.tudelft.jpacman.level.PlayerFactory;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

new *
public class consumedAPelletTest {
    2 usages
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    3 usages
    private Player ThePlayer = Factory.createPacMan();
    1 usage
    int points = 200;
    1 usage
    Pellet createdPellet = new Pellet(points, SPRITE_STORE.getPelletSprite());
    1 usage
    DefaultPointCalculator pointCalculator = new DefaultPointCalculator();

    new *
    @Test
    void testScore(){
        assertThat(ThePlayer.getScore()).isEqualTo( expected: 0);
        pointCalculator.consumedAPellet(ThePlayer, createdPellet);
        assertThat(ThePlayer.getScore()).isEqualTo( expected: 200);
    }
}
```

For testing the method `consumedAPellet()` in the `points` package, the goal was to ensure that the right number of points is calculated after “consuming a pellet.” I first instantiated the necessary object to create a `Player`, and also a `Pellet` object. By creating a `DefaultPointCalculator`, I was able to call the `consumedAPellet()`, and assert that the player has 0 points before consuming a pellet and 200 points after consuming, since each pellet had a value of 200.

After third test:

Element	Class, %	Method...	Line, %
nl.tudelft.jpacman	21% (12/...	12% (38/...	9% (106/...
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
ui	0% (0/6)	0% (0/31)	0% (0/127)
sprite	83% (5/6)	48% (22/...	53% (68/...
points	50% (1/2)	14% (1/7)	10% (2/20)
npc	0% (0/10)	0% (0/47)	0% (0/23...
level	23% (3/13)	11% (9/78)	5% (21/3...
integration	0% (0/1)	0% (0/4)	0% (0/6)
game	0% (0/3)	0% (0/14)	0% (0/37)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
board	30% (3/10)	11% (6/53)	10% (15/1...

```
1 package nl.tudelft.jpacman.sprite;
2 import org.junit.jupiter.api.Test;
3 import static org.assertj.core.api.AssertionsForClassTypes.assertThat;
4 import static org.mockito.Mockito.mock;
5 import static org.mockito.Mockito.when;
6
7 new *
8 public class EmptySpritesplitTest {
9     2 usages
10    int w=1;
11    2 usages
12    int x=2;
13    2 usages
14    int y=3;
15    2 usages
16    int z=4;
17    2 usages
18    EmptySprite expectedSprite;
19    2 usages
20    EmptySprite blank = mock(EmptySprite.class);
21    new *
22    @Test
23    void TestSplit() {
24        when(blank.split(w,x,y,z)).thenReturn(expectedSprite);
25        Sprite actualSprite = blank.split(w,x,y,z);
26        assertThat(actualSprite).isEqualTo(expectedSprite);
27    }
28 }
```

For testing the method `split()` in the `sprite` package, the goal was to successfully duplicate the same `Sprite` object that was returned in said method. To do this, I used isolation to determine if invoking the method isolated would return the same object as doing it not isolated. Then, I would assert that both returned `Sprite` objects would be equal.

TASK 3:

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		67%		57%	74	155	104	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		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	1,213 of 4,694	74%	293 of 637	54%	293	590	229	1,039	51	268	6	47

There are some areas where the Jacoco and IntelliJ are similar, but other areas are not. I think this is because of the branch calculations that are included in Jacoco.

Yes, I find it very helpful that Jacoco visualizes the uncovered branch coverage. It seems that I still have a lot of branches to cover.

I prefer IntelliJ's coverage window more. Although the bar visualization is helpful, having to open the Jacoco .html repeatedly seems like a hassle. Also, IntelliJ's is straight-to-the-point, I know exactly where my unit tests have covered.

TASK 4:

```
new *
def test_from_dict(self):
    """ Test account from dict """
    data = {'name': 'Jennifer Smith',
            'email': 'stevensjennifer@example.org',
            'phone_number': '351.317.1639x79470',
            'disabled': True}
    account = Account(**data)
    account.from_dict(data)
    self.assertEqual(account.name, 'Jennifer Smith')
    self.assertEqual(account.email, 'stevensjennifer@example.org')
    self.assertEqual(account.phone_number, '351.317.1639x79470')
    self.assertEqual(account.disabled, True)
```

```
new *
def test_update_valid(self):
    """ Test updating an account """
    data = {'name': 'Jennifer Smith',
            'email': 'stevensjennifer@example.org',
            'phone_number': '351.317.1639x79470',
            'disabled': True}
    account = Account(**data)
    account.from_dict(data)
    account.create()
    account.phone_number = '351.317.1639x79479'
    account.update()
    self.assertNotEqual(account.phone_number, '351.317.1639x79470')
```

```
new *
def test_update_empty(self):
    emptyaccount = Account()
    with self.assertRaises(DataValidationError):
        emptyaccount.update()
```

```

new *
def test_delete(self):
    data = {'name': 'Jennifer Smith',
            'email': 'stevensjennifer@example.org',
            'phone_number': '351.317.1639x79470',
            'disabled': True}
    account = Account(**data)
    account.from_dict(data)
    account.create()
    account.phone_number = '351.317.1639x79479'
    account = account.delete()
    self.assertIsNone(account)

```

```

new *
def test_find(self):
    data = {'name': 'Jennifer Smith',
            'email': 'stevensjennifer@example.org',
            'phone_number': '351.317.1639x79470',
            'disabled': True}
    account = Account(**data)
    account.from_dict(data)
    account.create()
    retval = Account.find(account.id)
    self.assertEqual(retval.id, account.id)

```

Name	Stmts	Miss	Cover	Missing
models__init__.py	7	0	100%	
models\account.py	40	0	100%	
TOTAL	47	0	100%	

Ran 9 tests in 0.785s

OK

TASK 5:

```
new *
def test_update_a_counter(self):
    """It should update a counter"""
    result = self.client.post('/counters/cat')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    baseline = COUNTERS['cat']
    result = self.client.put('/counters/cat')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(baseline+1, COUNTERS['cat'])
```

```
lime1@WS-5 MINGW64 ~/GIT CLONES/tdd (main)
$ 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\lime1\GIT CLONES\tdd\tests\test_counter.py", line 45, in test_update_a_counter
    self.assertEqual(result.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
```

For making `test_update_a_counter`, I first made a counter by calling the `post()` method with the name 'cat'. I then asserted that the counter was made successfully. Then I made the variable 'baseline' to keep track of the counter value, ensuring that `COUNTERS` was imported. Using the `put()` method to update the counter, I then asserted that the `.put` was successful and that the counter value was +1 compared to the baseline. The exception that was shown was the `AssertionError: 405 != 200`, and that a counter should be updated.


```
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update the counter"""
    app.logger.info(f"Request to update counter: {name}")
    global COUNTERS
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

```
lime1@WS-5 MINGW64 ~/GIT CLONES/tdd (main)
$ nosetests
```

Counter tests

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

For making `update_counter`, I realized that this function and the `create_counter` method share similarities. Instead of initializing `COUNTERS[name]` to 0, I would just increment it by 1. Additionally, the method would check for PUT. No exceptions were shown

```
def test_read_a_counter(self):
    """It should read a counter"""
    result = self.client.post('/counters/chameleon')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    baseline = COUNTERS['chameleon']
    result = self.client.get('/counters/chameleon')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(baseline, COUNTERS['chameleon'])
```


```
limel@WS-5 MINGW64 ~/GIT CLONES/tdd (main)
$ nosetests

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

=====
FAIL: It should read a counter
-----

Traceback (most recent call last):
  File "C:\Users\limel\GIT CLONES\tdd\tests\test_counter.py", line 54, in test_read_a_counter
    self.assertEqual(result.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
```

For `test_read_a_counter`, I made a new counter, and asserted that the counter was made successfully. I then held the counter value in a new variable. Next, I used the HTTP GET method to read in the data. After that, I asserted that the read was successful and that the baseline was equal to the counter value that was just read in. The exception that was shown was also the `AssertionError: 405 != 200`, and that a counter should be read.

```
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Update the counter"""
    app.logger.info(f"Request to read counter: {name}")
     global COUNTERS
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

```
limel@WS-5 MINGW64 ~/GIT CLONES/tdd (main)
$ 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

For making read_counter, it was almost the same as the other functions, other than the fact that COUNTERS[name] would not change. The method would check for 'GET' in this case. No exceptions were shown.