

Pre-Conditions Report

Fork Repository Link: <https://github.com/cpilande/jpacman>

Task 2.1

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	3.6% (2/55)	1.5% (5/326)	1.2% (14/1161)

As seen in the above screenshot, before adding unit tests, the test coverage values were quite small. I decided to test the following methods.

```
@Test
void testWithinBordersIsFalse() {
    assertThat(board.withinBorders(x: 1, y: 1)).isEqualTo(expected: false);
}
```

The method **WithinBorders()** checks whether the selected x and y values are within the borders of the board. In this unit test, I checked the sad path where the coordinates are out of bounds.

```
@Test
void testConsumedAPelletUpdateScore() {
    calc.consumedAPellet(player, pellet2);
    assertEquals(player.getScore(), testScore);
}
```

The method **ConsumedAPellet()** is called when a player comes into contact with a pellet, updating the player's score. In this unit test, I checked whether the player's score was updated correctly and matches the test score.

```

@Test
void testAddPointsBy5() {
    ThePlayer.addPoints(5);
    assertEquals(ThePlayer.getScore(), actual: 5);
}

```



















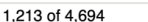

The method **testAddPointsBy5()** adds a value of 5 to the player's score to check if it is updating correctly. In this unit test, I checked whether the player's score was updated correctly.

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	21.8% (12/55)	11.7% (38/326)	8.9% (105/1177)

After adding the unit tests, as seen above, my coverage increased significantly in percentage.

Task 3

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

The coverage results are similar. For example, the Sprite package had similar coverage around 84%. However, there were different results related to branch coverage since IntelliJ does not show what branch coverage results are.

I did find the source code visualization helpful on uncovered branches because it gave me more insight on whether my tests are covering all possible avenues my program can take. This is useful because I can know the probability of a situation occurring that I did not test for and work on decreasing that probability. I preferred the JaCoCo visualization because it made it easier to understand how many more tests I need to add and how much coverage I already have.

Task 4

```
def test_update(self):
    data = ACCOUNT_DATA[self.rand]

    #create a test account with ID 24
    testAccount = Account()
    testAccount = Account(**data)
    testAccount.id=24
    testAccount = testAccount.to_dict()

    #create new account, update with id of 24 also
    account = Account(**data)
    account.id=24 #should access testAccount
    account.update()

    #checking if they are equal
    self.assertEqual(account.name, testAccount["name"])
    self.assertEqual(account.email, testAccount["email"])
    self.assertEqual(account.phone_number, testAccount["phone_number"])
    self.assertEqual(account.disabled, testAccount["disabled"])
    self.assertEqual(account.date_joined, testAccount["date_joined"])

def test_update_emptyIDfield(self):
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)

    #checks that the data validation error was raised
    with self.assertRaises(DataValidationError):
        account.update()
```

The method **update()** updates an account in the database, given a specific ID. In this unit test, I created two accounts with an ID of 24. When I update the new account, it should match the test account since they have the same id. When I check, both accounts have equal values. I also added a check for a sad path when the ID field is empty to ensure that a Data Validation Error is raised.

```
def test_find(self):
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.id = 5
    testAccount = Account()
    testAccount = Account.find(5)

    #Can't find an instance with ID of 5 here, returns none
    self.assertEqual(testAccount, None)
```

find() is used to find a specific account in the class. In this test, I search for an account with an ID of 5. Since there is no instance of 5 here, then no account is returned, meaning that the function works correctly.

```
def test_from_dict(self):
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    #from the dictionary, creating the account
    account = account.to_dict()

    data2 = ACCOUNT_DATA[self.rand] # get a random account
    testAccount = Account(**data2)
    #using .from_dict to create the account
    testAccount.from_dict(account)

    #checking if they are equal
    self.assertEqual(testAccount.name, account["name"])
    self.assertEqual(testAccount.email, account["email"])
    self.assertEqual(testAccount.phone_number, account["phone_number"])
    self.assertEqual(testAccount.disabled, account["disabled"])
    self.assertEqual(testAccount.date_joined, account["date_joined"])
```

from_dict() sets attributes from a dictionary. In this test, I created two accounts. I created a dictionary from the first account. From that dictionary, I set the attributes of the second account. I then check if these two accounts are the same because that means the attributes were set correctly.

```

def test_delete(self):
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()
    prevSize = len(Account.all())

    account.delete()

    self.assertEqual(prevSize - 1, len(Account.all()))

```

`delete()` removes an account from the database. In this test, I created an account and recorded the size of the number of accounts. I then deleted that account and checked if the size equalled the previous size minus one account and the length of all accounts right now.

```

(base) carmelapilande@Carmelas-MBP test_coverage % nosetests
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- delete
- find
- from dict
- Test the representation of an account
- Test account to dict
- update
- update emptyIDfield

Name                               Stmts   Miss  Cover    Missing
-----
models/__init__.py                  6       0   100%
models/account.py                  46       0   100%
-----
TOTAL                               52       0   100%
-----
Ran 9 tests in 0.261s

OK

```

After adding the unit tests, the program achieved 100% coverage and all tests passed.

Task 5

In this test, I checked **update()** by creating a new counter to update. I checked the status code for successful creation, and if it was valid, I called the PUT request. To compare the results, I manually incremented the result value.

```
def test_update_a_counter(self):
    client = app.test_client()
    result = client.post('/counters/new')

    result["new"] = result["new"] + 1 # updates the dictionary value by 1

    if result.status_code == status.HTTP_201_CREATED:
        result2 = self.client.put('/counters/new') #calls update function

        self.assertEqual(result2.status_code, status.HTTP_200_OK)
```

```
Traceback (most recent call last):
  File "/Users/carmelapilande/Documents/GitHub/tdd/tests/test_counter.py", line 44, in test_update_a_counter
    result["new"] = result["new"] + 1 # updates the dictionary value by 1
TypeError: 'WrapperTestResponse' object is not subscriptable
```

I received an exception though because the WrapperTestResponse object is not subscriptable. As a result, I **REFACTORED** the code to include **json.loads** to parse the JSON string into a Python dictionary.

After adding the correct parsing, I received an Assertion Error and entered the **RED** phase since the test did not pass successfully.

```
def test_update_a_counter(self):
    client = app.test_client()
    result = client.post('/counters/new')

    check = json.loads(result.data)
    check["new"] = check["new"] + 1 # updates the dictionary value by 1

    if result.status_code == status.HTTP_201_CREATED:
        result2 = self.client.put('/counters/new') #calls update function

        self.assertEqual(result2.status_code, status.HTTP_200_OK)

        actual = json.loads(result2.data)

        self.assertEqual(actual, check)
```

```
Traceback (most recent call last):
  File "/Users/carmelapilande/Documents/GitHub/tdd/tests/test_counter.py", line 50, in test_update_a_counter
    self.assertEqual(result2.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
----- >> begin captured stdout << -----
{'foo': 0, 'bar': 0, 'new': 0}
```

This is because I had to create the update endpoint in **counter.py** to increment the counter and return a successful status. After doing so, the tests passed, and I entered the **GREEN** phase.

```
def update_counter(name):  
    COUNTERS[name] = COUNTERS[name] + 1  
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

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


| Name           | Stmts | Miss | Cover | Missing |
|----------------|-------|------|-------|---------|
| src/counter.py | 16    | 0    | 100%  |         |
| src/status.py  | 6     | 0    | 100%  |         |
| TOTAL          | 22    | 0    | 100%  |         |

  
Ran 3 tests in 0.094s  
OK
```

In the following test, I created a new counter, created a GET request for it, and checked if the status was successful. An assertion error occurred since no endpoint was created, making the program enter the **RED** phase.

```
def test_read_counter(self):  
    client = app.test_client()  
    result = client.post('/counters/new2')  
    result = client.get('/counters/new2')  
  
    self.assertEqual(result.status_code, status.HTTP_200_OK)
```

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

Name	Stmts	Miss	Cover	Missing
src/counter.py	16	0	100%	
src/status.py	6	0	100%	
TOTAL	22	0	100%	

```
=====
```

FAIL: test_read_counter (test_counter.CounterTest)

Traceback (most recent call last):
 File "/Users/carmelapilande/Documents/GitHub/tdd/tests/test_counter.py", line 61, in test_read_counter
 self.assertEqual(result.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200

I then created the endpoint where if the name was present in the dictionary, the name was printed and the counter and success status returned. This allowed us to enter the **GREEN** phase.

```
@app.route('/counters/<name>', methods=['GET'])

def read_counter(name):
    if name in COUNTERS.keys():
        print(COUNTERS[name])
        return {name: COUNTERS[name]}, status.HTTP_200_OK
```

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

Name	Stmts	Miss	Cover	Missing
src/counter.py	21	0	100%	
src/status.py	6	0	100%	
TOTAL	27	0	100%	

```
Ran 4 tests in 0.093s

OK
```

Finally, I **REFACTORED** the code in order to test for when a GET request is tried on a counter that does not exist. An assertion error is also created here since there is no NOT FOUND status returned yet in the endpoint. This puts us in the **RED** phase.

```
def test_read_counter(self):
    client = app.test_client()
    result = client.post('/counters/new2')
    result = client.get('/counters/new2')

    self.assertEqual(result.status_code, status.HTTP_200_OK)

    result2 = client.get('/counters/doesnotexist')
    self.assertEqual(result2.status_code, status.HTTP_404_NOT_FOUND)
```



```
def read_counter(name):
    if name in COUNTERS.keys():
        print(COUNTERS[name])
        return {name: COUNTERS[name]}, status.HTTP_200_OK

    return name, status.HTTP_404_NOT_FOUND
```

After updating the `read_counter` function, we enter the **GREEN** phase again, all tests are passed, and the coverage is at 100%.

Counter tests

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

Name	Stmts	Miss	Cover	Missing
src/counter.py	21	0	100%	
src/status.py	6	0	100%	
TOTAL	27	0	100%	

Ran 4 tests in 0.093s

OK