

Testing Lab

Task 2.1: Coverage on JPacman

Before beginning the project we were at nearly 0% test coverage. When Implementing the given function, the overall class test coverage was bumped up from 3% to 8%.

Coverage Tests in 'jpacman.test' x			
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)

Coverage Tests in 'jpacman.test' x			
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)
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	0% (0/1)	0% (0/3)	0% (0/5)
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	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)

One of the test coverage implementations I covered was the **setKiller()** function in the Player.java file. Under the same PlayerTest.java file we created with the given code I added the function **test_setKiller()**. Here I simply created a test ghost which I set as the killer to the player. I then made sure that the setKiller function actually returned the test ghost.

```
public class PlayerTest {  
    1 usage  
    private static final PacManSprites SPRITE_STORE = new PacManSprites();  
    1 usage  
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);  
    3 usages  
    private Player ThePlayer = Factory.createPacMan();  
  
    2 usages  
    private Ghost testGhost;  
  
    new *  
    @Test  
    void testAlive(){  
        |    assertThat(ThePlayer.isAlive()).isEqualTo( expected: true);  
    }  
    new *  
    @Test  
    void test_setKiller(){  
        |    ThePlayer.setKiller(testGhost);  
        |    assertThat(ThePlayer.getKiller()).isEqualTo(testGhost);  
    }  
}
```

This brought up the method coverage in the Player to 50%.

✓ nl.tudelft.jpacman	14% (8/55)	10% (32/312)	8% (96/1151)
> 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)	8% (7/78)	4% (16/350)
CollisionInteractor	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteract	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	0% (0/1)	0% (0/3)	0% (0/5)
Player	100% (1/1)	50% (4/8)	45% (11/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
> 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)
> ...	0% (0/0)	0% (0/0)	0% (0/0)

After taking a look at the Board section I noticed that there was an example test of the direction North. This test made sure that the direction value along with its delta value actually went hand-in-hand. Having looked at this I made two more test cases for the directions South and East. I called these two functions **testSouth()** and **testEast()**. I followed the same format but noticed that if North contained the value -1, South would have to be the value 1. While I did not implement the testWest function the same would have applied for the east and west values. This made the method coverage for the Direction file jump from 50% to 100%.

```
22
23     new *
24     @Test
25     void testSouth() {
26         Direction south = Direction.valueOf(name: "SOUTH");
27         assertThat(south.getDeltaY()).isEqualTo(expected: 1);
28     }
29     new *
30     @Test
31     void testEast() {
32         Direction east = Direction.valueOf(name: "EAST");
33         assertThat(east.getDeltaX()).isEqualTo(expected: 1);
34     }
35 }
```

Before testEast() and testSouth() were implemented:

Element ^	Class, %	Method, %	Line, %
▼ nl.tudelft.jpacman	20% (11/55)	11% (34/307)	8% (97/1129)
▼ board	40% (4/10)	11% (6/53)	10% (15/141)
Board	100% (1/1)	14% (1/7)	5% (1/18)
BoardFactory	0% (0/3)	0% (0/11)	0% (0/26)
BoardFactoryTest	0% (0/1)	0% (0/6)	0% (0/18)
BoardTest	0% (0/1)	0% (0/3)	0% (0/3)
Direction	100% (1/1)	50% (2/4)	81% (9/11)
Square	100% (1/1)	12% (1/8)	4% (1/23)
SquareTest	0% (0/1)	0% (0/4)	0% (0/13)
Unit	100% (1/1)	20% (2/10)	13% (4/29)
> 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)	10% (8/73)	4% (16/328)
> 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 testEast() and testSouth() were implemented:

Element ^	Class, %	Method, %	Line, %
▼ nl.tudelft.jpacman	14% (8/55)	10% (33/312)	8% (97/1151)
▼ board	20% (2/10)	11% (6/53)	10% (15/141)
Board	0% (0/1)	0% (0/7)	0% (0/17)
BoardFact	0% (0/3)	0% (0/11)	0% (0/27)
BoardFact	0% (0/1)	0% (0/6)	0% (0/18)
BoardTest	0% (0/1)	0% (0/3)	0% (0/3)
Direction	100% (1/1)	100% (4/4)	100% (11/11)
Square	0% (0/1)	0% (0/8)	0% (0/23)
SquareTes	0% (0/1)	0% (0/4)	0% (0/13)
Unit	100% (1/1)	20% (2/10)	13% (4/29)
> 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	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)
LauncherSmc	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfi	0% (0/1)	0% (0/2)	0% (0/4)

Task 3: JaCoCo Report on JPacman

The coverage results from JaCoCo are similar from the IntelliJ in the sense that they both display the covered results of each file. However, I did find the IntelliJ is nowhere near as descriptive as JaCoCo. On JaCoCo it not only tells you missed lines but missed branches and missed instructions. I found the visualization of JaCoCo very helpful especially on the uncovered branches. I always find that the pictures and graphs are easier to follow. JaCoCo also goes line by line highlighting in green which were covered and which weren't. IntelliJ I found to be more confusing and harder to understand. Therefore I prefer the JaCoCo report. For reference, here is my report.

Level

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• move(Unit, Direction)		82%		61%	7	10	1	18	0	1
• Level(Board, List, List, CollisionMap)		85%		57%	6	8	0	18	0	1
• registerPlayer(Player)		85%		60%	4	6	0	10	0	1
• updateObservers()		80%		87%	1	5	2	9	0	1
• removeObserver(Level, LevelObserver)		0%		n/a	1	1	2	2	1	1
• remainingPellets()		92%		83%	2	7	0	10	0	1
• stopNPCs()		86%		66%	2	4	0	6	0	1
• start()		86%		50%	1	2	1	8	0	1
• static {...}		75%		50%	1	2	0	1	0	1
• startNPCs()		100%		100%	0	2	0	7	0	1
• stop()		100%		100%	0	2	0	7	0	1
• isAnyPlayerAlive()		100%		100%	0	3	0	5	0	1
• addObserver(Level, LevelObserver)		100%		n/a	0	1	0	2	0	1
• getBoard()		100%		n/a	0	1	0	1	0	1
• isInProgress()		100%		n/a	0	1	0	1	0	1
Total	59 of 451	86%	24 of 80	70%	25	55	6	105	1	15

Task 4: Working with Python Test Coverage

We began this task with a starting coverage of 72%. In which we ran the nosetests and noticed lines 26, 30, 34-35, 45-48, 52-54, and 74-75 were missing. The code for lines 26 and 30 were given to us which bumped our coverage score from 72% to 76% and removed the two code lines from the missing section. I then moved forward to examining lines 34 - 35. In which I noticed that this function was setting attributes from a dictionary. What I did was added the function **test_from_dict()**. This creates a mock dictionary with my information, sets it to account, and calls the from_dict() function in account.py. This ensures that the information from the dictionary was properly set. This bumped the coverage score to 81% and removed the lines 34 - 36.

test_from_dict() code:

```
new *
def test_from_dict(self):
    """ Test account from dict """
    mockaccount = Account()
    dataDic = {
        "name": "Dafne Gonzalez",
        "email": "gonzad28@unlv.nevada.edu",
        "phone_number": "7022221122",
        "disabled": False
    }
    mockaccount.from_dict(dataDic)
    self.assertEqual(mockaccount.name, dataDic["name"])
    self.assertEqual(mockaccount.email, dataDic["email"])
    self.assertEqual(mockaccount.phone_number, dataDic["phone_number"])
    self.assertEqual(mockaccount.disabled, dataDic["disabled"])
```

Nosetest Results:

```
Name                               Stmts  Miss  Cover   Missing
-----
models/__init__.py                  7      0   100%
models/account.py                  40      9    78%   45-48, 52-54, 74-75
-----
TOTAL                               47      9    81%
-----

Ran 5 tests in 0.899s
```

I then examined the lines 45 - 48 in the accounts.py file. These lines of code took the new updated account information and updated it to the known account. I created the **test_update()** function which tests updating the name of the account. However, by updating the name my coverage only went up to 87% and line 48 was still missing.

Nosetests results after name update:

Name	Stmts	Miss	Cover	Missing
models/__init__.py	7	0	100%	
models/account.py	40	6	85%	48, 52-54, 74-75
TOTAL	47	6	87%	

To fix this issue, I tested to see if there was an exception thrown when no id was set. This brought my coverage score to 89% and now lines 45 - 48 were removed from the missing section.

test_update() code:

```
def test_update(self):
    """ Test info was updated correctly """
    updateAccount = Account()

    dataDic = {
        "name": "Dafne Gonzalez",
        "email": "gonzad28@unlv.nevada.edu",
    }

    updateAccount.from_dict(dataDic)

    updateAccount.id = 26
    updateAccount.name = "dafne G Mena"
    updateAccount.update()

    self.assertEqual(updateAccount.name, second: "dafne G Mena")

    # Check if there is exception when no id is set
    updateAccount.id = None
    with self.assertRaises(DataValidationError):
        updateAccount.update()
```

nosetests results after name update and id exception:

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test info was updated correctly
- Test the representation of an account
- Test account to dict

Name                StmtS  Miss  Cover  Missing
-----
models/__init__.py    7      0  100%
models/account.py    40      5   88%  52-54, 74-75
-----
TOTAL                47      5   89%
-----

Ran 5 tests in 0.911s

OK
```

Moving on to line 52 - 54. These lines of code in the `account.py` file removes an account from the database. To test this, I created the `test_delete()`, which grabbed a random account, created it, then deleted it. To make sure it was successfully deleted I ensured that the length was equal to 0. This covered lines 52 - 54 and bumped the coverage score to 96%.

`test_delete()` code:

```
def test_delete(self):
    """ Test Account creation using known data """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()
    account.delete()
    self.assertEqual(len(Account.all()), second: 0)
```


Nosetests results:

```
Test Account Model
```

- Test creating multiple Accounts
- Test Account creation using known data
- Test Account creation using known data
- Test account from dict
- Test the representation of an account
- Test account to dict
- Test info was updated correctly

Name	Stmts	Miss	Cover	Missing
models/__init__.py	7	0	100%	
models/account.py	40	2	95%	74-75
TOTAL	47	2	96%	

Ran 7 tests in 0.668s

Now, the remaining lines 74 - 75 focused on finding an account by its ID. To test this I created the function called `test_find()`. Here I created a very basic mock account information which stored the name. I then set the id to a random number, in this case 114. Since there is only one account stored the first find should match the ID number 114.

`test_find()` code:

```
new *
def test_find(self):
    mockaccount = Account()
    dataDic = {
        "name": "Dafne Gonzalez",
    }
    mockaccount.from_dict(dataDic)
    mockaccount.create()
    mockaccount.id = 114
    self.assertEqual(mockaccount.find(1), mockaccount)
```

`nosetests` coverage:

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

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 0.743s
```

All in all, I implemented four functions which has a final coverage of 100%: `test_from_dict`, `test_update`, `test_delete`, `test_find`.

Task 5: TDD

First I began by making the **test_update_a_counter(self)** function in the test_counter file following the provided checklist step-by step. This includes starting with making a call to a counter. I then ensured that it returned a successful code. Following that, I checked the counter value and assigned it as a baseline. I made a call to update the counter I created and ensured it returned a successful code. At last I checked that the counter value is one more than the baseline value. The code for the **test_update_a_counter** function can be found here:

```
new *
def test_update_a_counter(self):
    # Make a call to Create a counter.
    result = self.client.post('/counters/daf')

    # Ensure that it returned a successful return code.
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    # Check the counter value as a baseline.
    baseline = result.json['daf']
    self.assertEqual(baseline, second: 0)

    # Make a call to Update the counter that you just created.
    updatecounter = self.client.put('/counters/daf')

    # Ensure that it returned a successful return code.
    self.assertEqual(updatecounter.status_code, status.HTTP_200_OK)

    # Check that the counter value is one more than the baseline you measured in step 3
    self.assertEqual(updatecounter.json['daf'], baseline + 1)

    # Completes the coverage. Tests for a name not in counter
    testPut = self.client.put('/counters/tati')
    self.assertEqual(testPut.status_code, status.HTTP_409_CONFLICT)
```

When running nosetests I was in the red phase:

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

=====
FAIL: test_update_a_counter (test_counter.CounterTest)
-----
Traceback (most recent call last):
  File "/Users/dafne/Desktop/CS472/Assign2/tdd/tests/test_counter.py", line 41, in test_update_a_counter
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
AssertionError: 409 != 201
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: foo
----- >> end captured logging << -----

Name          Stmts   Miss  Cover   Missing
-----
src/counter.py    11      0   100%
src/status.py      6      0   100%
-----
TOTAL              17      0   100%
-----

Ran 3 tests in 0.310s

FAILED (failures=1)
```

I then created a function called **update_counter(name)** in the counter.py file. To create the update counter I created a route for method PUT, made sure the name was not already in counter, incremented the counter by 1, and returned the new counter with an OK status. Code for **update_counter(name)**:

```
new *
@app.route(rule: '/counters/<name>', methods=['PUT'])
def update_counter(name):
    # Create a route for method PUT on endpoint /counters/<name>.
    app.logger.info(f"Request to update counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
        return {"Message": f"Counter {name} does not exists"}, status.HTTP_409_CONFLICT
    # Increment the counter by 1.
    COUNTERS[name] = COUNTERS[name] + 1
    # Return the new counter and a 200_OK return code.
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

When running nosetests the first time I noticed that my test coverage was still not at 100%. This was because I was failing to read a test case that had a name that was not already in counter. Which is why I added the last assert in the test_update_a_counter. Here is the coverage after

doing nosetests

```
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	18	0	100%	
src/status.py	6	0	100%	
TOTAL	24	0	100%	

```
Ran 3 tests in 0.287s
```

```
OK
```

```
dafne@Dafnes-MacBook-Pro tdd %
```

I then got started with the test for reading a counter. This function is called **test_readCounter(self)**. I began by making a counter using put, ensured that the code returned successful, used 'get' to retrieve the value of the counter and tested for a successful code 'OK.' Additionally, I tested a name that was not in counter to make sure I correctly returned an unsuccessful code.

```

new *
def test_readCounter(self):
    """It should read a counter"""
    # Makes a counter
    result = self.client.post('/counters/nom')

    # Ensure that it returned a successful return code.
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    # Gets the value of counter
    getResult = self.client.get('/counters/nom')
    self.assertEqual(getResult.json['Count'], second: "0")

    # Ensure that it returned a successful return code.
    self.assertEqual(getResult.status_code, status.HTTP_200_OK)

    # Get a test call that doesn't exist
    badTest = self.client.get('/counters/hii')

    # Ensure that it returned a unsuccessful return code.
    self.assertEqual(badTest.status_code, status.HTTP_409_CONFLICT)

```

Running nosetests **without** readCounter function in counter.py caused a Red phase error when trying to read the counter.

```

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

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

Traceback (most recent call last):
  File "/Users/dafne/Desktop/CS472/Assign2/tdd/tests/test_counter.py", line 71, in test_readCounter
    self.assertEqual(getResult.json['nomi'], 0)
TypeError: 'NoneType' object is not subscriptable
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: nomi
----- >> end captured logging << -----

Name           Stmts  Miss  Cover   Missing
-----
src/counter.py    18     0   100%
src/status.py     6     0   100%
-----

```

To fix this, I created the function **readCounter(name)**. This was a fairly straightforward function which was very similar to the updateCounter without having to update the counter. I begin by creating a route, checking it isn't already a name in the counter, returns that count and returns an OK error code. The code is here:

```

new *
@app.route(rule: '/counters/<name>', methods=['GET'])
def read_counter(name):
    # Create a route for method PUT on endpoint /counters/<name>.
    app.logger.info(f"Request to read counter: {name}")
    global COUNTERS
    # Checks the name is not in counters
    if name not in COUNTERS:
        return {"Message": f"Counter {name} does not exists"}, status.HTTP_409_CONFLICT

    # Return the counter and a 200_OK return code.
    return {"Count": f"{COUNTERS[name]}"}, status.HTTP_200_OK

```

Doing a nosetests after completing the readCounter:

```
dafne@Dafnes-MacBook-Pro tdd % nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates
- It should read a counter
- 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 0.303s
```

Dafne Gonzalez

Github repo: https://github.com/Aligary/CS472_Group1.git