Fernando Rojas
CS 472-1001
2/6/2024

Unit Testing

Task 1:
We can see the code coverage in the image below. This is not acceptable coverage as almost none of the code is being tested.

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	3% (2/55)	1% (5/313)	1% (14/1138)

Task 2:
For reference you can see the coverage below after the isAlive() method is tested.

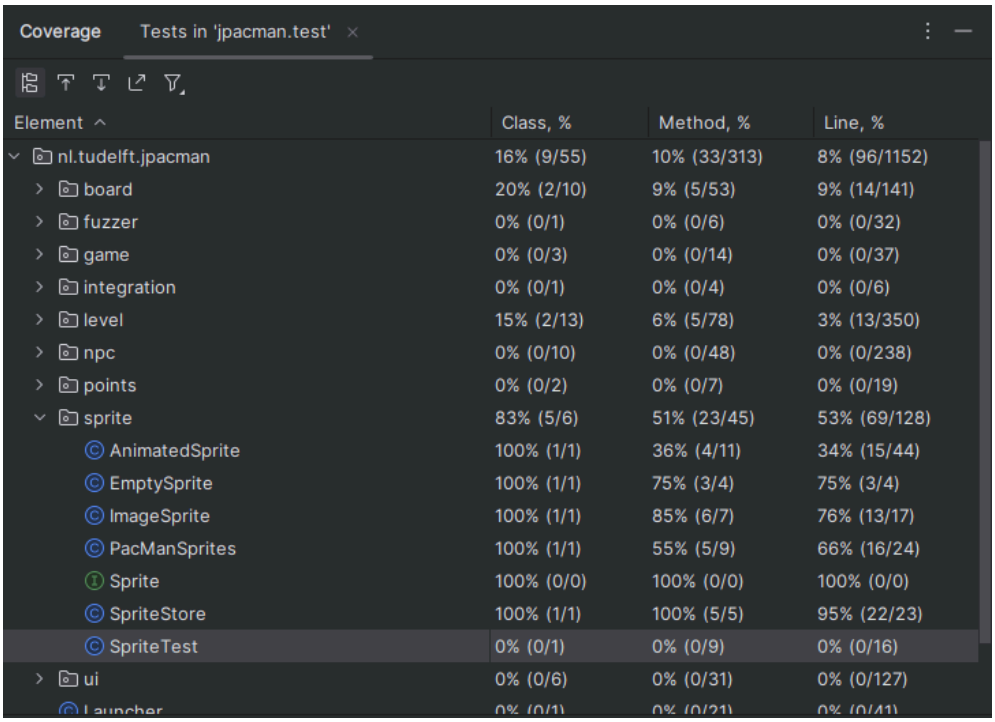
Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	14% (8/55)	9% (30/313)	8% (93/1152)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	15% (2/13)	6% (5/78)	3% (13/350)
npc	0% (0/10)	0% (0/48)	0% (0/238)
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)

For this section of the assignment I wrote more than three method tests. Here is a table to summarize my additions:

Class	Method
EmptySprite	split(int x, int y, int width, int height)
EmptySprite	getWidth()
EmptySprite	getHeight()

Player	setAlive(boolean isAlive)
Player	getKiller()
Player	setKiller(Unit killer)
Player	getScore()
Player	getSprite()
Player	addPoints(int points)
GhostFactory	createBlinkyTest()
GhostFactory	createPinkyTest()
GhostFactory	createInkyTest()
GhostFactory	createClydeTest()

EmptySprite method tests added. (Increase in sprite code coverage)



Coverage | Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	16% (9/55)	10% (33/313)	8% (96/1152)
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% (0/48)	0% (0/238)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	83% (5/6)	51% (23/45)	53% (69/128)
AnimatedSprite	100% (1/1)	36% (4/11)	34% (15/44)
EmptySprite	100% (1/1)	75% (3/4)	75% (3/4)
ImageSprite	100% (1/1)	85% (6/7)	76% (13/17)
PacManSprites	100% (1/1)	55% (5/9)	66% (16/24)
Sprite	100% (0/0)	100% (0/0)	100% (0/0)
SpriteStore	100% (1/1)	100% (5/5)	95% (22/23)
SpriteTest	0% (0/1)	0% (0/9)	0% (0/16)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)

GhostFactory method tests added. (Increase in npc code coverage)

Coverage GhostTest

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	23% (13/55)	12% (38/313)	9% (111/1132)
board	20% (2/10)	7% (4/53)	9% (13/136)
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% (0/329)
npc	70% (7/10)	31% (15/48)	14% (35/244)
ghost	66% (6/9)	31% (14/44)	12% (30/236)
Ghost	100% (1/1)	25% (1/4)	62% (5/8)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	66% (4/6)	42% (19/45)	49% (63/128)
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)
PacmanConfiguratior	0% (0/1)	0% (0/2)	0% (0/4)

Player method tests added. (Increase in level code coverage)

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	29% (16/55)	18% (57/313)	13% (155/1158)
board	20% (2/10)	11% (6/53)	10% (15/141)
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)	14% (11/78)	8% (29/350)
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)	100% (8/8)	100% (24/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	70% (7/10)	31% (15/48)	14% (35/244)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	83% (5/6)	55% (25/45)	59% (76/128)
ui	0% (0/6)	0% (0/31)	0% (0/127)

Task 3:

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		67%		59%	70	155	100	344	20	69	4	12
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.ui		78%		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		67%		25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.sprite		87%		59%	27	70	8	113	2	38	0	5
nl.tudelft.jpacman.points		59%		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	1,224 of 4,755	74%	290 of 637	54%	286	590	222	1,039	47	268	6	47

The coverage results from JaCoCo are not similar to the ones from the IntelliJ task. I think this is because JoCoCo uses missed instructions and branches in its report vs. IntelliJ which uses classes, methods, and lines for its percentages.

The visualization is pretty useful for seeing explicitly what lines of code were covered or missed. I found it much easier to visualize than trying to find the missed code in IntelliJ.

I think I would prefer IntelliJ for general test unit writing. JaCoCo would be ideal if I really needed to track down what specific branches in the code are going untested.

Task 4:

```
def test_from_dict(self):
    """ Test account from dict """
    data = ACCOUNT_DATA[self.rand]
    account = Account()
    result = account.from_dict(data)
    self.assertEqual(account.name, data["name"])
    self.assertEqual(account.email, data["email"])
    self.assertEqual(account.phone_number, data["phone_number"])
    self.assertEqual(account.disabled, data["disabled"])
```

```
def test_update_an_account(self):
    """ Test updating account in database"""
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()
    account.update()
    self.assertEqual(len(Account.all()), 1)
```

```
def test_update_an_account_fail(self):
    """ Test updating account error """
    account = Account()
    with self.assertRaises(DataValidationError):
        account.update()
```

```python
def test_delete_account(self):
    """ Test deleting account in database"""
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()
    account.delete()
    self.assertEqual(len(Account.all()), 0)
```

```python
def test_find_account(self):
    """ Test finding account in database"""
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()
    result = Account.find(account.id)
    self.assertEqual(result, account)
```

Test Results:

```
PS C:\Users\Fern\Documents\Program Developments\test_coverage> nosetests

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deleting account in database
- Test finding account in database
- Test account from dict
- Test the representation of an account
- Test account to dict
- Test updating account in database
- Test updating account error

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 0.557s

OK
```

Task 5:

Adding the updating a counter test will put us in a RED PHASE

```python
def test_updating_a_counter(self):
    """It should update a counter"""
    update = self.client.post('/counters/qux')
    self.assertEqual(update.status_code, status.HTTP_201_CREATED)
    baseline = update.json["qux"]
    self.assertEqual(baseline, 0)
    update = self.client.put('/counters/qux')
    self.assertEqual(update.status_code, status.HTTP_200_OK)
    self.assertEqual(update.json["qux"], baseline+1)
```

```
Counter
- 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\Fern\Documents\Program Developments\tdd\tests\test_counter.py", line 45, in test_updating_a_counter
    self.assertEqual(update.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: qux
-------------------- >> end captured logging << --------------------

Name              Stmts   Miss  Cover   Missing
---------------------------------------------
src\counter.py       12      0   100%
src\status.py         6      0   100%
---------------------------------------------
TOTAL                18      0   100%
----------------------------------------------------------------------
Ran 3 tests in 0.264s

FAILED (failures=1)
```

REFACTORING counter.py will put us in a GREEN PHASE

```python
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """ Update a Counter """
    app.logger.info(f"Request to update a counter: {name}")
    global COUNTERS
    if name in COUNTERS:
        COUNTERS[name] = COUNTERS[name] + 1
        return {name: COUNTERS[name]}, status.HTTP_200_OK
    return {"Message":f"Counter {name} found"}, status.HTTP_404_NOT_FOUND
```

```
Counter
- 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       19      1    95%   29
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                25      1    96%
----------------------------------------------------------------------
Ran 3 tests in 0.206s

OK
```

This fulfills requirement 1.

Adding the getting a counter test will put us in a RED PHASE

```
def test_get_a_counter(self):
    """It should get a counter"""
    result = self.client.get('/counters/bar')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
```

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

======================================================================
FAIL: It should get a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Fern\Documents\Program Developments\tdd\tests\test_counter.py", line 53, in test_get_a_counter
    self.assertEqual(result.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200

Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\counter.py       19      1    95%   29
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                25      1    96%
----------------------------------------------------------------------
Ran 4 tests in 0.210s

FAILED (failures=1)
```

REFACTORING counter.py will put us in a GREEN PHASE

```
PS C:\Users\Fern\Documents\Program Developments\tdd> nosetests

Counter
- 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      2    92%   30, 39
src\status.py         6      0   100%
----------------------------------------------
TOTAL                30      2    93%
----------------------------------------------------------------------
Ran 4 tests in 0.201s

OK
```

This fulfills requirement 2.

To ensure full coverage, we need to add versions of the tests where an invalid counter is retrieved.

```
def test_updating_a_nonexistent_counter(self):
    """It should return an error for updating nonexistent counter"""
    update = self.client.put('/counters/qqq')
    self.assertEqual(update.status_code, status.HTTP_404_NOT_FOUND)
```

```
def test_get_a_nonexistent_counter(self):
    result = self.client.get('/counters/qqq')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
```

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

Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\counter.py       24      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                30      0   100%
----------------------------------------------------------------------
Ran 6 tests in 0.207s

OK
```