UNIT TESTING REPORT by Uyen Tran
Link to repo: https://github.com/jenlcmc/SeniorDesign2.git

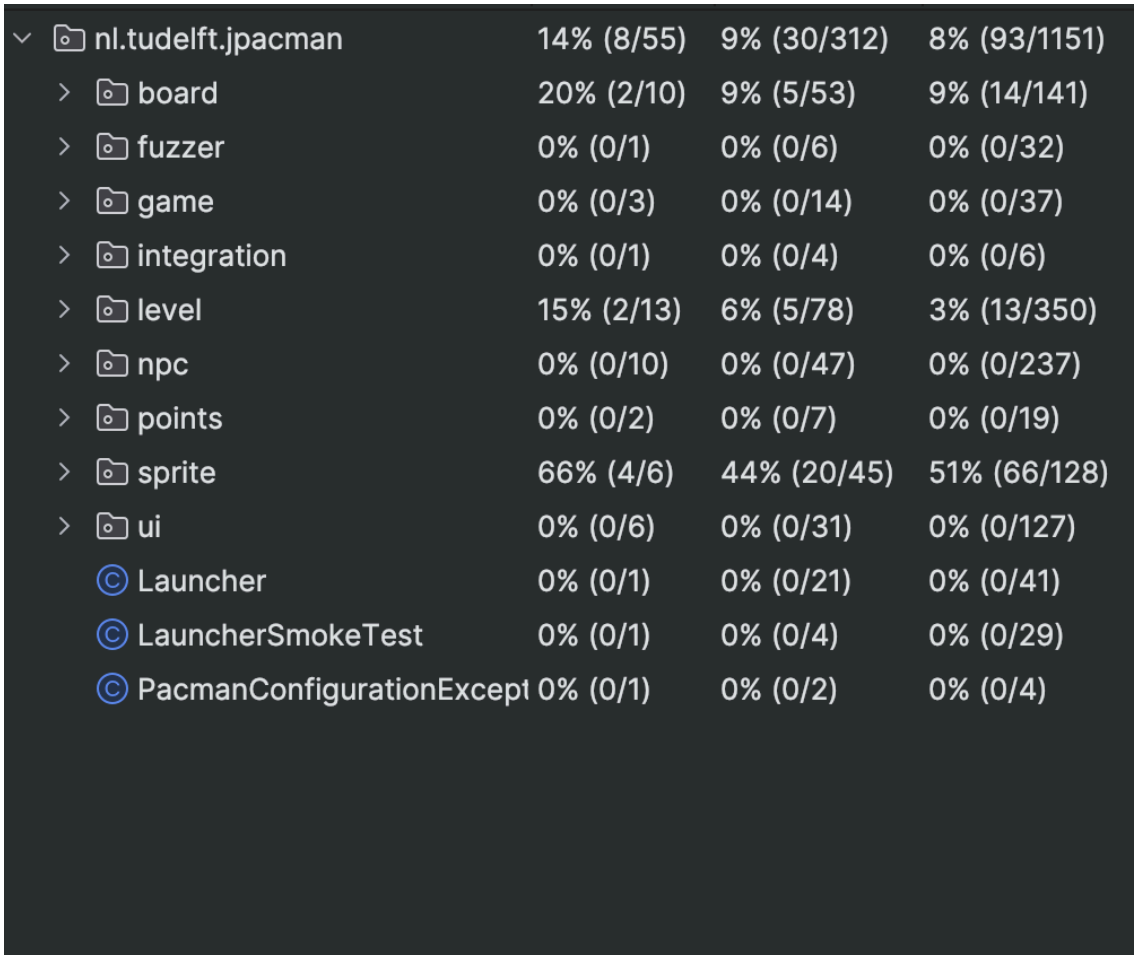
Tasks 2.1 and 3

```java
@Test
    public void testCreatePellet() {
        // Arrange
        PacManSprites sprites = new PacManSprites();
        GhostFactory ghostFactory = new GhostFactory(sprites); // create a ghost factory with the default sprites
        // create a level factory with the default sprites, ghost factory and point
        // calculator
        LevelFactory levelFactory = new LevelFactory(sprites, ghostFactory, new DefaultPointCalculator());

        // Act
        Pellet pellet = levelFactory.createPellet();

        // Assert
        // check if the pellet is not null, has the expected value and sprite
        assertNotNull(pellet, "Created pellet should not be null");
        assertEquals(10, pellet.getValue(), "Created pellet's value should be equal to 10");
        assertEquals(sprites.getPelletSprite(), pellet.getSprite(),
                "Created pellet's sprite should be equal to the expected pellet sprite");
    }
```

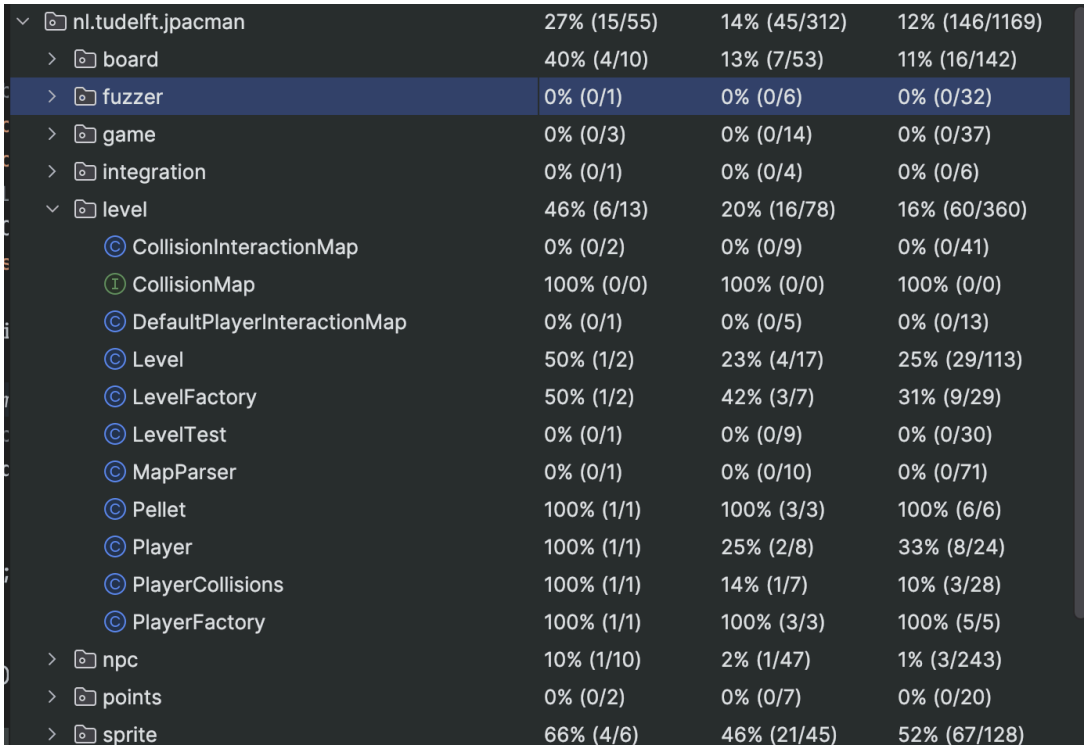
```java
@Test
    void testRegisterPlayer() {
        // Arrange
        // create a player, square, board, ghost, start squares, collision map and level
        Player player = mock(Player.class);
        Square square = mock(Square.class);
        // use Collections.singletonList to create a list with a single element
        List<Square> startSquares = Collections.singletonList(square);
        Board board = mock(Board.class);
        // use Collections.emptyList to create an empty list
        List<Ghost> ghosts = Collections.emptyList();
        // create a collision map
        CollisionMap collisionMap = mock(CollisionMap.class);
        // create a level with the board, ghosts, start squares and collision map
        Level level = new Level(board, ghosts, startSquares, collisionMap);

        // Act
        level.registerPlayer(player);

        // Assert
        verify(player).occupy(square); // check if the player occupies the expected square
    }
```

```java
@Test
    void testCreateLevel() {
        // Arrange
        PacManSprites sprites = new PacManSprites();
        GhostFactory ghostFactory = new GhostFactory(sprites);
        LevelFactory levelFactory = new LevelFactory(sprites, ghostFactory, new DefaultPointCalculator());
        Board board = mock(Board.class);
        Ghost ghost = mock(Ghost.class);
        // use Collections.singletonList to create a list with a single element
        List<Ghost> ghosts = Collections.singletonList(ghost);
        Square square = mock(Square.class);
        // use Collections.singletonList to create a list with a single element
        List<Square> startPositions = Collections.singletonList(square);

        // Act
        Level level = levelFactory.createLevel(board, ghosts, startPositions);

        // Assert
        // check if the level is not null and has the expected board
        assertNotNull(level, "Created level should not be null");
        assertEquals(board, level.getBoard(), "Created level should have the expected board");
        // check if the level has the expected ghosts
        verify(ghost).equals(ghosts.get(0));

    }
```

Before result:



After result:



nl.tudelft.jpacman	27% (15/55)	14% (45/312)	12% (146/1169)
board	40% (4/10)	13% (7/53)	11% (16/142)
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	46% (6/13)	20% (16/78)	16% (60/360)
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	50% (1/2)	23% (4/17)	25% (29/113)
LevelFactory	50% (1/2)	42% (3/7)	31% (9/29)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	100% (1/1)	100% (3/3)	100% (6/6)
Player	100% (1/1)	25% (2/8)	33% (8/24)
PlayerCollisions	100% (1/1)	14% (1/7)	10% (3/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	10% (1/10)	2% (1/47)	1% (3/243)
points	0% (0/2)	0% (0/7)	0% (0/20)
sprite	66% (4/6)	46% (21/45)	52% (67/128)

So all 3 tests I did are in the level folder (level and levelFactory files). In general, we can see the difference between before and after adding tests. Before, my level folder only had 15% in class, 6% in Methods, and 3% in lines. However, after adding tests, my result increased to 46% for class, 20% for Methods, and then 16% for lines

After trying the JaCoCo, my result is different compared to the IntelliJ one as for JaCoCo, I have 67% on Coverage of the missed Instructions and 57% of missed branches. I then looked at the missed lines and saw a different result. For JaCoCo, I have 6/105, while for IntelliJ, I have 29/113 for the lines. These 2 have different ways to test the coverage.

The source code visualization from JaCoCo is really helpful, and I prefer JaCoCo more as it provides me the coverage bar beside % and is more in-depth with different options that I can check. Also, seeing which part of the code pass, half-pass or not, is really useful.

Task 4

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deleting an account
- Test finding an account
- from dict
- Test the representation of an account
- Test account to dict
- Test updating an account and updating without an ID

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 0.395s

OK
```

Here is my result at the end for task 4 in which that I have 100% coverages

```python
def test_from_dict(self):
        data = ACCOUNT_DATA[self.rand]
        account = Account()
        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_account(self):
        """ Test updating an account and updating without an ID """
        # Test updating an account
        data = ACCOUNT_DATA[self.rand]
        account = Account(**data)
        account.create()
        account.disabled = True
        account.update()
        self.assertEqual(account.disabled, True)
        # Test updating without an ID
        account = Account()
        account.name = "Foo"
        with self.assertRaises(DataValidationError):
            account.update()

    def test_delete_account(self):
        """ Test deleting an account """
        data = ACCOUNT_DATA[self.rand]
        account = Account(**data)
        account.create()
        account.delete()
        self.assertEqual(len(Account.all()), 0)

    def test_find_account(self):
        """ Test finding an account """
        data = ACCOUNT_DATA[self.rand]
        account = Account(**data)
        account.create()
        result = Account.find(account.id)
        self.assertEqual(account, result)
```

Here is my code for task 4. We can see that for 1st test, I will just create a new account and then call the function, and then use the assertEqual() to make sure that it is the same for testing. 2nd test will do the same by calling create() to create a new account and then using .disable and update() to test whether it shows the same result. I also test the without ID to make sure that it should not do it. The rest are the same with these 2 as I will create a new account and then call those functions that I am supposed to test before using asserEqual() to make sure that it shows the same result.

Task 5:

```
(.venv) ┌─uyentran@jenlcmc ~/Desktop/tdd  ‹main*›
└─➤  nosetests

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

======================================================================
FAIL: It should create a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/uyentran/Desktop/tdd/tests/test_counter.py", line 69, in test_create_a_counter
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
AssertionError: 404 != 201

======================================================================
FAIL: It should return an error for duplicates
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/uyentran/Desktop/tdd/tests/test_counter.py", line 30, in test_duplicate_a_counter
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
AssertionError: 404 != 201

======================================================================
FAIL: It should read a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/uyentran/Desktop/tdd/tests/test_counter.py", line 55, in test_read_a_counter
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
AssertionError: 404 != 201

======================================================================
FAIL: It should update a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/uyentran/Desktop/tdd/tests/test_counter.py", line 38, in test_update_a_counter
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
AssertionError: 404 != 201

Name             Stmts   Miss  Cover   Missing
----------------------------------------------
src/counter.py       4      0   100%
src/status.py        6      0   100%
----------------------------------------------
TOTAL               10      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.211s

FAILED (failures=4)
```

Here are the result when it fail all tests

```
└─➤  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 (FAILED)

======================================================================
FAIL: It should read a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/uyentran/Desktop/tdd/tests/test_counter.py", line 58, in test_read_a_counter
    self.assertEqual(response.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: test_read_counter
--------------------- >> end captured logging << ---------------------

======================================================================
FAIL: It should update a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/uyentran/Desktop/tdd/tests/test_counter.py", line 43, in test_update_a_counter
    self.assertEqual(response.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: test_counter
--------------------- >> 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 4 tests in 0.234s

FAILED (failures=2)
```

This is result before I implement those 2 functions that I supposed to do

```
(.venv) ┌─uyentran@jenlcmc ~/Desktop/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 "/Users/uyentran/Desktop/tdd/tests/test_counter.py", line 58, in test_read_a_counter
    self.assertEqual(response.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: test_read_counter
--------------------- >> end captured logging << ---------------------

Name             Stmts   Miss  Cover   Missing
----------------------------------------------
src/counter.py      17      0   100%
src/status.py        6      0   100%
----------------------------------------------
TOTAL               23      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.217s

FAILED (failures=1)
```

Here is the result before implement last testing function which is read counter

```
(.venv) ┌─uyentran@jenlcmc ~/Desktop/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

Name             Stmts   Miss  Cover   Missing
----------------------------------------------
src/counter.py      22      0   100%
src/status.py        6      0   100%
----------------------------------------------
TOTAL               28      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.215s

OK
```

Here is the result with extra tests to achieve the 100% coverage for counter.py

```python
"""
Test Cases for Counter Web Service

Create a service that can keep a track of multiple counters
- API must be RESTful - see the status.py file. Following these guidelines, you can make assumptions about
how to call the web service and assert what it should return.
- The endpoint should be called /counters
- When creating a counter, you must specify the name in the path.
- Duplicate names must return a conflict error code.
- The service must be able to update a counter by name.
- The service must be able to read the counter
"""
from unittest import TestCase

# we need to import the unit under test - counter
from src.counter import app

# we need to import the file that contains the status codes
from src import status

class CounterTest(TestCase):
    """Counter tests"""

    def setUp(self):
        self.client = app.test_client()

    def test_duplicate_a_counter(self):
        """It should return an error for duplicates"""
        result = self.client.post('/counters/bar')
        self.assertEqual(result.status_code, status.HTTP_201_CREATED)
        result = self.client.post('/counters/bar')
        self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)

    def test_update_a_counter(self):
        """It should update a counter"""
        # Create a counter
        response = self.client.post('/counters/test_counter')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        # Check the counter value as a baseline
        baseline = response.json['test_counter']
        # Update the counter
        response = self.client.put('/counters/test_counter')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        # Check that the counter value is one more than the baseline
        updated_value = response.json['test_counter']
        self.assertEqual(updated_value, baseline + 1)
        # Try to update a counter that does not exist
        response = self.client.put('/counters/nonexistent_counter')
        self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)

    def test_read_a_counter(self):
        """It should read a counter"""
        # Create a counter
        response = self.client.post('/counters/test_read_counter')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        # Read the counter
        response = self.client.get('/counters/test_read_counter')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        # Check that the counter value is correct
        self.assertEqual(response.json['test_read_counter'], 0)
        # Try to read a counter that does not exist
        response = self.client.get('/counters/nonexistent_counter')
        self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)

    def test_create_a_counter(self):
     """It should create a counter"""
     client = app.test_client()
     result = client.post('/counters/foo')
     self.assertEqual(result.status_code, status.HTTP_201_CREATED)

```

Here is my test counter file with comments for readability. For the update counter test, I use .sjson to check the baseline which is before increment or update. Then I perform those updates and then recheck it with .json(). I also tested the test that the update counter does not exist and this one is one of those tests that help me achieve 100% coverage. The read counter also somewhat the same outline with update counter too and the last test which is check for counter not exist are one of the two that help me score 100% on cover

```python
from flask import Flask

# we need to import the file that contains the status codes
from src import status

app = Flask(__name__)

COUNTERS = {}

# We will use the app decorator and create a route called slash counters.
# specify the variable in route <name>
# let Flask know that the only methods that is allowed to called
# on this function is "POST".
@app.route('/counters/<name>', methods=['POST'])
def create_counter(name):
    """Create a counter"""
    app.logger.info(f"Request to create counter: {name}")
    global COUNTERS

    if name in COUNTERS:
        return {"Message":f"Counter {name} already exists"}, status.HTTP_409_CONFLICT

    COUNTERS[name] = 0
    return {name: COUNTERS[name]}, status.HTTP_201_CREATED

@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    global COUNTERS

    if name not in COUNTERS:
        return {"Message": f"Counter {name} does not exist"}, status.HTTP_404_NOT_FOUND

    COUNTERS[name] += 1

    return {name: COUNTERS[name]}, status.HTTP_200_OK

@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Read a counter"""
    global COUNTERS

    if name not in COUNTERS:
        return {"Message": f"Counter {name} does not exist"}, status.HTTP_404_NOT_FOUND

    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

Here is my counter file, which I use the app decorator and create a route for each of my test. I also added the duplicate test for all of the routes like the update and read counter ones to make sure that no duplicates that could cause errors.