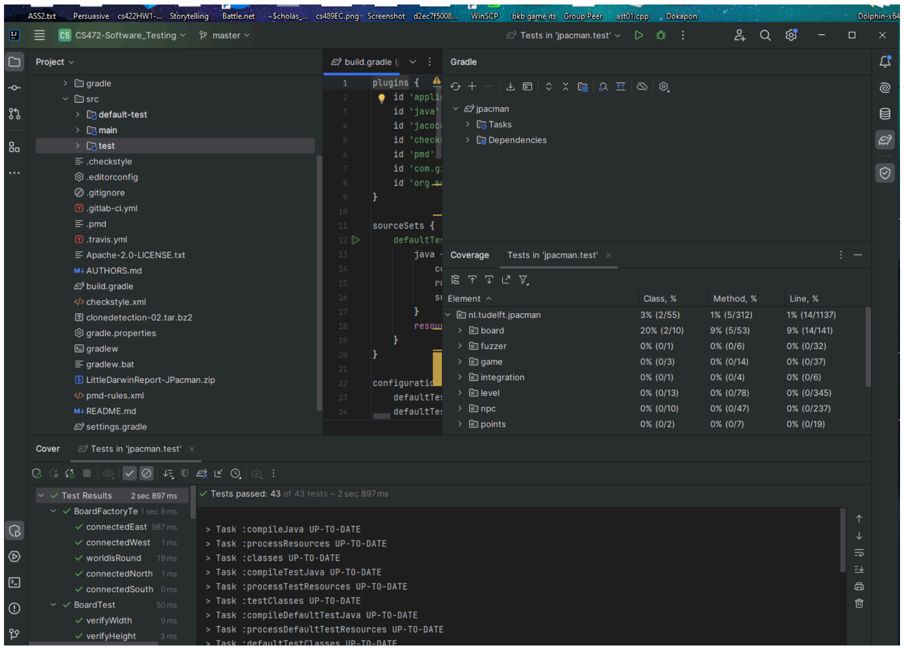
Link to Forked Repository: https://github.com/Nick8120/Group8

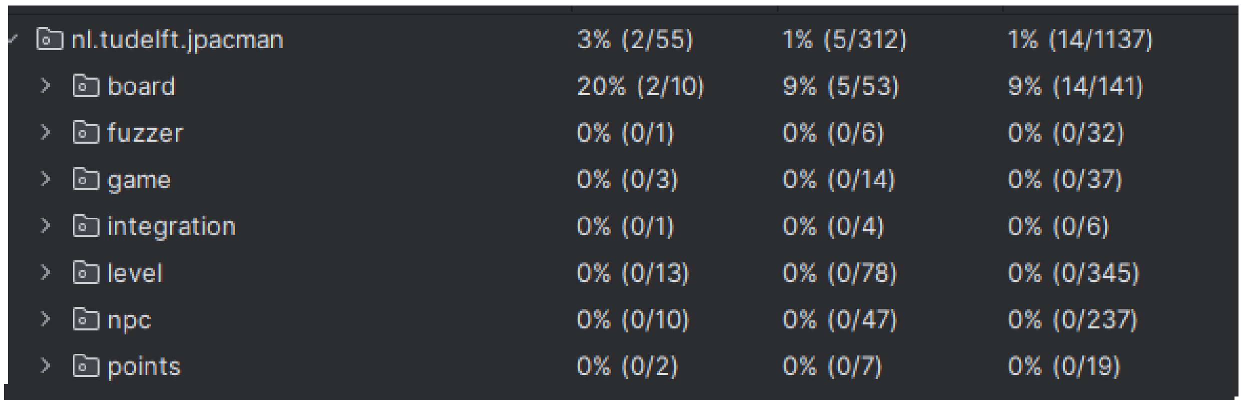
Task 1 – JPacman Test Coverage



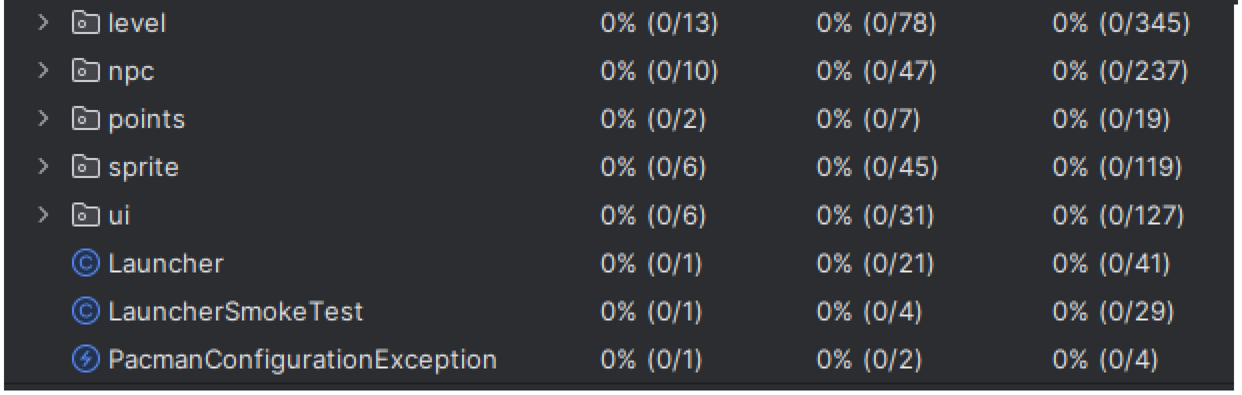
Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	0% (0/13)	0% (0/78)	0% (0/345)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)

```
Test Results                2 sec 897 ms
  BoardFactoryTe            1 sec 8 ms
    connectedEast           987 ms
    connectedWest           1 ms
    worldIsRound            19 ms
    connectedNorth          1 ms
    connectedSouth          0 ms
  BoardTest                 50 ms
    verifyWidth             9 ms
    verifyHeight            3 ms
```

```
Tests passed: 43 of 43 tests – 2 sec 897 ms

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources UP-TO-DATE
> Task :testClasses UP-TO-DATE
> Task :compileDefaultTestJava UP-TO-DATE
> Task :processDefaultTestResources UP-TO-DATE
> Task :defaultTestClasses UP-TO-DATE
```



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)



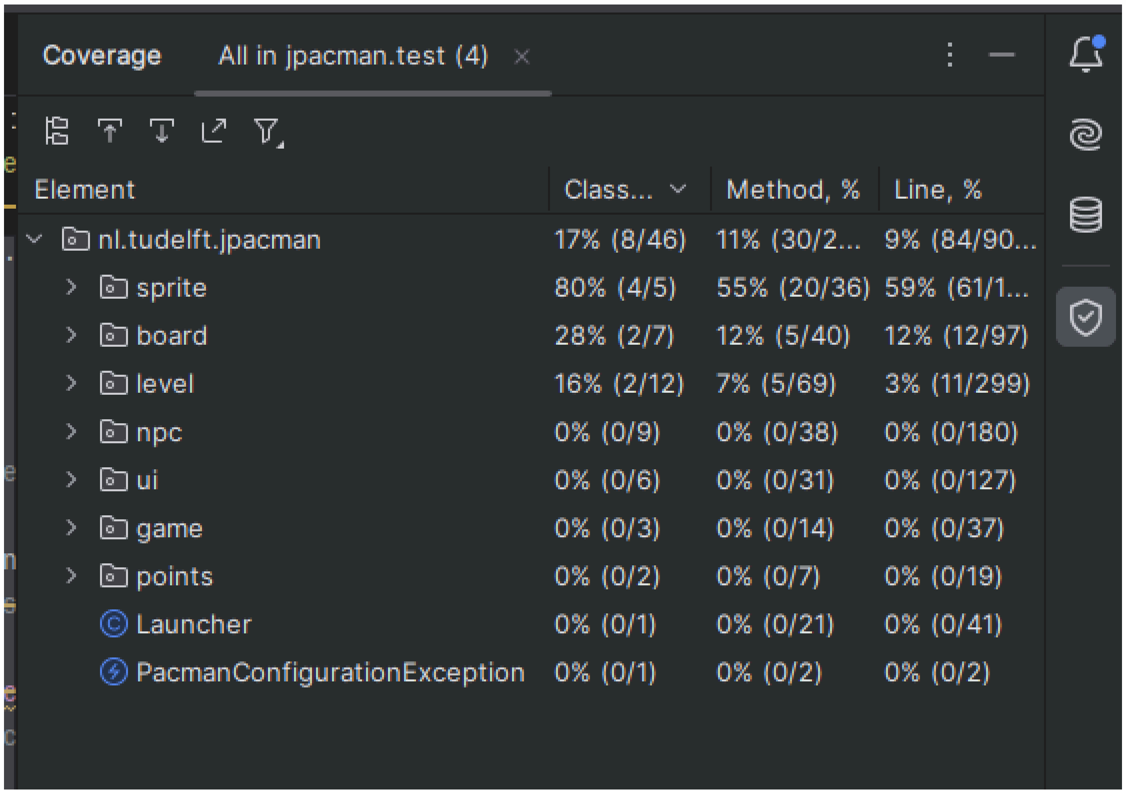
level	0% (0/13)	0% (0/78)	0% (0/345)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	0% (0/6)	0% (0/45)	0% (0/119)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

Question:

- Is the coverage good enough?

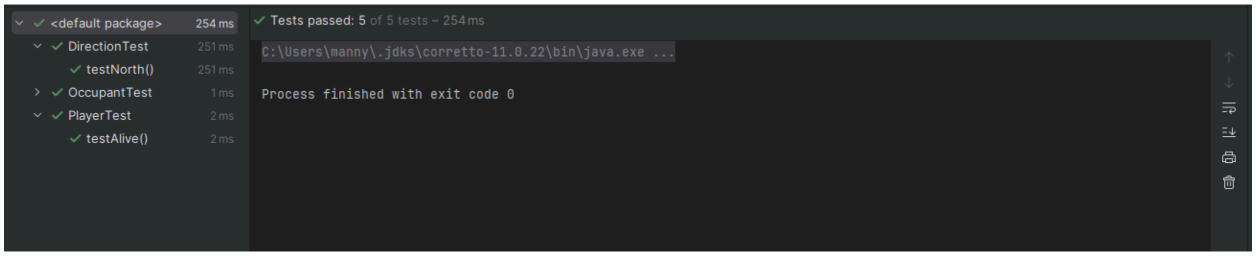
No, the test coverage is extremely low covering 3% for class and 1% for method and line coverage.

Task 2 – Increasing Coverage on JPacman



Coverage | All in jpacman.test (4)

Element	Class...	Method, %	Line, %
nl.tudelft.jpacman	17% (8/46)	11% (30/2...	9% (84/90...
sprite	80% (4/5)	55% (20/36)	59% (61/1...
board	28% (2/7)	12% (5/40)	12% (12/97)
level	16% (2/12)	7% (5/69)	3% (11/299)
npc	0% (0/9)	0% (0/38)	0% (0/180)
ui	0% (0/6)	0% (0/31)	0% (0/127)
game	0% (0/3)	0% (0/14)	0% (0/37)
points	0% (0/2)	0% (0/7)	0% (0/19)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/2)

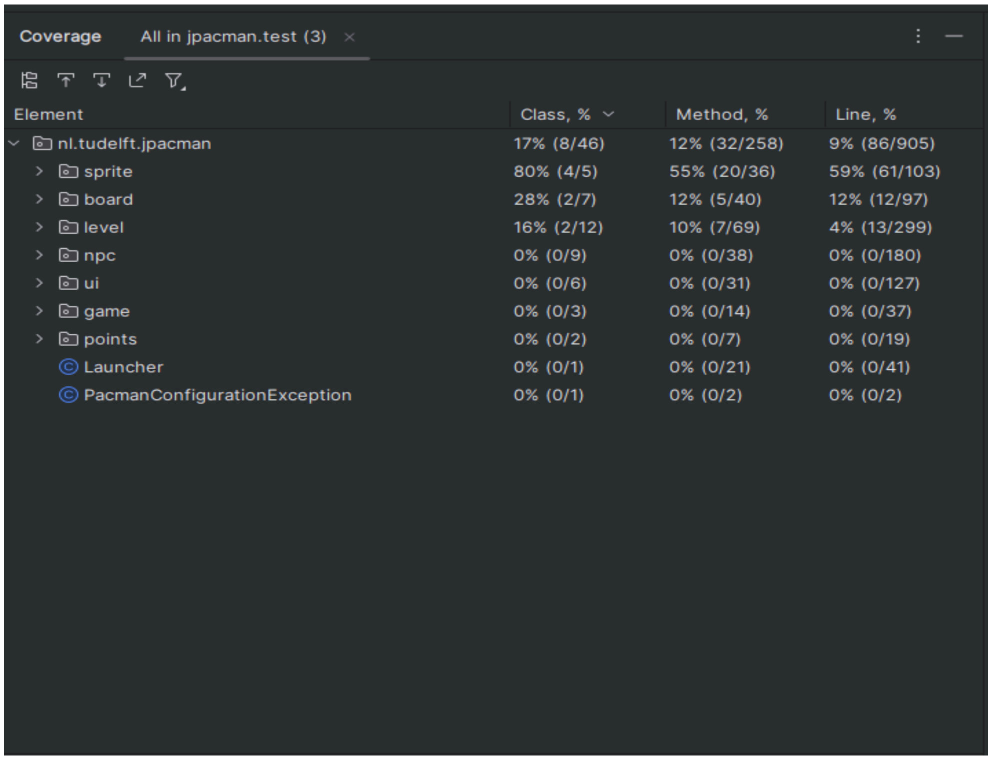


```
✓ <default package>    254 ms
  ✓ DirectionTest      251 ms
    ✓ testNorth()      251 ms
  ✓ OccupantTest         1 ms
  ✓ PlayerTest           2 ms
    ✓ testAlive()        2 ms
```

```
✓ Tests passed: 5 of 5 tests – 254 ms

C:\Users\manny\.jdks\corretto-11.0.22\bin\java.exe ...

Process finished with exit code 0
```

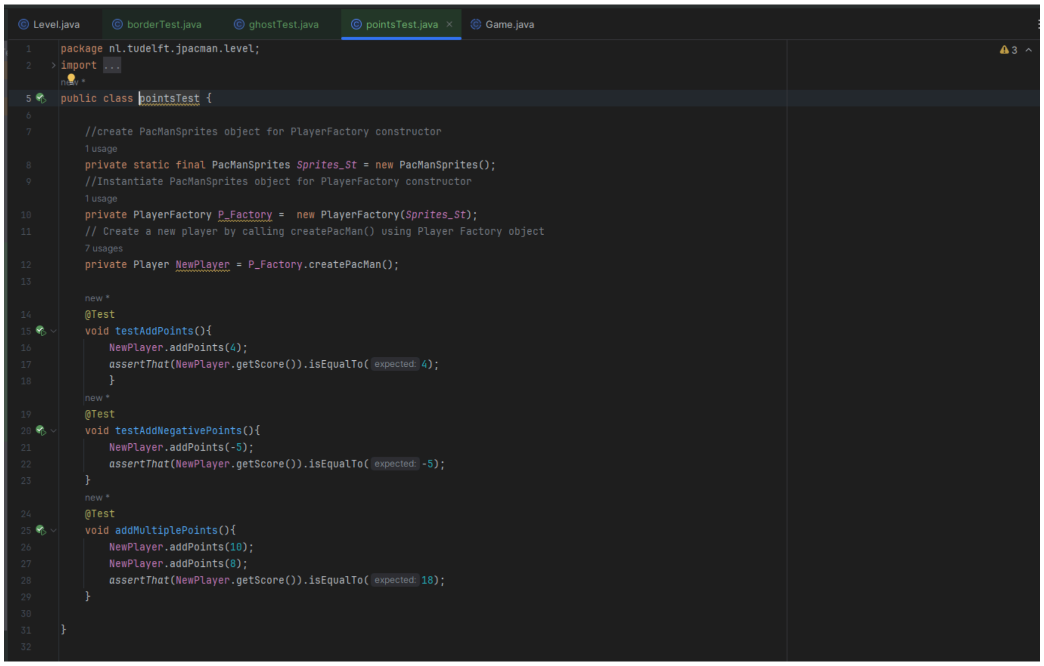
Task 2.1

Add points method code/results:



Coverage All in jpacman.test (3)

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	17% (8/46)	12% (32/258)	9% (86/905)
sprite	80% (4/5)	55% (20/36)	59% (61/103)
board	28% (2/7)	12% (5/40)	12% (12/97)
level	16% (2/12)	10% (7/69)	4% (13/299)
npc	0% (0/9)	0% (0/38)	0% (0/180)
ui	0% (0/6)	0% (0/31)	0% (0/127)
game	0% (0/3)	0% (0/14)	0% (0/37)
points	0% (0/2)	0% (0/7)	0% (0/19)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/2)

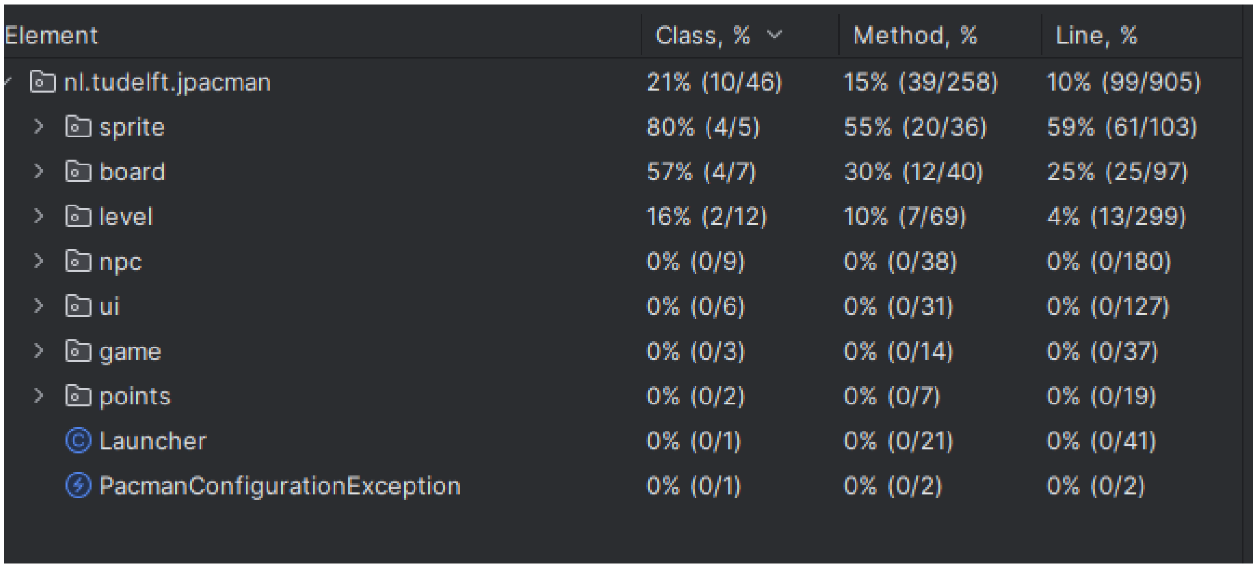


Level.java borderTest.java ghostTest.java pointsTest.java Game.java

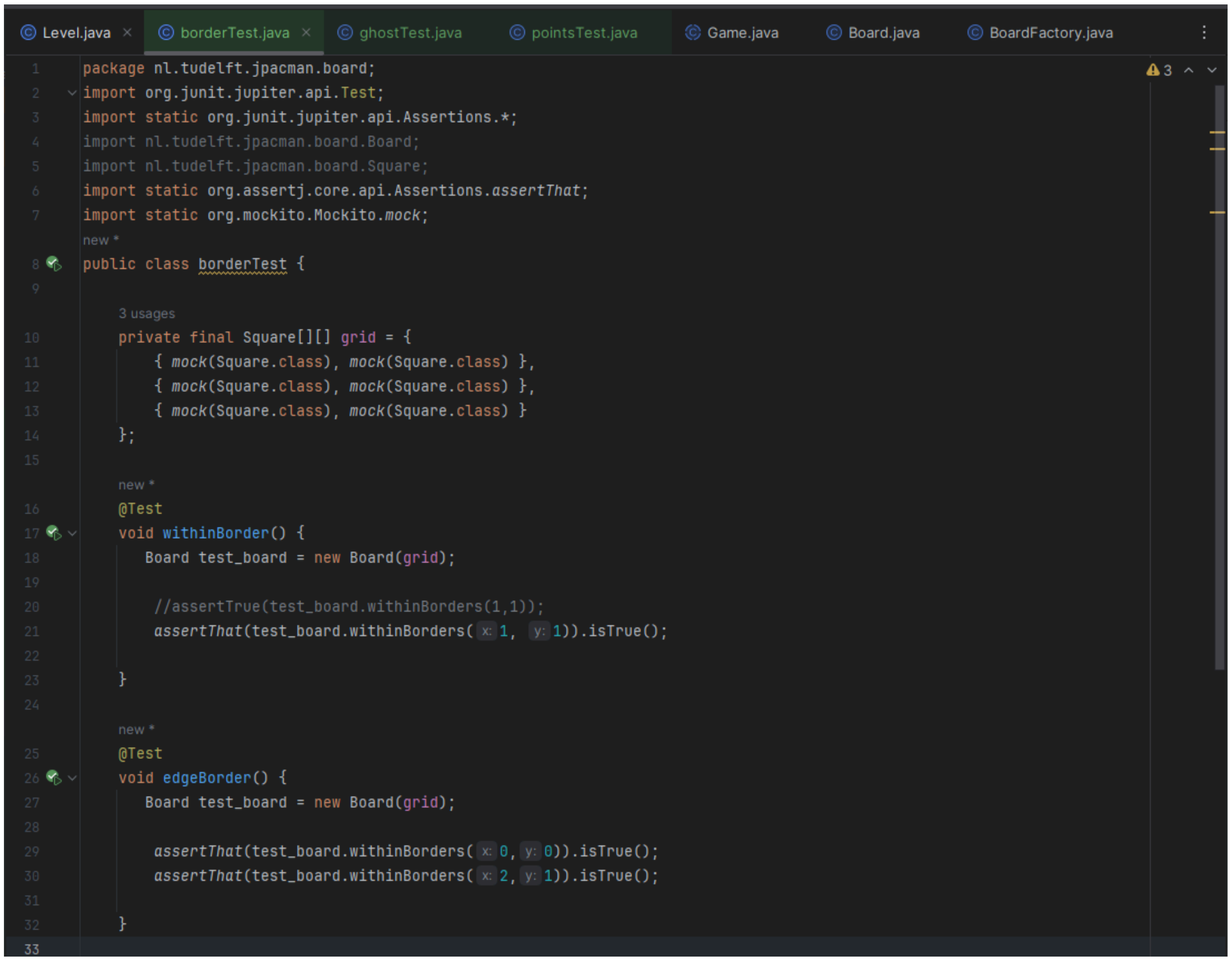
```java
package nl.tudelft.jpacman.level;
import ...
public class pointsTest {

    //create PacManSprites object for PlayerFactory constructor
    private static final PacManSprites Sprites_St = new PacManSprites();
    //Instantiate PacManSprites object for PlayerFactory constructor
    private PlayerFactory P_Factory =  new PlayerFactory(Sprites_St);
    // Create a new player by calling createPacMan() using Player Factory object
    private Player NewPlayer = P_Factory.createPacMan();

    @Test
    void testAddPoints(){
        NewPlayer.addPoints(4);
        assertThat(NewPlayer.getScore()).isEqualTo(4);
        }
    @Test
    void testAddNegativePoints(){
        NewPlayer.addPoints(-5);
        assertThat(NewPlayer.getScore()).isEqualTo(-5);
    }
    @Test
    void addMultiplePoints(){
        NewPlayer.addPoints(10);
        NewPlayer.addPoints(8);
        assertThat(NewPlayer.getScore()).isEqualTo(18);
    }

}
```

The first method that I attempted above was to write a test for was that of the adding points function. Here I covered three cases for adding points. One covering the basic addtion of points, the second testing the addition of negative points, and lastly the adding of multiple points at a time. This was done by creating a player and using the object of the Player class to call add points to said players score. This did not increase the coverage very much however only slightly improving it.

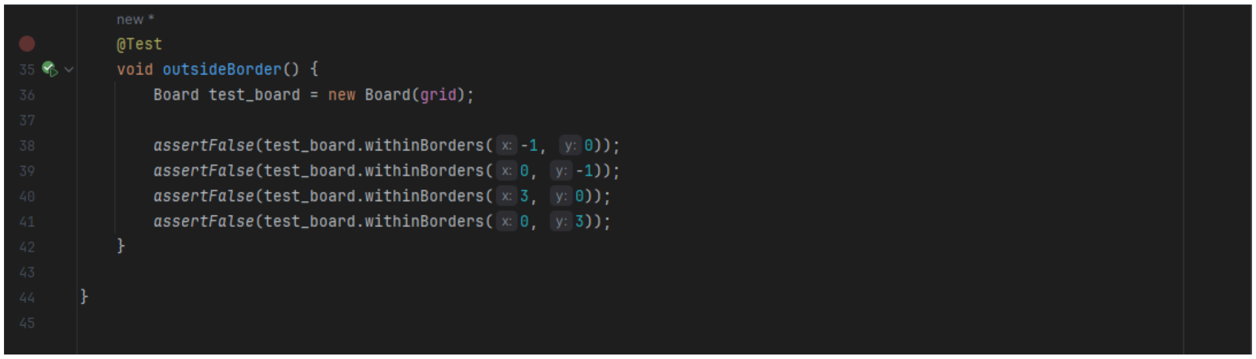
withinBoarder method code/results:



Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	21% (10/46)	15% (39/258)	10% (99/905)
sprite	80% (4/5)	55% (20/36)	59% (61/103)
board	57% (4/7)	30% (12/40)	25% (25/97)
level	16% (2/12)	10% (7/69)	4% (13/299)
npc	0% (0/9)	0% (0/38)	0% (0/180)
ui	0% (0/6)	0% (0/31)	0% (0/127)
game	0% (0/3)	0% (0/14)	0% (0/37)
points	0% (0/2)	0% (0/7)	0% (0/19)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/2)



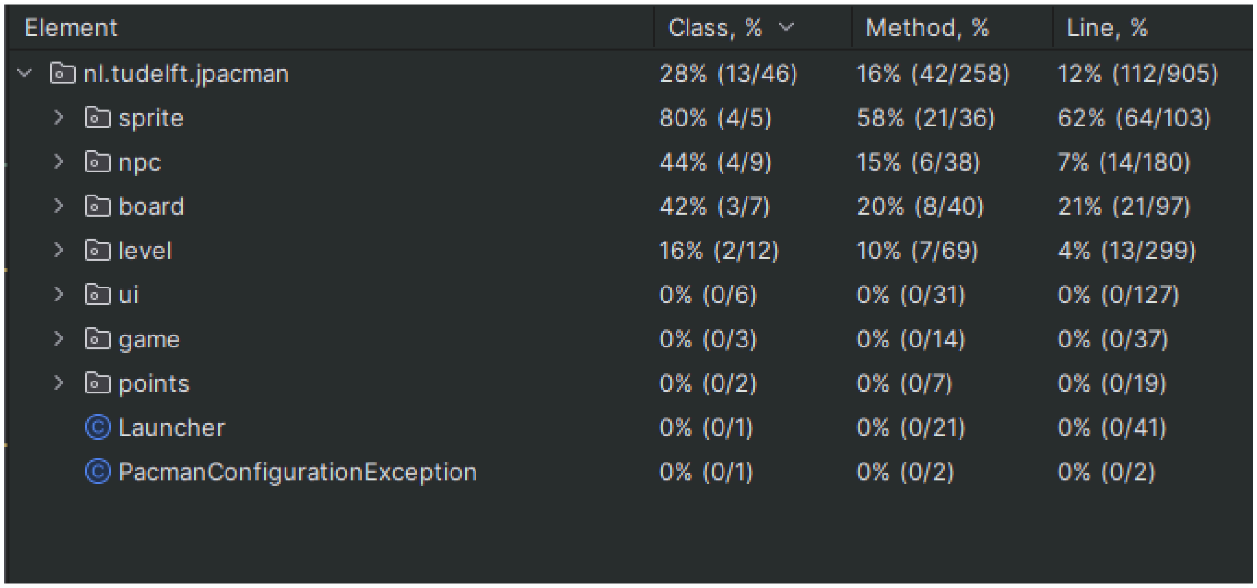
```
package nl.tudelft.jpacman.board;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import nl.tudelft.jpacman.board.Board;
import nl.tudelft.jpacman.board.Square;
import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.mock;
new *
public class borderTest {

    3 usages
    private final Square[][] grid = {
        { mock(Square.class), mock(Square.class) },
        { mock(Square.class), mock(Square.class) },
        { mock(Square.class), mock(Square.class) }
    };

    new *
    @Test
    void withinBorder() {
       Board test_board = new Board(grid);

        //assertTrue(test_board.withinBorders(1,1));
        assertThat(test_board.withinBorders( x: 1,  y: 1)).isTrue();

    }

    new *
    @Test
    void edgeBorder() {
       Board test_board = new Board(grid);

        assertThat(test_board.withinBorders( x: 0, y: 0)).isTrue();
        assertThat(test_board.withinBorders( x: 2, y: 1)).isTrue();

    }
```



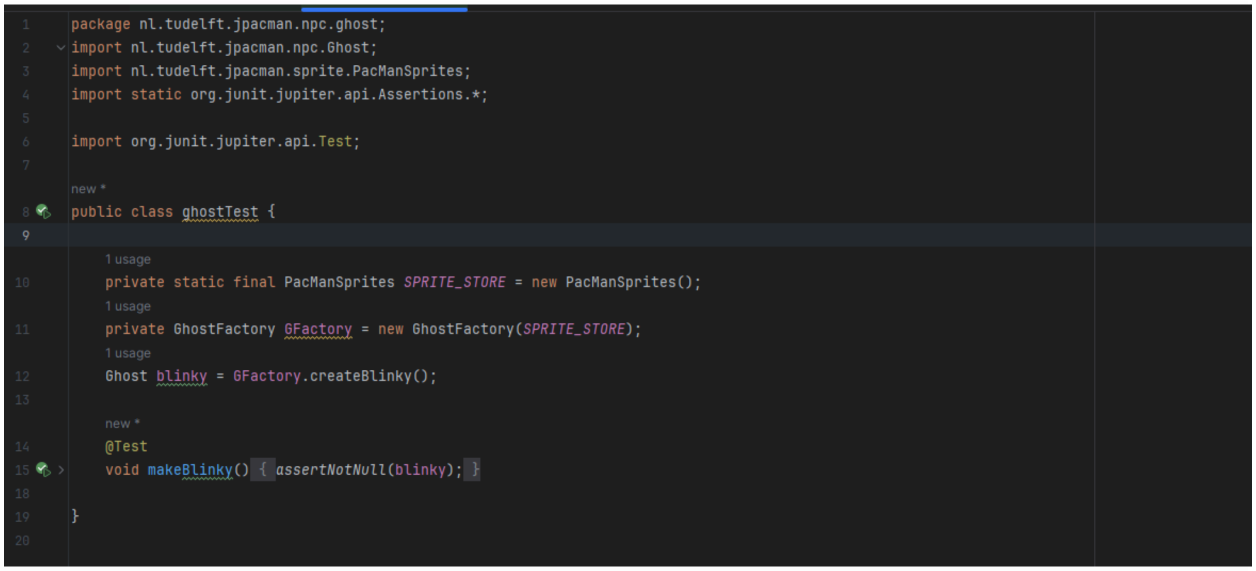
```
new *
@Test
void outsideBorder() {
    Board test_board = new Board(grid);

    assertFalse(test_board.withinBorders( x: -1,  y: 0));
    assertFalse(test_board.withinBorders( x: 0,  y: -1));
    assertFalse(test_board.withinBorders( x: 3,  y: 0));
    assertFalse(test_board.withinBorders( x: 0,  y: 3));
}

}
```

The next method I did a test for was that of withinBoarder. For this test I covered three potential cases. One case was a basic call using coordinates that were within boarders of the mock board I created. The second case covered if the coordinates were on the edge of the grid created. The last case covered if the coordinates were in fact outside of the grid boarders by asserting false to signify the coordinates were not valid within the boarders of the mock board. The results for the 'board' package coverage increased significantly after this test was implemented.

<u>createBlinky method code/results</u>:



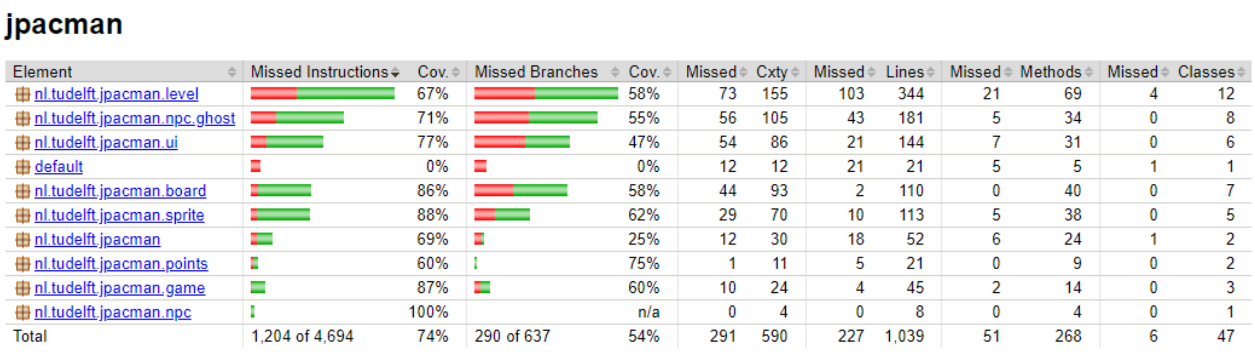
Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	28% (13/46)	16% (42/258)	12% (112/905)
sprite	80% (4/5)	58% (21/36)	62% (64/103)
npc	44% (4/9)	15% (6/38)	7% (14/180)
board	42% (3/7)	20% (8/40)	21% (21/97)
level	16% (2/12)	10% (7/69)	4% (13/299)
ui	0% (0/6)	0% (0/31)	0% (0/127)
game	0% (0/3)	0% (0/14)	0% (0/37)
points	0% (0/2)	0% (0/7)	0% (0/19)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/2)



```
package nl.tudelft.jpacman.npc.ghost;
import nl.tudelft.jpacman.npc.Ghost;
import nl.tudelft.jpacman.sprite.PacManSprites;
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

new *
public class ghostTest {

    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private GhostFactory GFactory = new GhostFactory(SPRITE_STORE);
    1 usage
    Ghost blinky = GFactory.createBlinky();

    new *
    @Test
    void makeBlinky() { assertNotNull(blinky); }

}
```

The last method that I covered was that of createBlinky. For this method I created a PacManSprites object to pass into a GhostFactory object. Then I used the GhostFactory object to call the creatBlinky() method to create a new blinky ghost. I then asserted that the new blinky was not null. Overall this test yielded significant increase in the test coverage for the npc package.

Task 3 – JaCoCo Report on JPacman



jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		67%		58%	73	155	103	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		88%		62%	29	70	10	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	1,204 of 4,694	74%	290 of 637	54%	291	590	227	1,039	51	268	6	47

Questions:

- Are the coverage results from `JaCoCo` similar to the ones you got from `IntelliJ` in the last task? Why so or why not?

    No, the numbers are not the same between IntelliJ and JaCoCo. To me based on my interpretation of JaCoCo the numbers seem to be better in some cases with better coverage percentage. This does cover the branching coverage as well so that could have impacted the coverage.

- Did you find helpful the source code visualization from `JaCoCo` on uncovered branches?

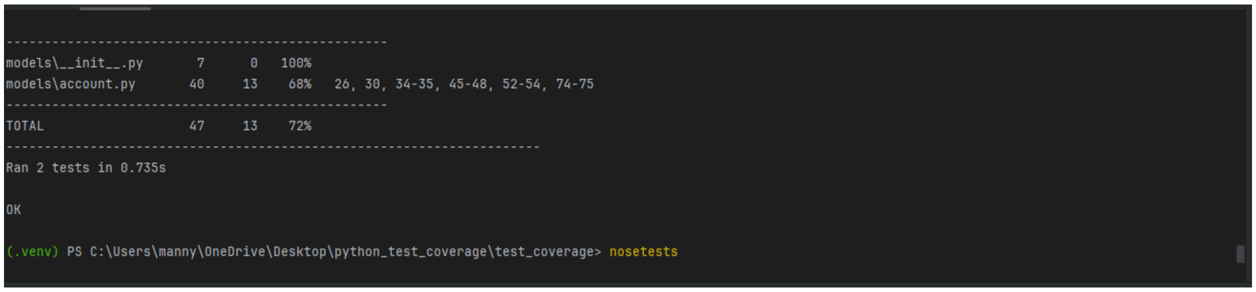
  Yes, having that additional coverage included is very helpful to understanding what code needs to have improved branch coverage which is something that can greatly affect coverage of a project overall.

- Which visualization did you prefer and why? `IntelliJ`'s coverage window or `JaCoCo`'s report?

  For quick feedback I would say I prefer IntelliJ's coverage window as it is fast and slightly more convenient, however for a more in depth analysis of the coverage I think JaCoCo's report is better overall.

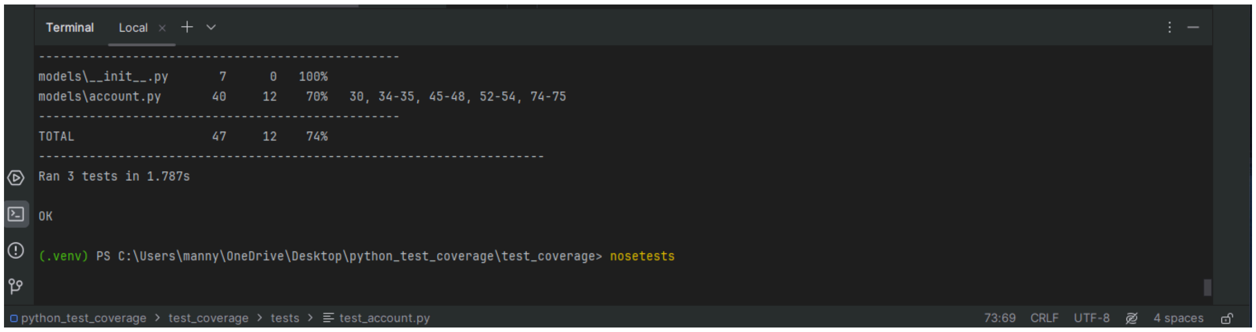
***More below**

Task 4 – Working with Python Test Coverage



```
--------------------------------------------------
models\__init__.py       7      0   100%
models\account.py       40     13    68%   26, 30, 34-35, 45-48, 52-54, 74-75
--------------------------------------------------
TOTAL                   47     13    72%
----------------------------------------------------------------------
Ran 2 tests in 0.735s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\python_test_coverage\test_coverage> nosetests
```

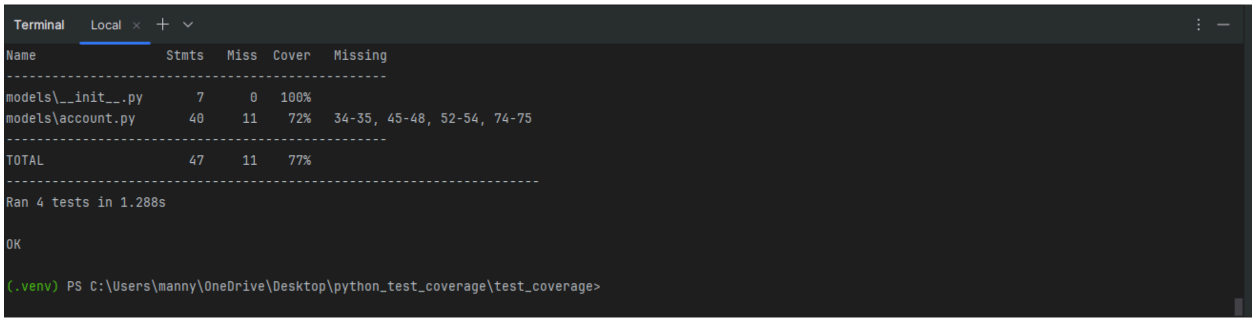
First run of nosetests



```
--------------------------------------------------
models\__init__.py       7      0   100%
models\account.py       40     12    70%   30, 34-35, 45-48, 52-54, 74-75
--------------------------------------------------
TOTAL                   47     12    74%
----------------------------------------------------------------------
Ran 3 tests in 1.787s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\python_test_coverage\test_coverage> nosetests
```

python_test_coverage > test_coverage > tests > test_account.py
73:69 CRLF UTF-8 4 spaces

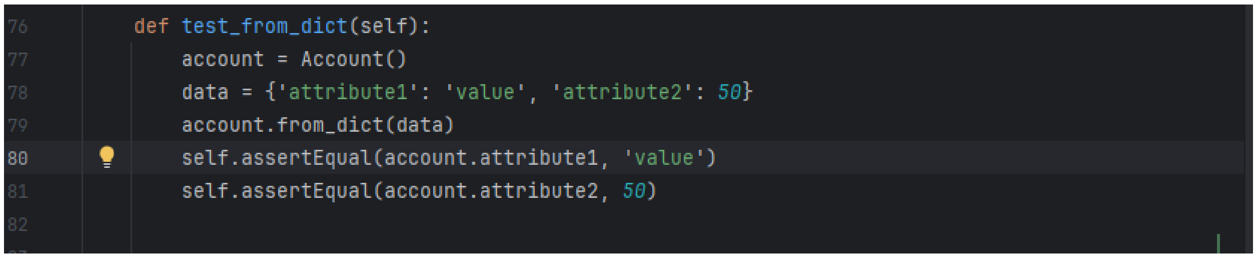
Run after adding test_repr



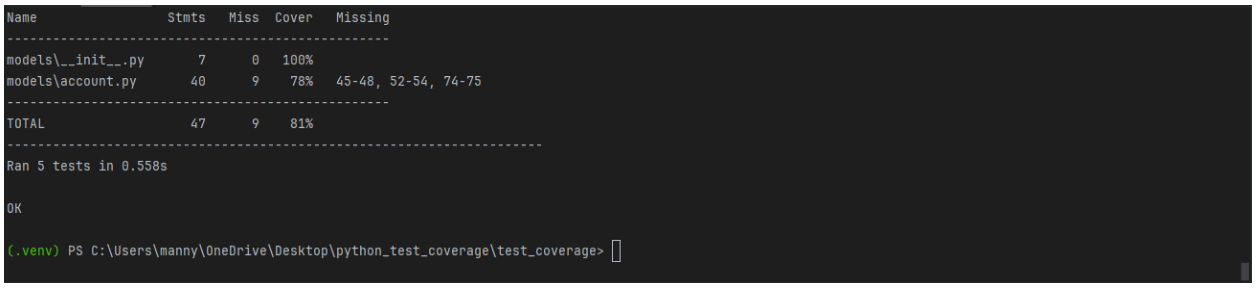
```
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models\__init__.py       7      0   100%
models\account.py       40     11    72%   34-35, 45-48, 52-54, 74-75
--------------------------------------------------
TOTAL                   47     11    77%
----------------------------------------------------------------------
Ran 4 tests in 1.288s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\python_test_coverage\test_coverage>
```

Run after addingn test_to_dict

My tests:

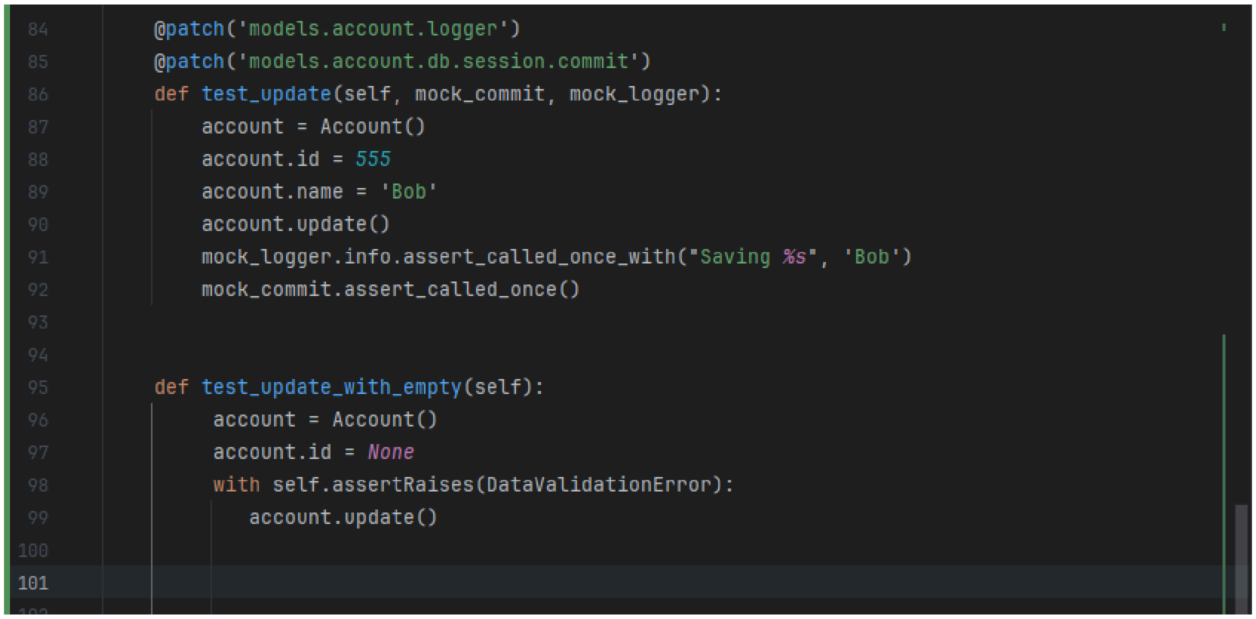


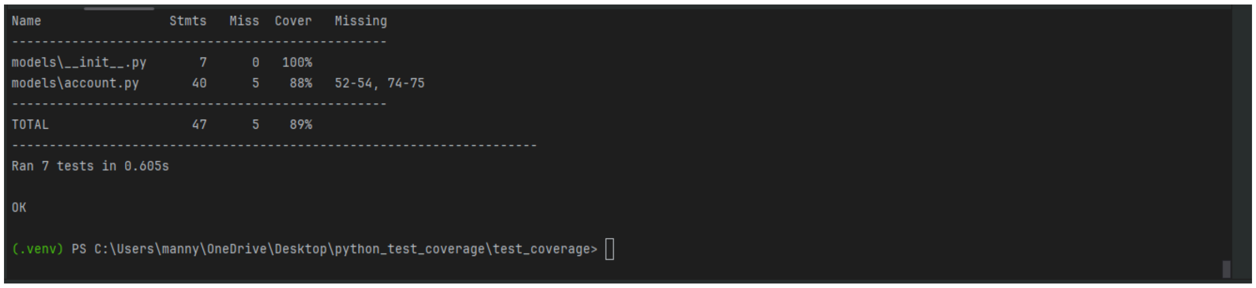
```
    def test_from_dict(self):
        account = Account()
        data = {'attribute1': 'value', 'attribute2': 50}
        account.from_dict(data)
        self.assertEqual(account.attribute1, 'value')
        self.assertEqual(account.attribute2, 50)
```



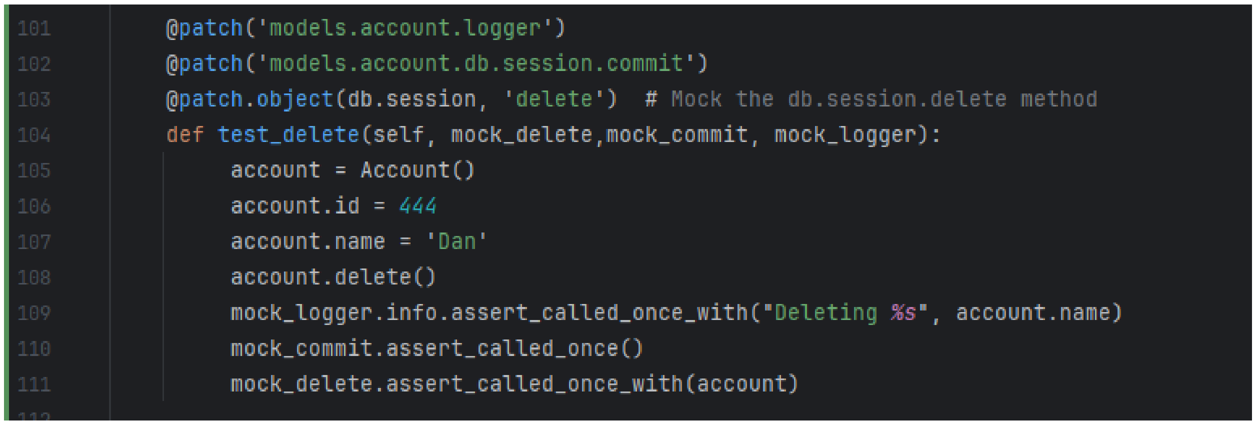
```
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models\__init__.py       7      0   100%
models\account.py       40      9    78%   45-48, 52-54, 74-75
--------------------------------------------------
TOTAL                   47      9    81%
----------------------------------------------------------------------
Ran 5 tests in 0.558s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\python_test_coverage\test_coverage>
```

For this test I tested the test_from_dict method. I created an dictionary with the attributes value and 50 and then passed it into the from_dict function and then asserted them using assertEqual. This ended up increasing the overall coverage by 4%.

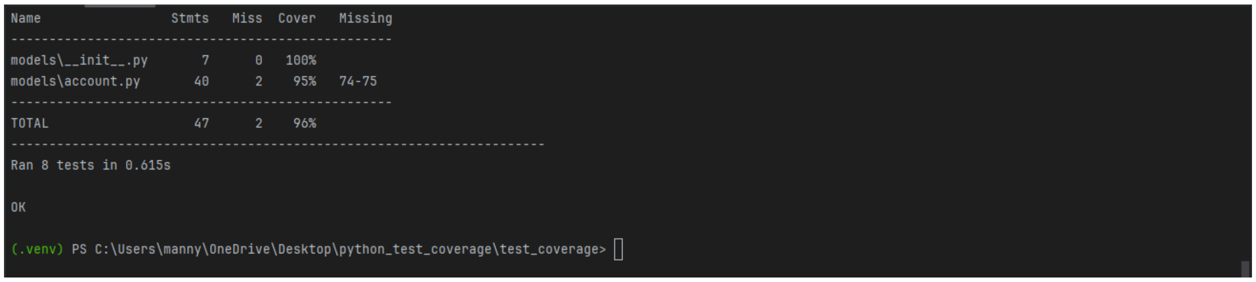

```python
    @patch('models.account.logger')
    @patch('models.account.db.session.commit')
    def test_update(self, mock_commit, mock_logger):
        account = Account()
        account.id = 555
        account.name = 'Bob'
        account.update()
        mock_logger.info.assert_called_once_with("Saving %s", 'Bob')
        mock_commit.assert_called_once()


    def test_update_with_empty(self):
         account = Account()
         account.id = None
         with self.assertRaises(DataValidationError):
             account.update()
```


```
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 7 tests in 0.605s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\python_test_coverage\test_coverage>
```

For the next lines that needed coverage I created to tests in test_update to update accounts and test_update_with_empty to account for trying to update an account with an empty id. I passed in the mock_commit and mock_logger using the Mock import. This allowed me to call the logger to print to the users console about the action that is happening as well as call a mock of commit() for committing account changes. For test_update I made an account and set the id as 555 and the name to Bob. I then called update as well as the logger and commit to simulate the updating of an account. This vastly improved the coverage increasing the total test coverage to 89%.

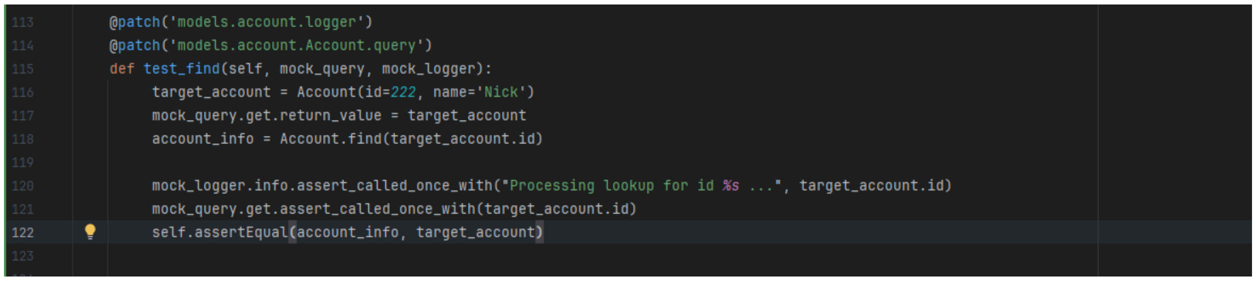


```
    @patch('models.account.logger')
    @patch('models.account.db.session.commit')
    @patch.object(db.session, 'delete')  # Mock the db.session.delete method
    def test_delete(self, mock_delete,mock_commit, mock_logger):
        account = Account()
        account.id = 444
        account.name = 'Dan'
        account.delete()
        mock_logger.info.assert_called_once_with("Deleting %s", account.name)
        mock_commit.assert_called_once()
        mock_delete.assert_called_once_with(account)
```

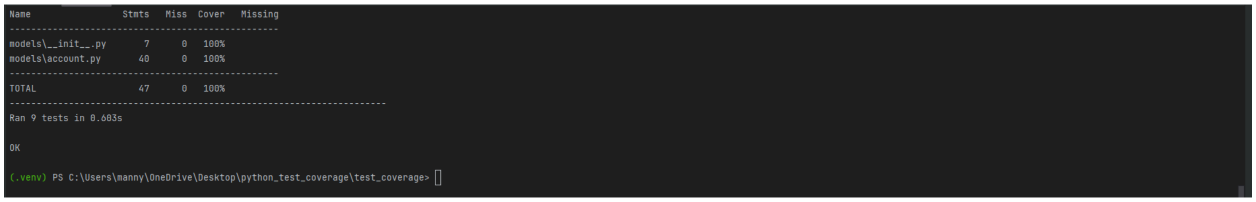


```
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models\__init__.py       7      0   100%
models\account.py       40      2    95%   74-75
--------------------------------------------------
TOTAL                   47      2    96%
----------------------------------------------------------------------
Ran 8 tests in 0.615s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\python_test_coverage\test_coverage>
```

The next test that I created was test_delete to test the deletion of an account. Similar to the previous tests I passed in mock_commit and mock_logger using the Mock import. This time I also passed in mock_delete which is an instance of MagicMock use to mock db.session.delete. Again I made an account calling Account() and this time made the id of this account 444 and the name Dan. I then called account delete to call the delete method on the account instance. I the called mock_logger.info.assert_called_once_with to assert that info of the mocked logger was called exactly once with the arguments needed. This is similar to how I called mock_commit.assert_called_once and mock_delete.assert_called_once_with(account) to assert that the commit and delete methods were called exactly once.



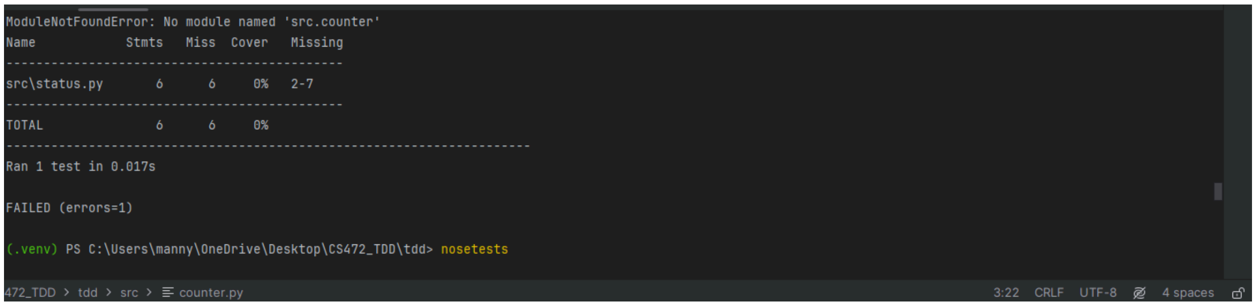
```python
    @patch('models.account.logger')
    @patch('models.account.Account.query')
    def test_find(self, mock_query, mock_logger):
        target_account = Account(id=222, name='Nick')
        mock_query.get.return_value = target_account
        account_info = Account.find(target_account.id)

        mock_logger.info.assert_called_once_with("Processing lookup for id %s ...", target_account.id)
        mock_query.get.assert_called_once_with(target_account.id)
        self.assertEqual(account_info, target_account)
```



```
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 0.603s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\python_test_coverage\test_coverage>
```

The last test I made was test_find to cover the find method. Here I passed in mock_query and mock_logger in addition to the ‘self’ parameter passed in all the test in all the functions. I did this by using decorator patches similar to the previous code. Next, I defined a target account by calling Account() to create an instance of the Account class or basically make an account. I then called the find method using Account to retrieve the account with the specified ID. I then called logger for the same reason as in the other tests but with an additional use of mock_query which asserts that the get method of the mocked query was called once. Lastly, I called assertEqual to assert that the results of the find method match that of the target account.

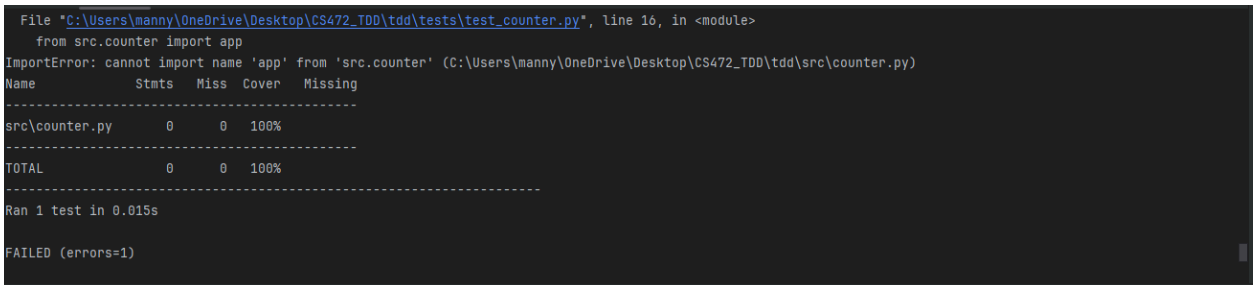
Task 5 - TDD



```
ModuleNotFoundError: No module named 'src.counter'
Name           Stmts   Miss  Cover   Missing
--------------------------------------------
src\status.py      6      6     0%   2-7
--------------------------------------------
TOTAL              6      6     0%
----------------------------------------------------------------------
Ran 1 test in 0.017s

FAILED (errors=1)

(.venv) PS C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd> nosetests
```

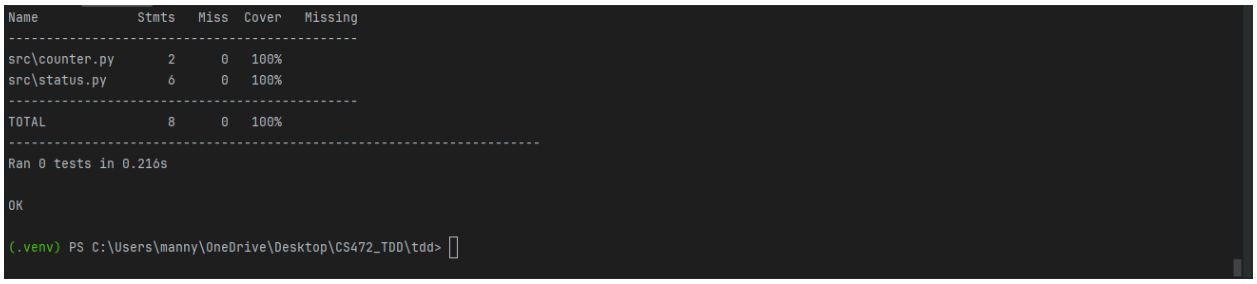
472_TDD > tdd > src > counter.py   3:22 CRLF UTF-8 4 spaces

ModuleNotFoundError: after adding imports and class CounterTest to test_counter.py



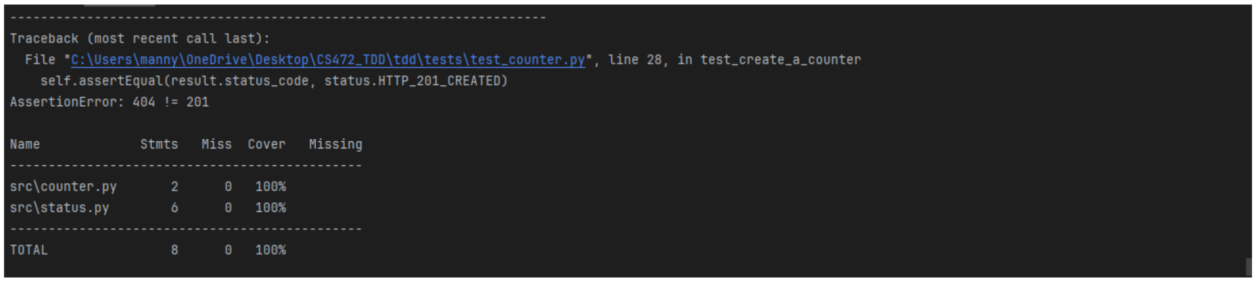
```
  File "C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd\tests\test_counter.py", line 16, in <module>
    from src.counter import app
ImportError: cannot import name 'app' from 'src.counter' (C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd\src\counter.py)
Name            Stmts   Miss  Cover   Missing
---------------------------------------------
src\counter.py      0      0   100%
---------------------------------------------
TOTAL               0      0   100%
----------------------------------------------------------------------
Ran 1 test in 0.015s

FAILED (errors=1)
```

ImportError: cannot import name 'app' from 'src.counter' after creating counter.py



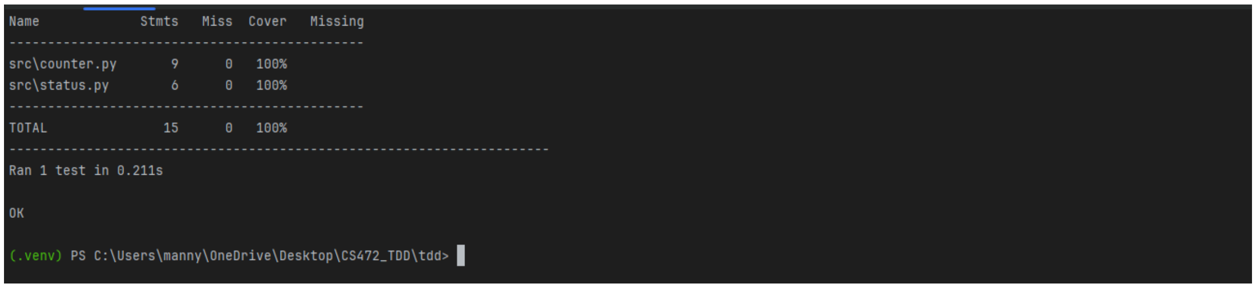
```
Name            Stmts   Miss  Cover   Missing
---------------------------------------------
src\counter.py      2      0   100%
src\status.py       6      0   100%
---------------------------------------------
TOTAL               8      0   100%
----------------------------------------------------------------------
Ran 0 tests in 0.216s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd> 
```

Successful run after adding Flask import to counter.py



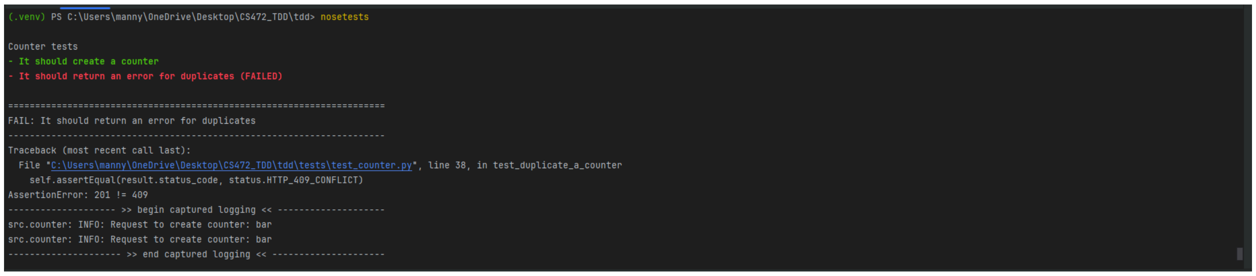
```
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd\tests\test_counter.py", line 28, in test_create_a_counter
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
AssertionError: 404 != 201

Name             Stmts   Miss  Cover   Missing
----------------------------------------------
src\counter.py       2      0   100%
src\status.py        6      0   100%
----------------------------------------------
TOTAL                8      0   100%
```

AssertionError: 404 !=201 after adding test_create_a_counter



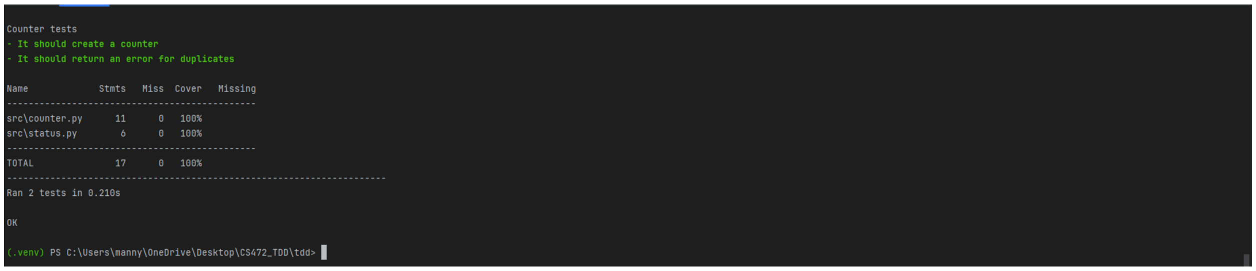
```
Name             Stmts   Miss  Cover   Missing
----------------------------------------------
src\counter.py       9      0   100%
src\status.py        6      0   100%
----------------------------------------------
TOTAL               15      0   100%
----------------------------------------------------------------------
Ran 1 test in 0.211s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd>
```

Successful run after adding create_counter code to counter.py to make endpoint



```
(.venv) PS C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd> nosetests

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

======================================================================
FAIL: It should return an error for duplicates
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd\tests\test_counter.py", line 38, in test_duplicate_a_counter
    self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
AssertionError: 201 != 409
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: bar
src.counter: INFO: Request to create counter: bar
--------------------- >> end captured logging << ---------------------
```

AssertionError: 201 != 409 after adding self() and test_duplicate_a_counter

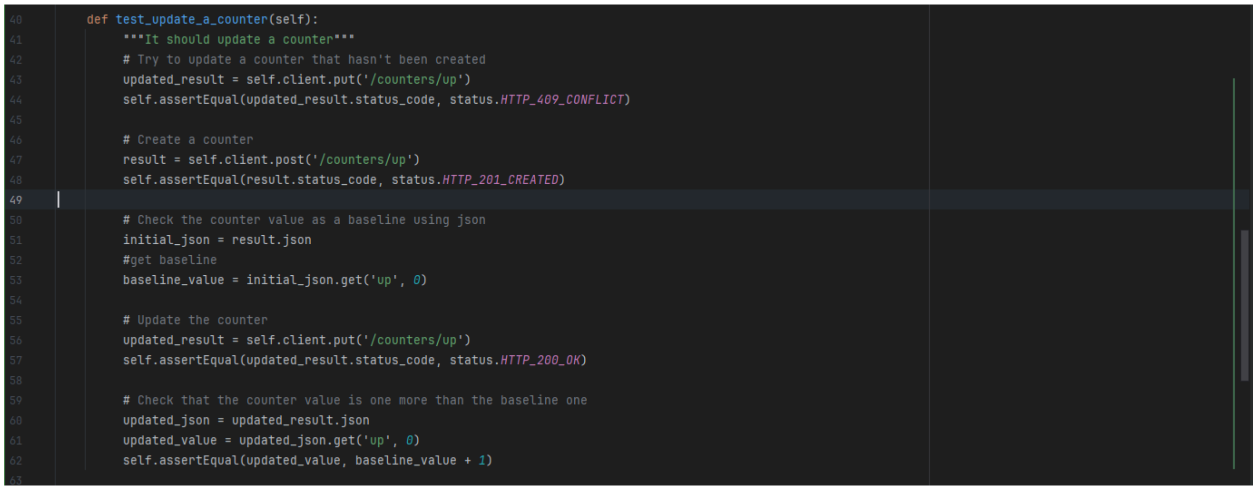


```
Counter tests
- It should create a counter
- It should return an error for duplicates

Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\counter.py       11      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                17      0   100%
----------------------------------------------------------------------
Ran 2 tests in 0.210s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd>
```

Success after adding check if counter exists before creating it

My code:

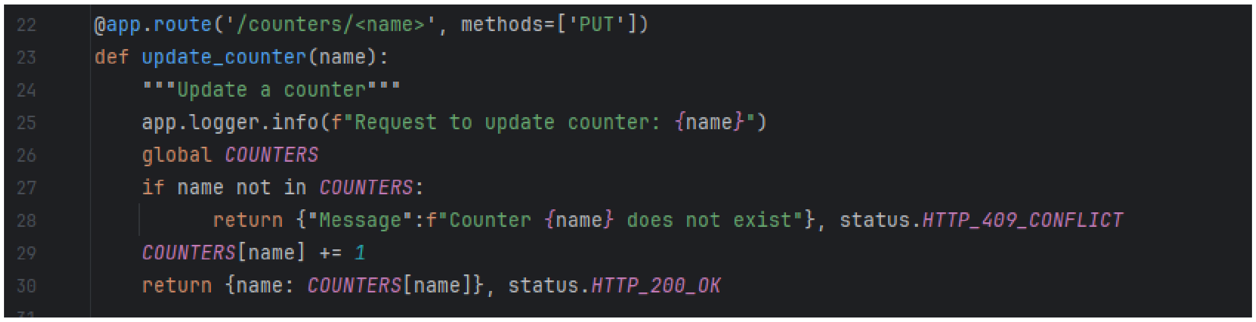
Code for test_update_a_counter in test_counter.py:



```python
    def test_update_a_counter(self):
        """It should update a counter"""
        # Try to update a counter that hasn't been created
        updated_result = self.client.put('/counters/up')
        self.assertEqual(updated_result.status_code, status.HTTP_409_CONFLICT)

        # Create a counter
        result = self.client.post('/counters/up')
        self.assertEqual(result.status_code, status.HTTP_201_CREATED)

        # Check the counter value as a baseline using json
        initial_json = result.json
        #get baseline
        baseline_value = initial_json.get('up', 0)

        # Update the counter
        updated_result = self.client.put('/counters/up')
        self.assertEqual(updated_result.status_code, status.HTTP_200_OK)

        # Check that the counter value is one more than the baseline one
        updated_json = updated_result.json
        updated_value = updated_json.get('up', 0)
        self.assertEqual(updated_value, baseline_value + 1)
```

This code covers a test to update a counter. Here I made a call to create a counter as provided in previous code. I then called assert to assert that counter was successfully created. For this next part I had some issues in the RED phase where my code was not working properly. I was eventually able to figure out that I could use json to parse the result into a python dictionary too contain the data return by the API. The using get() on the dictionary allowed me to retrieve the value associated with the key 'up' and if the key is not present in the dictionary to return the value 0 by default. After retrieving the baseline value and storing it I used the PUT method to request an update for the counter associated with 'up' endpoint. Then I called assert using self again but this time passing the HTTP_200_OK code to suggest that the counter was successfully updated. Finally, using json I retrieve the updated counter in a python dictionary and call assert to suggest that the updated counter value is one more than the baseline counter value.

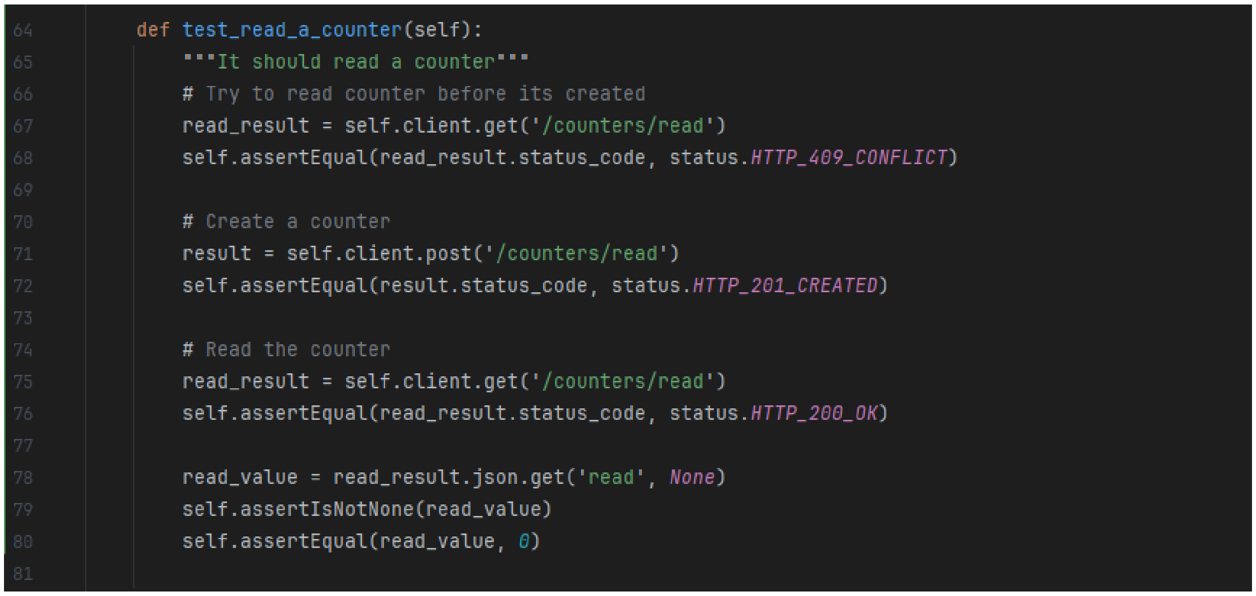
Code for update_counter in counter.py:



```
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    app.logger.info(f"Request to update counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
        return {"Message":f"Counter {name} does not exist"}, status.HTTP_409_CONFLICT
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

This is the endpoint code for counter/up. In this code I followed a similar pattern to the code provided in the tasks. I established a route to /counters/name using the PUT method in regards to REST API. I then called the logger to log the action being performed. Next, I incremented the counter by 1 to update it and returned the counter name and a 200 status to suggest the operation was successful. Finally, I entered a REFRACTOR phase and added a check to see if the counter was not created yet and if it wasn't to send a 409 code suggesting there was an error when attempting update said counter.

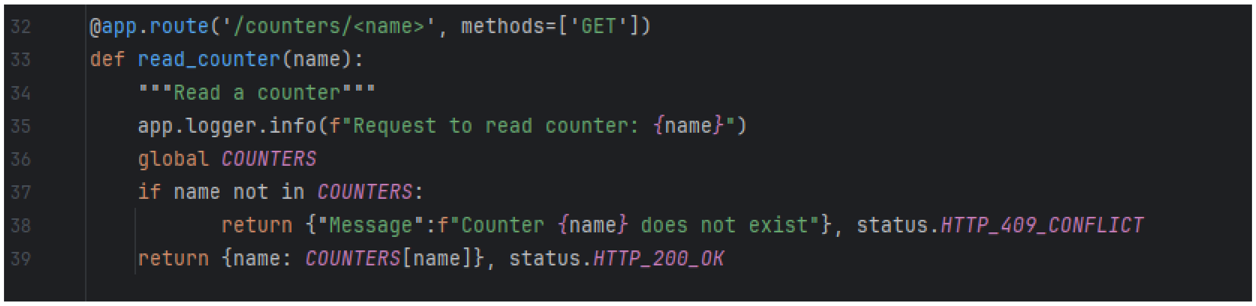
Code for test_read_a_counter in test_counter.py:



```
    def test_read_a_counter(self):
        """It should read a counter"""
        # Try to read counter before its created
        read_result = self.client.get('/counters/read')
        self.assertEqual(read_result.status_code, status.HTTP_409_CONFLICT)

        # Create a counter
        result = self.client.post('/counters/read')
        self.assertEqual(result.status_code, status.HTTP_201_CREATED)

        # Read the counter
        read_result = self.client.get('/counters/read')
        self.assertEqual(read_result.status_code, status.HTTP_200_OK)

        read_value = read_result.json.get('read', None)
        self.assertIsNotNone(read_value)
        self.assertEqual(read_value, 0)
```

This is the code I wrote for a test case to cover reading a counter. Similarly, I modeled this test like the previous code and created a counter asserting a 201_CREATED status to suggest that the counter was successfully created. Then I used get since this required the get method on the endpoint counters/read to read the result and return a 200_OK status to again suggest that the counter was successfully read. After that I parsed the read result as a python dictionary using json and used get() to retrieve the value associated with the key 'read' in the data. 'None' is returned if the key 'read' does not exist. I then called assert to make sure that the read value is not None. In addition I used the last statement assertEqual to verify that the read value is equal to 0 too signify the value of the counter before any updates. Finally, after some test and time in the RED phase I added a read for a counter that did not exist and returned a 409_CONFLICT status to cover all possible cases. This helped shift me into the GREEN phase where I had no errors.

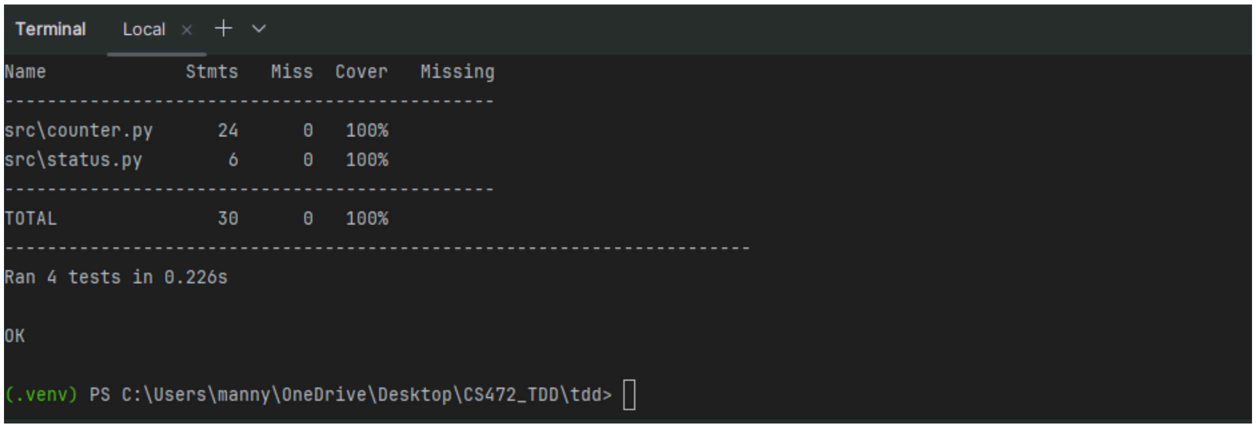
Code for read_counter in counter.py:



```
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Read a counter"""
    app.logger.info(f"Request to read counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
            return {"Message":f"Counter {name} does not exist"}, status.HTTP_409_CONFLICT
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

For the endpoint portion of the read counter implementation I first established a route and used the GET method for it. I then called the logger like before and declared global counters. Then I returned the name and a 200_OK status to indicte a successfully read counter. After this I entered a REFRACTOR phase where I added a check to see if the counter did not exist meaning it could not be read and needed to return a 409_CONFLICT status. This change also helped me enter the GREEN phase a push to 100% test coverage.

Final Coverage:



```
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 0.226s

OK

(.venv) PS C:\Users\manny\OneDrive\Desktop\CS472_TDD\tdd>
```

This is my final coverage for the TDD portion of the software testing assignment with 100% total coverage.