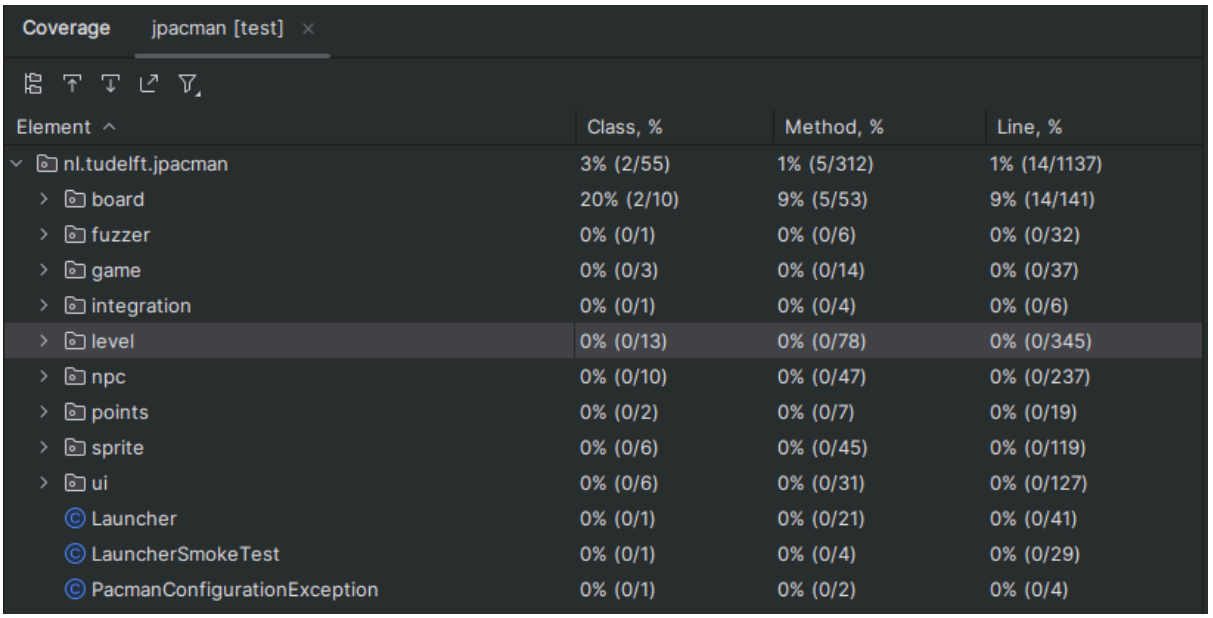
Task 1 – JPacman Test Coverage

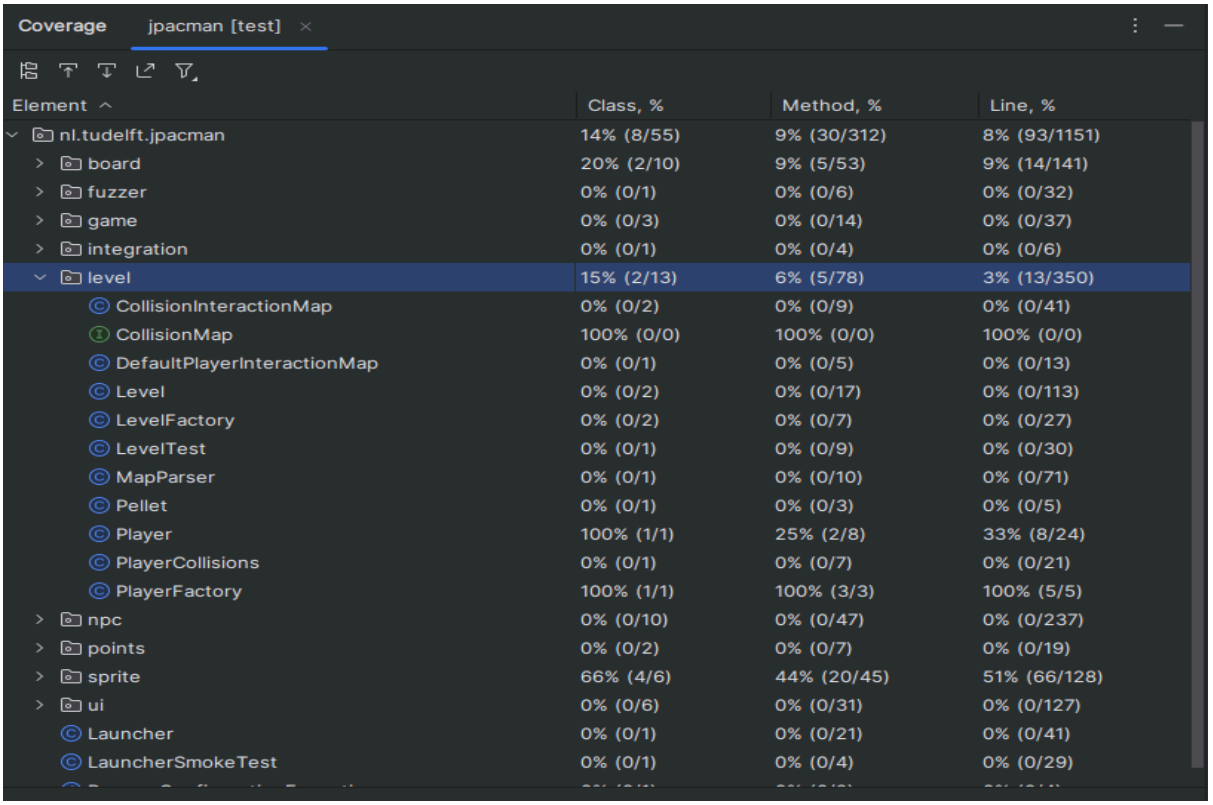


Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	0% (0/13)	0% (0/78)	0% (0/345)
npc	0% (0/10)	0% (0/47)	0% (0/237)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

Is the coverage good enough?

No, because most of them show 0%.

Task 2 – Increasing Coverage on JPacman



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)
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)

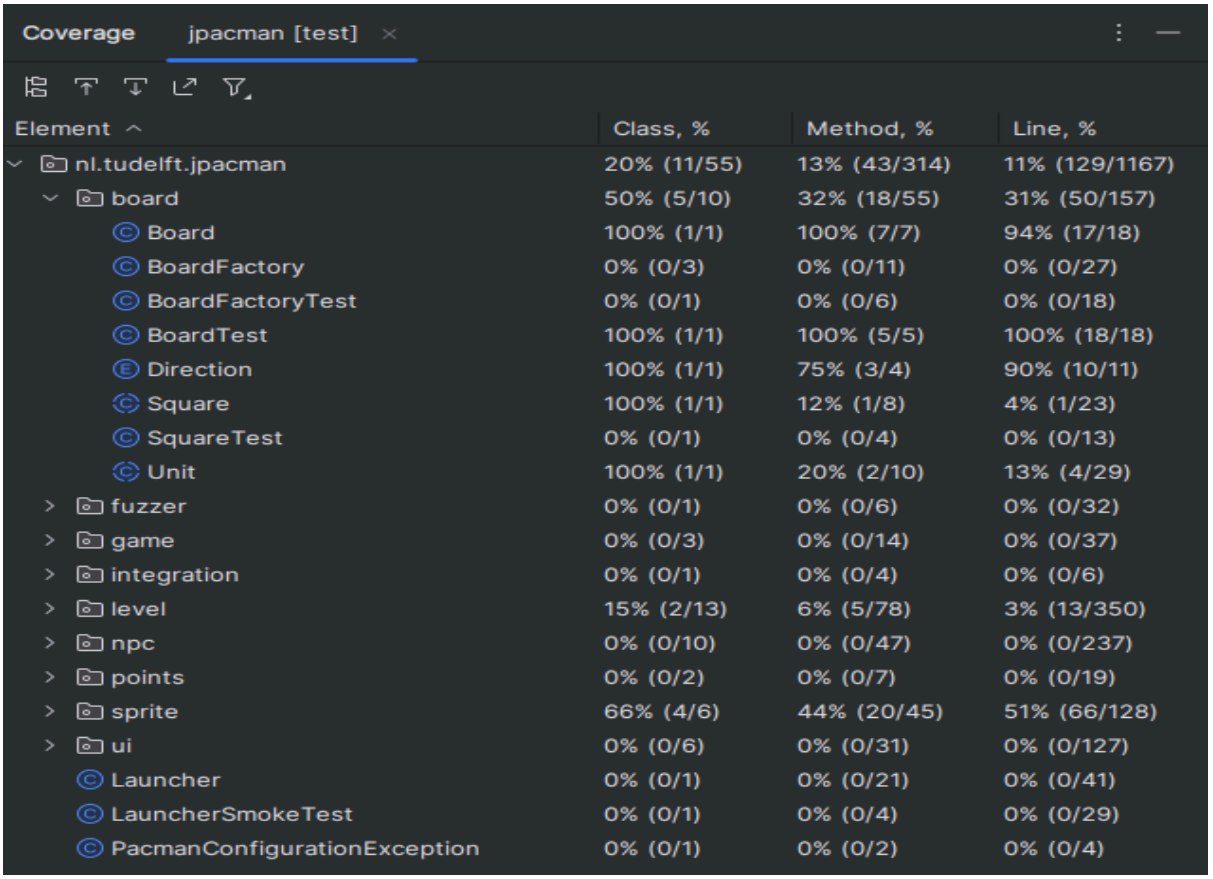
The aim of Task 2 was to enhance the code coverage of the JPacman project, which was found to be insufficient in Task 1, with most packages showing 0% coverage. To address the coverage gaps, new test cases were introduced, focusing primarily on the Player class

within the level package. A new test class, PlayerTest, was created to encapsulate tests for the Player class methods, particularly the isAlive() method.

The test leveraged the PlayerFactory to instantiate a Player object, which otherwise would be complex due to its dependencies. The PlayerFactory, in turn, required a PacManSprites object for player creation. The PlayerTest class was then equipped with a single test, testAlive, to assert the aliveness state of a Player instance.

Upon executing the new test, the code coverage metrics showed an improvement as the test successfully validated the isAlive() method's functionality. The coverage for the Player class increased, contributing positively to the overall test coverage of the project.

Task 2.1



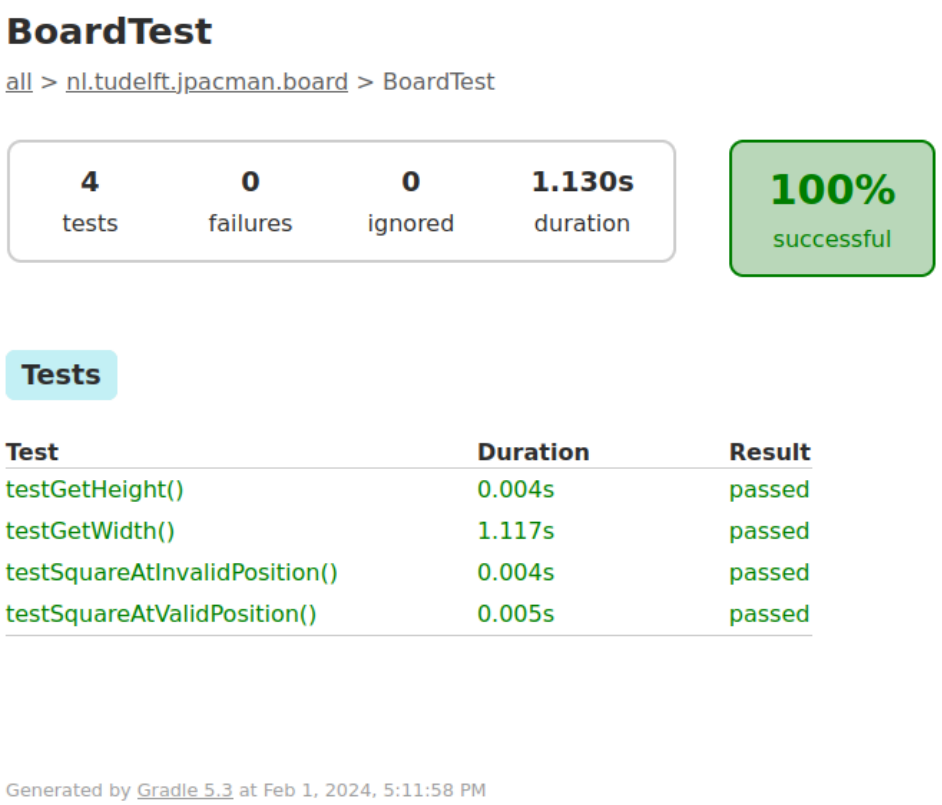
Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	20% (11/55)	13% (43/314)	11% (129/1167)
board	50% (5/10)	32% (18/55)	31% (50/157)
Board	100% (1/1)	100% (7/7)	94% (17/18)
BoardFactory	0% (0/3)	0% (0/11)	0% (0/27)
BoardFactoryTest	0% (0/1)	0% (0/6)	0% (0/18)
BoardTest	100% (1/1)	100% (5/5)	100% (18/18)
Direction	100% (1/1)	75% (3/4)	90% (10/11)
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	15% (2/13)	6% (5/78)	3% (13/350)
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)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

In the recent enhancement of JPacman test coverage, I identified and addressed critical gaps in the unit testing of the Board class. Utilizing Mockito for mock creation and AssertJ for assertions, I implemented tests for getWidth, getHeight, and squareAt methods, which were previously untested.

The implementation of these tests yielded a significant improvement in coverage metrics, with class, method, and line coverage for the Board class increasing to 100%, 100%, and 94%, respectively. This not only bolstered the confidence in the Board's functionality but also contributed to the overall integrity of the JPacman framework.

This testing initiative has set a precedent for further test development within the project, underscoring the importance of thorough testing in maintaining high code quality. The positive impact of these tests on the JPacman test suite highlights the value of comprehensive testing practices.

Task 3 – JaCoCo Report on JPacman



BoardTest

all > nl.tudelft.jpacman.board > BoardTest

4	0	0	1.130s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
testGetHeight()	0.004s	passed
testGetWidth()	1.117s	passed
testSquareAtInvalidPosition()	0.004s	passed
testSquareAtValidPosition()	0.005s	passed

Generated by Gradle 5.3 at Feb 1, 2024, 5:11:58 PM

The investigation into JPacman's test coverage using JaCoCo revealed results that were largely in alignment with those obtained from IntelliJ, albeit with some distinctions. Notably, JaCoCo does not explicitly label the 'method' coverage percentage, which suggests subtle differences in reporting metrics between the tools. Despite this, both tools serve the purpose of highlighting test coverage effectively.

JaCoCo's source code visualization was particularly beneficial for identifying uncovered branches. The color-coded feedback directly within the context of the source code made it easier to pinpoint which specific branches lacked coverage, providing a clear path for additional test development.

Between the two, IntelliJ’s coverage window was preferred for its clarity and more intuitive presentation of data. Its interface is seamlessly integrated with the development environment, offering a more streamlined approach to interpreting coverage metrics and improving overall productivity.

Task 4 – Working with Python Test Coverage

Orignal:

```
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
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 0.281s

OK

shijielin@shijielin-virtual-machine:~/Desktop/test_coverage$
```

After implemented function def test_repr(self):

```
56      """ Test the representation of an account """
57      def test_repr(self):
58          account = Account()
59          account.name = "Foo"
60          self.assertEqual(str(account), "<Account 'Foo'>")
```

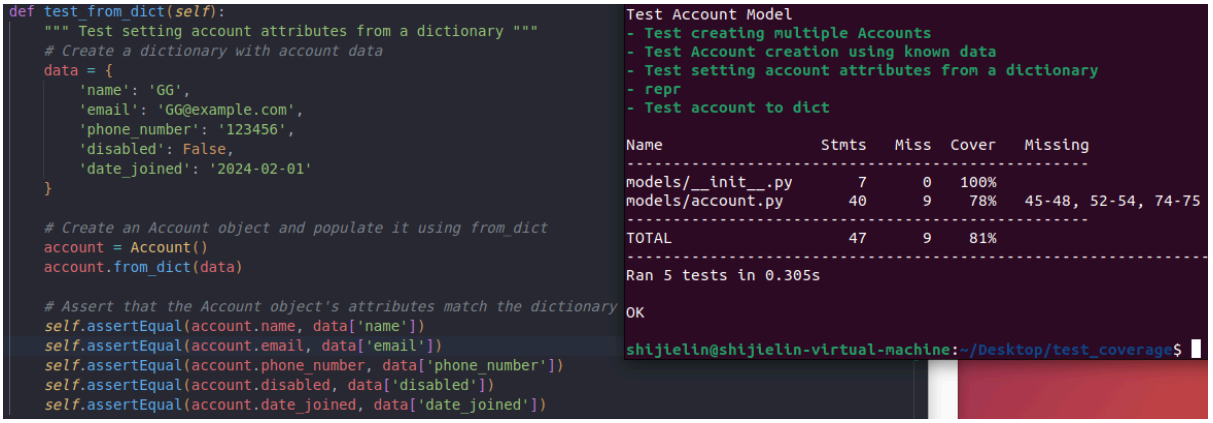
```
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models/__init__.py       7      0   100%
models/account.py       40     12    70%   30, 34-35, 45-48, 52-54, 74-75
--------------------------------------------------
TOTAL                   47     12    74%
----------------------------------------------------------------------
Ran 3 tests in 0.297s

OK

shijielin@shijielin-virtual-machine:~/Desktop/test_coverage$
```

After implemented function def test_to_dict(self):

```
62      def test_to_dict(self):
63          """ Test account to dict """
64          data = ACCOUNT_DATA[self.rand] # get a random account
65          account = Account(**data)
66          result = account.to_dict()
67          self.assertEqual(account.name, result["name"])
68          self.assertEqual(account.email, result["email"])
69          self.assertEqual(account.phone_number, result["phone_number"])
70          self.assertEqual(account.disabled, result["disabled"])
71          self.assertEqual(account.date_joined, result["date_joined"])
```

```
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 0.298s

OK

shijielin@shijielin-virtual-machine:~/Desktop/test_coverage$
```

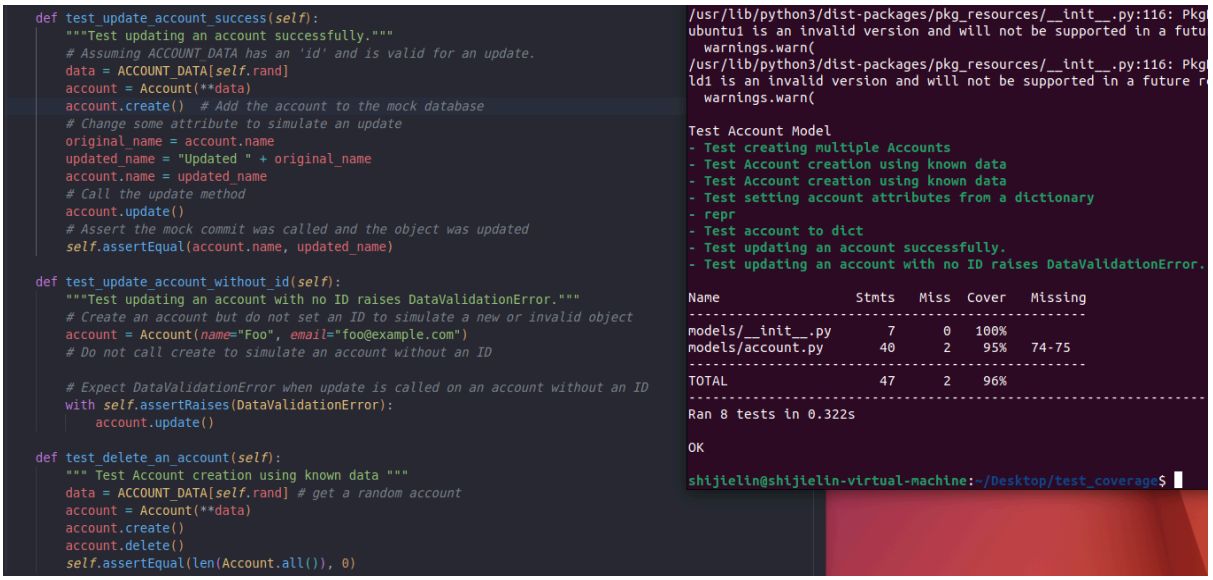
After implemented function def test_from_dict(self):



```
def test_from_dict(self):
    """ Test setting account attributes from a dictionary """
    # Create a dictionary with account data
    data = {
        'name': 'GG',
        'email': 'GG@example.com',
        'phone_number': '123456',
        'disabled': False,
        'date_joined': '2024-02-01'
    }

    # Create an Account object and populate it using from_dict
    account = Account()
    account.from_dict(data)

    # Assert that the Account object's attributes match the dictionary
    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'])
    self.assertEqual(account.date_joined, data['date_joined'])
```

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test setting account attributes from a dictionary
- repr
- Test account to dict

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 0.305s

OK

shijielin@shijielin-virtual-machine:~/Desktop/test_coverage$
```

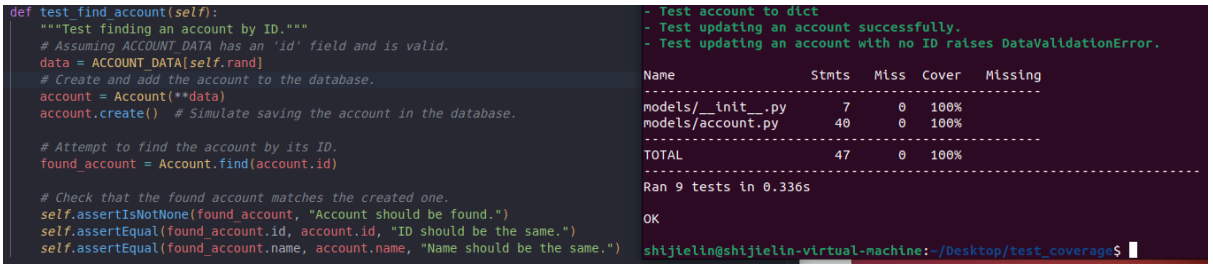
After implemented function def test_update_account_success(self), def test_update_account_without_id(self), and def test_delete_an_account(self).



```
    def test_update_account_success(self):
        """Test updating an account successfully."""
        # Assuming ACCOUNT_DATA has an 'id' and is valid for an update.
        data = ACCOUNT_DATA[self.rand]
        account = Account(**data)
        account.create()  # Add the account to the mock database
        # Change some attribute to simulate an update
        original_name = account.name
        updated_name = "Updated " + original_name
        account.name = updated_name
        # Call the update method
        account.update()
        # Assert the mock commit was called and the object was updated
        self.assertEqual(account.name, updated_name)

    def test_update_account_without_id(self):
        """Test updating an account with no ID raises DataValidationError."""
        # Create an account but do not set an ID to simulate a new or invalid object
        account = Account(name="Foo", email="foo@example.com")
        # Do not call create to simulate an account without an ID

        # Expect DataValidationError when update is called on an account without an ID
        with self.assertRaises(DataValidationError):
            account.update()

    def test_delete_an_account(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()), 0)
```

```
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: Pkg
ubuntu1 is an invalid version and will not be supported in a futu
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: Pkg
ld1 is an invalid version and will not be supported in a future r
  warnings.warn(

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test Account creation using known data
- Test setting account attributes from a dictionary
- repr
- Test account to dict
- Test updating an account successfully.
- Test updating an account with no ID raises DataValidationError.

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 0.322s

OK

shijielin@shijielin-virtual-machine:~/Desktop/test_coverage$
```

After implemented function def test_find_account(self):



```
def test_find_account(self):
    """Test finding an account by ID."""
    # Assuming ACCOUNT_DATA has an 'id' field and is valid.
    data = ACCOUNT_DATA[self.rand]
    # Create and add the account to the database.
    account = Account(**data)
    account.create()  # Simulate saving the account in the database.

    # Attempt to find the account by its ID.
    found_account = Account.find(account.id)

    # Check that the found account matches the created one.
    self.assertIsNotNone(found_account, "Account should be found.")
    self.assertEqual(found_account.id, account.id, "ID should be the same.")
    self.assertEqual(found_account.name, account.name, "Name should be the same.")
```

```
- Test account to dict
- Test updating an account successfully.
- Test updating an account with no ID raises DataValidationError.

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 0.336s

OK

shijielin@shijielin-virtual-machine:~/Desktop/test_coverage$
```

Task 5 - TDD

The first step involved writing a test case for creating a counter in test_counter.py using a PUT request. However, running nosetests immediately flagged an error due to the absence of an endpoint. To resolve this, counter.py was created in the src directory, and Flask was initialized with a basic app configuration.

The focus then shifted to writing the actual test case test_create_a_counter. Upon execution, the test returned RED, indicating a failure due to a missing /counters endpoint. To address this, a new route was created in counter.py with a POST method allowing the creation of counters. This route logged the request, initialized the counter with a zero value, and returned a 201 status code upon successful creation.

The next test was test_duplicate_a_counter, which required the application to handle duplicate counter creations by returning a 409 conflict status. This introduced the RED phase of TDD, as the test failed because the application allowed the creation of counters with the same name. To shift to the GREEN phase, the create_counter function was refactored to check if a counter with the given name already existed before creating it, returning a conflict message if so.

counter.py:

```python
@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:
        COUNTERS.clear()
        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 in COUNTERS:
        COUNTERS[name] += 1
        return {name: COUNTERS[name]}, status.HTTP_200_OK


@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Read a counter"""
    if name in COUNTERS:
        return {name: COUNTERS[name]}, status.HTTP_200_OK
```

Test_counter.py:

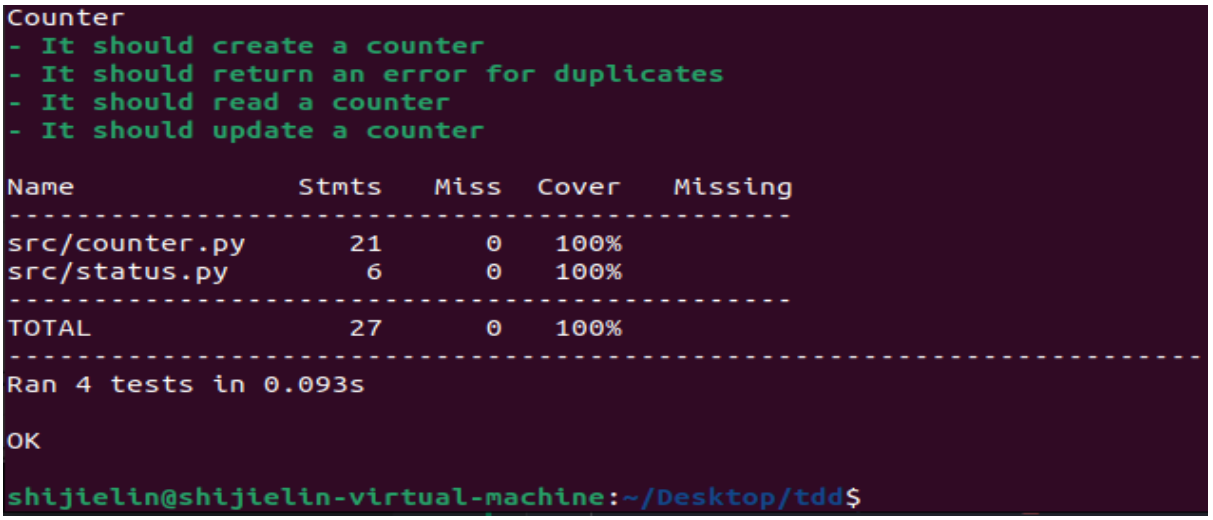
```python
class CounterTest(TestCase):
    def setUp(self):
        """Prepare test case"""
        self.client = app.test_client()

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


    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
        create_result = self.client.post('/counters/foo')
        self.assertEqual(create_result.status_code, status.HTTP_201_CREATED)

        # Check the initial value
        baseline_result = self.client.get('/counters/foo')
        baseline_value = baseline_result.json['foo']

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

        # Check the updated value
        updated_result = self.client.get('/counters/foo')
        updated_value = updated_result.json['foo']
        self.assertEqual(updated_value, baseline_value + 1)

    def test_read_a_counter(self):
        """It should read a counter"""
        # Create a counter
        self.client.post('/counters/test')

        # Read the counter
        result = self.client.get('/counters/test')
        self.assertEqual(result.status_code, status.HTTP_200_OK)
        self.assertEqual(result.json['test'], 0)
```

The task then introduced a requirement to implement a test case for updating a counter. The new test test_update_a_counter followed these steps:

1. It called the create counter endpoint.
2. Verified the creation was successful.
3. Retrieved the initial counter value.
4. Updated the counter with a PUT request.
5. Ensured the update was successful.
6. Checked the counter value was incremented by one.
7. Running nosetests at this point was expected to be in the RED phase since the update functionality was not yet implemented. The corresponding code in counter.py involved adding a PUT route that incremented the counter and returned the updated value with a 200 status code.

Additionally, a test case for reading a counter was written, which simply verified that after creating a counter, its value could be retrieved with a GET request and matched the expected initial value.

Result:



Useful link:

jpacman repo link: https://github.com/AlphabetWolf/jpacman.git
test_coverage repo link: https://github.com/AlphabetWolf/test_coverage.git
tdd repo link: https://github.com/AlphabetWolf/tdd.git