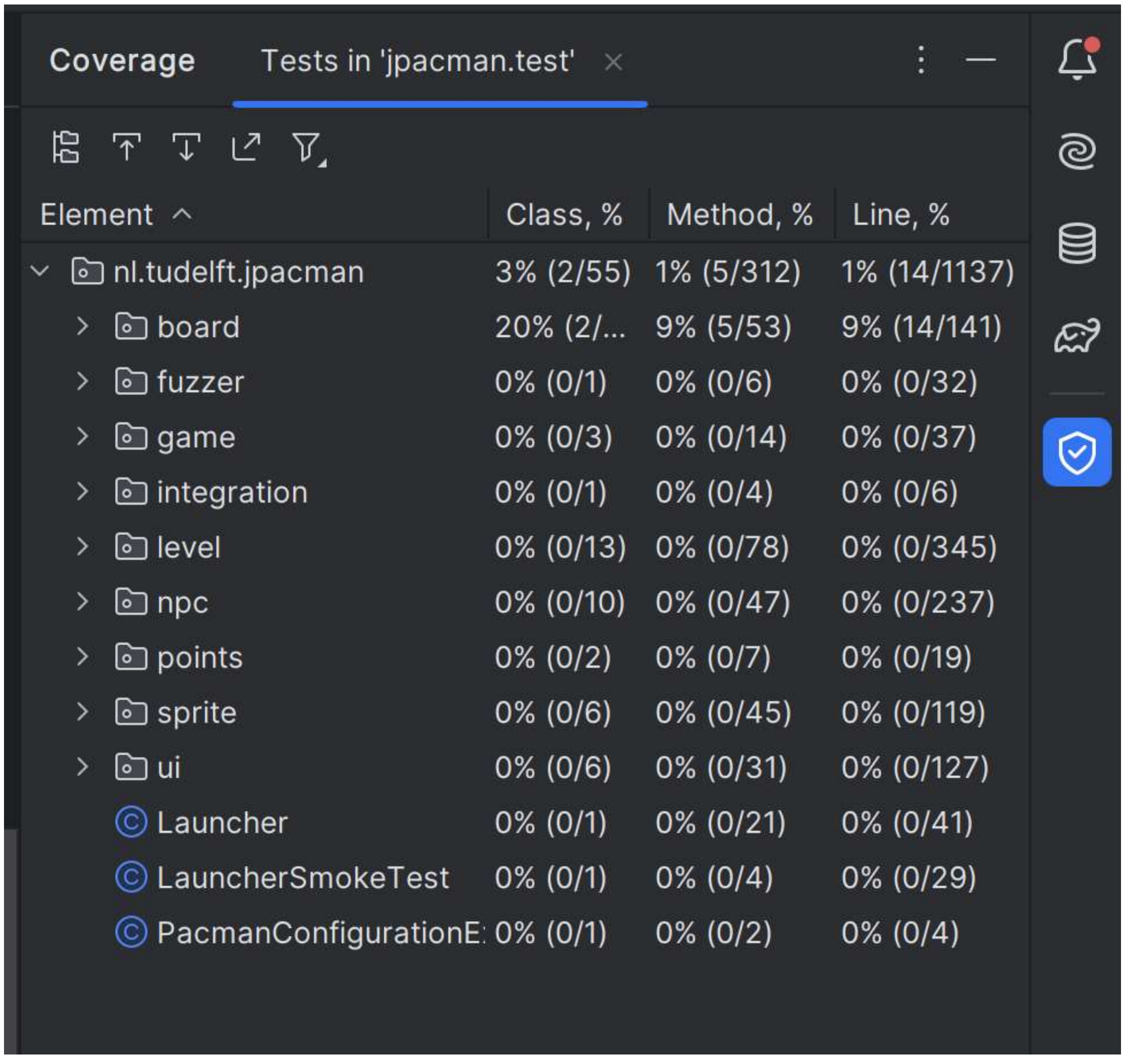
Cade Aihara 02/05/2024

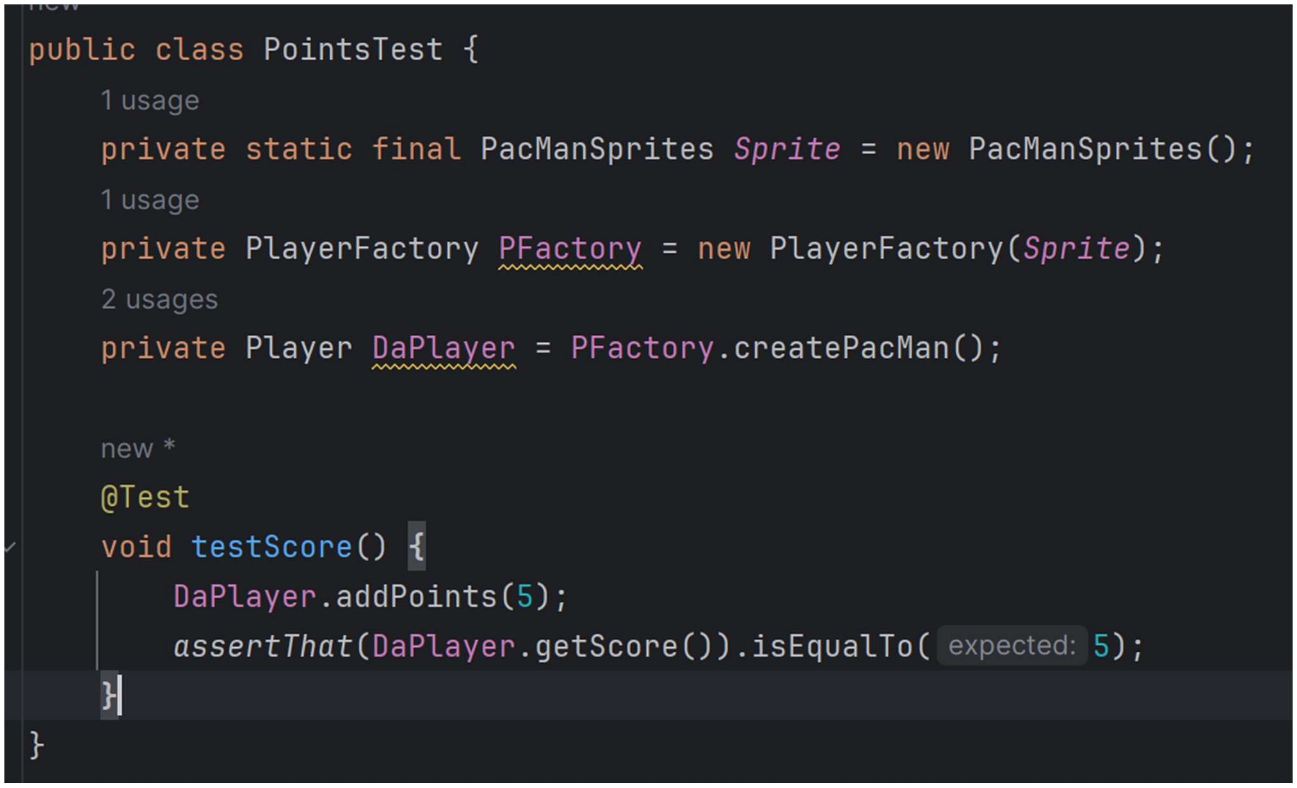
The picture below shows the coverage before doing my tests



Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
board	20% (2/...	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)
PacmanConfigurationE:	0% (0/1)	0% (0/2)	0% (0/4)

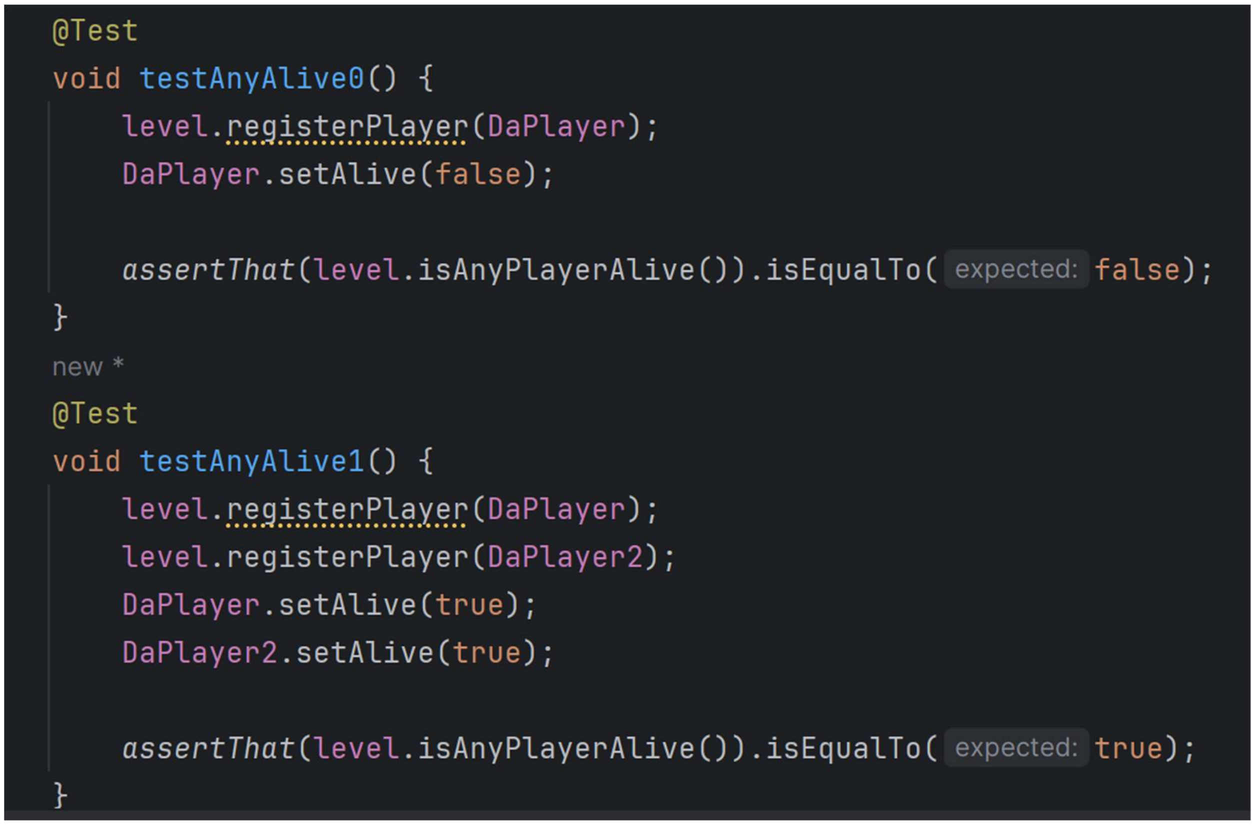
Here is a code snippet of me testing addPoints()

This works by adding 5 points to the default of 0 and confirming that there is 5 points. It has confirmed addPoints works properly.

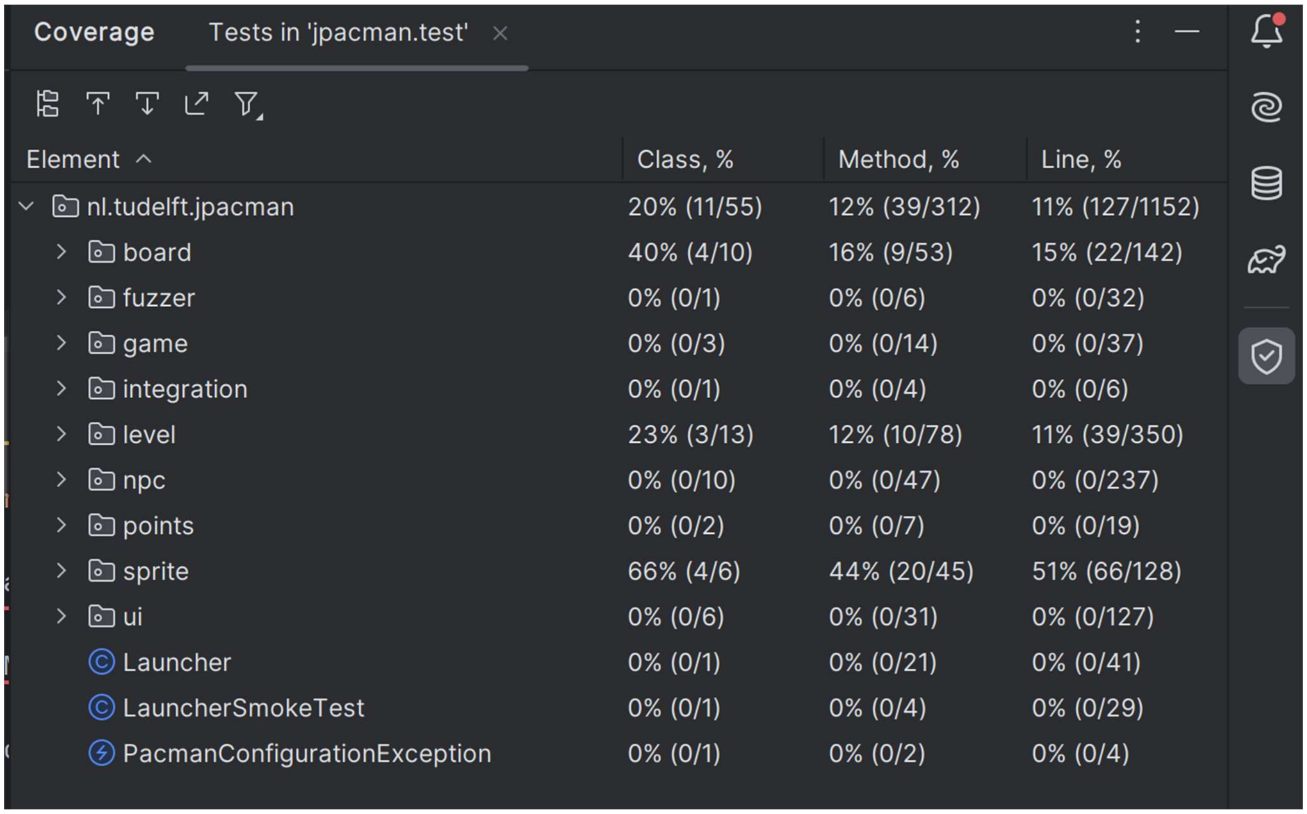


```
public class PointsTest {
    1 usage
    private static final PacManSprites Sprite = new PacManSprites();
    1 usage
    private PlayerFactory PFactory = new PlayerFactory(Sprite);
    2 usages
    private Player DaPlayer = PFactory.createPacMan();

    new *
    @Test
    void testScore() {
        DaPlayer.addPoints(5);
        assertThat(DaPlayer.getScore()).isEqualTo( expected: 5);
    }
}
```

Here is a code snippet of me testing both registerPlayer() and isAnyPlayerAlive(). This works by registering players and making sure they are in the level. Then it tests to see if the players are alive or not respectively. As long as one player is alive, isAnyPlayerAlive() will return true.

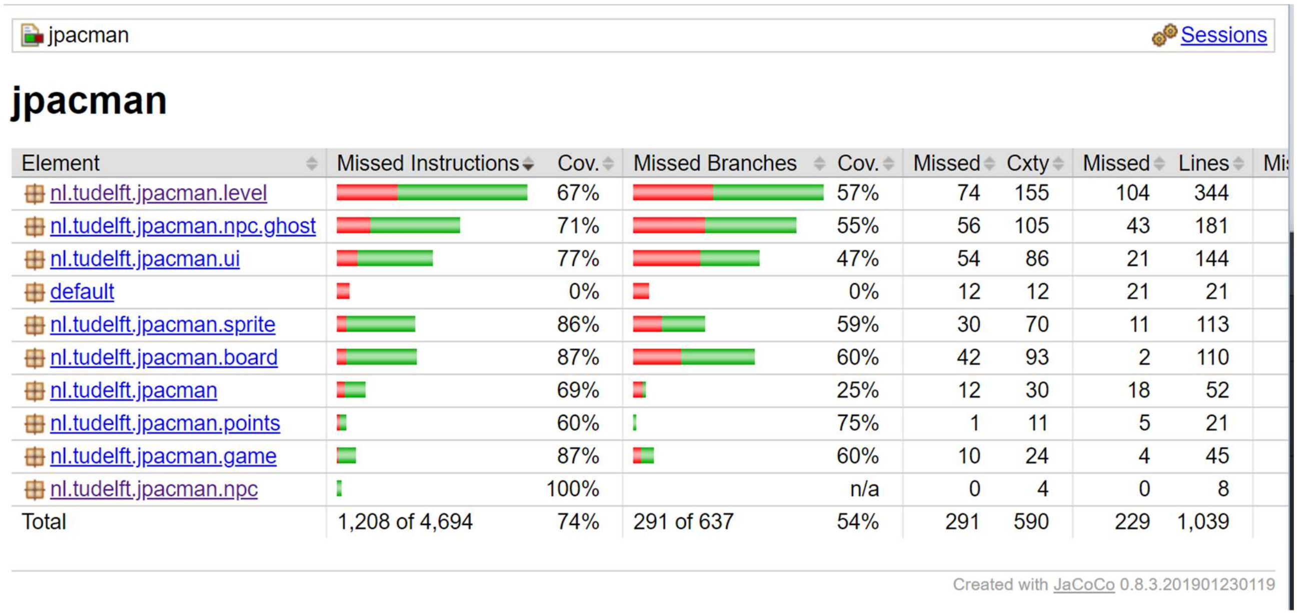

```java
@Test
void testAnyAlive0() {
    level.registerPlayer(DaPlayer);
    DaPlayer.setAlive(false);

    assertThat(level.isAnyPlayerAlive()).isEqualTo( expected: false);
}
new *
@Test
void testAnyAlive1() {
    level.registerPlayer(DaPlayer);
    level.registerPlayer(DaPlayer2);
    DaPlayer.setAlive(true);
    DaPlayer2.setAlive(true);

    assertThat(level.isAnyPlayerAlive()).isEqualTo( expected: true);
}
```

After making my unit tests, we can see an increase in coverage. This shows that the unit tests were successful in ensuring that the methods ran properly. I am happy with the results.


Coverage Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	20% (11/55)	12% (39/312)	11% (127/1152)
board	40% (4/10)	16% (9/53)	15% (22/142)
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	23% (3/13)	12% (10/78)	11% (39/350)
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)

Image from JaCoCo


jpacman Sessions

jpacman

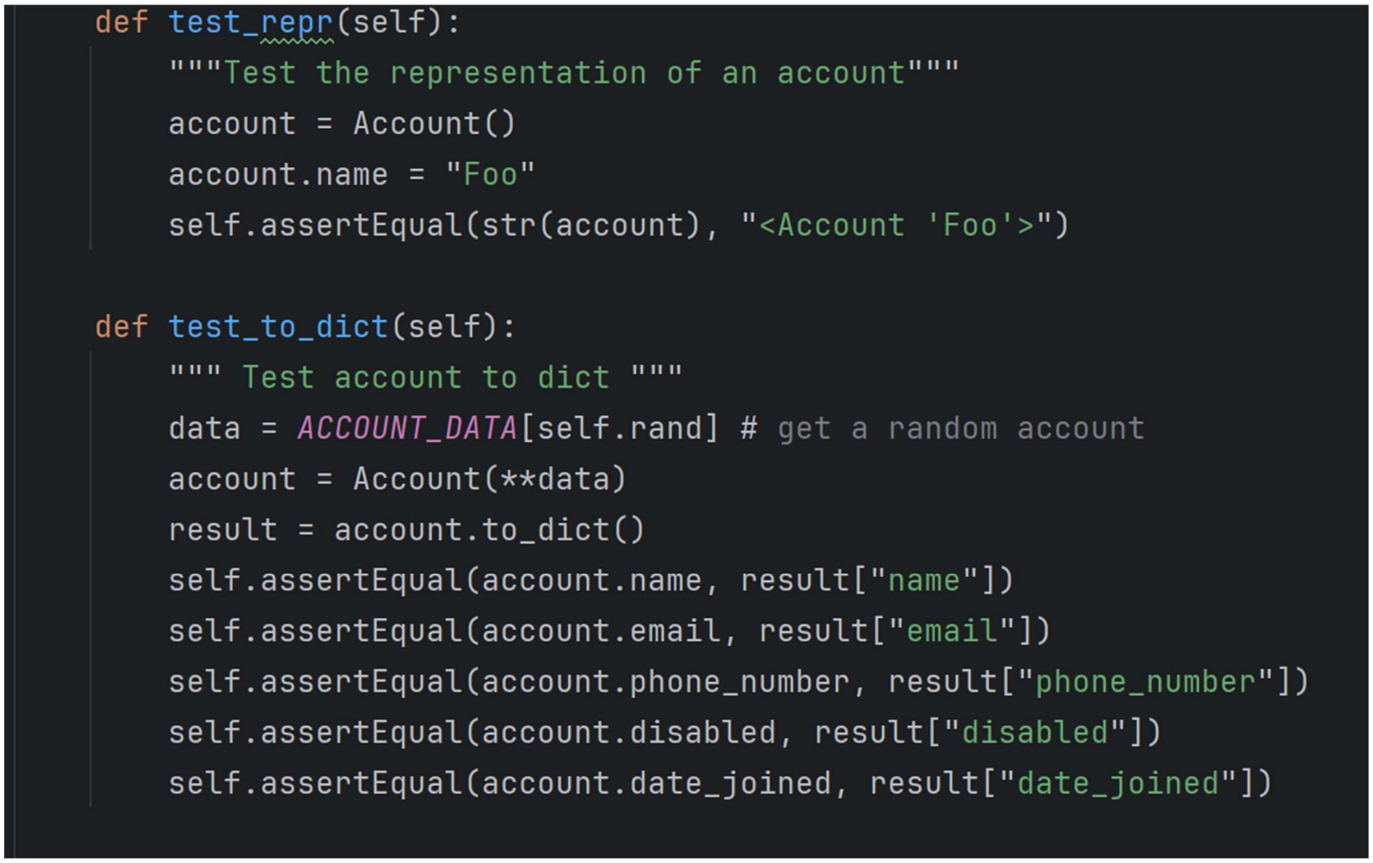
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Mi
nl.tudelft.jpacman.level		67%		57%	74	155	104	344	
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	
nl.tudelft.jpacman.ui		77%		47%	54	86	21	144	
default		0%		0%	12	12	21	21	
nl.tudelft.jpacman.sprite		86%		59%	30	70	11	113	
nl.tudelft.jpacman.board		87%		60%	42	93	2	110	
nl.tudelft.jpacman		69%		25%	12	30	18	52	
nl.tudelft.jpacman.points		60%		75%	1	11	5	21	
nl.tudelft.jpacman.game		87%		60%	10	24	4	45	
nl.tudelft.jpacman.npc		100%		n/a	0	4	0	8	
Total	1,208 of 4,694	74%	291 of 637	54%	291	590	229	1,039	

Created with JaCoCo 0.8.3.201901230119

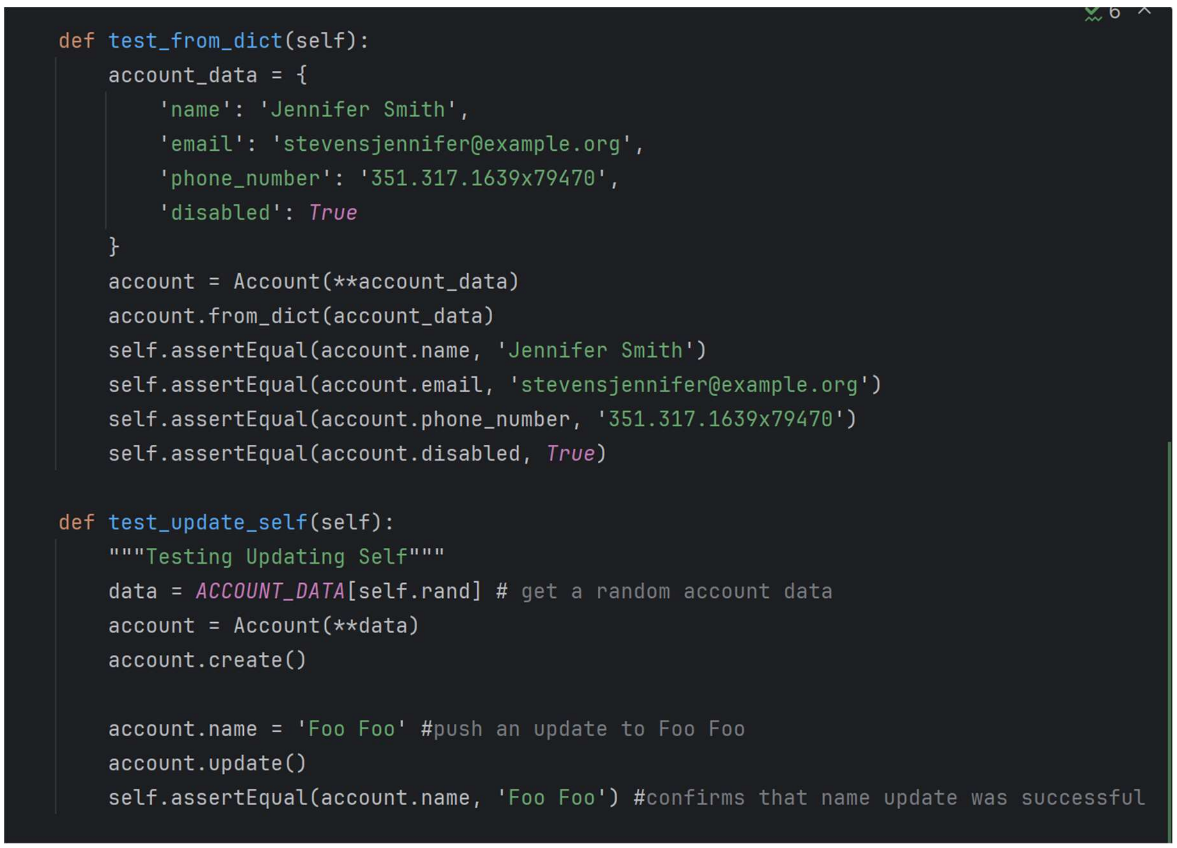
Questions :

The coverage results seem much different from JaCoCo to IntelliJ. JaCoCo is showing a higher coverage percentage than IntelliJ. It seems that different metrics to decide coverage may be used. I did find the JaCoCo visualization to be useful because it explicitly highlighted what parts were covered or not. It made it useful to see where exactly I needed to apply testing. I preferred JaCoCo because of the simplicity and ability to pinpoint where exactly I needed work to be done.

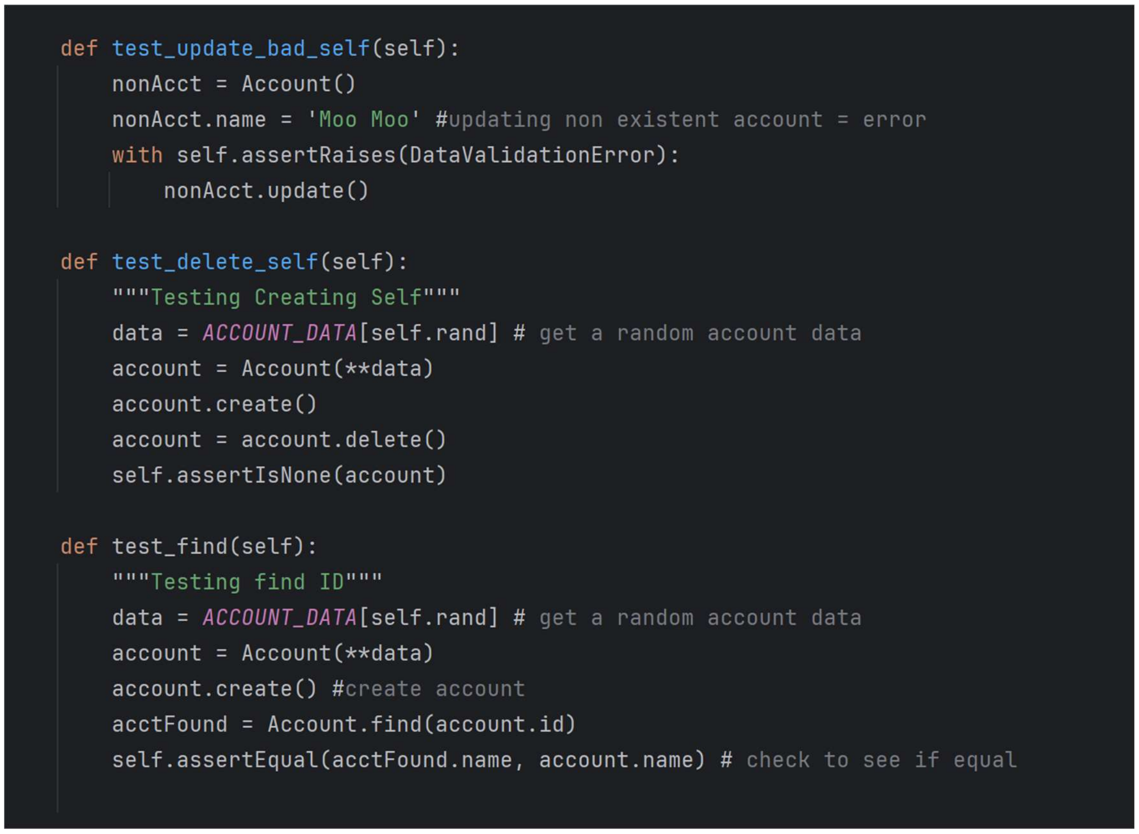
Below are code fragments of my ways of running tests to make sure things are running properly



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

def test_to_dict(self):
    """ Test account to dict """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    result = account.to_dict()
    self.assertEqual(account.name, result["name"])
    self.assertEqual(account.email, result["email"])
    self.assertEqual(account.phone_number, result["phone_number"])
    self.assertEqual(account.disabled, result["disabled"])
    self.assertEqual(account.date_joined, result["date_joined"])
```

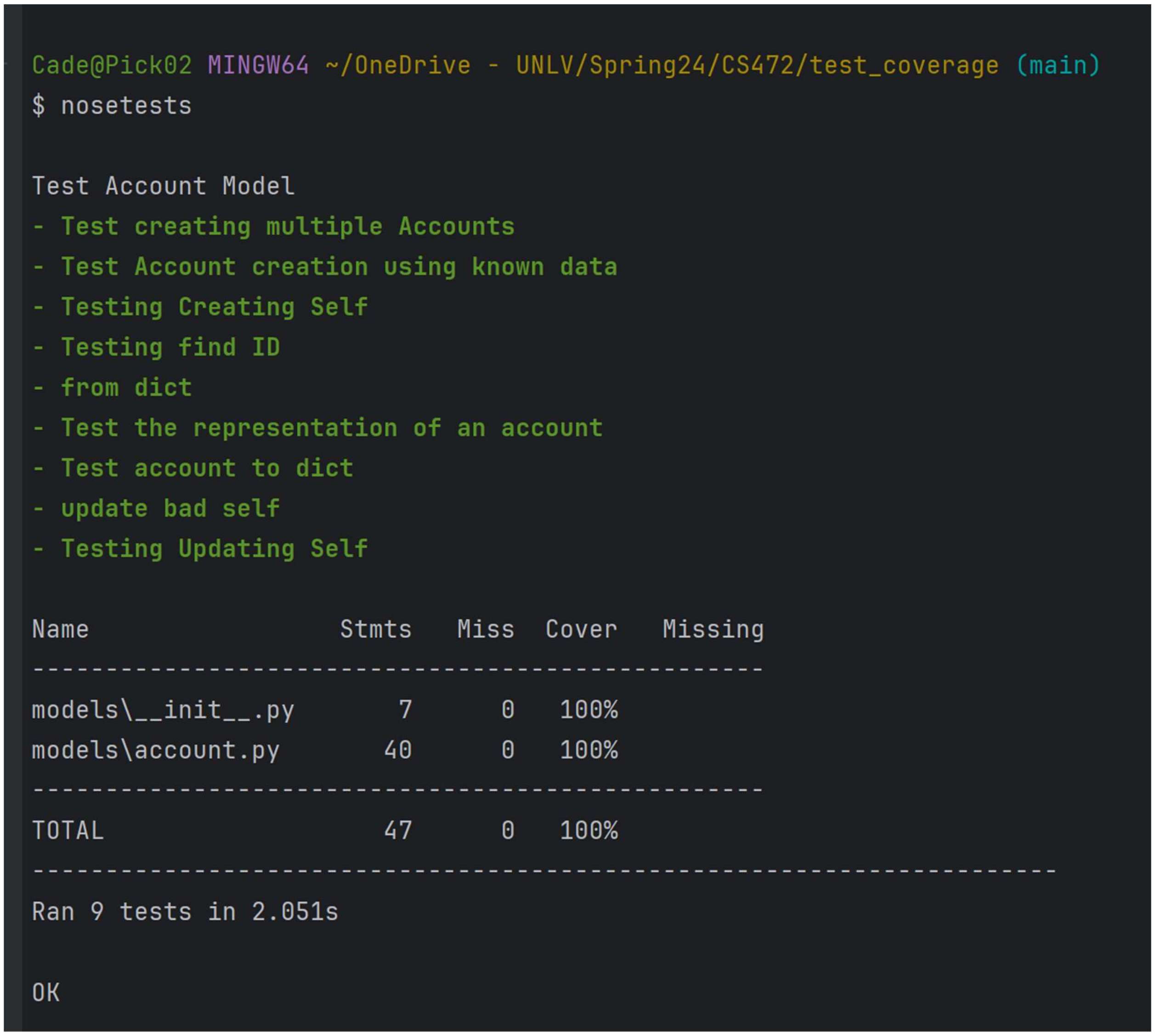


```python
def test_from_dict(self):
    account_data = {
        'name': 'Jennifer Smith',
        'email': 'stevensjennifer@example.org',
        'phone_number': '351.317.1639x79470',
        'disabled': True
    }
    account = Account(**account_data)
    account.from_dict(account_data)
    self.assertEqual(account.name, 'Jennifer Smith')
    self.assertEqual(account.email, 'stevensjennifer@example.org')
    self.assertEqual(account.phone_number, '351.317.1639x79470')
    self.assertEqual(account.disabled, True)

def test_update_self(self):
    """Testing Updating Self"""
    data = ACCOUNT_DATA[self.rand] # get a random account data
    account = Account(**data)
    account.create()

    account.name = 'Foo Foo' #push an update to Foo Foo
    account.update()
    self.assertEqual(account.name, 'Foo Foo') #confirms that name update was successful
```



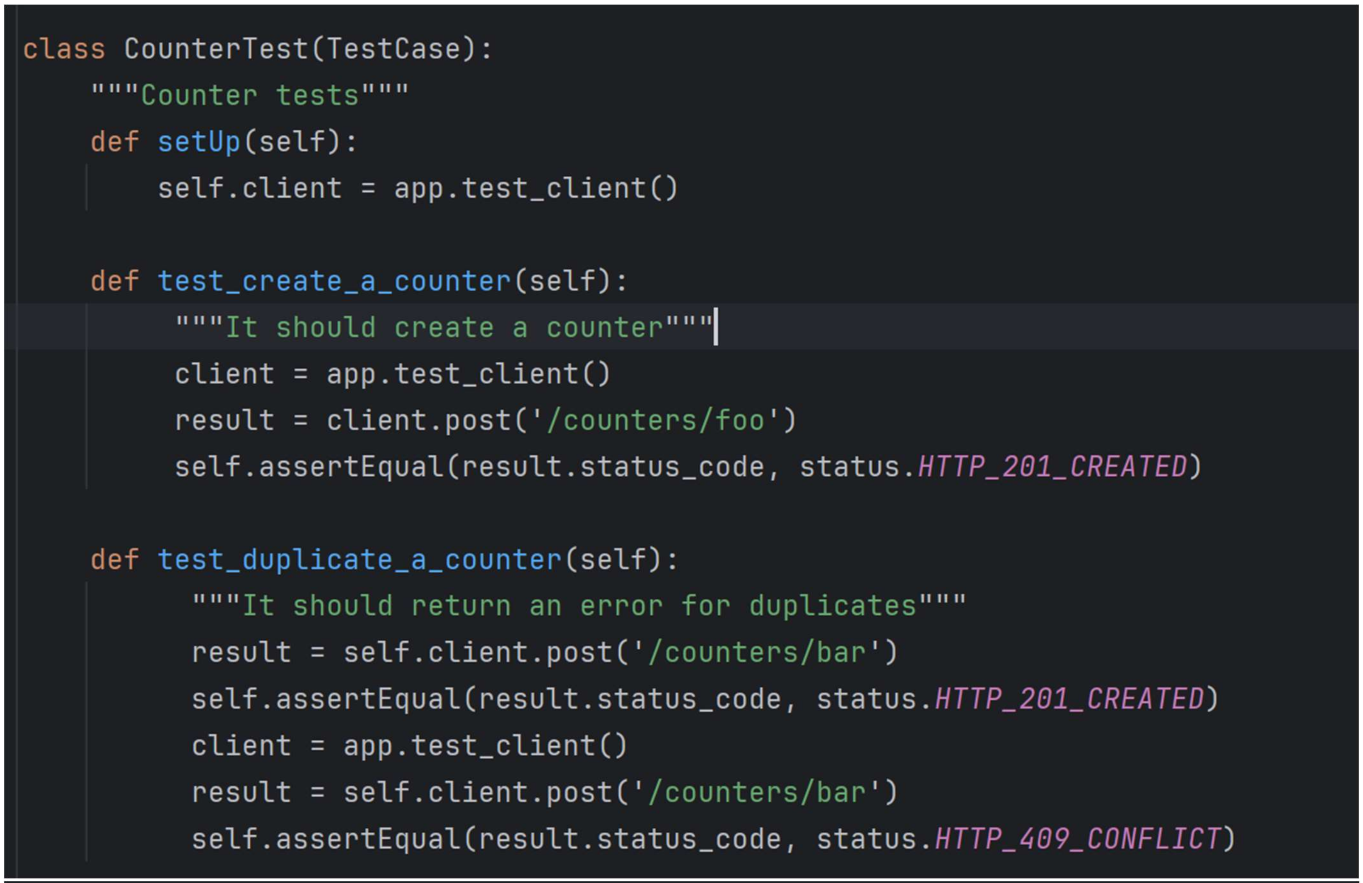
```python
def test_update_bad_self(self):
    nonAcct = Account()
    nonAcct.name = 'Moo Moo' #updating non existent account = error
    with self.assertRaises(DataValidationError):
        nonAcct.update()

def test_delete_self(self):
    """Testing Creating Self"""
    data = ACCOUNT_DATA[self.rand] # get a random account data
    account = Account(**data)
    account.create()
    account = account.delete()
    self.assertIsNone(account)

def test_find(self):
    """Testing find ID"""
    data = ACCOUNT_DATA[self.rand] # get a random account data
    account = Account(**data)
    account.create() #create account
    acctFound = Account.find(account.id)
    self.assertEqual(acctFound.name, account.name) # check to see if equal
```

Below are my results from running tests. It shows that my tests were sufficient to cover 100% of the code. That means we have completed checking for proper functioning of the code.

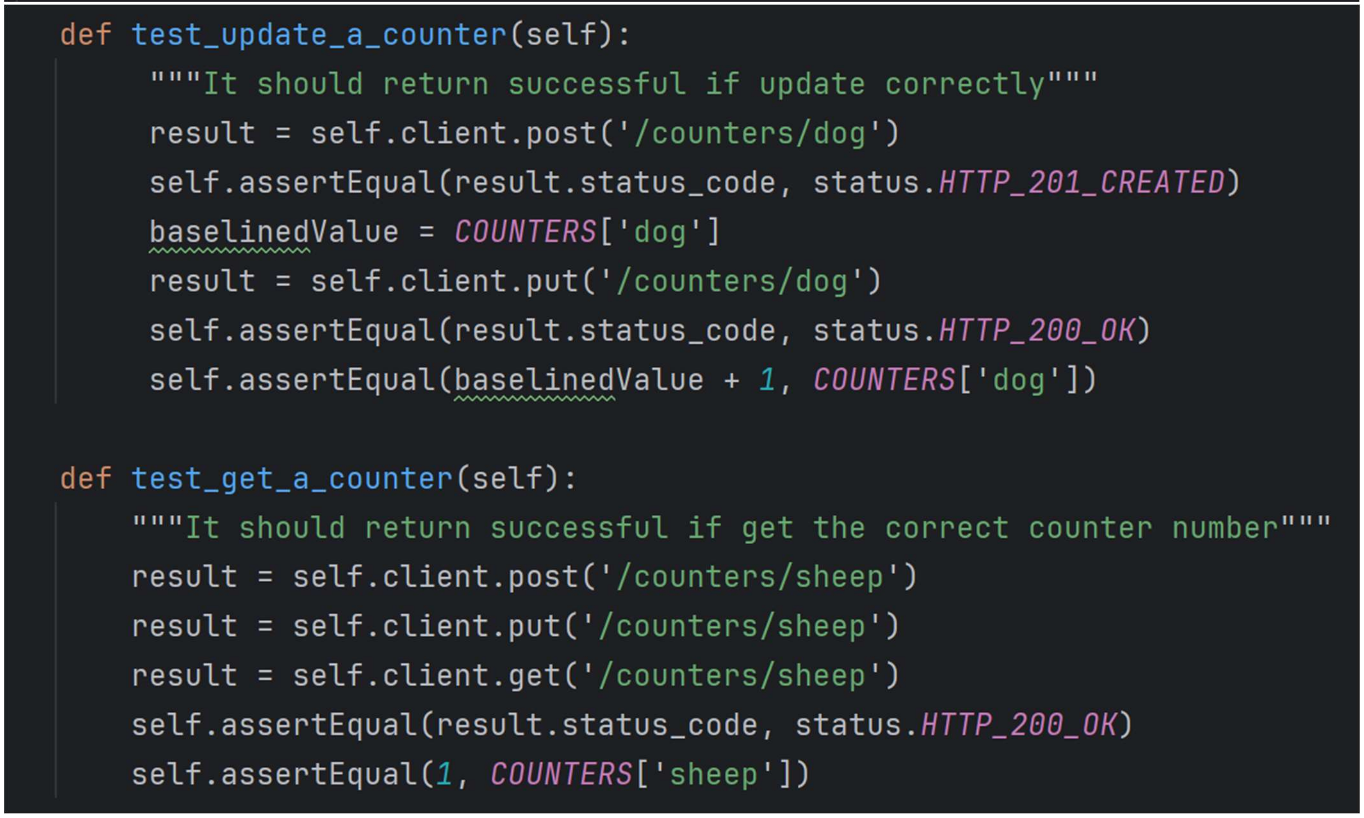


```
Cade@Pick02 MINGW64 ~/OneDrive - UNLV/Spring24/CS472/test_coverage (main)
$ nosetests

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Testing Creating Self
- Testing find ID
- from dict
- Test the representation of an account
- Test account to dict
- update bad self
- Testing Updating Self

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 2.051s

OK
```

Below are my snippets of code showing the tests I came up with to test the coverage. It works by comparing the exit codes to make sure it exits properly. It can also check the values following the action we want to check

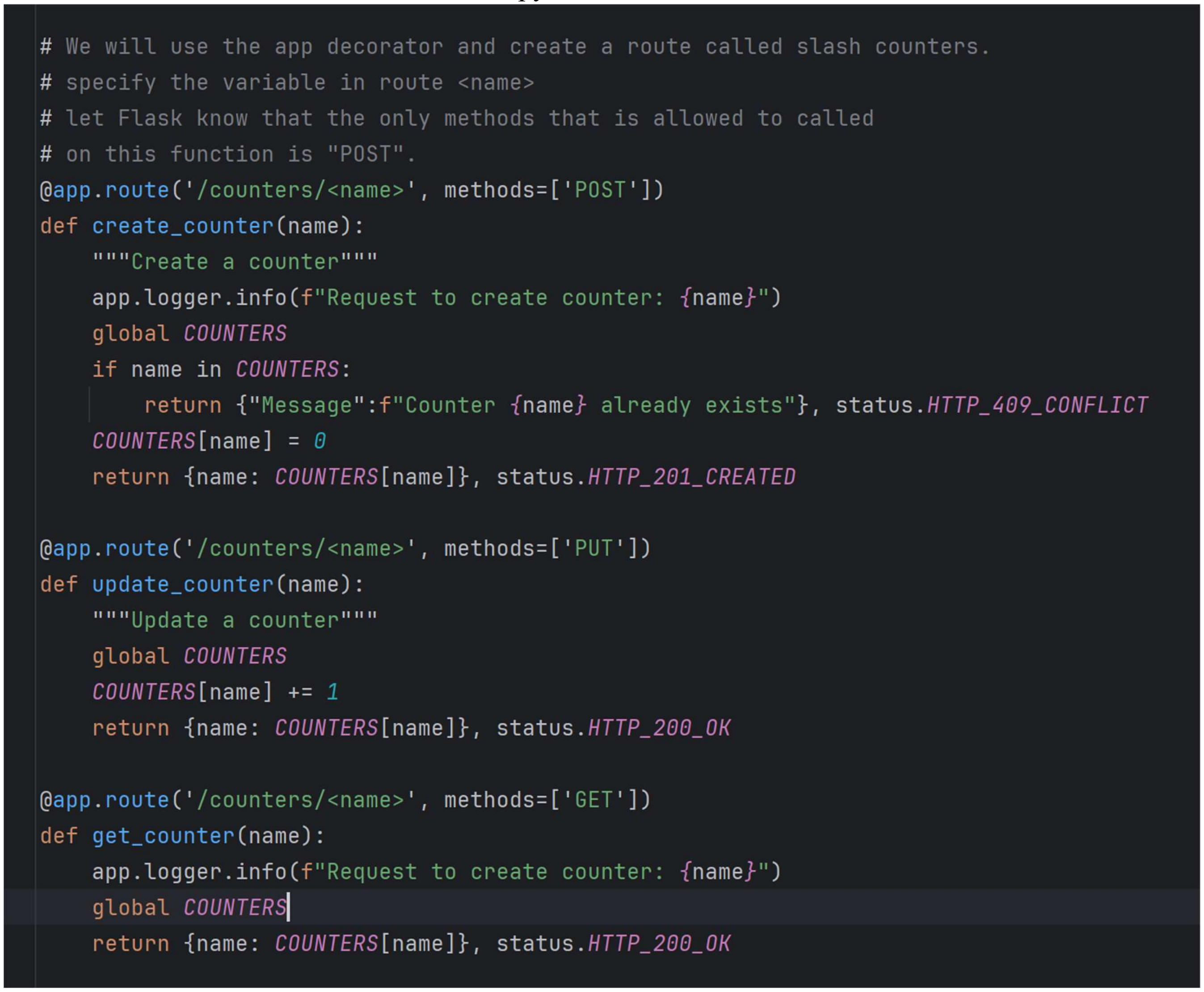


```python
class CounterTest(TestCase):
    """Counter tests"""
    def setUp(self):
        self.client = app.test_client()

    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)

    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)
          client = app.test_client()
          result = self.client.post('/counters/bar')
          self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
```

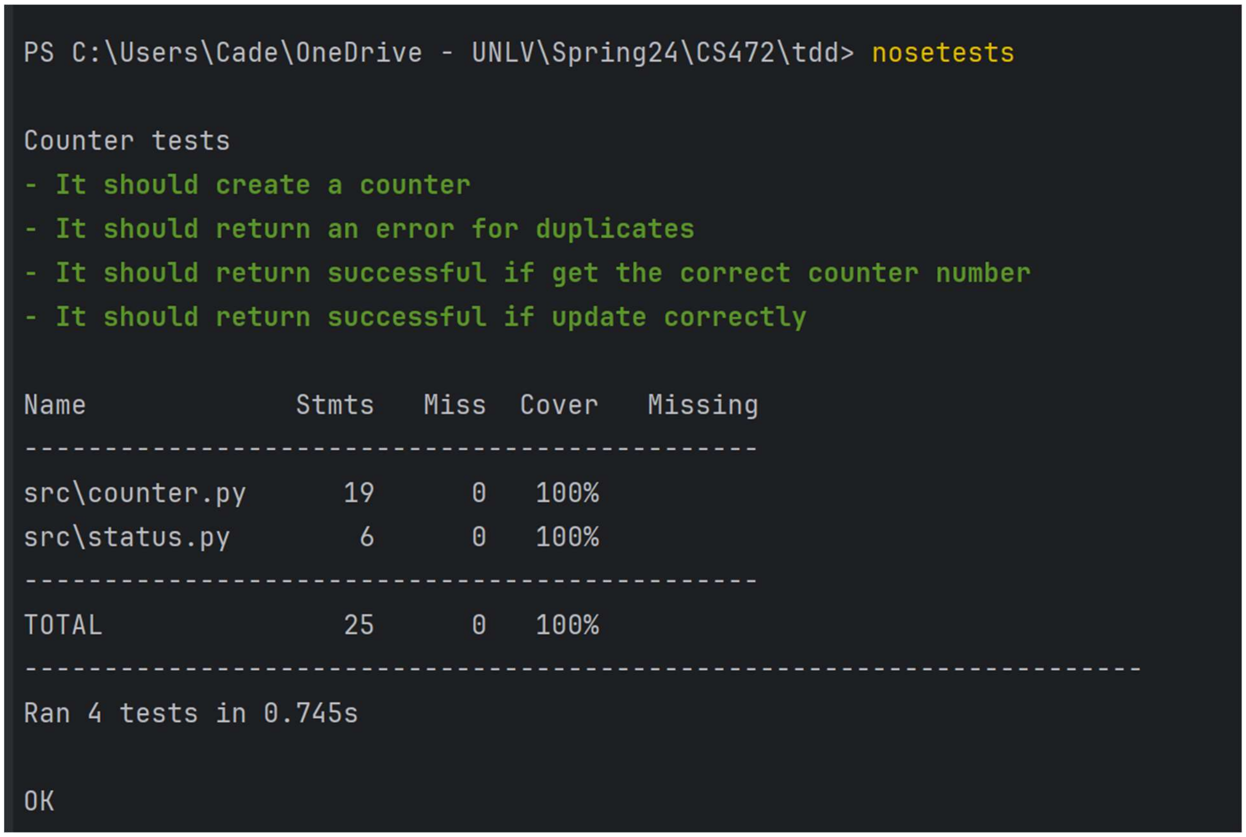


```python
    def test_update_a_counter(self):
         """It should return successful if update correctly"""
         result = self.client.post('/counters/dog')
         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
         baselinedValue = COUNTERS['dog']
         result = self.client.put('/counters/dog')
         self.assertEqual(result.status_code, status.HTTP_200_OK)
         self.assertEqual(baselinedValue + 1, COUNTERS['dog'])

    def test_get_a_counter(self):
        """It should return successful if get the correct counter number"""
        result = self.client.post('/counters/sheep')
        result = self.client.put('/counters/sheep')
        result = self.client.get('/counters/sheep')
        self.assertEqual(result.status_code, status.HTTP_200_OK)
        self.assertEqual(1, COUNTERS['sheep'])
```

Below is the code that was coded on counter.py



```python
# 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
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK

@app.route('/counters/<name>', methods=['GET'])
def get_counter(name):
    app.logger.info(f"Request to create counter: {name}")
    global COUNTERS
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```



```
PS C:\Users\Cade\OneDrive - UNLV\Spring24\CS472\tdd> nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates
- It should return successful if get the correct counter number
- It should return successful if update correctly

Name                Stmts   Miss  Cover   Missing
----------------------------------------------
src\counter.py         19      0   100%
src\status.py           6      0   100%
----------------------------------------------
TOTAL                  25      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.745s

OK
```

Here is the results after doing the coverage tests. 100% shows that we have successfully checked if our code would function.

This lab was helpful in understanding how to check to see if error checking is done correctly Complete error checking will test all lines of code to see if they function as intended. It was tricky to come up with ways to test the code fragments at times but this is an important step. We want to change the red sections to green to show that we covered the code sufficiently.