Task 2.1 & 3:

Question Answers:

- The results seemed somewhat different between JaCoCo and IntelliJ because of the way it is presented, but otherwise similar.
- I myself did not find the source code visualization very helpful, because there is just too much going on in the 'xml' file.
- I prefer the IntelliJ coverage window because it is more straightforward and requires less analyzing.

```java
public class PlayerTest {
    @Test
    void testIsAlive() {
        PacManSprites sprites = new PacManSprites();
        PlayerFactory playerFactory = new PlayerFactory(sprites);
        Player player = playerFactory.createPacMan();

        assertThat(player.isAlive()).isTrue();
    }

    @Test
    void testGetKiller() {
        PacManSprites sprites = new PacManSprites();
        PlayerFactory playerFactory = new PlayerFactory(sprites);
        Player player = playerFactory.createPacMan();
        Player killer = playerFactory.createPacMan();
        player.setKiller(killer);
        assertThat(player.getKiller()).isEqualTo(killer);
    }

    @Test
    void testSetKiller() {
        PacManSprites sprites = new PacManSprites();
        PlayerFactory playerFactory = new PlayerFactory(sprites);
        Player player = playerFactory.createPacMan();
        Player killer = playerFactory.createPacMan();
        player.setKiller(killer);
        assertThat(player.getKiller()).isEqualTo(killer);
    }
}
```

```java
public class SpriteTest {
    @Test
    void testGetSprite() {
        PacManSprites sprites = new PacManSprites();
        PlayerFactory playerFactory = new PlayerFactory(sprites);
        Player player = playerFactory.createPacMan();

        assertThat(player.getSprite()).isNotNull();
    }
}
```

Task 4:

- Reached 100% test coverage for the python tests

```python
def test_from_dict(self):
    """Test setting attributes from a dictionary"""
    data = {
        "name": "John Doe",
        "email": "john.doe@example.com",
        "phone_number": "1234567890",
        "disabled": False,
        "date_joined": "2022-01-01"
    }
    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"])
    self.assertEqual(account.date_joined, data["date_joined"])

def test_update_account(self):
    """ Test updating an account """
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()
    account.name = "Foo"
    account.update()
    self.assertEqual(len(Account.all()), 1)
    self.assertEqual(account.name, "Foo")

def test_update_account_with_empty_id(self):
    """ Test updating an account with empty ID """
    account = Account()
    with self.assertRaises(DataValidationError):
        account.update()
```

```python
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_by_id(self):
    """ Test finding an account by ID """
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()
    found_account = Account.find(account.id)
    self.assertEqual(found_account, account)

def test_find_nonexistent_account(self):
    """ Test finding a nonexistent account """
    nonexistent_account = Account.find(9999)
    self.assertIsNone(nonexistent_account)
```

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deleting an account
- Test finding an account by ID
- Test finding a nonexistent account
- Test setting attributes from a dictionary
- Test the representation of an account
- Test account to dict
- Test updating an account
- Test updating an account with empty ID

Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models/__init__.py       7      0   100%
models/account.py       40      0   100%
--------------------------------------------------
TOTAL                   47      0   100%
--------------------------------------------------
```

Task 5:

- Writing test cases and code for updating a counter

```python
def test_update_a_counter(self):
    """It should update a counter"""
    # Delete any existing counter named 'foo'
    self.client.delete('/counters/foo')

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

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

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

    # Check that the counter value is one more than the baseline
    updated_result = self.client.get('/counters/foo')
    updated_value = updated_result.json['foo']
    self.assertEqual(updated_value, baseline_value + 1)
```

```python
@app.route('/counters/<string:name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    global COUNTERS
    if name in COUNTERS:
        # Increment the counter by 1
        COUNTERS[name] += 1
        return {name: COUNTERS[name]}, status.HTTP_200_OK
    else:
        return '', status.HTTP_404_NOT_FOUND
```

- Writing test cases and code for reading a counter

```python
def test_read_a_counter(self):
    """It should read a counter"""
    # Delete any existing counter named 'foo'
    self.client.delete('/counters/foo')

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

    # Read the counter
    result = self.client.get('/counters/foo')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
```

```python
@app.route('/counters/<string:name>', methods=['GET'])
def read_counter(name):
    if name in COUNTERS:
        # Return the counter
        return {name: COUNTERS[name]}, status.HTTP_200_OK
    else:
        return '', status.HTTP_404_NOT_FOUND
```

- There was an issue that required me to define a function to delete an existing counter because a counter would already exist when trying to test more than once.

```python
@app.route('/counters/<string:name>', methods=['DELETE'])
def delete_counter(name):
    if name in COUNTERS:
        del COUNTERS[name]
        return '', status.HTTP_204_NO_CONTENT
    else:
        return '', status.HTTP_404_NOT_FOUND
```

- In order to get to 100% test coverage, I had to make tests that dealt solely with the scenario in which a counter did not exist.

```python
def test_update_nonexistent_counter(self):
    """It should return an error for updating a nonexistent counter"""
    # Delete any existing counter named 'nonexistent'
    self.client.delete('/counters/nonexistent')

    # Update the nonexistent counter
    result = self.client.put('/counters/nonexistent')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
```

- In the end, taking these steps got my test coverage to 100%

```
@B-Timok ➜/workspaces/CS472_Group1/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
- It should return an error for updating a nonexistent counter

Name             Stmts   Miss  Cover   Missing
----------------------------------------------
src/counter.py      28      1    96%   39
src/status.py        6      0   100%
----------------------------------------------
TOTAL               34      1    97%
----------------------------------------------------------------------
Ran 5 tests in 0.187s

OK
```

Link to my fork → https://github.com/B-Timok/CS472_Group1