Peyton Noel (5005559697)
CS 472 - 1001

Unit Testing

Link to repository: https://github.com/uid106/CS472_LABS.git

Task 1

Package nl.tudelft.jpacman.board

all > nl.tudelft.jpacman.board

4	0	0	0.098s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
DirectionTest	1	0	0	0.096s	100%
OccupantTest	3	0	0	0.002s	100%

- Is the coverage good enough? No, there should be significantly more coverage given the number of packages and classes within them to only have four tests.

Task 2

Test Summary

5	0	0	0.100s
tests	failures	ignored	duration

100% successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
nl.tudelft.jpacman.board	4	0	0	0.099s	100%
nl.tudelft.jpacman.level	1	0	0	0.001s	100%

- Additional coverage for isAlive() on the level package.

Peyton Noel (5005559697)
CS 472 - 1001

Task 2.1

```java
@BeforeEach
void setUp() {
    /*
      Setting up mocking for simulating classes
     */
    playerInstance = mock(PlayerFactory.class);
    levelInstance = mock(Level.class);
    pointCalculator = mock(PointCalculator.class);

    /*
      Game Factory Initialized with player factory mock
     */
    factoryInstance = new GameFactory(playerInstance);
}

@Test
void testCreateSinglePlayerGame() {

    Player testPacMan = mock(Player.class);
    /*
      When creating pac man is called then returning mock class
     */
    when(playerInstance.createPacMan()).thenReturn(testPacMan);
    /*
      Creating Test Game
     */
    Game testGame = factoryInstance.createSinglePlayerGame(levelInstance, pointCalculator);
    /*
      Performing Checks
     */
    assertNotNull(testGame);
    assertTrue(testGame instanceof SinglePlayerGame);
    verify(playerInstance).createPacMan();
}
```

```java
public class pointCalculatorTest {

    /**
     * Creating Instances
     */
    private DefaultPointCalculator calculatorInstance;  3 usages
    private Pellet pelletInstance;  1 usage
    private Player playerInstance;  5 usages

    @BeforeEach
    void setUp(){
        /*
          Creating mock class
         */
        calculatorInstance = new DefaultPointCalculator();
        playerInstance = mock(Player.class);
        pelletInstance = mock(Pellet.class);
    }

    @Test
    void testCollidedWithAGhost(){
        Ghost ghostInstance = mock(Ghost.class);
        calculatorInstance.collidedWithAGhost(playerInstance,ghostInstance);
        /*
          Makes sure nothing was changed in player as no points should be added
         */
        verifyZeroInteractions(playerInstance);
    }

    @Test
    void testPacmanMoved(){
        Direction directionInstance = Direction.NORTH;
        calculatorInstance.pacmanMoved(playerInstance,directionInstance);
        /*
          Makes sure nothing was changed in player as no points should be added
         */
        verifyZeroInteractions(playerInstance);
    }
}
```

Test Summary

8	0	0	0.856s
tests	failures	ignored	duration

100% successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
nl.tudelft.jpacman.board	4	0	0	0.103s	100%
nl.tudelft.jpacman.game	1	0	0	0.697s	100%
nl.tudelft.jpacman.level	1	0	0	0.001s	100%
nl.tudelft.jpacman.points	2	0	0	0.055s	100%

- Created testing methods for CreateSinglePlayerGame(), CollideWithAGhost(), and PacmanMoved(). The first creates a game using instance of level and point calculator and checking for null status, the next checks an instance of calculator against collidedwithghost function to check that no points are added to ensure a ghost wasn't collided with as it should not have occurred, and last sets the direction to north and calls packmanmoved with the instance of direction, and ensures that no points were added as there are no points given for moving.

Peyton Noel (5005559697)
CS 472 - 1001

Task 3

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		86%		59%	30	70	11	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	1,210 of 4,694	74%	292 of 637	54%	292	590	228	1,039	51	268	6	47

- The results are pulling similar information to IntelliJ but in much more extreme detail, it shows all the packages and their functions inside of them as well as the missed and covered tests.
- The source code visualization for uncovered branches on JaCoCo is very helpful, it shows the code highlighted for all sections that have not been covered by tests in red so they're easy to create tests for.
- Personally, I prefer JaCoCo's coverage as the visualization of which code wasn't covered and the bar graph of covered and not covered is useful. Seeing the code live in a browser as opposed to line numbers that were missed is significantly easier.

Peyton Noel (5005559697)
CS 472 - 1001

Task 4

```python
def test_from_dict():
    """Test from dict"""
    data = {
        "name": "fake name",
        "email": "fake email",
        "phone_number": "9999999999",
        "disabled": False,
        "date_joined": datetime( year: 2024, month: 1, day: 1)
    }

    account = Account()
    account.from_dict(data)

    assert account.name == data["name"]
    assert account.email == data["email"]
    assert account.phone_number == data["phone_number"]
    assert account.disabled == data["disabled"]
    assert account.date_joined == data["date_joined"]
```

```python
def test_update():
    """Test updates"""
    data = {
        "name": "fake name",
        "email": "fake email",
        "phone_number": "9999999999",
        "disabled": False,
        "date_joined": datetime( year: 2024, month: 1, day: 1),
        "id": 1
    }

    account = Account(**data)
    account.create()
    account.name = "New Fake Name"
    account.update()

    account_updated = db.session.get(Account, account.id)

    assert account_updated.name == "New Fake Name"
    assert account_updated.email == data["email"]
    assert account_updated.phone_number == data["phone_number"]
    assert account_updated.disabled == data["disabled"]
    assert account_updated.date_joined == data["date_joined"]

    validationErrorTest = Account(name="FAKE NAME",email="FAKE EMAIL")
    with pytest.raises(DataValidationError) as testing:
        validationErrorTest.update()

    assert str(testing.value) == "Update called with empty ID field"
```

```python
def test_delete():
    """Test delete"""
    data = {
        "name": "fake name",
        "email": "fake email",
        "phone_number": "9999999999",
        "disabled": False,
        "date_joined": datetime( year: 2024, month: 1, day: 1),
    }

    account = Account(**data)
    account.create()

    assert db.session.get(Account, account.id) is not None
    account.delete()
    assert db.session.get(Account, account.id) is None
```

```python
def test_find():
    """Test find"""
    data = {
        "name": "fake name",
        "email": "fake email",
        "phone_number": "9999999999",
        "disabled": False,
        "date_joined": datetime( year: 2024, month: 1, day: 1),
    }

    account = Account(**data)
    account.create()

    findingAccount = Account.find(account.id)

    assert findingAccount.id == account.id
    assert findingAccount.name == account.name
    assert findingAccount.email == account.email
    assert findingAccount.phone_number == account.phone_number
    assert findingAccount.disabled == account.disabled
    assert findingAccount.date_joined == account.date_joined

    assert Account.find(55) is None
```

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

- Four functions have been created for testing to get the coverage to 100%, including testing from_dict(), update(), find() and delete(). The from_dict test creates test data and creates an account, then inserts the test data into the account using from_dict and checks that the account data matches the data fields. The delete creates an account with data and ensures it exists, then deletes the account and checks that it no longer exists. The find creates an account and finds it using the account's ID and then ensures that the stored found account matches the data that was initally inputted and also attempts to find a non-existent ID to test failure. The update creates an account, and then updates the name data field, and copies it from the database to check that all of the information continues to match it's previous values as well as the updated name value, then creates an account without an ID and attempts an update to test the error message without the function.

Peyton Noel (5005559697)
CS 472 - 1001

Task 5

```python
def test_update_counter(self, client):
    """Testing Update"""
    client.post('/counters/test')
    result = client.put('/counters/test')
    assert result.status_code == status.HTTP_200_OK
    assert result.json == {'test': 1}
    result = client.put('/counters/test')
    assert result.status_code == status.HTTP_200_OK
    assert result.json == {'test': 2}

    result = client.put('/counters/nonexistent')
    assert result.status_code == status.HTTP_404_NOT_FOUND
```

First the test is created for update, post is called to create a counter, and then put is called and stored in result, the status is then checked to be okay and the number in the counter is checked, put is then called again and the counter is checked again to make sure it increased. A counter that hasn't been created is tried with put to ensure that it fails checks.

```python
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update 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_404_NOT_FOUND
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

Function can update counter with name called using put and can check if name is not in counter for 404 error, if the name does exist and update then status will be 200.

Peyton Noel (5005559697)
CS 472 - 1001

```python
def test_read_counter(self, client):
    """Testing Read Counter"""
    client.post('/counters/test2')
    result = client.get('counters/test2')
    assert result.status_code == status.HTTP_200_OK
    assert result.json == {'test2':0}

    result = client.get('counters/fake')
    assert result.status_code == status.HTTP_404_NOT_FOUND
```

The test for reading the counter first creates a counter, and then calls using get, and checks that the status is 200 and that it contains the initialized value of 0. Then a call with get on a counter that hasn't been created is called to ensure that the 404 error is functioning.

```python
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Counter Read"""
    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_404_NOT_FOUND
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

The function can read the counter value, it checks if the name does not exist in the counter, and if it does not return error 202. Access to global counter allows reading of the counter value.