Michael Razon
CS 472, Section 1001
Dr. John Businge
February 5th, 2024

Unit Testing

Task 1

The coverage is not good enough. We discussed in lecture that we should aim for 90% coverage to be excellent. Just 3% coverage is not nearly enough.

Task 2.1

For this task, I created test cases for the methods for the `DefaultPointsCalculator.java` file in the `points` folder. These methods required a lot of setup as the methods called `Player`, `Ghost`, and `Pellet` objects in addition to `Direction` variables. Similar to the process for the `PlayerTest` exercise, I needed to create a `GhostFactory` and `LevelFactory` using the same sprite store as `PlayerFactory`.

```java
private PacManSprites TestSprite = new PacManSprites();

private PlayerFactory TestFactory = new PlayerFactory(TestSprite);
private GhostFactory TestGhostFactory = new GhostFactory(TestSprite);
private DefaultPointCalculator TestCalculator = new DefaultPointCalculator();
private LevelFactory TestLevelFactory = new LevelFactory(TestSprite, TestGhostFactory, TestCalculator);

private Player CollisionPlayer = TestFactory.createPacMan();
private Ghost TestBlinky = TestGhostFactory.createBlinky();
private Pellet TestPellet = TestLevelFactory.createPellet();
private Direction testDirection = Direction.EAST;
```

After the set-up was complete, I investigated the methods in the file. Each method modified `Player`'s `score` based on the type of interaction that occurred. `collidedWithAGhost()` and `pacmanMoved()` do not modify `score` while `consumedAPellet()` adds the `Pellet` object's value to `score`.

To verify the methods worked as intended, I compared `Player` initial `score` with the consequent `score` following the call to the method. I used `assertThat().isEqualTo()` from the previous exercise to compare the pre- and post-call `scores`. I made sure `consumedAPellet()` caused the score to increment by the Pellet value, and the other two methods did not modify it. The code worked as expected with all methods passing the tests.

```java
@Test
void testCollidedWithAGhost(){
    int initialScore = CollisionPlayer.getScore();
    TestCalculator.collidedWithAGhost(CollisionPlayer, TestBlinky);
    assertThat(CollisionPlayer.getScore()).isEqualTo(initialScore);
}
@Test
void testConsumedAPellet(){
    int initialScore = CollisionPlayer.getScore();
    int pelletScore = TestPellet.getValue();

    TestCalculator.consumedAPellet(CollisionPlayer, TestPellet);
    assertThat(CollisionPlayer.getScore()).isEqualTo( expected: initialScore + pelletScore);
}
@Test
void testPacmanMoved(){
    int initialScore = CollisionPlayer.getScore();
    TestCalculator.pacmanMoved(CollisionPlayer, testDirection);
    assertThat(CollisionPlayer.getScore()).isEqualTo(initialScore);
}
```

Task 3

The coverage results from JaCoCo are similar to the ones I got from IntelliJ, but the numbers are different. For example, the number of methods is different probably because each one has different metrics for deciding what is a method. JaCoCo considers constructors to be methods while IntelliJ doesn't. I find IntelliJ's interpretation to be more useful. Aside from this, JaCoCo appears to have a lot more depth and interactivity. JaCoCo has page navigation which allows me to look within each file to get a breakdown of which methods were hit and which ones were missed. It also allows me to click each function to view the code with highlighted lines.

Element ∧	Class, %	Method, %	Line, %
∨ ▣ nl.tudelft.jpacman	27% (15/55)	15% (47/312)	11% (133/1161)
> ▣ 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	30% (4/13)	14% (11/78)	7% (28/353)
> ▣ npc	40% (4/10)	12% (6/47)	6% (17/243)
∨ ▣ points	50% (1/2)	42% (3/7)	20% (4/20)
◎ DefaultPointCalculator	100% (1/1)	100% (3/3)	100% (4/4)
① PointCalculator	100% (0/0)	100% (0/0)	100% (0/0)
◎ PointCalculatorLoader	0% (0/1)	0% (0/4)	0% (0/16)
> ▣ sprite	66% (4/6)	48% (22/45)	54% (70/128)
> ▣ 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)

I find the source code visualization from JaCoCo to be helpful. In addition to showing which lines were hit and missed, JaCoCo also highlights any branching code with paths that were not hit. When I scroll over, it also shows how many branches were hit.

```
if ("DefaultPointCalculator".equals(strategyToLoad)) {
    return DefaultPointCalculator.class;
}
```
1 of 2 branches missed.
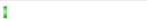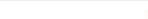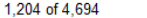
I preferred JaCoCo's report because of its depth. I like that I can see which methods were hit instead of IntelliJ's general overview. I think the color coding is nicer too since it highlights the entire line whereas IntelliJ only highlights line numbers. This is useful for me since I often lose track of which line I am looking at especially in denser files. Moreover, I can have it in a separate window. For IntelliJ, I have to close and open the report and expand the window to see all the detailed percentages.

DefaultPointCalculator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
consumedAPellet(Player, Pellet)		100%		n/a	0	1	0	2	0	1
DefaultPointCalculator()		100%		n/a	0	1	0	1	0	1
collidedWithAGhost(Player, Ghost)		100%		n/a	0	1	0	1	0	1
pacmanMoved(Player, Direction)		100%		n/a	0	1	0	1	0	1
Total	0 of 10	100%	0 of 0	n/a	0	4	0	5	0	4

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

Task 4

This was the result after following the instructions to set up this task. I worked on increasing coverage by interpreting from_dict().

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\test_coverage> py -m nose

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test the representation of an account
- Test account to dict

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 1.006s

OK
```

from_dict() creates an account based on an inputted dictionary data type variable. Thus, the test had to ensure the variables set by the method actually coincided with the passed dict. I tested name, email, phone_number, and disabled since id and date_joined only populate when an account is added to the database, which from_dict() is not responsible for.

```python
def test_from_dict(self):
    """ Test dict to account """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account()                 # initialize blank account
    account.from_dict(data)             # populate account details

    # assert if account details match the dict values
    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"])
```

This test was successful at covering lines 34-35. Next, I worked on coverage for lines 45-48 which was the update() method.

```
Test Account Model
 - Test creating multiple Accounts
 - Test Account creation using known data
 - Test dict to account
 - Test the representation of an account
 - 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 0.845s

OK
```

update() saves any changes to accounts by projecting the edits into the database. It also makes sure that it doesn't try to update an account that doesn't actually exist. This meant I had to consider both scenarios when designing the test case. After creating an account and putting it into the database, I saved the name before then changing it and calling update(). I verified if it successfully worked by checking if the account is represented by the new name rather than the saved old name. I also tested if it would not update a nonexistent account by creating an account then calling update on it. I did this with assertRaises() which required me to pass the expected error, the function name, and any parameters.

```python
def test_update(self):
    """ Test account update """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()                   # push into database

    oldName = account.name             # save old name
    account.name = "new name"          # change name
    account.update()                   # push name change to database

    self.assertNotEqual("<Account '%s'>" % oldName, str(account))       # ensure new name is not old name

    unregisteredAccount = Account(**data)
    self.assertRaises(DataValidationError, unregisteredAccount.update)  # check if error raise happens
```

My test was successful at covering the lines, so I worked on lines 52-54.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\test coverage> py -m nose

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test dict to account
- Test the representation of an account
- Test account to dict
- Test 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 6 tests in 0.869s

OK
```

Lines 52-54 are the delete() method. It removes an account from the database. To test this, I created an account, added it to the database, then deleted it with the function call. Afterwards, I used the all() function to get a list of the accounts in the database. I used the in keyword to check if the name of the deleted account was in the list. It should not be, so I paired it with an assert to ensure the search would come out as false.

```python
def test_delete(self):
    """ Test account deletion """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()                # push into db
    accountName = account           # save account name

    account.delete()                # delete account from db
    allAccounts = Account.all()     # get list of accounts in db

    self.assertEqual(accountName in allAccounts, False) # assert if account is not in the db
```

This test case was successful at covering the lines. Now all that was left was lines 74-75, which was the find() class method.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\test_coverage> py -m nose

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test account deletion
- Test dict to account
- Test the representation of an account
- Test account to dict
- Test account update

Name                    Stmts   Miss  Cover   Missing
----------------------------------------------------
models\__init__.py          7      0   100%
models\account.py          40      2    95%   74-75
----------------------------------------------------
TOTAL                      47      2    96%
----------------------------------------------------
Ran 7 tests in 0.980s

OK
```

This method was easy to interpret because there was a detailed comment. It takes an account id and searches for it. If it is found, it returns the instance. If it is not found, then it returns None. For the test case, I created an account, added it to the database, got the id, then searched for it. If the returned account was the same as the searched account, then the assertion would pass. I also included another test case for if the account was not found. I did this by deleting the account. Then, comparing the return value should match false.

```python
def test_find(self):
    """ Test account find """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()                    # push into db
    accountId = account.id              # get id
    self.assertEqual(account, Account.find(accountId))    # assert if correct account returned
    account.delete()                    # delete the account
    self.assertEqual(None, Account.find(accountId))       # assert if no account was found
```

With that, I achieved 100% coverage of the code.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\test_coverage> py -m nose

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test account deletion
- Test account find
- Test dict to account
- Test the representation of an account
- Test account to dict
- Test account update

Name                    Stmts   Miss  Cover   Missing
--------------------------------------------------------
models\__init__.py         7      0   100%
models\account.py         40      0   100%
--------------------------------------------------------
TOTAL                     47      0   100%
--------------------------------------------------------------------------
Ran 8 tests in 0.812s

OK
```

Task 5

Here are the results of pynose after given code was inserted. It put me in GREEN, so I had to REFACTOR by adding an implementation for PUT. Before that, I had to create the test case.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\tdd> py -m nose

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 0.297s

OK
```

To test updating a counter, I followed the instructions given. The part that was difficult was finding out how to save the baseline of the counter. I used get_json() to do this. After saving the baseline, I called PUT and ensured it was successful. I then verified the counter was actually updated by comparing the new value with the original value. I also added another section of code since the REST guidelines state a counter should be created if it didn't already exist. Thus, I added a check that a counter was successfully created.

```python
def test_update_a_counter(self):
    """It should update a counter"""
    result = self.client.post('/counters/generic')                # Create a counter
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)  # Ensure successful return code
    counterValue = result.get_json()['generic']                    # Save baseline value

    result = self.client.put('/counters/generic')                  # Update the counter
    self.assertEqual(result.status_code in [status.HTTP_200_OK, status.HTTP_204_NO_CONTENT], True)   # Ensure successful return code
    counterValueUpdate = result.get_json()['generic']              # Save new value

    self.assertEqual(counterValueUpdate, (counterValue + 1))       # Ensure counter was updated

    result = self.client.put('/counters/newCounter')               # Should create a new counter if it doesnt exist
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)  # Check if counter was created
```

After running, pynose returned an AssertionError since there was no code to carry out. The assertion failed because the resulting status code was not a successful one. I was in RED.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\tdd> py -m nose

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


======================================================================
FAIL: It should update a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\ocmic\Documents\SPRING2024\CS472\testing\tdd\tests\test_counter.py", line 48, in test_update_a_counter
    self.assertEqual(result.status_code in [status.HTTP_200_OK, status.HTTP_204_NO_CONTENT], True)   # Ensure successful return code
AssertionError: False != True
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: generic
-------------------- >> end captured logging << --------------------

Name               Stmts   Miss  Cover   Missing
--------------------------------------------------------
src\counter.py        11      0   100%
src\status.py          6      0   100%
--------------------------------------------------------
TOTAL                 17      0   100%
--------------------------------------------------------
Ran 3 tests in 0.300s

FAILED (failures=1)
```

I followed the given instructions to create a function that implements PUT. If the counter existed, I incremented its value and returned 200_OK. I also added an additional section to create the counter if it doesn't already exist as per REST guidelines

```python
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    app.logger.info(f"Request to update counter: {name}")
    if name in COUNTERS:
        COUNTERS[name] += 1
        return {name: COUNTERS[name]}, status.HTTP_200_OK
    # Create a counter if it doesn't already exist
    COUNTERS[name] = 1
    return {name: COUNTERS[name]}, status.HTTP_201_CREATED
```

This put me in GREEN after running pynose, allowing me to REFACTOR for the GET portion.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\tdd> py -m nose

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

Name              Stmts   Miss   Cover   Missing
-------------------------------------------------
src\counter.py      19      0    100%
src\status.py        6      0    100%
-------------------------------------------------
TOTAL               25      0    100%
-------------------------------------------------------------------
Ran 3 tests in 0.324s

OK
```

I followed REST guidelines to create the test case for reading a counter. After calling GET, it should return 200_OK, so I added an assertion to check for that. GET should also return an error if the counter doesn't exist, so I also put an assertion for the error.

```python
def test_read_a_counter(self):
    """It should read a counter"""
    self.client.post('/counters/bob')                                    # Create a counter
    read = self.client.get('/counters/bob')                              # Read a counter
    self.assertEqual(read.status_code, status.HTTP_200_OK)               # Ensure successful return code
    result = self.client.get('/counters/unrealCounter')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)      # If object doesn't exist, should return error
```

This put me in RED with an AssertionError. The program returned a 405 method not allowed error instead of a 200_OK status, so I worked on implementing GET.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\tdd> py -m nose

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


======================================================================
FAIL: It should read a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\ocmic\Documents\SPRING2024\CS472\testing\tdd\tests\test_counter.py", line 60, in test_read_a_counter
    self.assertEqual(read.status_code, status.HTTP_200_OK)        # Ensure successful return code
AssertionError: 405 != 200
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: bob
-------------------- >> end captured logging << ---------------------

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 0.309s

FAILED (failures=1)
```

If the counter was found, it would return that same counter along with a 200_OK status code. Otherwise, it should return a 404_NOT_FOUND.

```python
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Read a counter"""
    app.logger.info(f"Request to read counter: {name}")
    if name in COUNTERS:
        return {name: COUNTERS[name]}, status.HTTP_200_OK
    else:
        return {"Message": "{name} not found"}, status.HTTP_404_NOT_FOUND
```

I completed implementing the methods as indicated by the successful tests and 100% coverage.

```
PS C:\Users\ocmic\Documents\SPRING2024\CS472\testing\tdd> py -m nose

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

Name             Stmts   Miss  Cover   Missing
----------------------------------------------------
src\counter.py      25      0   100%
src\status.py        6      0   100%
----------------------------------------------------
TOTAL               31      0   100%
----------------------------------------------------
Ran 4 tests in 0.327s

OK
```