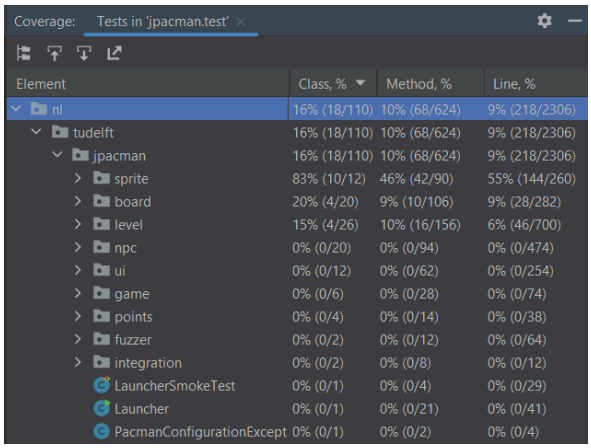
Report

Fork repository: https://github.com/ederic-oytas/CS472TeamsRepo

Task 1

Below is a screenshot of the coverage:



Coverage: Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl	16% (18/110)	10% (68/624)	9% (218/2306)
tudelft	16% (18/110)	10% (68/624)	9% (218/2306)
jpacman	16% (18/110)	10% (68/624)	9% (218/2306)
sprite	83% (10/12)	46% (42/90)	55% (144/260)
board	20% (4/20)	9% (10/106)	9% (28/282)
level	15% (4/26)	10% (16/156)	6% (46/700)
npc	0% (0/20)	0% (0/94)	0% (0/474)
ui	0% (0/12)	0% (0/62)	0% (0/254)
game	0% (0/6)	0% (0/28)	0% (0/74)
points	0% (0/4)	0% (0/14)	0% (0/38)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
integration	0% (0/2)	0% (0/8)	0% (0/12)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
PacmanConfigurationExcept	0% (0/1)	0% (0/2)	0% (0/4)

Question: Is the coverage good enough?

- Answer: No, because the tests done so far only cover 9% of all lines in 'nl'. For some files, it is completely zero.

Task 2

I made tests for:

- src/main/java/nl/tudelft/jpacman/level/Pellet.getValue
- src/main/java/nl/tudelft/jpacman/level/Pellet.getSprite
- src/main/java/nl/tudelft/jpacman/level/LevelFactory.createPellet

These are located in jpacman/src/test/java/nl/tudelft/jpacman/level/EdericOytas_UnitTest.java

Below are the three unit tests I wrote:

```
@Test
void testPelletGetValue() {
   Sprite pellet_sprite = SPRITES.getPelletSprite();
   Pellet pellet = new Pellet(15, pellet_sprite);
   assertThat(pellet.getValue()).isEqualTo(15);
}

@Test
void testPelletGetSprite() {
   Sprite pellet_sprite = SPRITES.getPelletSprite();
   Pellet pellet = new Pellet(15, pellet_sprite);
   assertThat(pellet.getSprite()).isEqualTo(pellet_sprite);
}

@Test
void testLevelFactoryCreatePellet() {

   GhostFactory ghostFactory = new GhostFactory(SPRITES);
   PointCalculator pointCalculator = new DefaultPointCalculator();
   LevelFactory levelFactory = new LevelFactory(
       SPRITES, ghostFactory, pointCalculator
   );

   Sprite pelletSprite = SPRITES.getPelletSprite();
   Pellet pellet1 = levelFactory.createPellet();
   Pellet pellet2 = levelFactory.createPellet();

   assertThat(pellet1.getValue()).isEqualTo(10);
   assertThat(pellet2.getValue()).isEqualTo(10);
   assertThat(pellet1.getSprite()).isEqualTo(pelletSprite);
   assertThat(pellet2.getSprite()).isEqualTo(pelletSprite);

}
```

Below is a screenshot of the code coverage without the three unit tests added.

Element	Class, %	Method, %	Line, %
nl	16% (18/110)	10% (68/624)	9% (218/2306)
tudelft	16% (18/110)	10% (68/624)	9% (218/2306)
jpacman	16% (18/110)	10% (68/624)	9% (218/2306)
sprite	83% (10/12)	46% (42/90)	55% (144/260)
board	20% (4/20)	9% (10/106)	9% (28/282)
level	15% (4/26)	10% (16/156)	6% (46/700)
Player	100% (1/1)	62% (5/8)	75% (18/24)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
Level	0% (0/2)	0% (0/17)	0% (0/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
DefaultPlayerInteractionN	0% (0/1)	0% (0/5)	0% (0/13)
npc	0% (0/20)	0% (0/94)	0% (0/474)
ui	0% (0/12)	0% (0/62)	0% (0/254)
game	0% (0/6)	0% (0/28)	0% (0/74)
points	0% (0/4)	0% (0/14)	0% (0/38)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
integration	0% (0/2)	0% (0/8)	0% (0/12)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
PacmanConfigurationExcepti	0% (0/1)	0% (0/2)	0% (0/4)

Below is with them added.

nl	23% (26/110)	13% (82/624)	10% (254/2328)
tudelft	23% (26/110)	13% (82/624)	10% (254/2328)
jpacman	23% (26/110)	13% (82/624)	10% (254/2328)
sprite	83% (10/12)	48% (44/90)	56% (146/260)
points	50% (2/4)	0% (0/14)	4% (2/42)
level	30% (8/26)	16% (26/156)	10% (72/706)
Pellet	100% (1/1)	100% (3/3)	100% (6/6)
Player	100% (1/1)	62% (5/8)	75% (18/24)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
LevelFactory	50% (1/2)	28% (2/7)	24% (7/29)
Level	0% (0/2)	0% (0/17)	0% (0/113)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
DefaultPlayerInteractionN	0% (0/1)	0% (0/5)	0% (0/13)
board	20% (4/20)	9% (10/106)	9% (28/282)
npc	10% (2/20)	2% (2/94)	1% (6/486)
ui	0% (0/12)	0% (0/62)	0% (0/254)
game	0% (0/6)	0% (0/28)	0% (0/74)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
integration	0% (0/2)	0% (0/8)	0% (0/12)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
PacmanConfigurationExcepti	0% (0/1)	0% (0/2)	0% (0/4)

As shown, the Pellet class went from 0% line coverage to 100% line coverage, and the LevelFactory class went from 0% line coverage to 24% line coverage.

Task 3

Below are the coverage result from JaCoCo at the top level.

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		68%		58%	72	155	102	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	1,206 of 4,694	74%	291 of 637	54%	291	590	227	1,039	51	268	6	47

Below are the coverage results from JaCoCo for the level package.

nl.tudelft.jpacman.level

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
CollisionInteractionMap		0%		0%	19	19	46	46	7	7	1	1
Level		86%		70%	25	55	6	105	1	15	0	1
LevelFactory.RandomGhost		0%		0%	6	6	12	12	3	3	1	1
DefaultPlayerInteractionMap		0%		n/a	5	5	17	17	5	5	1	1
MapParser		87%		78%	7	26	7	69	1	10	0	1
PlayerCollisions		75%		57%	5	14	6	28	1	7	0	1
CollisionInteractionMap.InverseCollisionHandler		0%		n/a	2	2	5	5	2	2	1	1
LevelFactory		89%		80%	1	8	1	17	0	4	0	1
Player		91%		83%	2	11	2	24	1	8	0	1
Level.NpcMoveTask		100%		100%	0	3	0	10	0	2	0	1
PlayerFactory		100%		n/a	0	3	0	5	0	3	0	1
Pellet		100%		n/a	0	3	0	6	0	3	0	1
Total	434 of 1,365	68%	68 of 165	58%	72	155	102	344	21	69	4	12

Question: Are the coverage results from `JaCoCo` similar to the ones you got from `IntelliJ` in the last task? Why so or why not?

- Answer: For some files, such as Level.java, the difference in line/instruction coverage is massive. For Level.java, it is 0% versus 86%. This may indicate that some other tests (like non-unit tests) are also being run when the JaCoCo makes a coverage report versus when it is done using IntelliJ, resulting in a higher line coverage.

Question: Did you find helpful the source code visualization from `JaCoCo` on uncovered branches?

- Answer: Yes, I found the source code visualization helpful to directly see which lines are not covered, so I may create tests to cover them.

Question: Which visualization did you prefer and why? `IntelliJ`'s coverage window or `JaCoCo`'s report?

- Answer: I preferred IntelliJ's coverage window because it is much more terse, has dropdown menus for the coverage (instead of going to a different page), and does not require a separate window to open.

Task 4

I created 3 unit tests in order to reach 100% code coverage.

To test Account.from_dict, I created this test to check if all the attributes are being set on the class from the given dict.

```
def test_from_dict():
    rand = randrange(0, len(ACCOUNT_DATA))  # Generate a random index
    data = ACCOUNT_DATA[rand]  # get a random account
    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"]  # not in test
data
```

To test Account.update and Account.find, I started with an initial Account object, created a new entry in the database, and tested that the Account object from the database account matches the initial account object I started with. I also tested that Account.update raises a DataValidationError before the account is created in the database.

```
def test_update_and_find():
    rand = randrange(0, len(ACCOUNT_DATA))  # Generate a random index
    data = ACCOUNT_DATA[rand]  # get a random account
    account = Account(**data)

    # Test error; account not in DB yet
    with pytest.raises(DataValidationError):
        account.update()

    account.create()

    account.name = "Timmy"
    account.email = "timmy@email.something"
    account.update()

    db_account = Account.find(account.id)
    assert db_account.name == account.name
    assert db_account.email == account.email
```

To test Account.delete (and also Account.find again), I started with an initial Account object, created a new entry in the database, tested that it existed, deleted it, then tested that it does not exist.

```
def test_delete_and_find():
    rand = randrange(0, len(ACCOUNT_DATA))  # Generate a random index
    data = ACCOUNT_DATA[rand]  # get a random account
    account = Account(**data)

    account.create()
    db_account = Account.find(account.id)
    assert db_account is not None

    account.delete()
    db_account = Account.find(account.id)
    assert db_account is None
```

Below is the coverage report adding the above code snippets.

```
---------- coverage: platform win32, python 3.11.4-final-0 -----------
Name                 Stmts   Miss  Cover   Missing
------------------------------------------------
models\__init__.py       7      0   100%
models\account.py       40      0   100%
------------------------------------------------
TOTAL                   47      0   100%
```

Task 5

Update Method

I wrote the following method to test updating a counter:

```
    def test_update_a_counter(self, client):
        result = client.post('/counters/fiz')
        assert result.status_code == status.HTTP_201_CREATED
        assert result.json['fiz'] == 0
        result = client.put('/counters/fiz')
        assert result.status_code == status.HTTP_200_OK
        assert result.json['fiz'] == 1
```

After running it, I got an AssertionError when checking the status code after the endpoint to update is called.

```
========================================================================== FAILURES ==========================================
_____________________________________________________________ TestCounterEndPoints.test_update_a_counter _____________________________

self = <tests.test_counter.TestCounterEndPoints object at 0x0000029B0DA4FE10>, client = <FlaskClient <Flask 'src.counter'>>

    def test_update_a_counter(self, client):
        result = client.post('/counters/fiz')
        assert result.status_code == status.HTTP_201_CREATED
        assert result.json['fiz'] == 0
        result = client.put('/counters/fiz')
>       assert result.status_code == status.HTTP_200_OK
E       assert 405 == 200
E        +  where 405 = <WrapperTestResponse streamed [405 METHOD NOT ALLOWED]>.status_code
E        +  and   200 = status.HTTP_200_OK

tests\test_counter.py:47: AssertionError

---------- coverage: platform win32, python 3.11.4-final-0 -----------
Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\__init__.py       0      0   100%
src\counter.py       11      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                17      0   100%

================================================================= short test summary info ==================================
FAILED tests/test_counter.py::TestCounterEndPoints::test_update_a_counter - assert 405 == 200
============================================================= 1 failed, 2 passed in 0.33s ==============================
```

Then I created a new function in counter.py, which covers this method.

```
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    app.logger.info(f"Request to create counter: {name}")
    global COUNTERS
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

This makes the test case pass.

```
collected 3 items

tests/test_counter.py::TestCounterEndPoints::test_create_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_duplicate_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_update_a_counter PASSED

---------- coverage: platform win32, python 3.11.4-final-0 -----------
Name              Stmts   Miss  Cover   Missing
---------------------------------------------
src\__init__.py       0      0   100%
src\counter.py       16      0   100%
src\status.py         6      0   100%
---------------------------------------------
TOTAL                22      0   100%


========================================================================== 3 passed in 0.27s =====
```

I noticed that the update method does not cover cases where the counter does not exist, so I created another test case:

```
    def test_update_not_found(self, client):
        result = client.put('/counters/buz')
        assert result.status_code == status.HTTP_404_NOT_FOUND
```

This resulted in an KeyError:

```
  File "C:\Users\ederi\home\cs\cs472\tdd\src\counter.py", line 29, in update_counter
    COUNTERS[name] += 1
    ~~~~~~~~~^^^^^^
KeyError: 'buz'

---------- coverage: platform win32, python 3.11.4-final-0 -----------
Name              Stmts   Miss  Cover   Missing
---------------------------------------------
src\__init__.py       0      0   100%
src\counter.py       16      0   100%
src\status.py         6      0   100%
---------------------------------------------
TOTAL                22      0   100%

================================================================ short test summary info =======
FAILED tests/test_counter.py::TestCounterEndPoints::test_update_not_found - assert 500 == 404
============================================================== 1 failed, 3 passed in 0.31s =====
```

So, I updated the code to include a check for such. This is the updated code:

```
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    app.logger.info(f"Request to create 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
```

Here is the test report:

```
collected 4 items

tests/test_counter.py::TestCounterEndPoints::test_create_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_duplicate_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_update_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_update_not_found PASSED

---------- coverage: platform win32, python 3.11.4-final-0 -----------
Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\__init__.py       0      0   100%
src\counter.py       18      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                24      0   100%


========================================================================== 4 passed in 0.26s ===
```

Read Method

To test the read method, I need to test two cases: the counter exists, and when it does not exist. So I wrote two test cases:

```
    def test_read_a_counter(self, client):
        result = client.post('/counters/raz')
        assert result.status_code == status.HTTP_201_CREATED
        assert result.json['raz'] == 0
        result = client.put('/counters/raz')
        assert result.status_code == status.HTTP_200_OK
        assert result.json['raz'] == 1
        result = client.get('/counters/raz')
        assert result.status_code == status.HTTP_200_OK
        assert result.json['raz'] == 1

    def test_read_not_found(self, client):
        result = client.get('/counters/jaz')
        assert result.status_code == status.HTTP_404_NOT_FOUND
```

Using pytest, I got two failed test cases:

```
E       +  and   404 = status.HTTP_404_NOT_FOUND

tests\test_counter.py:67: AssertionError

---------- coverage: platform win32, python 3.11.4-final-0 -----------
Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\__init__.py       0      0   100%
src\counter.py       18      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                24      0   100%

=================================================================== short test summary info ===
FAILED tests/test_counter.py::TestCounterEndPoints::test_read_a_counter - assert 405 == 200
FAILED tests/test_counter.py::TestCounterEndPoints::test_read_not_found - assert 405 == 404
=================================================================== 2 failed, 4 passed in 0.35s =
```

Then, I added a new function to counter.py:

```
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Read a counter"""
    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
```

This passed all test cases:

```
tests/test_counter.py::TestCounterEndPoints::test_create_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_duplicate_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_update_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_update_not_found PASSED
tests/test_counter.py::TestCounterEndPoints::test_read_a_counter PASSED
tests/test_counter.py::TestCounterEndPoints::test_read_not_found PASSED

---------- coverage: platform win32, python 3.11.4-final-0 -----------
Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\__init__.py       0      0   100%
src\counter.py       24      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                30      0   100%


====================================================================== 6 passed in 0.27s =====
```