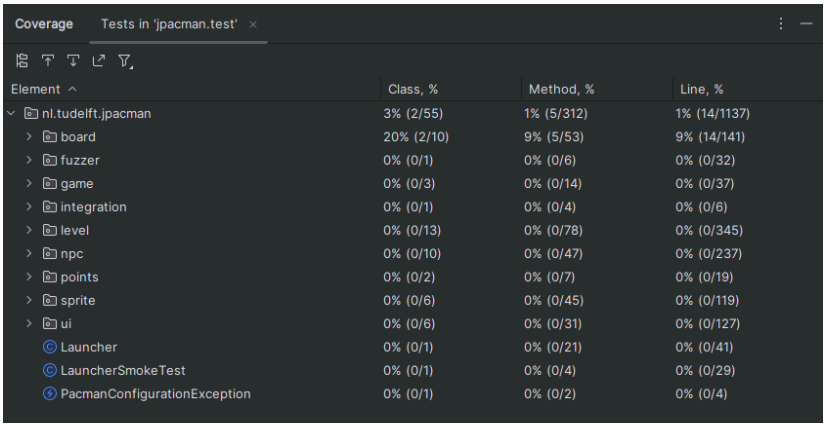
Edward Kim
CS472

Fork Repository: https://github.com/KeenParchment/munch

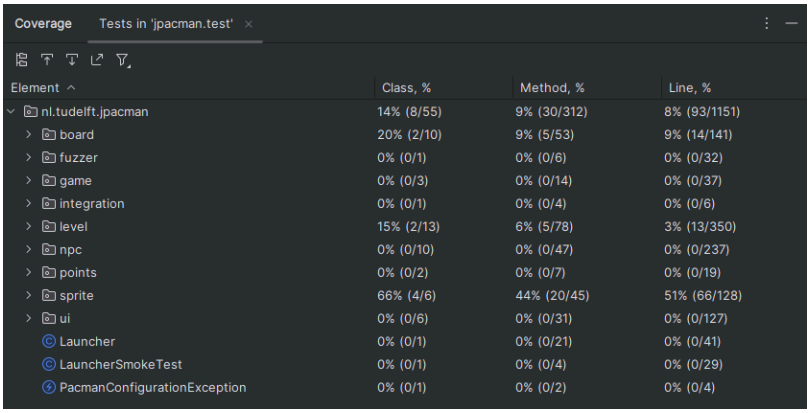
Task 1 - JPacman Test Coverage



- *Is the coverage good enough?*
    - No, the coverage is not good enough. They are in fact all really low.

Task 2.1 - Increasing Coverage on JPacman

- Jpacman Test Coverage after writing level/PlayerTest



- Unit Test 1: src/main/java/nl/tudelft/jpacman/npc.ghost/CyldeTest

```
new *
public class ClydeTest {

    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();

    1 usage
    private final GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);

    1 usage
    private final Clyde clyde = (Clyde) ghostFactory.createClyde();

    new *
    @Test
    void ClydeIsNotNull() { assertThat(clyde).isNotNull(); }
}
```

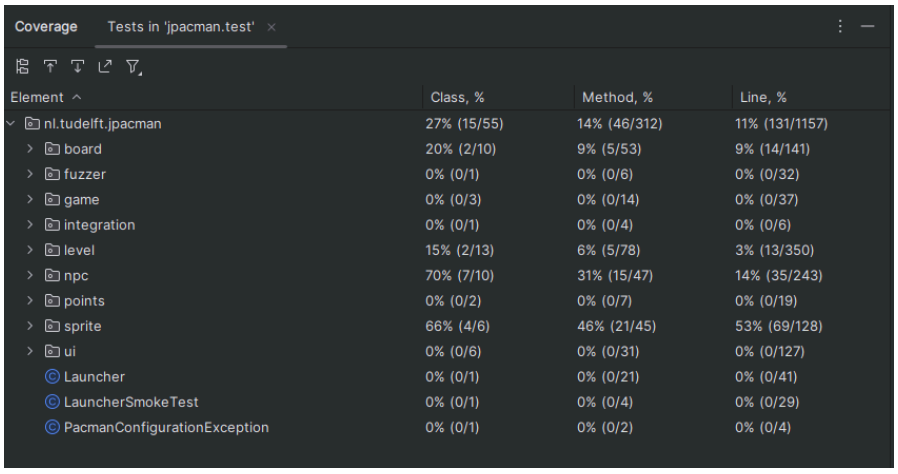
- Unit Test 2: src/main/java/nl/tudelft/jpacman/npc.ghost/InkyTest

```
new *
public class InkyTest {

    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();

    1 usage
    private final GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);

    1 usage
    private final Inky inky = (Inky) ghostFactory.createInky();

    new *
    @Test
    void InkyIsNotNull() { assertThat(inky).isNotNull(); }
}
```

- Unit Test 3: src/main/java/nl/tudelft/jpacman/npc.ghost/PinkyTest

```
new *
public class PinkyTest {

    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();

    1 usage
    private final GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);

    1 usage
    private final Pinky pinky = (Pinky) ghostFactory.createPinky();

    new *
    @Test
    void PinkyIsNotNull() { assertThat(pinky).isNotNull(); }
}
```

- Jpacman Test Coverage after writing 3 unit tests



Edward Kim
CS472

Task 3 - JaCoco Report on JPacman

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		67%		57%	74	155	104	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	1,213 of 4,694	74%	293 of 637	54%	293	590	229	1,039	51	268	6	47

- *Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?*
    - The coverage outcomes reported by JaCoCo differ significantly from those generated by IntelliJ. Specifically, IntelliJ's coverage with my test code falls short compared to JaCoCo, which provides a more comprehensive assessment overall.

- *Did you find helpful the source code visualization from JaCoCo on uncovered branches?*
    - Yes, the visualization in JaCoCo is helpful in seeing what new branches have test codes. This is especially true when working with a large number of contributors to the source code, where a visualization can give a good gauge to the project overall. However, I do wish there was more information regarding the meaning of JaCoCo, such as a legend for the green and red bars. This would allow any user to visit the JaCoCo report quickly.

- *Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?*
    - I prefer a bit of both. They both can provide good information to the user. However, I slightly prefer the IntelliJ coverage window as it gives enough information for me, while JaCoCo is a bit harder to understand. However, JaCoCo does provide more comprehensive coverage than IntelliJ. If I was working with a large number of contributors, I would prefer to use JaCoCo's report for an overall representation of the coverage.

Edward Kim
CS472

Task 4 - Working with Python Test Coverage

- Nosetests coverage report without tests cases

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data

Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models\__init__.py      7      0   100%
models\account.py      40     13    68%   26, 30, 34-35, 45-48, 52-54, 74-75
-------------------------------------------------
TOTAL                  47     13    72%
----------------------------------------------------------------------
Ran 2 tests in 1.185s

OK
```

- Test Case: test_from_dict()

```python
def test_from_dict(self):
    """ Test creating an Account from a dictionary """
    data = ACCOUNT_DATA[self.rand]
    original_account = Account(**data)

    # Act
    new_account = Account()
    new_account.from_dict(original_account.to_dict())

    # Check if the attributes of the original and new accounts match
    self.assertEqual(original_account.name, new_account.name)
    self.assertEqual(original_account.email, new_account.email)
    self.assertEqual(original_account.phone_number, new_account.phone_number)
    self.assertEqual(original_account.disabled, new_account.disabled)
    self.assertEqual(original_account.date_joined, new_account.date_joined)
```

- Test Case: test_update()

```python
def test_update(self):
    """Test successful update of an account"""
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()

    # Act
    updated_name = "Updated Name"
    account.name = updated_name
    account.update()

    # Check If the name of the account has been successfully updated
    updated_account = Account.find(account.id)
    self.assertEqual(updated_account.name, updated_name)
```

- Test Case: test_update_empty_id()

```
def test_update_empty_id(self):
    """ Test update of an account with no ID """
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)

    # Attempt to update an account with an ID, should raise DataValidationError
    with self.assertRaises(DataValidationError):
        account.update()
```

- Test Case: test_delete()

```
def test_delete(self):
    """ Test deletion of an account """
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()

    # Act
    account_id = account.id
    account.delete()

    # Check if the account has been successfully deleted
    self.assertIsNone(Account.find(account_id))
```

- Nosetests coverage report with 100% coverage after implementation of test cases

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deletion of an account
- Test creating an Account from a dictionary
- Test the representation of an account
- Test account to dict
- Test successful update of an account
- Test update of an account with no ID

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 1.815s

OK
```

Edward Kim
CS472

Task 5 - TDD

- Test Case RED phase: test_create_a_counter()

```
def test_create_a_counter(self):
    """It should create a counter"""
    result = self.client.post('/counters/foo')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
```

- Test Case RED phase: test_duplicate_a_counter()

```
def test_duplicate_a_counter(self):
    """It should return an error for duplicates"""
    self._create_counter_and_assert( name: 'bar_duplicate', status.HTTP_201_CREATED)
    result = self.client.post('/counters/bar_duplicate')
    self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
```

- Test Case RED phase: test_read_a_counter()

```
def test_read_a_counter(self):
    """It should read a counter"""
    counter_name = 'bar_read'
    self._create_counter_and_assert(counter_name, status.HTTP_201_CREATED)

    result = self.client.get(f'/counters/{counter_name}')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    value = json.loads(result.data)[counter_name]
    self.assertEqual(value,  second: 0)

    no_result = self.client.get('/counters/no_result_read')
    self.assertEqual(no_result.status_code, status.HTTP_404_NOT_FOUND)
    response_data = json.loads(no_result.data)
    self.assertEqual(response_data["error"],  second: "Counter not found")
```

- Test Case RED phase: test_update_a_counter()

```
def test_update_a_counter(self):
    """It should update a counter"""
    counter_name = 'bar_update'
    self._create_counter_and_assert(counter_name, status.HTTP_201_CREATED)

    base_result = self.client.get(f'/counters/{counter_name}')
    base_value = json.loads(base_result.data)[counter_name]

    update_result = self.client.put(f'/counters/{counter_name}')
    self.assertEqual(update_result.status_code, status.HTTP_200_OK)

    new_result = self.client.get(f'/counters/{counter_name}')
    new_value = json.loads(new_result.data)[counter_name]
    self.assertEqual(new_value, base_value + 1)

    no_result = self.client.put('/counters/no_result_update')
    self.assertEqual(no_result.status_code, status.HTTP_404_NOT_FOUND)
    response_data = json.loads(no_result.data)
    self.assertEqual(response_data["error"],  second: "Counter not found")
```

- Test Case GREEN & REFACTOR phase: create_counter()

```python
@app.route( rule: '/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
```

- Test Case GREEN & REFACTOR phase: read_counter()

```python
@app.route( rule: '/counters/<name>', methods=['GET'])
def read_counter(name):
    """Read a counter value"""
    global COUNTERS
    if name not in COUNTERS:
        return {"error": "Counter not found"}, status.HTTP_404_NOT_FOUND
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

- Test Case GREEN & REFACTOR phase: update_counter()

```python
@app.route( rule: '/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    global COUNTERS
    if name not in COUNTERS:
        return {"error": "Counter not found"}, status.HTTP_404_NOT_FOUND

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

- Nosetests coverage report after implementation of test cases

```
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      22      0   100%
src\status.py        6      0   100%
------------------------------------------------
TOTAL               28      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.149s

OK
```

Edward Kim
CS472

- Exceptions you encountered while running nosetests?
    - Encountered what method ‘PUT’ ‘GET’ POST’ to use.
    - Required to have REFACTOR in test cases, otherwise, while nosetests may report the coverage to be 100%, however, the text would output RED.