Jadon Loi
Github Repo: https://github.com/noodaj/munch

Task 2.1

No, the coverage is not enough.

Coverage | Tests in 'jpacman'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
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	0% (0/13)	0% (0/78)	0% (0/345)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	0% (0/6)	0% (0/45)	0% (0/119)
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)

As we can see there is only 10% of the code that is tested throughout the entire source code.

Coverage | Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	29% (16/...	18% (58/311)	15% (177/11...
board	60% (6/10)	38% (20/52)	41% (61/147)
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	15% (2/13)	12% (10/78)	7% (26/350)
npc	40% (4/10)	12% (6/47)	6% (17/243)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	66% (4/6)	48% (22/45)	57% (73/128)
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)
PacmanConfigurationExce	0% (0/1)	0% (0/2)	0% (0/4)

```java
package nl.tudelft.jpacman.board;

import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;
public class BoardTest {
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    public Square s1 = new BasicSquare();
    public Square s2 = new BasicSquare();
    public BoardFactory board;

    @Test
    void testBoard() {
        //test 3
        board = new BoardFactory(SPRITE_STORE);
        Board mockBoard = board.createBoard(new Square[][]{{s1}});

        assertThat(mockBoard.getHeight()).isEqualTo(1);
        assertThat(mockBoard.getWidth()).isEqualTo(1);
        assertThat(mockBoard.invariant()).isEqualTo(true);
    }

}
```

I created 3 unit tests for and the ones I chose are from the Board Class, the getWidth and getHeight function. I also did unit tests in the Player class for the functions setAlive and getScore.

Task 3

There are some similarities with some of the unit tests that I wrote. Most of the code though is still not covered so it is not as good as JaCoCo's unit tests.
Yes, the visualization is nice seeing what new branches have test codes. It makes the person reviewing the pull request's life a lot easier since they have test cases to help verify that their code works well.
I like JaCoCo's report more because of the bar it provides as well as what functions were tested. They both have their main focus to be how many lines were covered throughout the entire code base which is the first thing that they show. Also another thing that is nice is the testing between branches. Some branches might have new features that the main branch might not have and these tests can be used in the main repository in the future if a pull request is made.

Task 4

```
Name                  Stmts   Miss  Cover   Missing
-----------------------------------------------
models/__init__.py        6      0   100%
models/account.py        40     15    62%   26, 30, 34-35, 45-48, 52-54, 63-64, 74-75
-----------------------------------------------
TOTAL                    46     15    67%
----------------------------------------------------------------------
Ran 2 tests in 0.288s
```

```
Name                  Stmts   Miss  Cover   Missing
-----------------------------------------------
models\__init__.py        6      0   100%
models\account.py        42      0   100%
-----------------------------------------------
TOTAL                    48      0   100%
----------------------------------------------------------------------
Ran 8 tests in 0.418s
```

Below are some code snippets

```
noodaj
def test_repr(self):
    """ Test account to repr"""
    account = Account()
    account.name = "Jadon"
    self.assertEqual(str(account),  second: "<Account 'Jadon'>")
```

```
noodaj
def test_to_dict(self):
    """ Test account to dict """
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    result = account.to_dict()
    self.assertEqual(account.name, result["name"])
    self.assertEqual(account.email, result["email"])
    self.assertEqual(account.phone_number, result["phone_number"])
    self.assertEqual(account.disabled, result["disabled"])
    self.assertEqual(account.date_joined, result["date_joined"])
```

Task 5

```
Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\counter.py       23      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                29      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.143s

OK
```

Code snippets

These are snippets for the create a counter both the green and red

```
new *
@app.route( rule: "/counters/<name>", methods=["POST"])
def create_counter(name):
    """ Create 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
```

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

```
new *
def test_create_a_counter(self):
    """ Counter should be created """
    result = self.client.post("/counters/foo")
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    result = self.client.post("/counters/foo")
    self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
```

These are snippets for the both the green and red update a counter

```
new *
@app.route( rule: '/counters/<name>', methods=["PUT"])
def update_count(name):
    if name not in COUNTERS:
        COUNTERS[name] = 1
        return {"Message": f"Counter {name} does not exist"}, status.HTTP_204_NO_CONTENT

    COUNTERS[name] += 1

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

```
def test_update_a_counter(self):
    """ Update create a counter """
    result = self.client.post("/counters/test")

    self.assertEqual(COUNTERS['test'],  second: 0)
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    result = self.client.put('/counters/test')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(COUNTERS['test'],  second: 1)

    result = self.client.put('/counters/test2')
    self.assertEqual(result.status_code, status.HTTP_204_NO_CONTENT)
```

These are the code snippets for the read a counter both the green and red

```
@app.route( rule: '/counters/<name>', methods=["GET"])
def read_a_counter(name):
    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
```

```
def test_get_a_counter(self):
    self.client.post('/counters/hello')
    self.client.put('/counters/hello')

    result = self.client.get('/counters/hello')
    self.assertEqual(result.status_code, status.HTTP_200_OK)

    result = self.client.get('/counters/hello2')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
```