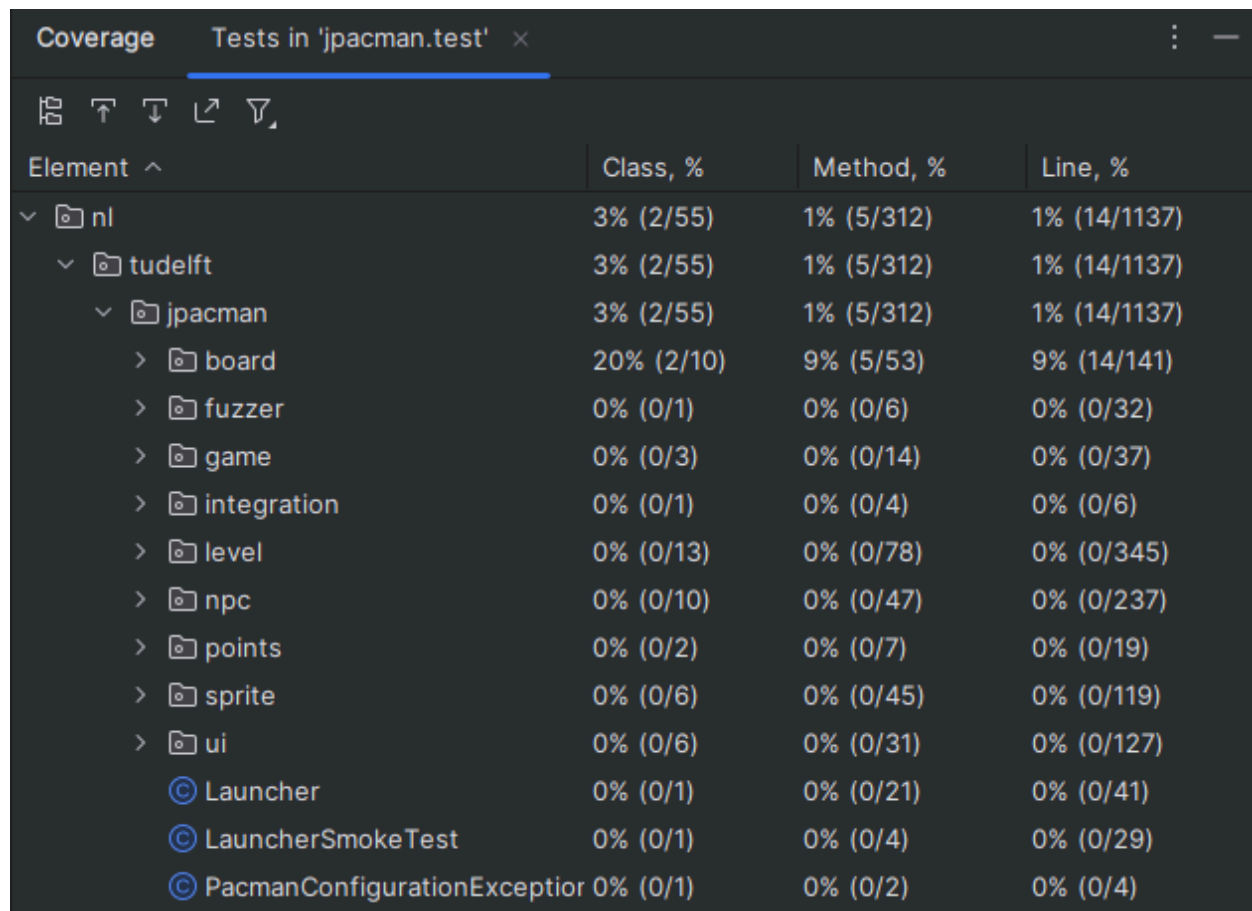


## Software Testing Lab Summary

My Forked Repo: <https://github.com/vicliyj/UNLV-S24-CS472-Group7>

### Task 1 - JPacman Test Coverage

This is the initial coverage of the included unit tests of the JPacman game. The coverage percentages are quite low, which isn't good enough! 1% or less of every method is tested.



The screenshot shows the 'Coverage' tool in IntelliJ IDEA, displaying test results for 'jpacman.test'. The table lists various elements (packages and classes) and their coverage percentages for Class, Method, and Line. The coverage is generally low, with most elements having 0% or 1% coverage. The 'board' package shows slightly higher coverage at 20% for Class, 9% for Method, and 9% for Line.

Element ^	Class, %	Method, %	Line, %
nl	3% (2/55)	1% (5/312)	1% (14/1137)
tudelft	3% (2/55)	1% (5/312)	1% (14/1137)
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)

### Task 2 - Increasing Coverage on JPacman

- Coverage after addition of PlayerTest (given unit test)

Element ^	Class, %	Method, %	Line, %
✓  nl	16% (9/55)	9% (30/312)	8% (95/1153)
✓  tudelft	16% (9/55)	9% (30/312)	8% (95/1153)
✓  jpacman	16% (9/55)	9% (30/312)	8% (95/1153)
>  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	15% (2/13)	6% (5/78)	3% (13/350)
>  npc	0% (0/10)	0% (0/47)	0% (0/237)
>  points	0% (0/2)	0% (0/7)	0% (0/19)
>  sprite	83% (5/6)	44% (20/45)	52% (68/130)
>  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)

- Coverage after addition of ProgressTest. Tested if the game .isInProgress()

Element ^	Class, %	Method...	Line, %
✓  nl	58% (32...	35% (112/...	30% (366...
✓  tudelft	58% (32...	35% (112/...	30% (366...
✓  jpacman	58% (32...	35% (112/...	30% (366...
>  board	70% (7/10)	52% (28/...	56% (81/1...
>  fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
>  game	100% (3/...	42% (6/14)	40% (18/4...
>  integration	0% (0/1)	0% (0/4)	0% (0/6)
>  level	53% (7/13)	29% (23/...	35% (128/...
>  npc	70% (7/10)	31% (15/47)	14% (35/2...
>  points	100% (2/...	57% (4/7)	54% (12/2...
>  sprite	83% (5/6)	53% (24/...	56% (74/1...
>  ui	0% (0/6)	0% (0/31)	0% (0/127)
Ⓢ Launcher	100% (1/1)	57% (12/21)	37% (18/48)
Ⓢ LauncherSmokeTes	0% (0/1)	0% (0/4)	0% (0/29)
⚡ PacmanConfiguratic	0% (0/1)	0% (0/2)	0% (0/4)

## Code for PlayerTest

```
package nl.tudelft.jpacman.game;

import nl.tudelft.jpacman.Launcher;
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;

public class ProgressTest {

    1 usage
    private Launcher game = new Launcher();

    1 usage
    private Game TheGame = game.makeGame();

    @Test
    void testProgress(){
        |    assertThat(TheGame.isInProgress()).isEqualTo( expected: false);
    }
}
```

- Code for GhostTypeTest unit test. This checks to make sure the ghosts are created correctly and doesn't return a null value.

```
package nl.tudelft.jpacman.npc.ghost;

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

import static org.assertj.core.api.Assertions.assertThat;

public class GhostTypeTest {
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private GhostFactory Factory = new GhostFactory(SPRITE_STORE);
    1 usage
    public Ghost blinky = Factory.createBlinky();

    @Test
    void testType(){ assertThat((blinky != null)); }

}
```

- Code for ScoreTest unit test. Checks that the score at the beginning of the game starts at 0 and isn't a negative value.

```
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

public class ScoreTest {
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    1 usage
    private Player ThePlayer = Factory.createPacMan();

    @Test
    void testScore() {
        int score = ThePlayer.getScore();
        assertThat(actual: score > -1);
    }
}
```

- The coverage after the addition of ProgressTest didn't change much, only about a 1% increase in methods tested. These were the final coverage values after GhostTypeTest and

## ScoreTest.

Element ^	Class, %	Method, %	Line, %
✓ nl	58% (32/55)	36% (113/312)	30% (367/1190)
✓ tudelft	58% (32/55)	36% (113/312)	30% (367/1190)
✓ jpacman	58% (32/55)	36% (113/312)	30% (367/1190)
> board	70% (7/10)	52% (28/53)	56% (81/144)
> fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> game	100% (3/3)	42% (6/14)	40% (18/45)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
> level	53% (7/13)	30% (24/78)	35% (129/360)
> npc	70% (7/10)	31% (15/47)	14% (35/243)
> points	100% (2/2)	57% (4/7)	54% (12/22)
> sprite	83% (5/6)	53% (24/45)	56% (74/130)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
Ⓢ Launcher	100% (1/1)	57% (12/21)	37% (18/48)
Ⓢ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
Ⓢ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

## 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

Created with JaCoCo 0.8.3.201901230119

The coverage from JaCoCo is indeed similar to the ones from IntelliJ, with a couple extra details.

The coverage report from IntelliJ only covers the general percentage while JaCoCo also states how many lines are missed. I don't think the JaCoCo report was very helpful, though I don't think I was using it correctly or to its full capabilities. The colors made it easier to visualize the percentage. In the end, there wasn't much of a difference between both visualizations. The IntelliJ interface was more updated - simple and easy to read while being highly accessible.

## Task 4 - Working with Python Test Coverage

- Code snippets from unit test cases

```

def test_from_dict(self):
    """ Test attributes to dict """
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    thisDict = {"name": "",
                "email": "",
                "phone_number": "",
                "disabled": "",
                "date_joined": ""}
    account.from_dict(thisDict)
    self.assertEqual(thisDict["name"], account.name)
    self.assertEqual(thisDict["email"], account.email)
    self.assertEqual(thisDict["phone_number"], account.phone_number)
    self.assertEqual(thisDict["disabled"], account.disabled)
    self.assertEqual(thisDict["date_joined"], account.date_joined)

```

```

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

    newName = "something else"
    account.name = newName
    account.update()

    self.assertEqual(newName, account.name)
    account.id = ""

    # ID field empty, DataValidationError should be raised
    with self.assertRaises(DataValidationError):
        account.update()

```

```
def test_delete(self):  
    """ Test delete self """  
  
    # Create a new account instance  
    account = Account()  
    account.name = "foo"  
    account.id = 1234  
    account.create()  
    self.assertIsNotNone(account.id, "No account ID")  
    account.delete()  
    self.assertIsNone(Account.find(account.id))
```

```
def test_find(self):  
    account = Account()  
    account.name = "no"  
    account.id = 5678  
    account.create()  
    result = Account.find(account.id)  
    self.assertIsNotNone(result)
```



- FINAL COVERAGE REPORT

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

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 0.123s
```

Task 5 - TDD

Counter updater method - This takes the current counter and increments it by 1, then saves the new value. It also returns a status of 200\_OK

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

Test case for the counter updater method. This test creates a counter, saves the original counter value, and calls the counter update method. The new counter value returned is then checked against the original value, making sure that it has been incremented by 1.

```
def test_update_a_counter(self):
    """Testing if counter is updated"""
    self.client = app.test_client()
    result = self.client.post('/counters/doo')
    currCount = result.json["doo"]
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    result2 = self.client.put('/counters/doo')
    updatedCount = result2.json["doo"]
    self.assertEqual(result2.status_code, status.HTTP_200_OK)
    self.assertEqual(currCount + 1, updatedCount)
```

Counter reader method. This method increments through all existing COUNTERS to see if the one requested exists. If it doesn't exist, then one is created. Then, the counter value is returned, along with a 200\_OK.

```
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    if name not in COUNTERS:
        COUNTERS[name] = 0
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

This is the test case for reading a counter. If a counter exists, then it returns the counter value and a 200\_OK status. If a counter doesn't exist yet, then one should be created and the same status should be returned.

```
def test_read_a_counter(self):
    """Testing if counter is read"""
    # self.client = app.test_client()
    # result = self.client.post('/counters/moo')
    # self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    result2 = self.client.get('/counters/moo')
    self.assertEqual(result2.status_code, status.HTTP_200_OK)
```