

Fork Link: [Github Project Fork](#)

## Task 1 – JPacman Test Coverage:

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)

The coverage is terrible because it only contains 1% coverage for the entire project. For relatively good coverage, 90% coverage is recommended.

## Task 2 - Increasing Coverage on JPacman

Element	Class, %	Method, %	Line, %
▼ nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (93/1151)
> sprite	66% (4/6)	44% (20/45)	51% (66/128)
> board	20% (2/10)	9% (5/53)	9% (14/141)
> level	15% (2/13)	6% (5/78)	3% (13/350)
> npc	0% (0/10)	0% (0/47)	0% (0/237)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
> game	0% (0/3)	0% (0/14)	0% (0/37)
> points	0% (0/2)	0% (0/7)	0% (0/19)
> fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
© LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
© Launcher	0% (0/1)	0% (0/21)	0% (0/41)
© PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

Increased the coverage for the **level** package by adding a unit test to check if the player is alive. Class, Method, and Line coverage went up from 0% for all to 15%, 6% and 3% respectively.

Fork Link: [Github Project Fork](#)

## Task 2.1:

Test Functions:

1)

Element ^	Class, %	Method, %	Line, %
nl.tudelft.jpacman	16% (9/55)	10% (32/312)	8% (96/1156)
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	66% (4/6)	44% (20/45)	51% (66/128)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	100% (1/1)	9% (2/21)	6% (3/46)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

I tested the `getSpriteStore()` function in the Launcher class. The Class, Method, and Line coverage for the Launcher class went from 0% for all to 100%, 9%, and 6% respectively.

2)

Element ^	Class, %	Method, %	Line, %
nl.tudelft.jpacman	16% (9/55)	10% (32/312)	8% (97/1156)
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	66% (4/6)	44% (20/45)	51% (66/128)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	100% (1/1)	14% (3/21)	8% (4/46)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

I tested the `getLevelMap()` function in the Launcher class. The Class, Method, and Line coverage for the Launcher class went from 100%, 9%, and 6% to 100%, 14%, and 8% respectively.

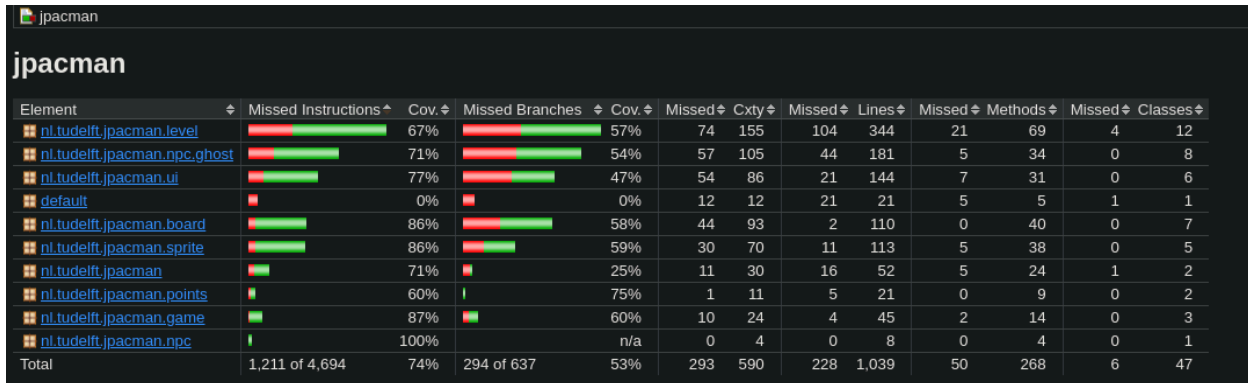
3)

Element ^	Class, %	Method, %	Line, %
nl.tudelft.jpacman	16% (9/55)	10% (34/312)	8% (99/1156)
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	66% (4/6)	44% (20/45)	51% (66/128)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	100% (1/1)	19% (4/21)	13% (6/46)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

I tested the `withMapFile()` function in the Launcher class. The Class, Method, and Line coverage for the launcher class went from 100%, 15%, and 8% to 100%, 19%, and 13%.

Fork Link: [Github Project Fork](#)

### Task 3 – JaCoCo Report on JPacman (10 points)



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%		54%	57	105	44	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		71%		25%	11	30	16	52	5	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	1,211 of 4,694	74%	294 of 637	53%	293	590	228	1,039	50	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?
  - Test coverage is slightly different between IntelliJ and JaCoCo due to their different focuses. IntelliJ coverage provides real-time feedback in the IDE, emphasizing class, method, and line coverage. In contrast, JaCoCo, offers more insights into bytecode instructions and branches, generating comprehensive reports post-build. The differences between the two would explain the slight differences in the reported coverage.
- Did you find helpful the source code visualization from JaCoCo on uncovered branches?
  - The source code visualization from JaCoCo is extremely useful if you want a GUI to see exactly what you're testing in your code. It can help with seeing what tests you have left instead of just seeing a completion count like in IntelliJ
- Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?
  - For initial testing, I would prefer IntelliJ's coverage window due to it being native to the application and it being easy to see coverage for the entire application. But when finalizing my testing infrastructure I'd want to use JaCoCo's report to get a more detailed coverage report to see what portions of code I'm possibly missing.

### Task 4 - Working with Python Test Coverage

#### 1) 72% Test coverage when running nosetests

```
danielshina@homepc ~/.../testing/test_coverage main nosetests
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 0.476s

OK
```

Fork Link: [Github Project Fork](#)

## 2) Adding test\_repr so test coverage is now 74%

```
danielshina@homepc > ~/.../testing/test_coverage > main • nosetests

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

Name          Stmts  Miss  Cover   Missing
-----
models/__init__.py    7     0   100%
models/account.py    40    12    70%   30, 34-35, 45-48, 52-54, 74-75
-----
TOTAL                47    12    74%
-----

Ran 3 tests in 0.500s

OK
```

## 3) Adding test\_to\_dict, test coverage is now 77%

```
danielshina@homepc > ~/.../testing/test_coverage > main • nosetests

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 0.520s

OK
```

## 4) Adding test\_from\_dict, test coverage improved to 81%

```
+ 62 def test_to_dict(self):
+ 63     """ Test account to dict """
+ 64     data = ACCOUNT_DATA[self.rand] # get a random account
+ 65     account = Account(**data)
+ 66     result = account.to_dict()
+ 67     self.assertEqual(account.name, result["name"])
+ 68     self.assertEqual(account.email, result["email"])
+ 69     self.assertEqual(account.phone_number, result["phone_number"])
+ 70     self.assertEqual(account.disabled, result["disabled"])
+ 71     self.assertEqual(account.date_joined, result["date_joined"])
+ 72
+ 73 def test_from_dict(self):
+ 74     """Test setting attributes from a dictionary using from_dict"""
+ 75     account_data = ACCOUNT_DATA[self.rand]
+ 76     account = Account()
+ 77     account.from_dict(account_data)
+ 78
+ 79     for key, value in account_data.items():
+ 80         self.assertEqual(getattr(account, key), value)
+ 81
+ 82 test_account.py 80:13 < 100% Δ 6 ✖ 4
```

```
danielshina@homepc > ~/.../testing/test_coverage > main • nosetests

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test setting attributes from a dictionary using from_dict
- 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 0.512s

OK
```

Fork Link: [Github Project Fork](#)

## 5) Added test\_update and test\_update\_with\_empty\_id, test coverage improved to 89%

```

+ 82 def test_update_account(self):
+ 83     """Test updating an Account in the database"""
+ 84     account_data = ACCOUNT_DATA[self.rand]
+ 85     account = Account()
+ 86     account.from_dict(account_data)
+ 87     account.create()
+ 88
+ 89     new_name = 'Updated Name'
+ 90     account.name = new_name
+ 91     account.update()
+ 92
+ 93     updated_account = Account.query.get(account.id)
+ 94     self.assertIsNotNone(updated_account)
+ 95     self.assertEqual(updated_account.name, new_name)
+ 96
+ 97 def test_update_account_with_empty_id(self):
+ 98     """Test updating an Account with an empty ID field"""
+ 99     account = Account()
+ 100
+ 101     with self.assertRaises(DataValidationError):
+ 102         account.update()

```

```

danielshina@homepc ~/.../testing/test_coverage 1 main • nosetests
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test setting attributes from a dictionary using from_dict
- Test the representation of an account
- Test account to dict
- Test updating an Account in the database
- Test updating an Account with an empty ID field

Name          Stmt% Miss Cover Missing
-----
models/__init__.py 7      0 100%
models/account.py 40      5  88% 52-54, 74-75
TOTAL          47      5  89%

Ran 7 tests in 0.557s
OK

```

## 6) Added test\_delete\_account, test coverage improved to 96%

```

+ 104 def test_delete_account(self):
+ 105     """Test deleting an Account from the database"""
+ 106     account_data = ACCOUNT_DATA[self.rand]
+ 107     account = Account()
+ 108     account.from_dict(account_data)
+ 109     account.create()
+ 110
+ 111     account.delete()
+ 112
+ 113     deleted_account = Account.query.get(account.id)
+ 114     self.assertIsNone(deleted_account)
+ 115
>>115

```

```

danielshina@homepc ~/.../testing/test_coverage 1 main • nosetests
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deleting an Account from the database
- Test setting attributes from a dictionary using from_dict
- Test the representation of an account
- Test account to dict
- Test updating an Account in the database
- Test updating an Account with an empty ID field

Name          Stmt% Miss Cover Missing
-----
models/__init__.py 7      0 100%
models/account.py 40      2  95% 74-75
TOTAL          47      2  96%

Ran 8 tests in 0.591s
OK

```

## 7) Added test\_find\_account\_by\_id, test coverage improved to 100%

```

+ 116 def test_find_account_by_id(self):
+ 117     """Test finding an Account by ID"""
+ 118     account_data = ACCOUNT_DATA[self.rand]
+ 119     account = Account()
+ 120     account.from_dict(account_data)
+ 121     account.create()
+ 122
+ 123     found_account = Account.find(account.id)
+ 124
+ 125     self.assertIsNotNone(found_account)
+ 126     self.assertEqual(found_account.id, account.id)

```

```

danielshina@homepc ~/.../testing/test_coverage 1 main • nosetests
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deleting an Account from the database
- Test finding an Account by ID
- Test setting attributes from a dictionary using from_dict
- Test the representation of an account
- Test account to dict
- Test updating an Account in the database
- Test updating an Account with an empty ID field

Name          Stmt% Miss Cover Missing
-----
models/__init__.py 7      0 100%
models/account.py 40      0 100%
TOTAL          47      0 100%

Ran 9 tests in 0.620s

```

Fork Link: [Github Project Fork](#)

## Task 5 - TDD

1) Create update counter test:

```
def test_update_a_counter(self):
    """It should update a counter"""
    # 1) Make a call to Create a counter.
    result = self.client.post('/counters/woo')
    # 2) Ensure that it returned a successful return code.
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    # 3) Check the counter value as a baseline.
    baseline = result.get_json()['woo']
    self.assertEqual(baseline, 0)

    increment = 1

    # 4) Make a call to Update the counter that you just created.
    result = self.client.put('/counters/woo', json=increment)
    # 5) Ensure that it returned a successful return code.
    self.assertEqual(result.status_code, status.HTTP_200_OK)

    updatedValue = result.get_json()['woo']
    # 6) Check that the counter value is one more than the baseline
    self.assertEqual(updatedValue, baseline + increment)

def test_update_a_nonexistent_counter(self):
    """It should return an error for non-existent counter"""
    result = self.client.post('/counters/doo')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

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

```
danielshina@homepc ~/.../testing/tdd main nosetests
Counter tests
- It should create a counter
- It should return an error for duplicates
- It should update a counter (FAILED)
- It should return an error for non-existent counter (FAILED)

=====
FAIL: It should update a counter
=====
Traceback (most recent call last):
  File "/home/danielshina/unlv/spring_2024/cs_472/labs/testing/tdd/tests/test_counter.py", line 57, in
    test_update_a_counter
    self.assertEqual(result.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: woo
----- >> end captured logging << -----

=====
FAIL: It should return an error for non-existent counter
=====
Traceback (most recent call last):
  File "/home/danielshina/unlv/spring_2024/cs_472/labs/testing/tdd/tests/test_counter.py", line 69, in
    test_update_a_nonexistent_counter
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
AssertionError: 405 != 404
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: doo
----- >> 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 4 tests in 0.116s

FAILED (failures=2)
```

I've written the `test_update_a_counter` and `test_update_a_nonexistent_counter` tests. As you can see it fails because I haven't implemented the PUT request handler yet.

Fork Link: [Github Project Fork](#)

## 2) Create PUT handler

```
29 @app.route('/counters/<name>', methods=['PUT'])
30 def update_counter(name):
31     """Check if counter exists"""
32
33     if name not in COUNTERS:
34         return {"Message": f"Counter {name} doesn't exist!"}, status.HTTP_404_NOT_FOUND
35
36     COUNTERS[name] += request.get_json()
37
38     return {name: COUNTERS[name]}, status.HTTP_200_OK
```

```
x danielshina@homepc ~/.../testing/tdd | main • nosetests

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

Name          Stmts   Miss  Cover   Missing
-----
src/counter.py    17      0  100%
src/status.py      6      0  100%
-----
TOTAL             23      0  100%
-----
Ran 4 tests in 0.121s

OK
```

After adding the handler to process a PUT request on route /counters/<name> the test now passes

## 3) Create get counter test:

```
def test_get_a_counter(self):
    """It should get a counter"""
    # Create new counter
    result = self.client.post('/counters/far')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    expectedValue = 0

    # Check counter is 0
    result = self.client.get('/counters/far')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(result.get_json()['far'], expectedValue)

    # Update counter by 1
    result = self.client.put('/counters/far', json=1)
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(result.get_json()['far'], expectedValue + 1)

    # Check counter is 1 after update
    result = self.client.get('/counters/far')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(result.get_json()['far'], expectedValue + 1)
```



Fork Link: [Github Project Fork](#)

```
def test_get_a_nonexistent_counter(self):
    """It should return an error for non-existent counter"""
    result = self.client.post('/counters/coo')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

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

```
danielshina@homepc ~/.../testing/tdd main nosetests

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

=====
FAIL: It should get a counter
-----
Traceback (most recent call last):
  File "/home/danielshina/unlv/spring_2024/cs_472/labs/testing/tdd/tests/test_counter.py", line 81, in
test_get_a_counter
    self.assertEqual(result.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: far
----- >> end captured logging << -----

=====
FAIL: It should return an error for non-existent counter
-----
Traceback (most recent call last):
  File "/home/danielshina/unlv/spring_2024/cs_472/labs/testing/tdd/tests/test_counter.py", line 100, i
n test_get_a_nonexistent_counter
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
AssertionError: 405 != 404
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: coo
----- >> end captured logging << -----

Name          Stmts  Miss  Cover   Missing
-----
src/counter.py   17     0   100%
src/status.py     6     0   100%
-----
TOTAL           23     0   100%
```

I've written the `test_get_a_counter` and `test_get_a_nonexistent_counter` tests. As you can see it fails because I haven't implemented the GET request handler yet.



Fork Link: [Github Project Fork](#)

#### 4) Create GET handler

```
@app.route('/counters/<name>', methods=['GET'])
def get_counter(name):
    """Check if counter exists"""

    if name not in COUNTERS:
        return {"Message": f"Counter {name} doesn't exist!"}, status.HTTP_404_NOT_FOUND

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

```
✖ danielshina@homepc ~/.../testing/tdd main nosetests

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

Name          Stmt%  Miss  Cover  Missing
-----
src/counter.py    22    0   100%
src/status.py     6     0   100%
-----
TOTAL            28    0   100%
-----

Ran 6 tests in 0.120s

OK
```

After adding the handler to process a GET request on route /counters/<name> the test now passes.