

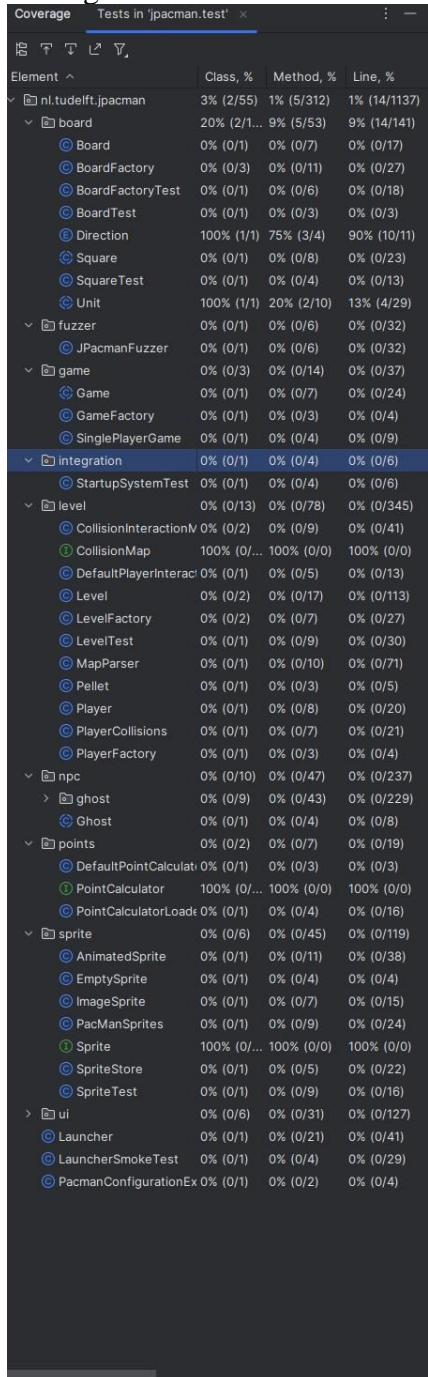
MacGyver Codilla

CS472

Task 1

Is the coverage good enough?

After cloning the repo and building with coverage, the coverage window reports that there is barely any code coverage. This is not acceptable. If code is not worth testing, it is not worth writing.



The screenshot shows a coverage report for tests in the 'jpacman.test' package. The report is organized into a tree view with columns for 'Element', 'Class, %', 'Method, %', and 'Line, %'. The overall coverage is very low, with most classes having 0% coverage. The 'integration' package shows 0% coverage for all its tests. The 'level' package shows 0% coverage for all its tests. The 'npc' package shows 0% coverage for all its tests. The 'sprite' package shows 0% coverage for all its tests. The 'ui' package shows 0% coverage for all its tests. The 'game' package shows 0% coverage for all its tests. The 'fuzzer' package shows 0% coverage for all its tests. The 'board' package shows 20% coverage for the 'board' class, but 0% for all other classes. The 'nl.tudelft.jpacman' package shows 3% coverage for the 'nl.tudelft.jpacman' class, but 0% for all other classes.

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
board	20% (2/11)	9% (5/53)	9% (14/141)
Board	0% (0/1)	0% (0/7)	0% (0/17)
BoardFactory	0% (0/3)	0% (0/11)	0% (0/27)
BoardFactoryTest	0% (0/1)	0% (0/6)	0% (0/18)
BoardTest	0% (0/1)	0% (0/3)	0% (0/3)
Direction	100% (1/1)	75% (3/4)	90% (10/11)
Square	0% (0/1)	0% (0/8)	0% (0/23)
SquareTest	0% (0/1)	0% (0/4)	0% (0/13)
Unit	100% (1/1)	20% (2/10)	13% (4/29)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
JPacmanFuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
Game	0% (0/1)	0% (0/7)	0% (0/24)
GameFactory	0% (0/1)	0% (0/3)	0% (0/4)
SinglePlayerGame	0% (0/1)	0% (0/4)	0% (0/9)
integration	0% (0/1)	0% (0/4)	0% (0/6)
StartupSystemTest	0% (0/1)	0% (0/4)	0% (0/6)
level	0% (0/13)	0% (0/78)	0% (0/345)
CollisionInteraction	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/1)	100% (0/0)	100% (0/0)
DefaultPlayerInteraction	0% (0/1)	0% (0/5)	0% (0/13)
Level	0% (0/2)	0% (0/17)	0% (0/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	0% (0/1)	0% (0/8)	0% (0/20)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	0% (0/1)	0% (0/3)	0% (0/4)
npc	0% (0/10)	0% (0/47)	0% (0/237)
ghost	0% (0/9)	0% (0/43)	0% (0/229)
Ghost	0% (0/1)	0% (0/4)	0% (0/8)
points	0% (0/2)	0% (0/7)	0% (0/19)
DefaultPointCalculator	0% (0/1)	0% (0/3)	0% (0/3)
PointCalculator	100% (0/1)	100% (0/0)	100% (0/0)
PointCalculatorLoader	0% (0/1)	0% (0/4)	0% (0/16)
sprite	0% (0/6)	0% (0/45)	0% (0/119)
AnimatedSprite	0% (0/1)	0% (0/11)	0% (0/38)
EmptySprite	0% (0/1)	0% (0/4)	0% (0/4)
ImageSprite	0% (0/1)	0% (0/7)	0% (0/15)
PacmanSprites	0% (0/1)	0% (0/9)	0% (0/24)
Sprite	100% (0/1)	100% (0/0)	100% (0/0)
SpriteStore	0% (0/1)	0% (0/5)	0% (0/22)
SpriteTest	0% (0/1)	0% (0/9)	0% (0/16)
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)
PacmanConfigurationEx	0% (0/1)	0% (0/2)	0% (0/4)

Task 2

In this task, I learned how to increase coverage of code by writing tests that target specific lines. I need to consider all possible inputs that a function consumes. This task also helped me learn how to write additional tests

Task 3

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

They are similar. The lines of code I've written tests for are highlighted in green. Any code that I did not write tests for are yellow.

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

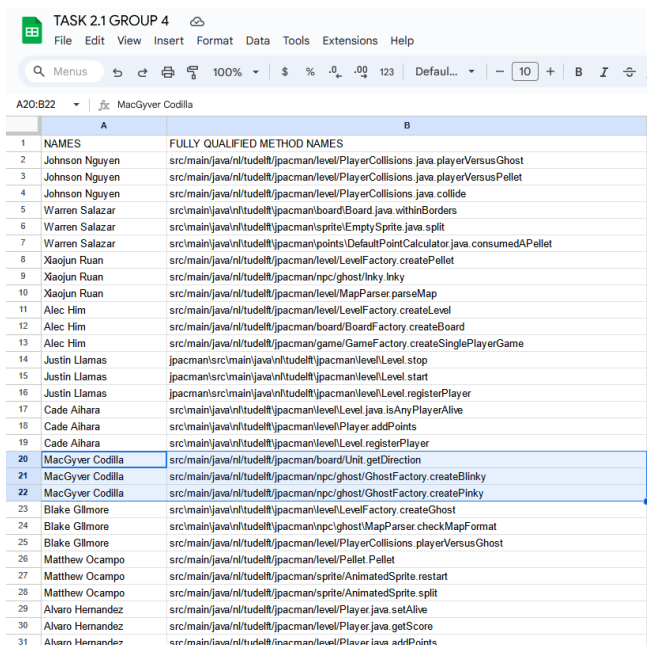
I have found it to be helpful. It would have been nice to know this existed before this task.

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

IntelliJ's coverage window only showed an approximation of the coverage as a percentage. JaCoCo showed which code was covered and which were missed. A visualization as well as a pointer as to where the coverage was missed would have been a lot more helpful than an approximation.

Code snippets:

I coordinated with my group to see which class methods have not had tests written for them. I wrote tests for these class methods.



The screenshot shows the IntelliJ IDEA interface with a table titled "FULLY QUALIFIED METHOD NAMES". The table has two columns: "A" (Names) and "B" (Fully Qualified Method Names). The table lists 31 entries, each with a line number, a name, and a fully qualified method name. The entries are as follows:

	A	B
1	NAMES	FULLY QUALIFIED METHOD NAMES
2	Johnson Nguyen	src/main/java/nl/tudelft/ipacman/level/PlayerCollisions.java.playerVersusGhost
3	Johnson Nguyen	src/main/java/nl/tudelft/ipacman/level/PlayerCollisions.java.playerVersusPellet
4	Johnson Nguyen	src/main/java/nl/tudelft/ipacman/level/PlayerCollisions.java.collide
5	Warren Salazar	src/main/java/nl/tudelft/ipacman/board/Board.java.withinBorders
6	Warren Salazar	src/main/java/nl/tudelft/ipacman/sprite/EmptySprite.java.split
7	Warren Salazar	src/main/java/nl/tudelft/ipacman/points/DefaultPointCalculator.java.consumedAPellet
8	XiaoJun Ruan	src/main/java/nl/tudelft/ipacman/level/LevelFactory.createPellet
9	XiaoJun Ruan	src/main/java/nl/tudelft/ipacman/npc/ghost/Inky.linky
10	XiaoJun Ruan	src/main/java/nl/tudelft/ipacman/level/MapParser.parseMap
11	Alec Him	src/main/java/nl/tudelft/ipacman/level/LevelFactory.createLevel
12	Alec Him	src/main/java/nl/tudelft/ipacman/board/BoardFactory.createBoard
13	Alec Him	src/main/java/nl/tudelft/ipacman/game/GameFactory.createSinglePlayerGame
14	Justin Llamas	ipacman/src/main/java/nl/tudelft/ipacman/level/Level.stop
15	Justin Llamas	ipacman/src/main/java/nl/tudelft/ipacman/level/Level.start
16	Justin Llamas	ipacman/src/main/java/nl/tudelft/ipacman/level/Level.registerPlayer
17	Cade Aihara	src/main/java/nl/tudelft/ipacman/level/Level.java.isAnyPlayerAlive
18	Cade Aihara	src/main/java/nl/tudelft/ipacman/level/Player.addPoints
19	Cade Aihara	src/main/java/nl/tudelft/ipacman/level/Level.registerPlayer
20	MacGyver Codilla	src/main/java/nl/tudelft/ipacman/board/Unit.getDirection
21	MacGyver Codilla	src/main/java/nl/tudelft/ipacman/npc/ghost/GhostFactory.createBlinky
22	MacGyver Codilla	src/main/java/nl/tudelft/ipacman/npc/ghost/GhostFactory.createPinky
23	Blake Gilmore	src/main/java/nl/tudelft/ipacman/level/LevelFactory.createGhost
24	Blake Gilmore	src/main/java/nl/tudelft/ipacman/npc/ghost/MapParser.checkMapFormat
25	Blake Gilmore	src/main/java/nl/tudelft/ipacman/level/PlayerCollisions.playerVersusGhost
26	Matthew Ocampo	src/main/java/nl/tudelft/ipacman/level/PelletPellet
27	Matthew Ocampo	src/main/java/nl/tudelft/ipacman/sprite/AnimatedSprite.restart
28	Matthew Ocampo	src/main/java/nl/tudelft/ipacman/sprite/AnimatedSprite.split
29	Alvaro Hernandez	src/main/java/nl/tudelft/ipacman/level/Player.java.setAlive
30	Alvaro Hernandez	src/main/java/nl/tudelft/ipacman/level/Player.java.getScore
31	Alvaro Hernandez	src/main/java/nl/tudelft/ipacman/level/Player.java.addPoints

```
UnitTest.java  GhostFactoryTest.java x  Pellet.java  Player.java  GhostFactoryTest.java

2
3 import nl.tudelft.jpacman.npc.Ghost;
4 import nl.tudelft.jpacman.sprite.PacManSprites;
5 import org.junit.jupiter.api.Test;
6
7 import static org.junit.jupiter.api.Assertions.assertTrue;
8
9 new *
10 public class GhostFactoryTest
11 {
12     2 usages
13     private static final PacManSprites SPRITE_STORE = new PacManSprites();
14
15     new *
16     @Test
17     public void createBlinkyTest()
18     {
19         GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);
20         Ghost blinky = ghostFactory.createBlinky();
21
22         assertTrue(blinky instanceof Blinky);
23     }
24
25     new *
26     @Test
27     public void createPinkyTest()
28     {
29         GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);
30         Ghost pinky = ghostFactory.createPinky();
31
32         assertTrue(pinky instanceof Pinky);
33     }
34 }
```

```
UnitTest.java x GhostFactoryTest.java Pellet.java Player.java GhostFactory.java
5 import nl.tudelft.jpacman.sprite.PacManSprites;
6 import org.junit.jupiter.api.Test;
7
8 import static org.junit.jupiter.api.Assertions.assertEquals;
9 import static org.junit.jupiter.api.Assertions.assertNull;
10
11 new *
12 public class UnitTest
13 {
14     1 usage
15     private static final PacManSprites SPRITE_STORE = new PacManSprites();
16
17     new *
18     @Test
19     public void testRandomMove()
20     {
21         GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);
22         Ghost inky = ghostFactory.createInky();
23
24         Direction direction = inky.getDirection();
25
26         if (direction == Direction.EAST)
27         {
28             assertEquals(direction, Direction.EAST);
29         }
30         else if (direction == Direction.NORTH)
31         {
32             assertEquals(direction, Direction.NORTH);
33         }
34         else if (direction == Direction.SOUTH)
35         {
36             assertEquals(direction, Direction.SOUTH);
37         }
38         else if (direction == Direction.WEST)
39         {
40             assertEquals(direction, Direction.WEST);
41         }
42         else
43         {
44             assertNull(direction);
45         }
46     }
47 }
```

Task 4

In task 4, I was tasked with improving the test coverage in this repository. Fortunately, the two methods to be tested are given which gave me enough working knowledge to work through the rest.

The basic idea is to request a random account and perform an operation on it using the class methods. If it calls for it, the database will be updated to reflect those changes.

```
# GIVEN 26
def test_repr(self):
    """Test the representation of an account"""
    account = Account()
    account.name = "Foo"
    self.assertEqual(str(account), "<Account 'Foo'>")

# GIVEN 30
def test_to_dict(self):
    """ Test account to dict """
    data = ACCOUNT_DATA[self.rand] # get a random account
    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"])
```

For my own test cases, I had requested a random account and performed an operation on that account. Some methods requested that an id is to be assigned, however, I could not find what that corresponded with. I was able to get a decent enough coverage at 92%. The only hiccups came from committing to the database. None of my group members could replicate the issue. I was running python 3.10.10...

```
M+ README.md  account.py  test_account.py ×
Plugins supporting *.py files found.  Install Python plugin  Ignore extension

37
38 #####
39 # TEST CASES
40 #####
41
42 # 74-75
43 def test_find(self):
44     """ Test account find """
45     idx = self.rand
46     data = ACCOUNT_DATA[idx] # get a random account
47     account = Account(**data)
48
49     result = account.Find(idx)
50     if result == None:
51         self.assertIsNone(result)
52     else:
53         d = account.to_dict()
54         self.assertEqual(account.name, d["name"])
55
56 # 52-54
57 def test_delete(self):
58     """ Test account delete """
59     idx = self.rand
60     data = ACCOUNT_DATA[idx] # get a random account
61     account = Account(**data)
62     account.delete()
63     account2 = ACCOUNT_DATA[idx]
64     assertNotEqual(account.name, account2.name)
65
66 # 45-48
67 def test_update(self):
68     """ Test account update """
69     idx = self.rand
70     data = ACCOUNT_DATA[idx] # get a random account
71     account = Account(**data)
72     result = account.to_dict()
73     self.assertEqual(account.name, result["name"])
74     account.name = "CAROL BURNETT"
75     account.id = idx
76     account.update()
77
78     idx = self.rand
79     data = ACCOUNT_DATA[idx] # get a random account
80     account = Account(**data)
81     with self.assertRaises(DataValidationError) as context:
82         account.update()
83
84
85 # 34-35
86 def test_from_dict(self):
87     """ Test account from dict """
88     data = ACCOUNT_DATA[self.rand] # get a random account
89     account = Account(**data)
90     result = account.to_dict()
91     result["name"] = "PAUL HARRELL"
92     account.from_dict(result)
93     self.assertEqual(account.name, result["name"])
94     self.assertEqual(account.email, result["email"])
95     self.assertEqual(account.phone_number, result["phone_number"])
96     self.assertEqual(account.disabled, result["disabled"])
97     self.assertEqual(account.date_joined, result["date_joined"])
98
99 # GIVEN 26
```

Task 5

This utilized a different development methodology called Test Driven Development. The task required a different frame of mind from multiple perspectives to approach the problems. Ideally someone would design the program, and a tester would take that design and write tests for it. The software developer codes the design to spec.

I've written the update method in counter.py to get the counter for the webpage. First, I fetched the counter from the list of available counters. If it did not exist, I returned an error message and status 404. I incremented the counter by 1, and returned the counter and a 200 status code. In the test, I made a call to create the counter, ensured it was successful and got the value, called the update, checked the response's status code, and then asserted that the counter was one more than the previous.

This task exposed me to different perspectives, and got me familiar with the thought process behind this methodology.

Link: <https://github.com/39otsu/cs472-group4/tree/main/jpacman>

M+ README.md
status.py
setup.cfg
test_counter.py
counter.py

i Plugins supporting *.py files found.
Install Python plugin
Ignore extension

```

1  """
2  Test Cases for Counter Web Service
3
4  Create a service that can keep a track of multiple counters
5  - API must be RESTful - see the status.py file. Following these guidelines, you can make ass
6  how to call the web service and assert what it should return.
7  - The endpoint should be called /counters
8  - When creating a counter, you must specify the name in the path.
9  - Duplicate names must return a conflict error code.
10 - The service must be able to update a counter by name.
11 - The service must be able to read the counter
12 """
13 from unittest import TestCase
14
15 # we need to import the unit under test - counter
16 from src.counter import app
17
18 # we need to import the file that contains the status codes
19 from src import status
20
21 class CounterTest(TestCase):
22     """Counter tests"""
23     def setUp(self):
24         self.client = app.test_client()
25
26     def test_create_a_counter(self):
27         """It should create a counter"""
28         result = self.client.post('/counters/foo')
29         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
30
31     def test_duplicate_a_counter(self):
32         """It should return an error for duplicates"""
33         result = self.client.post('/counters/bar')
34         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
35         result = self.client.post('/counters/bar')
36         self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
37
38     def test_update_a_counter(self):
39         """It should update the counter"""
40         result = self.client.post('/counters/cherry')
41         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
42         firstTimeCounter = result.json['cherry']
43         result = self.client.put('/counters/cherry')
44         secondTimeCounter = result.json['cherry']
45         self.assertEqual(result.status_code, status.HTTP_200_OK)
46         self.assertEqual(secondTimeCounter, firstTimeCounter + 1)
47         result = self.client.put('/counters/cherries')
48         self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)

```


README.md × status.py setup.cfg test_counter.py counter.py ×

Plugins supporting *.py files found.

Install Python plugin

Ignore extension

```
1 from flask import Flask
2 import status
3 app = Flask(__name__)
4
5 COUNTERS = {}
6
7 # We will use the app decorator and create a route called slash counters.
8 # specify the variable in route <name>
9 # let Flask know that the only methods that is allowed to called
10 # on this function is "POST".
11 @app.route('/counters/<name>', methods=['POST'])
12 def create_counter(name):
13     """Create a counter"""
14     app.logger.info(f"Request to create counter: {name}")
15     global COUNTERS
16     if name in COUNTERS:
17         return {"Message": f"Counter {name} already exists"}, status.HTTP_409_CONFLICT
18     COUNTERS[name] = 0
19     return {name: COUNTERS[name]}, status.HTTP_201_CREATED
20
21 @app.route('/counters/<name>', methods=['PUT'])
22 def update_counter(name):
23     """update a counter"""
24     app.logger.info(f"Request to update counter: {name}")
25     global COUNTERS
26     if name not in COUNTERS:
27         return {"Message": f"Counter {name} does not exist."}, status.HTTP_404_NOT_FOUND
28     COUNTERS[name] = COUNTERS[name] + 1
29     return {name: COUNTERS[name]}, status.HTTP_200_OK
```