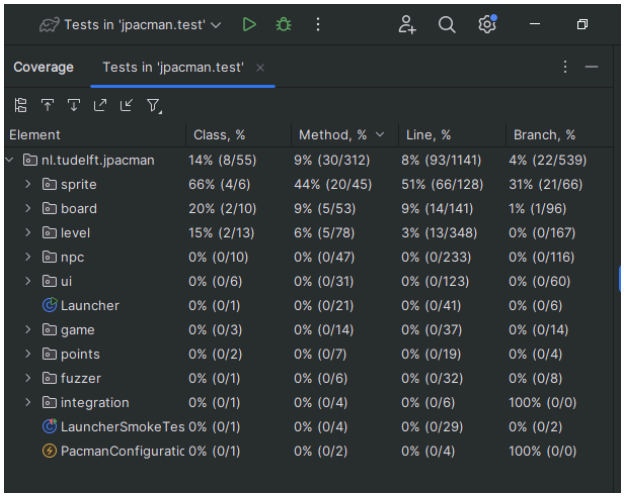
testSetAlive()



Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	10% (32/312)	9% (104/1141)	5% (27/539)
sprite	66% (4/6)	46% (21/45)	54% (70/128)	33% (22/66)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
level	15% (2/13)	7% (6/78)	5% (20/348)	2% (4/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
LauncherSmokeTes	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfiguratic	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

testScore()

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	10% (32/312)	9% (104/1141)	5% (27/539)
sprite	66% (4/6)	46% (21/45)	54% (70/128)	33% (22/66)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
level	15% (2/13)	7% (6/78)	5% (20/348)	2% (4/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
LauncherSmokeTes	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfiguratic	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	10% (33/312)	9% (105/1141)	5% (27/539)
sprite	66% (4/6)	46% (21/45)	54% (70/128)	33% (22/66)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
level	15% (2/13)	8% (7/78)	6% (21/348)	2% (4/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
LauncherSmokeTes	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfiguratio	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

testSetScore()

Element	Class, %	Method...	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	10% (33/312)	9% (105/1141)	5% (27/539)
sprite	66% (4/6)	46% (21/45)	54% (70/128)	33% (22/66)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
level	15% (2/13)	8% (7/78)	6% (21/348)	2% (4/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
LauncherSmokeTe	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurat	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

Element	Class, %	Method...	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	10% (34/312)	9% (107/1141)	5% (27/539)
sprite	66% (4/6)	46% (21/45)	54% (70/128)	33% (22/66)
level	15% (2/13)	10% (8/78)	6% (23/348)	2% (4/167)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
LauncherSmokeTe	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurat	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

```
19      @Test  David
20      void testSetAlive() {
21          player.setAlive(true);
22          assertThat(player.isAlive()).isEqualTo( expected: true);
23
24          player.setAlive(false);
25          assertThat(player.isAlive()).isEqualTo( expected: false);
26      }
27
28      @Test  David
29      void testScore() {
30          // Upon initilization, score should be 0
31          assertThat(player.getScore()).isEqualTo( expected: 0);
32      }
33
34      @Test  David
35      void testSetScore() {
36          int points = player.getScore();
37          player.addPoints(1);
38
39          assertThat(player.getScore()).isEqualTo( expected: points + 1);
40      }
41  }
```

Method testing functions above.

- Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?
    - JaCoCo and the InteliJ coverage test are very similar however, it appears that InteliJ recognized a few files missing testing that JaCoCo did not. Notable, these files appear to only be test files; for example, within the “~jpacman.sprite” JaCoCo has all except for “SpriteTest.” It can be said that either JaCoCo is inferior to IntelliJ for not recognizing this file or that it is smarter because it recognized that this is a test file.

- Did you find helpful the source code visualization from JaCoCo on uncovered branches?
    - It was very nice to see the red and green bar as the test filled up; however, InteliJ’s UI is not bad either. They are both very helpful in conveying information with no real preference for one or the other.

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

- For mere convenience, I believe InteliJ's coverage window is more convenient because the window is within the same screen. It may be a little more difficult to see some information like the exact count of files counted for in testing and total files but it is more compact.

Task 4

The test for the account would ensure that each added account correctly displays the information used to create it. Notable, for the "from a dictionary" function, we first create a dictionary with the following information, create the account, and then check to see if what we actually want was used to create it.

```
def test_from_dict():
    rand = randrange(0, len(ACCOUNT_DATA))      # Generate a random index
    data = ACCOUNT_DATA[rand]       # Get a random account
    account = Account(**data)

    now = datetime.datetime.now()

    dict = {
        'name' : 'account_Name',
        'email': 'account_Email',
        'phone_number' : '7021234567',
        'disabled' : True,
        'date_joined' : now
    }

    account.from_dict(dict)
    account.create()

    assert account.name == 'account_Name'
    assert account.email == 'account_Email'
    assert account.phone_number == '7021234567'
    assert account.disabled == True
    assert account.date_joined == now


def test_update():
    rand = randrange(0, len(ACCOUNT_DATA))      # Generate a random index
    data = ACCOUNT_DATA[rand]       # Get a random account
    account = Account(**data)

    # Test exception with empty id
    try:
        account.update()
    except DataValidationError as error:
        assert str(error) == "Update called with empty ID field"

    # Get a random id and do an update???
    account.id = 59498
    account.name = "new name"
```

It is worth mentioning that for each variable, a specific datatype is enforced such as string for name, bool for disabled, and datetime for date_joined. The dictionary reflects this and the function checks to see if these data types are being used.

Currently, due to some issues with account ID, the coverage for this portion is only at 98%. The issue with the ID being, the Account.all() and Account.find(any_Num) are unable to produce any accounts at all. Making it very difficult to test some functions as it relies on an ID that simply does not exist.

Task 5

Within this section, we are asked to use the custom endpoints, use HTTP methods, and follow REST guidelines to create, test, and ensure the full functionality of the counter.py file. A create, update, and get counter function is created and does their respective task, the test code for each is as follows:

```
31      def test_create_a_counter(self, client):
32          """It should create a counter"""
33          result = client.post('/counters/bar')
34          assert result.status_code == status.HTTP_201_CREATED
35
36          result = client.post('/counters/bar')
37          assert result.status_code == status.HTTP_409_CONFLICT
38
39      def test_update_a_counter(self, client):
40          # Create a new counter
41          result = client.post('/counters/count')
42          assert result.status_code == status.HTTP_201_CREATED
43
44          # Value returned by the endpoint is a byte. We must parse this into
45          # a dictionary and attempt to retrieve usable data from it
46          # ast will do this conversion into a dicitonary and then
47          # grab the only key in it for the value
48          count = ast.literal_eval(result.data.decode('UTF-8'))['count']
49
50          # Check for OK code from post
51          result = client.put('/counters/count')
52          assert result.status_code == status.HTTP_200_OK
53
54          # Ensure the value was incremented from before
55          result = ast.literal_eval(result.data.decode('UTF-8'))
56          assert count + 1 == result['count']
57
58          # Assert on attempting to update non-existing counters
59          result = client.put('/counters/unknownCounter')
60          assert result.status_code == status.HTTP_404_NOT_FOUND
61
62      def test_retrieve_counter(self, client):
63          result = client.get('/counters/unKnownCounter')
64          assert result.status_code == status.HTTP_404_NOT_FOUND
65
66          result = client.get('/counters/count')
67          assert result.status_code == status.HTTP_200_OK
68
69          # This one must be 1 because it was incremented previously
70          content = ast.literal_eval(result.data.decode('UTF-8'))
71          assert 1 == content['count']
72
```

Each function would test the and check for expected output from the counter.py with some duplications like the post function being called and checked in multiple places. The most notable

part of this code is on line 48. This part is necessary because, for unknown reasons, the endpoint returns a byte time despite being given a Dictionary type in the code. Thus, we must parse this byte type into more usable information and preferably back into a Dictionary type. The ast will do exactly that and once we have our information back, we can check to see if the value is being correctly modified according to their respective function.