Johnson Nguyen
CS 472
Dr. Businge
https://github.com/nguyenj21/cs472-group4

Unit Testing Lab Report

Executive Summary 2.1 & 3:

The project code contains JUnit tests for the PlayerCollisions class, examining player-ghost, player-pellet, and generic collisions. Using Mockito, mock instances of Player, Ghost, and Pellet are created. The setup method initializes the test environment by mocking the PointCalculator class and creating a PlayerCollisions instance. Tests validate expected behaviors, such as interactions with PointCalculator, player state changes, and collisions involving the player, ghosts, and pellets.

There were minor discrepancies in covered branches between JaCoCo and IntelliJ with a preference for JaCoCo due to its detailed information and visual appeal. Despite the preference, both tools are recognized for their value, and the approach showcases a comprehensive testing strategy with attention to functionality and coverage metrics.

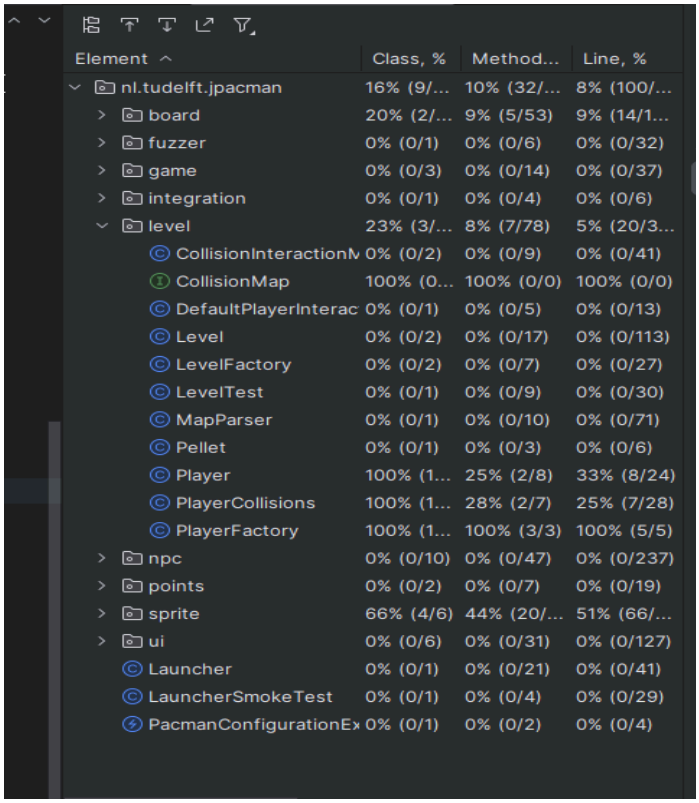
Executive Summary 4 & 5:

Red Phase:
The test is initiated to ensure the update functionality of a counter. It starts by creating a counter and expects a 201_CREATED status code. The baseline counter value is recorded for later comparison.

Green Phase:
The code is implemented to handle counter updates. A PUT route on /counters/<name> is created, incrementing the counter by 1. The test is rerun, now expecting a 200_OK status during the update. The updated counter value is verified to be one more than the baseline.

Refactor Phase:
The existing code structure, including duplicate check during counter creation, is found suitable. No major changes are needed besides checking to see if it already exists. The code ensures correct counter creation and updates while adhering to REST API guidelines.

Code Snippet 1

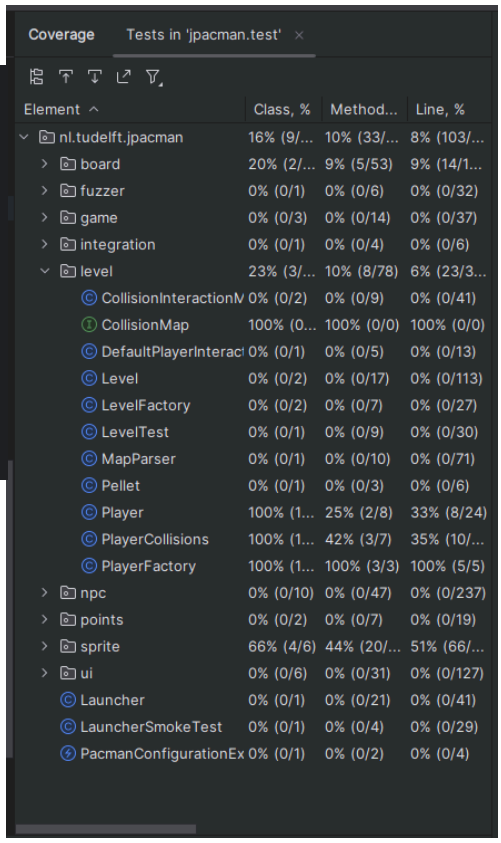


Element	Class, %	Method...	Line, %
nl.tudelft.jpacman	16% (9/...	10% (32/...	8% (100/...
board	20% (2/...	9% (5/53)	9% (14/1...
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	23% (3/...	8% (7/78)	5% (20/3...
CollisionInteractionM	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0...	100% (0/0)	100% (0/0)
DefaultPlayerInterac	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% (0/6)
Player	100% (1...	25% (2/8)	33% (8/24)
PlayerCollisions	100% (1...	28% (2/7)	25% (7/28)
PlayerFactory	100% (1...	100% (3/3)	100% (5/5)
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/...	51% (66/...
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)

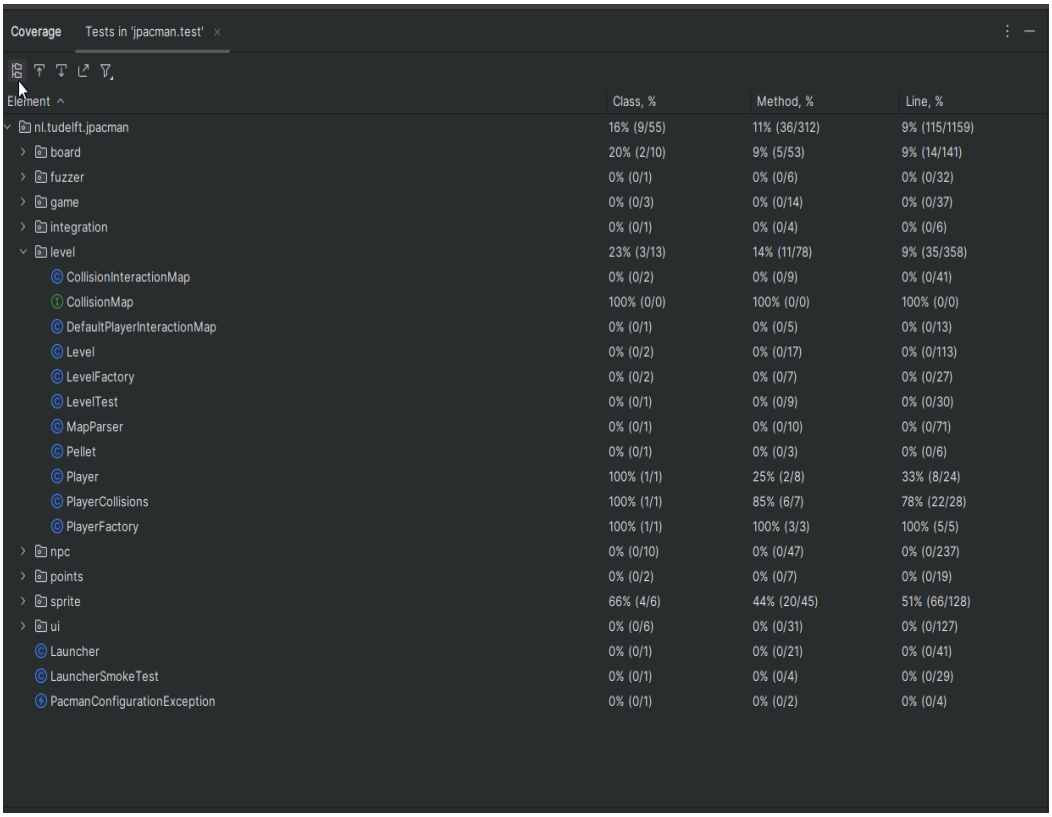
```java
new *
public class PlayerCollisionsTest {

    5 usages
    private PlayerCollisions playerCollisions;
    4 usages
    private PointCalculator pointCalculator;

    new *
    @BeforeEach
    void setup(){
        pointCalculator = mock(PointCalculator.class);
        playerCollisions = new PlayerCollisions((pointCalculator));
    }
    new *
    @Test
    void playerVersusGhostTest(){
        Player player = mock(Player.class);
        Ghost ghost = mock(Ghost.class);

        playerCollisions.playerVersusGhost(player,ghost);

        verify(pointCalculator).collidedWithAGhost(player, ghost);
        verify(player).setAlive(false);
        verify(player).setKiller(ghost);



    }
```

Code Snippet 2

```
new *
@Test
void playerVersusPelletTest(){
    Player player = mock(Player.class);
    Pellet pellet = mock(Pellet.class);
    when(pellet.getValue()).thenReturn( value: 0);

    playerCollisions.playerVersusPellet(player,pellet);

    verify(pointCalculator).consumedAPellet(player, pellet);


}
```



Code Snippet 3




```
new *
@Test
void collideTest() {
    Player player = mock(Player.class);
    Ghost ghost = mock(Ghost.class);
    //Pellet pellet = mock(Pellet.class);

    playerCollisions.collide(player,ghost);
    playerCollisions.collide(ghost,player);


    verify(player,times( wantedNumberOfInvocations: 2)).setAlive(false);
    verify(player,times( wantedNumberOfInvocations: 2)).setKiller(ghost);


    //playerCollisions.collide(player,pellet);
    //verify(player).addPoints(anyInt());
    //verify(pellet).leaveSquare();


}
```

Task 4

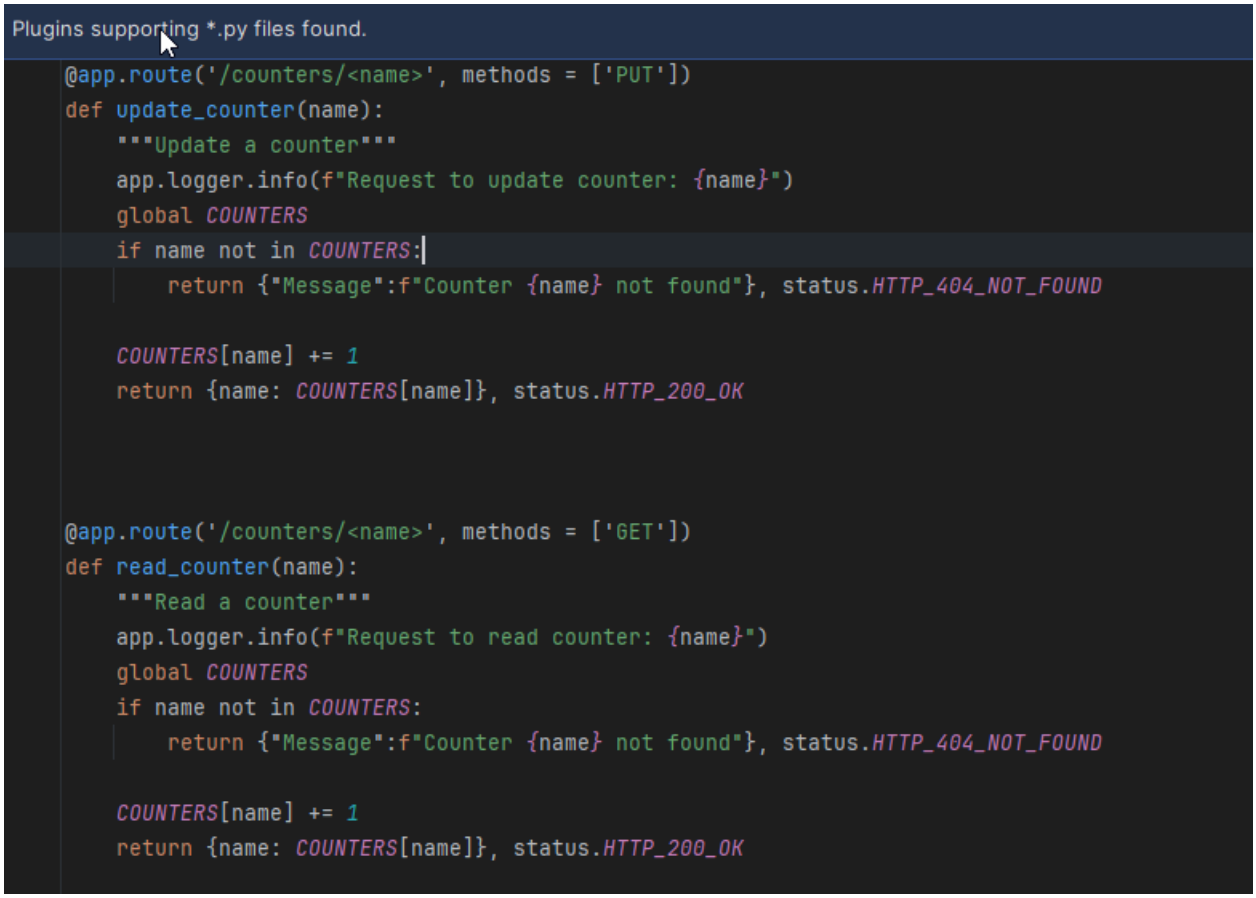
```python
def test_from_dict(self):
    """ Test account from dict """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account()
    account.from_dict(data)
    self.assertEqual(account.name, data.get("name"))
    self.assertEqual(account.email, data.get("email"))
    self.assertEqual(account.phone_number, data.get("phone_number"))
    self.assertEqual(account.disabled, data.get("disabled"))
    self.assertEqual(account.date_joined, data.get("date_joined"))

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

    account.name = "Updated Test"
    try:
        account.update()
    except DataValidationError as e:
        self.fail(f"Update failed with DataValidationError: {e}")

    retrieveAccount = Account.find(account.id)

    self.assertIsNotNone(retrieveAccount)
    self.assertEqual(account.name, retrieveAccount.name)
    self.assertEqual(account.email, retrieveAccount.email)
    self.assertEqual(account.disabled, retrieveAccount.disabled)
    self.assertEqual(account.phone_number, retrieveAccount.phone_number)
```

```python
def test_delete(self):
    """ Test account deletion """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()

    account.delete()

    retrieveAccount = Account.find(account.id)

    self.assertIsNone(retrieveAccount)


def test_find(self):
    """ Test account find """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()

    retrieveAccount = Account.find(account.id)

    self.assertIsNotNone(retrieveAccount)
    self.assertEqual(account.name, retrieveAccount.name)
    self.assertEqual(account.email, retrieveAccount.email)
    self.assertEqual(account.disabled, retrieveAccount.disabled)
    self.assertEqual(account.phone_number, retrieveAccount.phone_number)
```

Task 5

```python
def test_update_a_counter(self):
    # creating a counter
    result = self.client.post('/counters/update')
    # ensure returned successful return code
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    # check counter value as baseline
   # result = self.client.get('/counters/update')
    # make a call to update counter we just created
    baselineValue = result.get_json()['update']
    result = self.client.put('/counters/empty')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
    # ensure returned successful return code
    # check counter is one more than baseline we measured in check counter value as baseline
    result = self.client.put('/counters/update')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    updateValue = result.get_json()['update']
    self.assertEqual(baselineValue + 1,updateValue)
```

```python
def test_read_counter(self):
    """Read a counter"""
    # creating a counter
    result = self.client.post('/counters/read')
    # ensure returned successful return code
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    # read counter
    result = self.client.get('/counters/read')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    counterValue = result.get_json()['read']
   # self.assertEqual(baselineValue + 1,updateValue)
    result = self.client.get('/counters/empty')
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
```


Plugins supporting *.py files found.
@app.route('/counters/<name>', methods = ['PUT'])
def update_counter(name):
"""Update a counter"""
app.logger.info(f"Request to update counter: {name}")
global COUNTERS
if name not in COUNTERS:
return {"Message":f"Counter {name} not found"}, status.HTTP_404_NOT_FOUND
COUNTERS[name] += 1
return {name: COUNTERS[name]}, status.HTTP_200_OK
@app.route('/counters/<name>', methods = ['GET'])
def read_counter(name):
"""Read a counter"""
app.logger.info(f"Request to read counter: {name}")
global COUNTERS
if name not in COUNTERS:
return {"Message":f"Counter {name} not found"}, status.HTTP_404_NOT_FOUND
COUNTERS[name] += 1
return {name: COUNTERS[name]}, status.HTTP_200_OK