Task 2.1 - Tests Files Created

BFactory = new BoardFactory(sprites);
var square1 = new BasicSquare();
var square2 = new BasicSquare();



```
assertTrue(board.withinBorders( x: 0, y: 0));
assertTrue(board.withinBorders( x: board.getWidth()-1, y: board.getHeight()-1));

@Test
void TestSetDirection(){

    /*
    *Set the player direction to North
    * Check if Player direction is North
    */

    Direction north = Direction.valueOf( name: "NORTH");
    ThePlayer.setDirection(north);
```

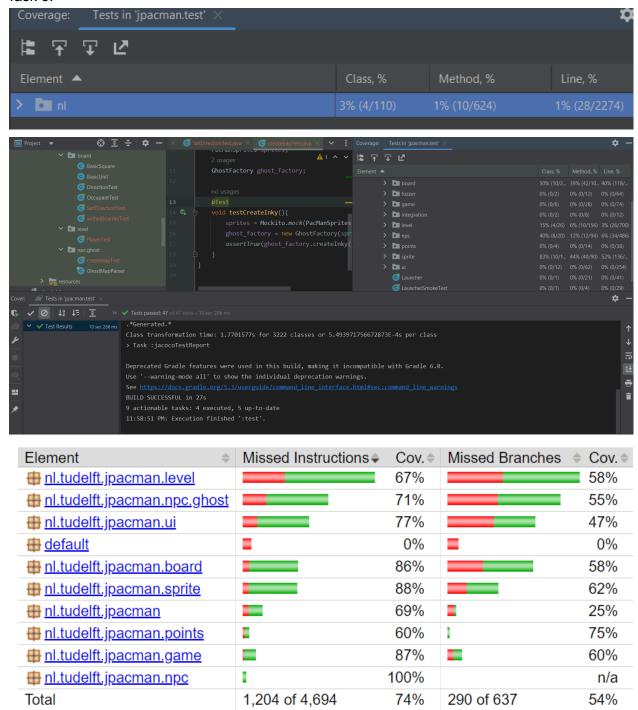
var board = BFactory.createBoard(new Square[][]{{square1}, {square2}});

```
public class createInkyTest {
    2 usages
    PacManSprites sprites;
    2 usages
    GhostFactory ghost_factory;

    no usages new *
    @Test
    void testCreateInky(){
        sprites = Mockito.mock(PacManSprites.class);
        ghost_factory = new GhostFactory(sprites);
        assertTrue(ghost_factory.createInky() instanceof Inky);
}
```

assertThat(ThePlayer.getDirection()).isEqualTo(Direction.NORTH);

Task 3:



The coverage results between intelliJ and JaCoCo are vastly different. You can see that the JaCoCo tracks bytecode instruction and statement branches, while intelliJ tracks each line. I find that JaCoCo's visualization is very helpful because it shows which statements have been covered and which have not been tested. I favor JaCoCo's coverage because it is a more specific version of the intelliJ. I like the feature that highlights statements as well.

## Task 4:

```
def test_from_dict(self):
    """ Test account dict"""
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.from_dict(data)
    result = account.to_dict()
    self.assertEqual(account.name, result["name"])

new *

def test_update(self):
    """ Test account update"""
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    with self.assertRaises(models.account.DataValidationError):
        account.update()
        account.create()
        account.update()
        result = account.to_dict()
        self.assertEqual(account.name, result["name"])
```

```
def test_delete(self):
    """ Test account deletion """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()
    account.delete()
    self.assertEqual(len(Account.all()), second: 0)

new *

def test_find(self):
    """ Test finding"""
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()
    self.assertEqual(Account.find(account.id), account)
```

Name	Stmts	Miss	Cover	Missing
models\initpy	7	0	100%	
models\account.py	40	Θ	100%	
TOTAL	47	0	100%	
Ran 8 tests in 1.467	S			

Task 5: Added test for POST and PUT.

```
def test_update_a_counter(self):
    """Test update counter. POST, PUT, GET"""

# Create Counter doo
    self.setUp()
    result = self.client.post('/counters/doo')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    self.assertEqual(COUNTERS.get('doo'), second: 0)

#Increment doo by 1
    result = self.client.put('/counters/doo')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(COUNTERS.get('doo'), second: 1)
```

## Added PUT

```
new *

@app.route( rule: '/counters/<name>', methods=['PUT'])

def update_counter(name):
    """Update a counter"""

app.logger.info(f"Request to update counter: {name}")

global COUNTERS

COUNTERS[name] = COUNTERS[name] + 1

return {name: COUNTERS[name]}, status.HTTP_200_0K
```

## Added GET and test for GET

```
@app.route( rule: '/counters/<name>', methods=['GET'])

def read_counter(name):
    """Read a counter"""
    app.logger.info(f"Request to read counter: {name}")
    global COUNTERS
    return {name: COUNTERS[name]}, status.HTTP_200_0K
#Read doo
```

```
#Read doo
result = self.client.get('/counters/doo')
self.assertEqual(result.status_code, status.HTTP_200_0K)
self.assertEqual(COUNTERS.get('doo'), second: 1)
```