Alexander Lauderdale
CS 472 1001
5 Feb 2024

Unit Testing Report

Link to fork repository: https://github.com/axelauda/SeniorDesign

Task 2.1:

Method 1: BoardFactory.createBoard()

Location: src/main/java/nl/tudelft/jpacman/board/BoardFactory

Test File: src/test/java/nl/tudelft/jpacman/board/CreateBoardTest.java

To test when a board is created, I needed to import a board factory that would create the board, while also initializing a 2D Square array called grid, filling it with a couple Square objects, and passing grid into the createBoard() method.

My test was for the proportions of the grid (its width and height) to match the proportions of the board created from it. To do that, I had two assert statements: The first would check if the board's width was the same as the grid's width. The second would check the height for both. If they are equal, then the board was created properly.

When the board is created, it also sets each square to know its neighboring squares, to the north, south, east, and west. To check this, I used the board.squareAt() function to get a square, and then the squares in each of the four cardinal directions, and then I check that with the middle square, that its neighbors (obtained by doing square.getSquareAt(Direction)) are the same as the the actual squares that it's surrounded by.

Alexander Lauderdale
CS 472 1001
5 Feb 2024

CreateBoardTest.java

```java
package nl.tudelft.jpacman.board;

import ...

public class CreateBoardTest {
    1 usage
    private final int width = 4;
    1 usage
    private final int height = 3;
    10 usages
    Square[][] grid = new Square[width][height];
    1 usage
    private final PacManSprites sprites = new PacManSprites();
    2 usages
    private final BoardFactory factory = new BoardFactory(sprites);

    /**
     * Is the width and height of the board the same as the grid it was created from?
     */
    @Test
    void checkBoardProportions() {
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[0].length; j++) {
                grid[i][j] = new BasicSquare();
            }
        }
        Board board = factory.createBoard(grid);
        assertThat(board.getWidth()).isEqualTo(grid.length);
        assertThat(board.getHeight()).isEqualTo(grid[0].length);
    }
```

```java
    /**
     * Are the neighbors of squares on the board linked properly?
     *   We check this by matching the correct neighbors of a square from
     *   the board to the stored neighbors map of that square.
     */
    @Test
    void checkNeighbors() {
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[0].length; j++) {
                grid[i][j] = new BasicSquare();
            }
        }
        Board board = factory.createBoard(grid);
        Square current = board.squareAt( x: 1,  y: 1);
        Square east = board.squareAt( x: 2,  y: 1);
        Square west = board.squareAt( x: 0,  y: 1);
        Square north = board.squareAt( x: 1,  y: 0);
        Square south = board.squareAt( x: 1,  y: 2);
        assertThat(current.getSquareAt(Direction.EAST)).isEqualTo(east);
        assertThat(current.getSquareAt(Direction.WEST)).isEqualTo(west);
        assertThat(current.getSquareAt(Direction.NORTH)).isEqualTo(north);
        assertThat(current.getSquareAt(Direction.SOUTH)).isEqualTo(south);
    }
}
```

Alexander Lauderdale
CS 472 1001
5 Feb 2024

Method 2: Unit.leaveSquare()

Location: src/main/java/nl/tudelft/jpacman/board/Unit

Test File: src/test/java/nl/tudelft/jpacman/board/UnitLeaveSquareTest.java

For testing this method, I needed to check that when a unit leaves a square that it has actually left the square. To start, I began by placing a unit on a square using unit.occupy(). I then checked on the square's end if it's being occupied by the unit with sq.getOccupants(). Since .getOccupants() returns a list of units, and there is one unit currently occupying, then it should be the case that the .isEmpty() of that list is false, which I check with an assert.

After that, I called unit.leaveSquare(), which should remove the unit from the square's occupant list. I then checked using .isEmpty() again, since when I used leaveSquare(), the occupant list should now be empty since it only contained one unit. Therefore, if .isEmpty() was true, then the test passed.

Alexander Lauderdale
CS 472 1001
5 Feb 2024

UnitLeaveSquareTest.java

```java
package nl.tudelft.jpacman.board;

import static org.assertj.core.api.Assertions.assertThat;
import java.util.List;

import org.junit.jupiter.api.Test;

public class UnitLeaveSquareTest {
    3 usages
    Square sq = new BasicSquare();
    2 usages
    Unit unit = new BasicUnit();

    /**
     * Check that when a unit leaves a square,
     * that the square isn't still being occupied by the unit.
     */
    @Test
    void leftSquare() {
        // Have a unit occupy a square with no occupants,
        // and check if the square is being occupied.
        unit.occupy(sq);
        List<Unit> occupants = sq.getOccupants();
        assertThat(occupants.isEmpty()).isEqualTo( expected: false);
        // Have a unit leave the square, and check if the square has no occupants now.
        unit.leaveSquare();
        occupants = sq.getOccupants();
        assertThat(occupants.isEmpty()).isEqualTo( expected: true);
    }
}
```

Alexander Lauderdale
CS 472 1001
5 Feb 2024

Method 3: PlayerCollisions.PlayerVersusGhost()

Location: src/main/java/nl/tudelft/jpacman/level/PlayerCollisions

Test File: src/test/java/nl/tudelft/jpacman/level/PlayerVersusGhostTest.java

For testing .PlayerVersusGhost() in the PlayerCollisions class, there were three areas I needed to check for the test to pass. That is, when a player and ghost collide:

(1). The score of the player stays the same.

(2). The player is killed.

(3). The player was killed by the ghost and not some other unit.

To check (1), since I had just created Player, its score should have been 0, so as long as after the player and ghost collide the Player.getScore() stays 0, then it passes.

To check (2), I just called Player.isAlive() and checked if it's false, meaning the player is dead.

To check (3), if Player.getKiller() is the ghost that it collided with, then it passes the last check.

Alexander Lauderdale
CS 472 1001
5 Feb 2024

PlayerVersusGhostTest.java

```java
package nl.tudelft.jpacman.level;
import ...

public class PlayerVersusGhostTest {
    2 usages
    private final PacManSprites sprites = new PacManSprites();
    1 usage
    private final PlayerFactory playerFactory = new PlayerFactory(sprites);
    1 usage
    private final GhostFactory ghostFactory = new GhostFactory(sprites);
    1 usage
    private final PointCalculator calculator = new DefaultPointCalculator();
    1 usage
    private final PlayerCollisions collisions = new PlayerCollisions(calculator);
    /*
    Check that when a player collides with a ghost, that:
        1). Their score stays the same.
        2). The player dies.
        3). The player was killed by the ghost.
     */
    @Test
    void testPlayer() {
        Player player = playerFactory.createPacMan();
        Ghost ghost = ghostFactory.createBlinky();

        collisions.playerVersusGhost(player, ghost);
        assertThat(player.getScore()).isEqualTo( expected: 0);
        assertThat(player.isAlive()).isEqualTo( expected: false);
        assertThat(player.getKiller()).isEqualTo(ghost);
    }
}
```

Alexander Lauderdale
CS 472 1001
5 Feb 2024

Coverage:

Initial:

Coverage - jpacman

Coverage | jpacman [test]

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (93/1151)
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)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	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	100% (1/1)	25% (2/8)	33% (8/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	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% (20/45)	51% (66/128)
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)

Alexander Lauderdale
CS 472 1001
5 Feb 2024

<u>After Method 1:</u>

Coverage - jpacman

Coverage | jpacman [test]

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	20% (11/55)	14% (46/312)	12% (140/115...
board	50% (5/10)	39% (21/53)	42% (61/144)
Board	100% (1/1)	100% (7/7)	94% (17/18)
BoardFactory	33% (1/3)	27% (3/11)	58% (17/29)
BoardFactoryTest	0% (0/1)	0% (0/6)	0% (0/18)
BoardTest	0% (0/1)	0% (0/3)	0% (0/3)
Direction	100% (1/1)	100% (4/4)	100% (11/11)
Square	100% (1/1)	62% (5/8)	52% (12/23)
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)
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	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)

Alexander Lauderdale
CS 472 1001
5 Feb 2024

After Method 2:

Coverage - jpacman

Coverage | jpacman [test]

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	20% (11/55)	16% (52/312)	13% (160/115...
board	50% (5/10)	50% (27/53)	56% (81/144)
Board	100% (1/1)	100% (7/7)	94% (17/18)
BoardFactory	33% (1/3)	27% (3/11)	58% (17/29)
BoardFactoryTest	0% (0/1)	0% (0/6)	0% (0/18)
BoardTest	0% (0/1)	0% (0/3)	0% (0/3)
Direction	100% (1/1)	100% (4/4)	100% (11/11)
Square	100% (1/1)	100% (8/8)	86% (20/23)
SquareTest	0% (0/1)	0% (0/4)	0% (0/13)
Unit	100% (1/1)	50% (5/10)	55% (16/29)
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	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)

Alexander Lauderdale
CS 472 1001
5 Feb 2024

After Method 3:

Coverage - jpacman

Coverage | jpacman [test]

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	30% (17/55)	21% (67/312)	17% (201/116...
board	50% (5/10)	50% (27/53)	56% (81/144)
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% (3/13)	14% (11/78)	8% (29/357)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	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	100% (1/1)	75% (6/8)	70% (17/24)
PlayerCollisions	100% (1/1)	28% (2/7)	25% (7/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	40% (4/10)	12% (6/47)	6% (17/243)
points	50% (1/2)	14% (1/7)	5% (1/20)
sprite	66% (4/6)	48% (22/45)	57% (73/128)
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)

Alexander Lauderdale
CS 472 1001
5 Feb 2024

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?

No, the coverage results are sometimes more or less than the coverage of my own tests. For example, the coverage of Board on JaCoCo is 81%, while IntelliJ says that I have 100% class and method coverage. However, I missed one line on JaCoCo in Board, which does correspond and equal the 94% (17/18) line coverage in IntelliJ. Another example is BoardFactory, which in JaCoCo has 94% coverage, but in IntelliJ I have 33% class coverage and 27% method coverage. I believe that JaCoCo is more accurate and determines coverage line by line to provide a total coverage, while in IntelliJ it checks if at least one line has been tested in a class or method to determine if it's been "covered." I also believe that a covered line has a different definition in JaCoCo compared to IntelliJ, with JaCoCo being more lenient on what's considered a line covered, which explains the differing results from time to time.

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

Yes, the way that the code section that isn't being covered is highlighted in yellow makes it easy to see where tests should be made to fix the coverage of the uncovered branch.

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

I prefer JaCoCo's report over IntelliJ's, since JaCoCo highlights code for you to see where coverage is or isn't being done. It also includes branch coverage, which IntelliJ doesn't have, and JaCoCo tells you how many Methods and Lines you've missed which is a bit clearer to me than the fraction and percentage representation in IntelliJ.

Alexander Lauderdale
CS 472 1001
5 Feb 2024

Task 4:

```
(venv) C:\Users\alexl\test_coverage>nosetests -s

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test account deletion
- Test account from dict
- Test the representation of an account
- Test account to dict
- Test updating an existing Account
- Test updating an account with an empty ID

Name                    Stmts   Miss  Cover   Missing
-------------------------------------------------------
models\__init__.py          7      0   100%
models\account.py          40      0   100%
-------------------------------------------------------
TOTAL                      47      0   100%
----------------------------------------------------------
Ran 8 tests in 1.232s

OK
```

I achieved 100% coverage after implementing the following tests:

```python
def test_from_dict(self):
    """ Test account from dict """
    data = ACCOUNT_DATA[self.rand]  # get a random account

    # Create an account from data dictionary
    account = Account()
    account.from_dict(data)

    # Check that attributes are the same as dictionary
    self.assertEqual(account.name, data["name"])
    self.assertEqual(account.email, data["email"])
    self.assertEqual(account.phone_number, data["phone_number"])
    self.assertEqual(account.disabled, data["disabled"])
```

Alexander Lauderdale
CS 472 1001
5 Feb 2024

```python
def test_update_account(self):
    """ Test updating an existing Account """
    # Create an account
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()

    # Update the account's attributes
    updated_name = "Updated Name"
    account.name = updated_name
    account.update()

    # Retrieve the updated account from the database
    updated_account = Account.find(account.id)

    # Check that the attributes were updated
    self.assertEqual(updated_account.name, updated_name)

def test_update_account_with_empty_id(self):
    """ Test updating an account with an empty ID """
    # Create an account without saving it to the database
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)

    # Attempt to update the account with an empty ID
    with self.assertRaises(DataValidationError):
        account.update()

    # Ensure that the account was not saved to the database
    self.assertEqual(len(Account.all()), 0)
```

```python
def test_delete(self):
    """ Test account deletion """
    # Create an account and save it to the database
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()

    # Delete it and check if it was removed from the database
    account.delete()
    self.assertEqual(len(Account.all()), 0)
```

Alexander Lauderdale
CS 472 1001
5 Feb 2024

Task 5:

test_update_a_counter(self):

```
def test_update_a_counter(self):
    result = self.client.post('/counters/baz')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    self.assertEqual(COUNTERS['baz'], 0)

    result = self.client.put('/counters/baz')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    self.assertEqual(COUNTERS['baz'], 1)
```

In the update counter test, I first created a new counter, and checked if the counter was initialized to 0. When I ran nosetests, it was GREEN, since creating a counter was previously tested and properly coded.

After creating the counter, I called the put method, which in counter.py would increment the counter by 1. I then checked if the status was OK, and that the counter was equal to 1. This gave me a RED in nosetests, and an AssertionError: 405 != 200, meaning that the return status of the put method wasn't OK but METHOD_NOT_ALLOWED.

This was because I now needed to implement update_counter() in counter.py:

```
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    global COUNTERS

    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

In update_counter(), I REFACTORED the code by incrementing the count of an existing counter by 1, then returning the name and the status OK.

Now when I ran the nosetests, I got GREEN, since now the previous assertion that failed is now 200=200. Lastly, the assertion self.assertEqual(COUNTERS[name],1), passes meaning that the put method was successful and properly covered.

Alexander Lauderdale
CS 472 1001
5 Feb 2024

test_read_a_counter(self):

```
def test_read_a_counter(self):
    result = self.client.post('/counters/boo')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    result = self.client.get('/counters/boo')
    self.assertEqual(result.status_code, status.HTTP_200_OK)

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

In the read counter test, we check that when reading a counter, that either it exists and the status is OK, or that it doesn't exist and it returns the status NOT_FOUND. I first start by creating a counter "boo", and assert check that it was created.

If that passes, I call the get method for "boo", which should return status OK since boo exists as a counter in COUNTERS.

Next, I try to read a counter called "bah", which was never a counter that I created. When I try to make an assertion after doing self.client.get('/counters/bah') that status_code = OK, I get RED, 404 != 200.

This is because inside the read_counter() method that I REFACTORED in counter.py:

```
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """Update a counter"""
    global COUNTERS

    if name not in COUNTERS:
        return {"Message":f"Counter {name} does not exist"}, status.HTTP_404_NOT_FOUND

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

It checks if the name that's passed to the get method exists as a counter in COUNTERS. If it doesn't, then it returns the status that it couldn't find the counter as 404_NOT_FOUND. When I ran the nosetests after fixing the last assertion so status_code = NOT_FOUND, I get GREEN; since reading the "boo" counter has status OK, and reading the "bah" counter returns status NOT_FOUND, the read_counter() get method has been properly covered.

Alexander Lauderdale
CS 472 1001
5 Feb 2024

```
(venv) C:\Users\alexl\tdd>nosetests -s

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

Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
src\counter.py       20      0   100%
src\status.py         6      0   100%
-----------------------------------------------
TOTAL                26      0   100%
------------------------------------------------
Ran 4 tests in 0.406s

OK
```

After writing the counter methods based on the test methods, I was sure to achieve 100% coverage.