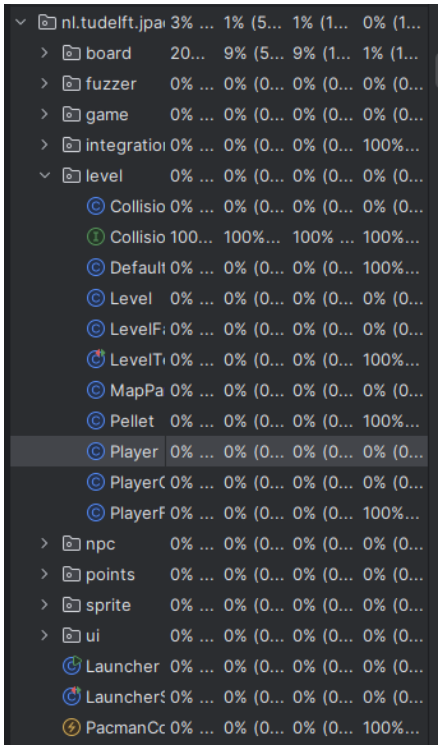
TASK 2.1



- Is the coverage good enough?

Absolutely not. The coverage the tests cover in the code is practically nothing with each category being <5% overall. The only package being tested was the board. At this coverage, we might as well not have any tests at all.

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (93/1141)	4% (22/539)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	15% (2/13)	6% (5/78)	3% (13/348)	0% (0/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	66% (4/6)	44% (20/45)	51% (66/128)	31% (21/66)
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)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

This is the coverage after completing the example test in the instructions. This is much better with us testing more than just the board package and even nearly completing the sprite package along the way. Still very bad overall with most things uncovered.

Pellet.getValue Test

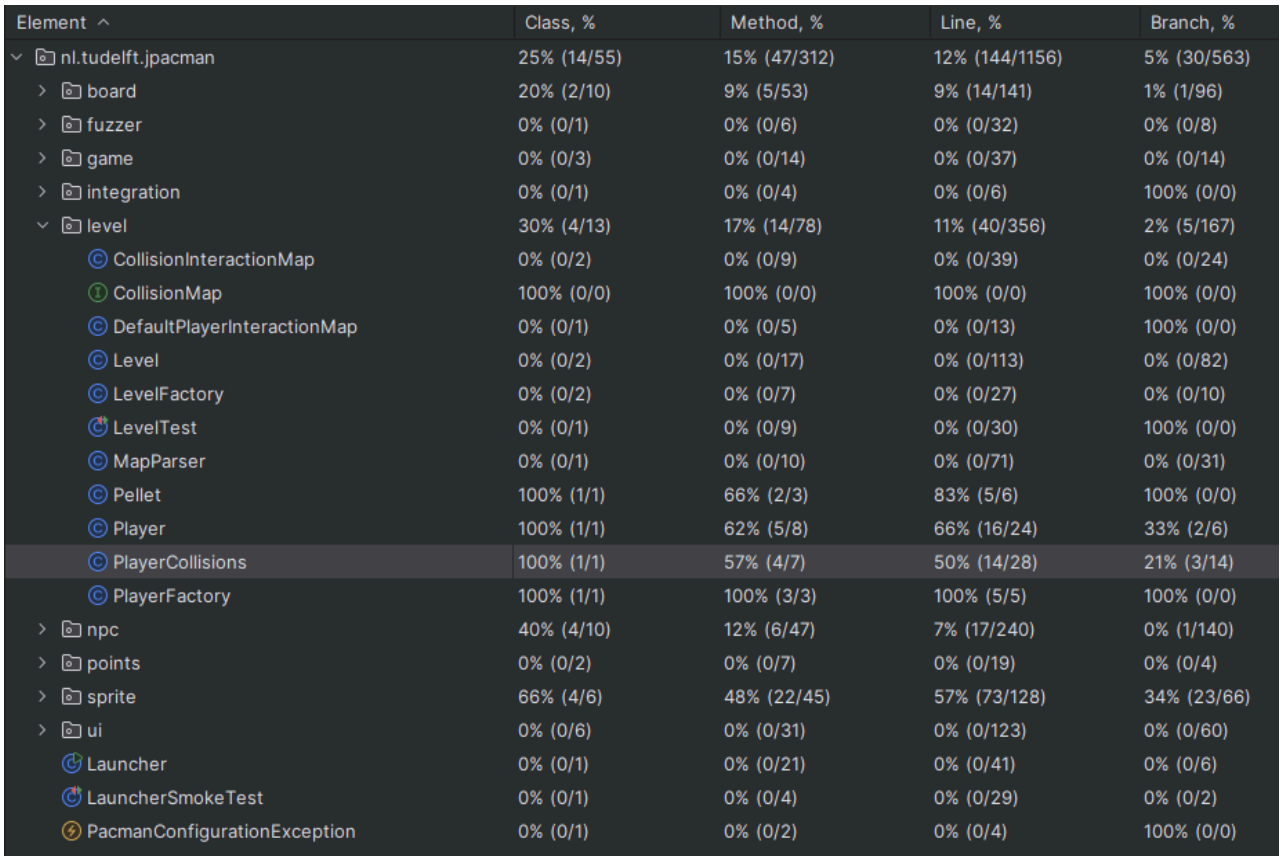
This was the first test completed for task 2.1 and made use of a parameter test in which I found an example of in BoardTest.java. This setup worked flawlessly and the coverage increased by several points in some small areas with the 10% of methods now being covered. However, it was pretty basic and light on code and barely covered anything.

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	16% (9/55)	10% (32/312)	8% (98/1142)	4% (22/539)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	23% (3/13)	8% (7/78)	5% (18/349)	0% (0/167)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/39)	0% (0/24)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)	100% (0/0)
Level	0% (0/2)	0% (0/17)	0% (0/113)	0% (0/82)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)	0% (0/10)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)	100% (0/0)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)	0% (0/31)
Pellet	100% (1/1)	66% (2/3)	83% (5/6)	100% (0/0)
Player	100% (1/1)	25% (2/8)	33% (8/24)	0% (0/6)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)	0% (0/14)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)	100% (0/0)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	66% (4/6)	44% (20/45)	51% (66/128)	31% (21/66)
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)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

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

import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.mock;

import nl.tudelft.jpacman.sprite.Sprite;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

public class PelletValueTest {

    // Mimics input to test to see what would happen
    // Idea sourced from BoardTest.java
    @ParameterizedTest
    @CsvSource({
        "1",
        "2",
        "3",
        "25",
        "367",
        "-5",
        "01",
        "20000",
        "-453"
    })
    void testPelletValue(int x) {

        // Makes a pellet object
        Pellet pelletPoint = new Pellet(x, mock(Sprite.class));

        // Checks that its value is the same as inputted
        assertThat(pelletPoint.getValue()).isEqualTo(x);
    }
}
```

PlayerCollisions.PlayerVersusGhost Test

Testing the PlayerVersusGhost method in PlayerCollisions was the second test I tackled. It seemed complicated at first but was surprisingly easy to complete. I took the method from the example test to create a Player object and expanded that basic idea to make Ghost and mocked a collision. I feel testing to ensure that the player was correctly marked as dead and that the ghost made for the test was the killer was appropriate and covered the area. The Class% covered by this was pretty good apparently, but coverage is still poor.

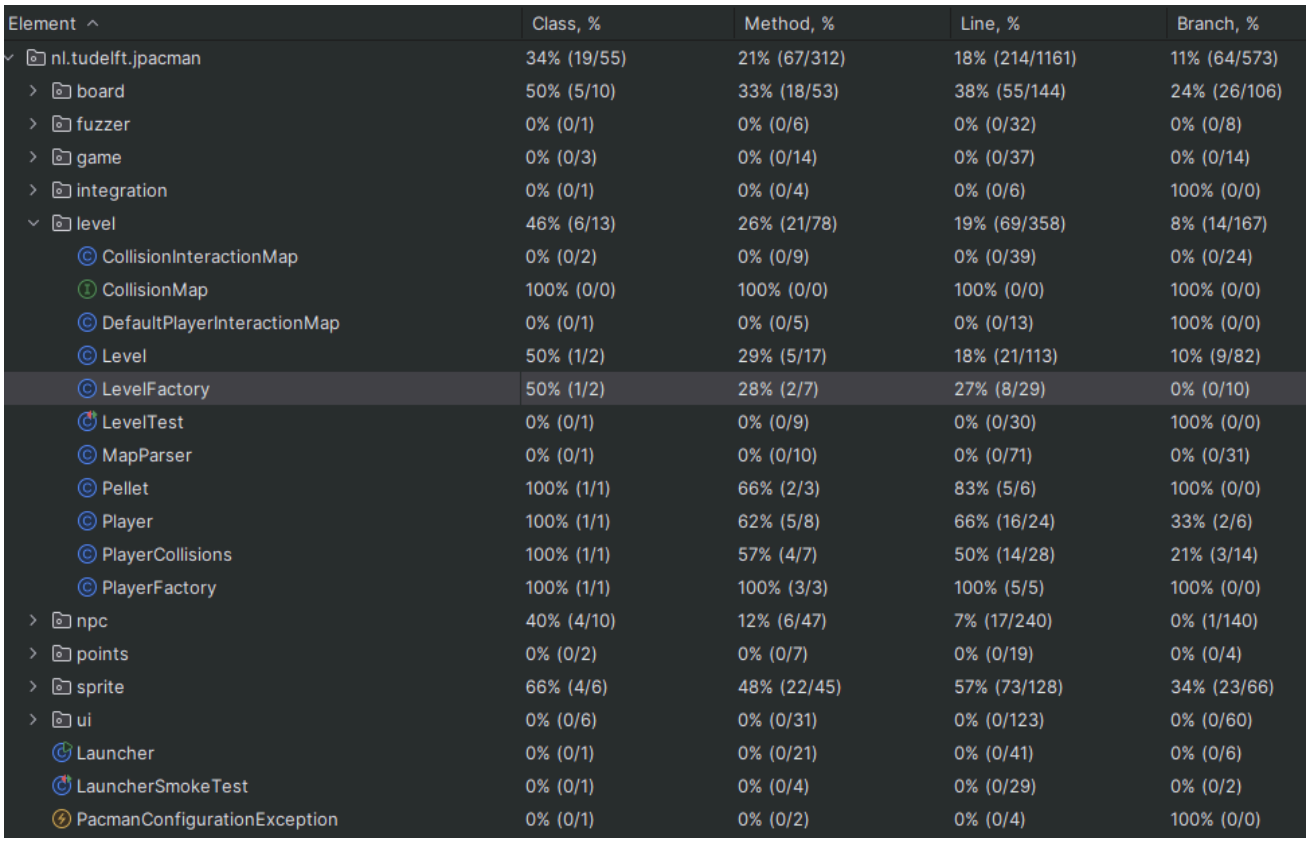


Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	25% (14/55)	15% (47/312)	12% (144/1156)	5% (30/563)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	30% (4/13)	17% (14/78)	11% (40/356)	2% (5/167)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/39)	0% (0/24)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)	100% (0/0)
Level	0% (0/2)	0% (0/17)	0% (0/113)	0% (0/82)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)	0% (0/10)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)	100% (0/0)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)	0% (0/31)
Pellet	100% (1/1)	66% (2/3)	83% (5/6)	100% (0/0)
Player	100% (1/1)	62% (5/8)	66% (16/24)	33% (2/6)
PlayerCollisions	100% (1/1)	57% (4/7)	50% (14/28)	21% (3/14)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)	100% (0/0)
npc	40% (4/10)	12% (6/47)	7% (17/240)	0% (1/140)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	66% (4/6)	48% (22/45)	57% (73/128)	34% (23/66)
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)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

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

import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.mock;

import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.npc.Ghost;

import nl.tudelft.jpacman.points.PointCalculator;

import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;

public class PvGCollisionsTest {

    @Test
    void playerAgainstGhost(){
        // Creates a player object
        // Given from Task 2 / PlayerTest.java
        PacManSprites pacSprites = new PacManSprites();
        PlayerFactory plFactory = new PlayerFactory(pacSprites);
        Player pl = plFactory.createPacMan();

        // Creates a Ghost via a GhostFactory
        GhostFactory ghFactory = new GhostFactory(pacSprites);
        Ghost gh = ghFactory.createBlinky();

        // Creates a PlayerCollision Object by creating a fake
        // PointCalculator and then mimics a collision between
        // the generated ghost and player
        PointCalculator calc = mock(PointCalculator.class);
        PlayerCollisions plCollide = new PlayerCollisions(calc);
        plCollide.collide(pl, gh);

        // Checks that the player was set to die
        // and that the killer is our ghost
        assertThat(pl.isAlive()).isEqualTo( expected: false);
        assertThat(pl.getKiller()).isEqualTo(gh);
    }
}
```

LevelFactory.createLevel Test

This particular test was pretty easy initially as I followed the pattern set forth from the previous tests and used similar means to make objects. However, one particular issue arose that took some time to figure out and it related to using BasicSquare.java. That particular asset would not properly import into my test and I found that if I change the package from Level to Board that the test was located that it would work without issue. Hence why the package is Board at the top and not Level, despite it testing a method in package Level. Simply testing if the level was made felt good enough and IntelliJ says I covered a decent chunk of LevelFactory in my one test.

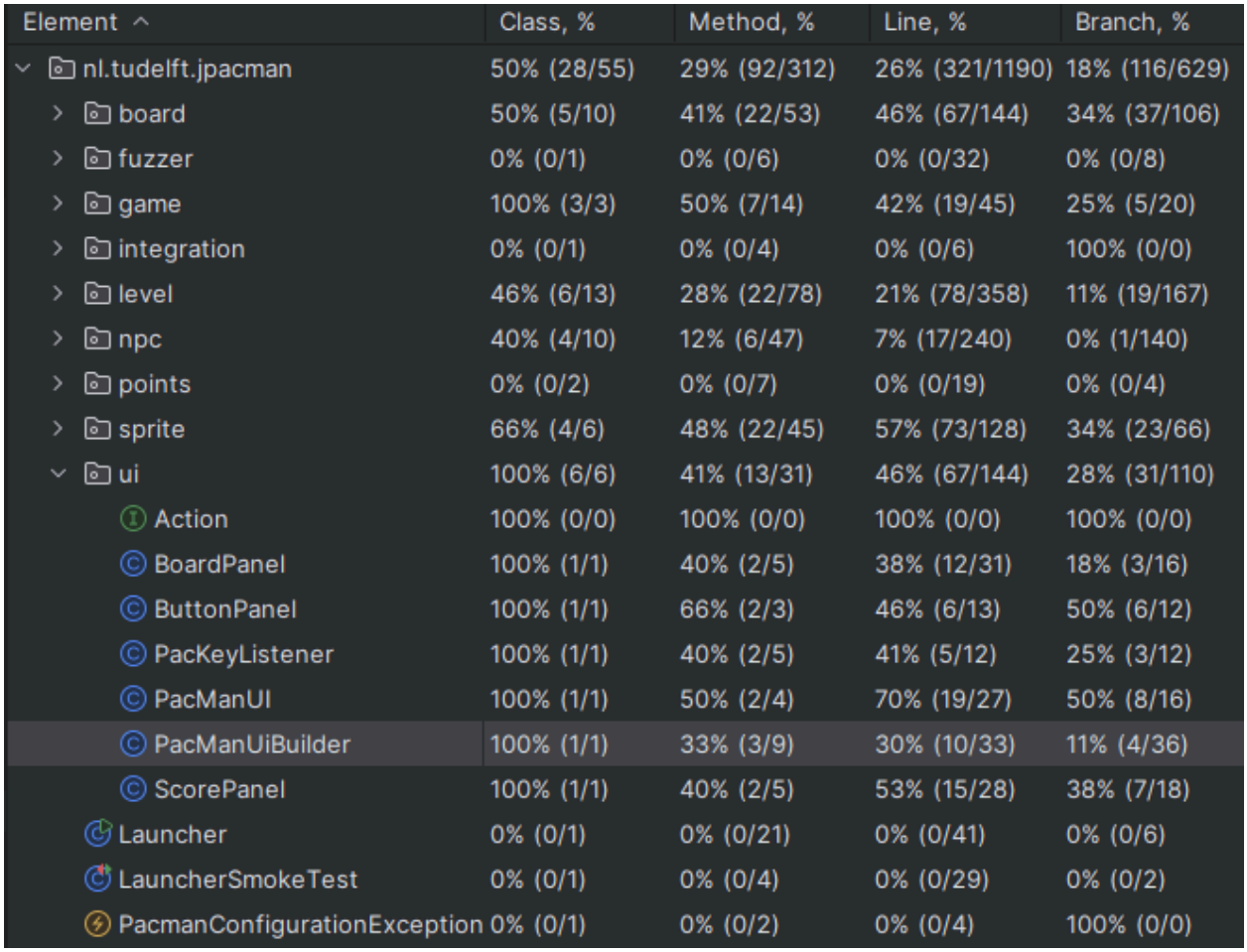


Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	34% (19/55)	21% (67/312)	18% (214/1161)	11% (64/573)
board	50% (5/10)	33% (18/53)	38% (55/144)	24% (26/106)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	46% (6/13)	26% (21/78)	19% (69/358)	8% (14/167)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/39)	0% (0/24)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)	100% (0/0)
Level	50% (1/2)	29% (5/17)	18% (21/113)	10% (9/82)
LevelFactory	50% (1/2)	28% (2/7)	27% (8/29)	0% (0/10)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)	100% (0/0)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)	0% (0/31)
Pellet	100% (1/1)	66% (2/3)	83% (5/6)	100% (0/0)
Player	100% (1/1)	62% (5/8)	66% (16/24)	33% (2/6)
PlayerCollisions	100% (1/1)	57% (4/7)	50% (14/28)	21% (3/14)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)	100% (0/0)
npc	40% (4/10)	12% (6/47)	7% (17/240)	0% (1/140)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	66% (4/6)	48% (22/45)	57% (73/128)	34% (23/66)
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)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

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

import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.mock;

import nl.tudelft.jpacman.npc.Ghost;
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.points.PointCalculator;
import nl.tudelft.jpacman.sprite.PacManSprites;
import nl.tudelft.jpacman.level.LevelFactory;
import nl.tudelft.jpacman.level.Level;

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.List;

public class LevelCreationTest {

    @Test
    void createALevel(){
        // Pre-requisites to making the LevelFactory
        PacManSprites pacSprites = new PacManSprites();
        GhostFactory ghFactory = new GhostFactory(pacSprites);
        PointCalculator calc = mock(PointCalculator.class);

        LevelFactory levelCreator = new LevelFactory(pacSprites, ghFactory, calc);

        // Pre-requisites to making the Level with the LevelFactory
        // Most of this is sourced from SquareTest.java
        Square s1 = new BasicSquare();
        BoardFactory boxMaker = new BoardFactory(pacSprites);
        Board box = boxMaker.createBoard(new Square[][]{{s1}});

        // Couple of blank lists to make the level
        List<Ghost> ghosts = new ArrayList<>();
        List<Square> startPositions = new ArrayList<>();

        Level map = null;
        map = levelCreator.createLevel(box, ghosts, startPositions);

        // Tests - first ensures a level was even made
        assertThat(map).isNotNull();
        // Checks that the board is still the same as generated
        assertThat(map.getBoard()).isEqualTo(box);
        // Level shouldn't have started
        assertThat(map.isInProgress()).isEqualTo( expected: false);
        // There are no players, so none should be alive
        assertThat(map.isAnyPlayerAlive()).isEqualTo( expected: false);
    }
}
```

PacManUIBuilder.build Test

This final test was done since the initial Pellet.GetValue test was too simple and a more complicated test needed to be done. I primarily used methods from previous tests to make what I needed, and especially took inspiration from how I made a level object in the above test. Notably, this included wanting to use BasicSquare.java again but this time of swapping packages, I simply remade it in the test file itself. I also underestimated how big this particular test was gonna be, but it never got too complicated. Had an issue come up where I needed something in the list for the start positions, but was easy to fix. Simply tested if the UI was made given that's the point of the method. Ended up hitting 50% in Class% covered according to IntelliJ and was beginning to look good for coverage at this point.



Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	50% (28/55)	29% (92/312)	26% (321/1190)	18% (116/629)
board	50% (5/10)	41% (22/53)	46% (67/144)	34% (37/106)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	100% (3/3)	50% (7/14)	42% (19/45)	25% (5/20)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	46% (6/13)	28% (22/78)	21% (78/358)	11% (19/167)
npc	40% (4/10)	12% (6/47)	7% (17/240)	0% (1/140)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	66% (4/6)	48% (22/45)	57% (73/128)	34% (23/66)
ui	100% (6/6)	41% (13/31)	46% (67/144)	28% (31/110)
Action	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
BoardPanel	100% (1/1)	40% (2/5)	38% (12/31)	18% (3/16)
ButtonPanel	100% (1/1)	66% (2/3)	46% (6/13)	50% (6/12)
PacKeyListener	100% (1/1)	40% (2/5)	41% (5/12)	25% (3/12)
PacManUI	100% (1/1)	50% (2/4)	70% (19/27)	50% (8/16)
PacManUiBuilder	100% (1/1)	33% (3/9)	30% (10/33)	11% (4/36)
ScorePanel	100% (1/1)	40% (2/5)	53% (15/28)	38% (7/18)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

```java
class SimpleSquare extends Square {  1 usage

    /**
     * Creates a new simple square.
     * Sourced from BasicSquare.java
     */
    SimpleSquare() { super(); }

    @Override
    public boolean isAccessibleTo(Unit unit) { return true; }

    @Override
    /return.type.incompatible/
    public Sprite getSprite() { return null; }
}
```

```java
public class pacManBuilder {

    @Test
    void buildAPacMan(){
        // Pre-requisites to create the Level Factory and Game Factory
        PacManSprites pacSprites = new PacManSprites();
        PlayerFactory plFactory = new PlayerFactory(pacSprites);
        GhostFactory ghFactory = new GhostFactory(pacSprites);
        PointCalculator calc = mock(PointCalculator.class);
        LevelFactory levelCreator = new LevelFactory(pacSprites, ghFactory, calc);

        // Pre-requisites to making the Level with the LevelFactory
        // Most of this is sourced from SquareTest.java
        Square s1 = new SimpleSquare();
        BoardFactory boxMaker = new BoardFactory(pacSprites);
        Board box = boxMaker.createBoard(new Square[][]{{s1}});

        // Couple of blank lists to make the level
        List<Ghost> ghosts = new ArrayList<>();
        List<Square> startPositions = new ArrayList<>();

        // Adds a square as a starting position
        startPositions.add(s1);

        // Creates a level
        Level map = levelCreator.createLevel(box, ghosts, startPositions);

        // Makes a game object
        GameFactory gameMaker = new GameFactory(plFactory);
        Game isGame = gameMaker.createSinglePlayerGame(map, calc);

        // Uses game object to make the UI
        PacManUiBuilder pacManMaker = new PacManUiBuilder();
        PacManUI pac = null;
        pac = pacManMaker.build(isGame);

        // Tests that something was made
        assertThat(pac).isNotNull();
    }
}
```

TASK 3

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

Absolutely not. The results given to me from JaCoCo seemed to surpass the reports from IntelliJ in every aspect. I did not spot a single instance where JaCoCo gave me a worse rating than IntelliJ in any category or overall.

Example:
From IntelliJ (Class %, Method %, Line %, Branch %)

Level	50% (1/2)	35% (6/17)	26% (30/113)	17% (14/82)
LevelFactory	50% (1/2)	28% (2/7)	27% (8/29)	0% (0/10)

From JaCoCo (Missed Branches, Missed Lines, Missed Methods, Missed Classes)

Level	70%	6	105	1	15	0	1
Level.NpcMoveTask	100%	0	10	0	2	0	1
LevelFactory	80%	1	17	0	4	0	1

This even extended to files I didn't believe had any tests performed like MapParser which came out to 0% in every category for IntelliJ but got nearly everything covered according to JaCoCo. One thing to note is that IntelliJ and JaCoCo must have slightly different rating systems in how they measure a line, class, branch, etc. because both come to different amounts in each category (as can be seen above).

Inspecting the actual code, it seems JaCoCo gives more credit to things than IntelliJ does. As an example, according to IntelliJ, in PlayerFactory under "createGhost", none of the lines had any coverage, but in JaCoCo, it was 100% covered. This is most likely why results are better in JaCoCo than in IntelliJ.

One interesting thing that did occur, however, was when I completed the test case for the PacManUIBuilder. Once completed, IntelliJ updated many random files in their coverage, including Level and LevelFactory, but JaCoCo did not have any changes, at least in those two files.

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

It was much easier to spot things I may have missed or hadn't covered as it's a bit more in your face and pops out more compared to the subdued approach IntelliJ takes. Though I can see how some may not like it as they may view it as obnoxious.

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

While I think IntelliJ is likely more accurate in what was actually being tested, I do prefer JaCoCo because it is easier to see what I may have missed and it has that yellow category for partial coverage which is nice and IntelliJ does not have that.

JaCoCo is also a bit more visually pleasing to me and the bar it has for some categories does more to help me visualize what's missing than simple numbers do.

TASK 4

From_Dict Test

Used the method in the given test to get a random account. I then made a new dict using the same keys as the one stored in the account and gave that as the argument to test from_dict. I then copied the output using to_dict and compared to see if the values were indeed overwritten.

```python
def test_from_dict():
    """ Test getting attributes from a dict """
    rand = randrange(0, len(ACCOUNT_DATA))  # Generate a random index
    data = ACCOUNT_DATA[rand]  # get a random account
    account = Account(**data)

    newAttr = {
        "name" : "1",
        "email" : "2",
        "phone_number" : "3",
        "disabled" : False,
        "date_joined" : db.DateTime
    }

    account.from_dict(newAttr)
    result = account.to_dict()

    assert account.name == result["name"]
    assert account.email == result["email"]
    assert account.phone_number == result["phone_number"]
    assert account.disabled == result["disabled"]
    assert account.date_joined == result["date_joined"]
```

```
---------- coverage: platform win32, python 3.12.0-final-0 -----------
Name                  Stmts   Miss  Cover   Missing
-----------------------------------------------------
models\__init__.py        7      0   100%
models\account.py        40      9    78%   45-48, 52-54, 74-75
-----------------------------------------------------
TOTAL                    47      9    81%
```

Update Test

Used the method in the given test to get a random account. Uses a try..exception to update the account and if it fails it would assign an ID and try again. This is because the error code on line 47 in the original method needs to be tested and this proved to be a sound way to implement it. The original account object doesn't actually have an ID value associated and so would trip the exception but try again with an ID and succeed. It then simply tests the data to see if there are any discrepancies. Notably, I ran pytest and recorded the results before implementing my try..exception, so the coverage is slightly outdated, but still shows progress was made.

```python
def test_update():
    """ Test updating an account in the database """
    rand = randrange(0, len(ACCOUNT_DATA))  # Generate a random index
    data = ACCOUNT_DATA[rand]  # get a random account
    account = Account(**data)

    try:
        account.update()
    except:
        account.id = 1
        account.update()

    result = account.to_dict()

    assert account.name == result["name"]
    assert account.email == result["email"]
    assert account.phone_number == result["phone_number"]
    assert account.disabled == result["disabled"]
    assert account.date_joined == result["date_joined"]
```

```
---------- coverage: platform win32, python 3.12.0-final-0 -----------
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models\__init__.py       7      0   100%
models\account.py       40      6    85%   47, 52-54, 74-75
--------------------------------------------------
TOTAL                   47      6    87%
```

Delete Test

Used the method in the given test to get a random account. Simply creates an account and then deletes it. Uses a try..exception to try and update the account as it expects an error and we don't want that to stop the test. Then simply checks that the length of the list holding the counters still matches. This is where I got the idea to use the try..exception that was later implemented in test_update, so pytest shows that it has been fixed at this point and line 47 is covered.

```python
def test_delete():
    """ Test deleting an account in the database """
    rand = randrange(0, len(ACCOUNT_DATA))  # Generate a random index
    data = ACCOUNT_DATA[rand]  # get a random account
    account = Account(**data)

    account.create()
    account.delete()
    try:
        account.update()
    except:
        assert len(Account.all()) == len(ACCOUNT_DATA)
```

```
---------- coverage: platform win32, python 3.12.0-final-0 -----------
Name                  Stmts   Miss  Cover   Missing
-----------------------------------------------------
models\__init__.py        7      0   100%
models\account.py        40      2    95%   74-75
-----------------------------------------------------
TOTAL                    47      2    96%
```

Find Test

Used the method in the given test to get a random account and then copy that account's ID. We then use it to find the account ID with the find method from the complete list of all account IDs. We then simply test that they match. This covered 100% of the statements according to the pytest.

```python
def test_find():
    """ Test finding an account in the database by its ID """
    rand = randrange(0, len(ACCOUNT_DATA))  # Generate a random index
    data = ACCOUNT_DATA[rand]  # get a random account
    account = Account(**data)
    accID = account.id

    queryID = Account.find(accID)

    assert queryID == account.id
```

```
---------- coverage: platform win32, python 3.12.0-final-0 -----------
Name                  Stmts   Miss  Cover   Missing
-----------------------------------------------------
models\__init__.py        7      0   100%
models\account.py        40      0   100%
-----------------------------------------------------
TOTAL                    47      0   100%
```

TASK 5

Update / Put Counter

Following the outline given to us in the instructions, the test and method being tested were easy to make and I even added my own test. The test attempts to access a counter that doesn't exist, then create that counter, check the value is at 0, then increment it by 1, then check the counter value has successfully incremented and check each of these steps along the way for the correct status code being returned. The method itself checks that the counter trying to be incremented exists and returns an error if it doesn't, then increments the counters value if it does and returns a successful status code. The only real issue came in the form of trying to access the value in the counters. I ended up importing the entire global COUNTERS from counters.py to access it.

```python
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """ Update a counter """
    global COUNTERS
    if not name in COUNTERS:
        return {"Message":f"Counter {name} doesn't exist"}, status.HTTP_404_NOT_FOUND
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

```python
def test_update_a_counter(self, client):
    """ It should update an existing counter """
    # Update Counter That Doesn't Exist
    result = client.put('/counters/uac')
    assert result.status_code == status.HTTP_404_NOT_FOUND

    # Make Counter To Exist
    result = client.post('/counters/uac')
    assert result.status_code == status.HTTP_201_CREATED

    # Check Counter Value
    assert COUNTERS["uac"] == 0

    # Update Counter
    result = client.put('/counters/uac')
    assert result.status_code == status.HTTP_200_OK

    # Test Counter Value
    assert COUNTERS["uac"] == 1
```

Read / Get Counter

Much the same as previously in the PUT test including me adding an extra test just because. The test attempts to read a counter that doesn't exist, make the counter an attempt to read it again, and then just checks if the value of the counter is indeed 0. A test is done along the way to ensure the correct status code is being returned. The method simply checks the counter being asked for exits and returns the counter if it does and errors if it doesn't. Again, I had an issue accessing the value in the counter, but importing the entire COUNTERS dict seemed to make it work and pass all tests.

```python
@app.route('/counters/<name>', methods=['GET'])
def read_counter(name):
    """ Read a counter """
    global COUNTERS
    if not name in COUNTERS:
        return {"Message":f"Counter {name} doesn't exist"}, status.HTTP_404_NOT_FOUND
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

```python
def test_get_a_counter(self, client):
    """ It should get an existing counter """
    # Read Counter That Doesn't Exist
    result = client.get('/counters/gac')
    assert result.status_code == status.HTTP_404_NOT_FOUND

    # Make Counter To Exist
    result = client.post('/counters/gac')
    assert result.status_code == status.HTTP_201_CREATED

    # Retrieve Counter
    result = client.get('/counters/gac')
    assert result.status_code == status.HTTP_200_OK

    # Read Counter
    assert COUNTERS["gac"] == 0
```

```python
# Import the dict that holds all the counters
from src.counter import COUNTERS
```

Overall, pytest claims 100% coverage and both the above tests and the given example tests all worked.

To expand on my issue trying to access the counters value, I kept getting an error saying the result was not subscriptable, implying it was not a dict or tuple. I also couldn't figure out the attribute name to access the counter or the key name at all. I'm still not entirely sure you could even access it via the result from the other methods, but I simply imported the COUNTERS dict to get around the issue.

```
---------- coverage: platform win32, python 3.12.0-final-0 -----------
Name              Stmts   Miss  Cover   Missing
-------------------------------------------------
src\__init__.py       0      0   100%
src\counter.py       22      0   100%
src\status.py         6      0   100%
-------------------------------------------------
TOTAL                28      0   100%
```

Fork Link: https://github.com/LilAttacker/jpacman