Part 1: Assess Quality of Tests using Line Coverage

1. Report three different scenarios you have tested and the corresponding coverage results.

1. A scenario which was tested was starting the game with the `start` button and then stopping the game with the `stop` button. This appears to freeze the game in its current state.

   This code is from: `/jpacman-framework/src/main/java/nl/tudelft/jpacman/ui/PacManUiBuilder.java`.

   ```
    1  /**
    2   * Adds a button with the caption {@value #STOP_CAPTION} that stops the
    3   * game.
    4   *
    5   * @param game
    6   *            The game to stop.
    7   */
    8  private void addStopButton(final Game game) {
    9      assert game != null;
   10
   11      buttons.put(STOP_CAPTION, new Action() {
   12          @Override
   13          public void doAction() {
   14              game.stop();
   15          }
   16      });
   17  }
   ```

   It appears that as the UI is built, the `stop` button is added to `this.buttons`. This functionality works but the test coverage is lacking. The action which is performed is `game.stop()` on line 14. After observing coverage results, it seems that line 9 doesn't have full branch coverage. The prediction here is that only one state of `game` is tested. Also on line 14 where `game.stop()` is called, no coverage exists. The prediction here is that no test exists where `doAction` is called.

2. A scenario tested was when the `stop` button was pressed, that no more moves could be made.

   This code is from `/jpacman-framework/src/main/java/nl/tudelft/jpacman/level/Level.java`.

```java
 1  /**
 2   * Moves the unit into the given direction if possible and handles all
 3   * collisions.
 4   *
 5   * @param unit
 6   *            The unit to move.
 7   * @param direction
 8   *            The direction to move the unit in.
 9   */
10  public void move(Unit unit, Direction direction) {
11      assert unit != null;
12      assert direction != null;
13
14      if (!isInProgress()) {
15          return;
16      }
17
18      synchronized (moveLock) {
19          ...
20      }
21  }
```

While this functionality works the branch coverage is lacking. On line 14, only one value of the predicate `isInProgress()` is used thus not coverage every branch test scenario.

3. A scenario tested was pressing the arrow keys to move pacman around the board.

   This code is from `/jpacman-framework/src/main/java/nl/tudelft/jpacman/ui/PacKeyListener.java`.

```java
@Override
public void keyPressed(KeyEvent e) {
    assert e != null;
    Action action = mappings.get(e.getKeyCode());
    if (action != null) {
        action.doAction();
    }
}
```

   This is run everytime a key is pressed from the keyboard. It handles mapping a `KeyEvent` is a specific action, as this functionality works, the entire function does not have test coverage.

2. Report the coverage percentage. Identify the three least covered application classes. Identify the three least covered application classes. Explain why the tests for them are adequate or how they can be improved.

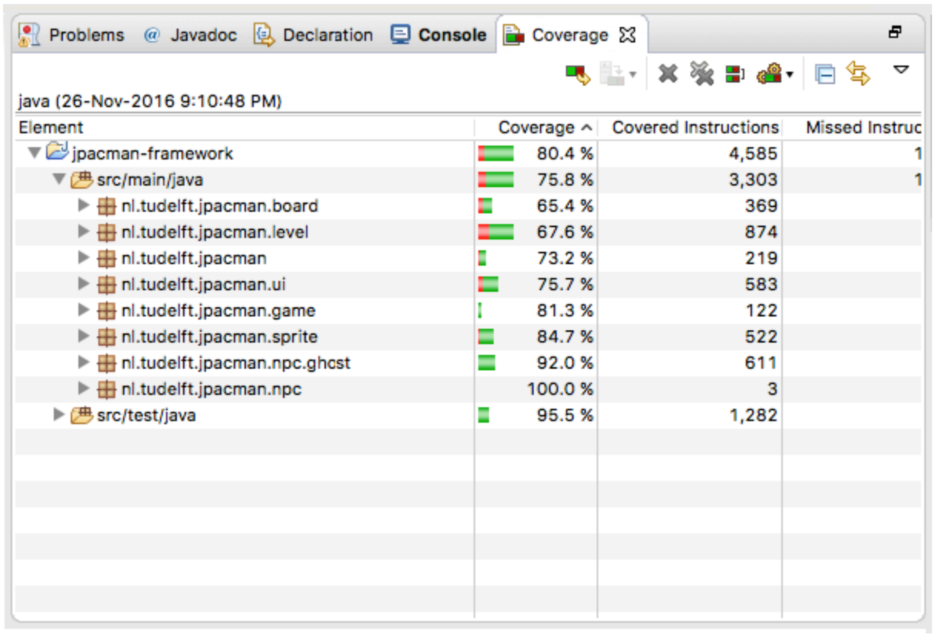
Three application classes exist with 0% coverage.

1. `/jpacman-framework/src/main/java/nl/tudelft/jpacman/level/CollisionInteractionMap.java`
   - Simply no tests import `CollisionInteractionMap` so none of its functions are used in tests.
2. `/jpacman-framework/src/main/java/nl/tudelft/jpacman/level/DefaultPlayerInteractionMap.java`
   - Simply no tests import `DefaultPlayerInteractionMap` so none of its functions are used in tests.
3. `/jpacman-framework/src/main/java/nl/tudelft/jpacman/PacmanConfigurationException.java`
   - Simply no tests import `PacmanConfigurationException` so none of its functions are used in tests.

3. Measure the code coverage again, but this time with a configuration that has runtime assertion enabled (add ‘ -

ea ’ as VM argument). To do this, right click on the project, select “Coverage As”, then go to “Coverage Configurations”. Then under “Arguments” add “ -ea ” to VM arguments. Explain the coverage changes you see.

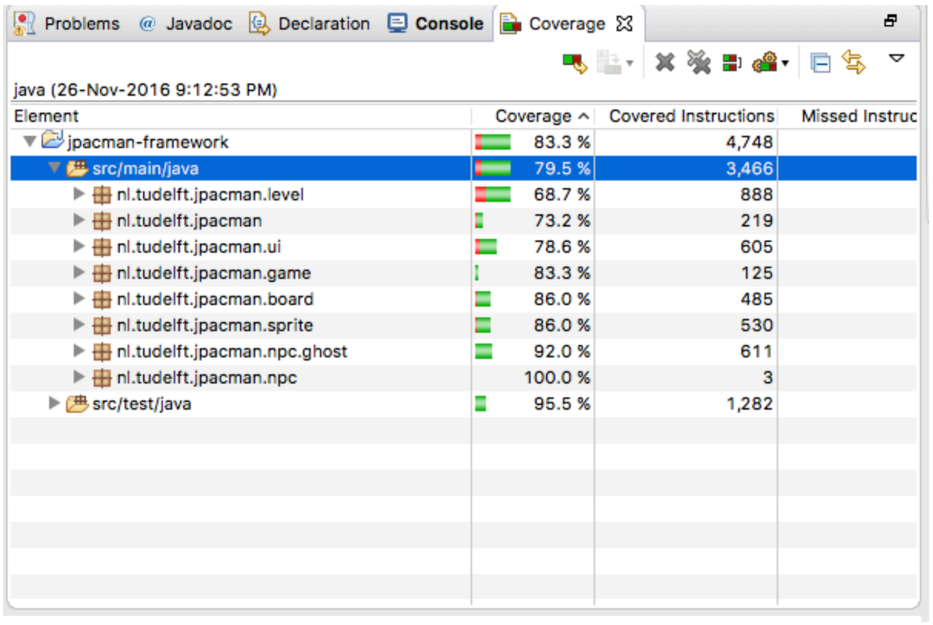
- Without `-ea`



java (26-Nov-2016 9:10:48 PM)

Element	Coverage	Covered Instructions	Missed Instruc
jpacman-framework	80.4 %	4,585	1
src/main/java	75.8 %	3,303	1
nl.tudelft.jpacman.board	65.4 %	369	
nl.tudelft.jpacman.level	67.6 %	874	
nl.tudelft.jpacman	73.2 %	219	
nl.tudelft.jpacman.ui	75.7 %	583	
nl.tudelft.jpacman.game	81.3 %	122	
nl.tudelft.jpacman.sprite	84.7 %	522	
nl.tudelft.jpacman.npc.ghost	92.0 %	611	
nl.tudelft.jpacman.npc	100.0 %	3	
src/test/java	95.5 %	1,282	

- With `-ea`

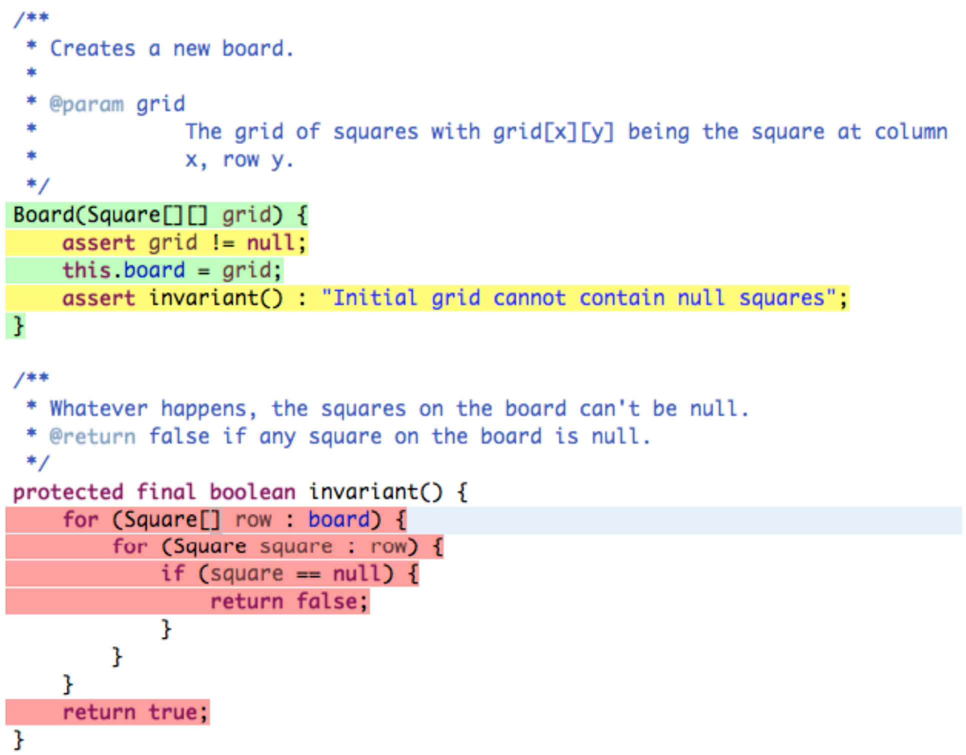


java (26-Nov-2016 9:12:53 PM)

Element	Coverage	Covered Instructions	Missed Instruc
jpacman-framework	83.3 %	4,748	
src/main/java	79.5 %	3,466	
nl.tudelft.jpacman.level	68.7 %	888	
nl.tudelft.jpacman	73.2 %	219	
nl.tudelft.jpacman.ui	78.6 %	605	
nl.tudelft.jpacman.game	83.3 %	125	
nl.tudelft.jpacman.board	86.0 %	485	
nl.tudelft.jpacman.sprite	86.0 %	530	
nl.tudelft.jpacman.npc.ghost	92.0 %	611	
nl.tudelft.jpacman.npc	100.0 %	3	
src/test/java	95.5 %	1,282	

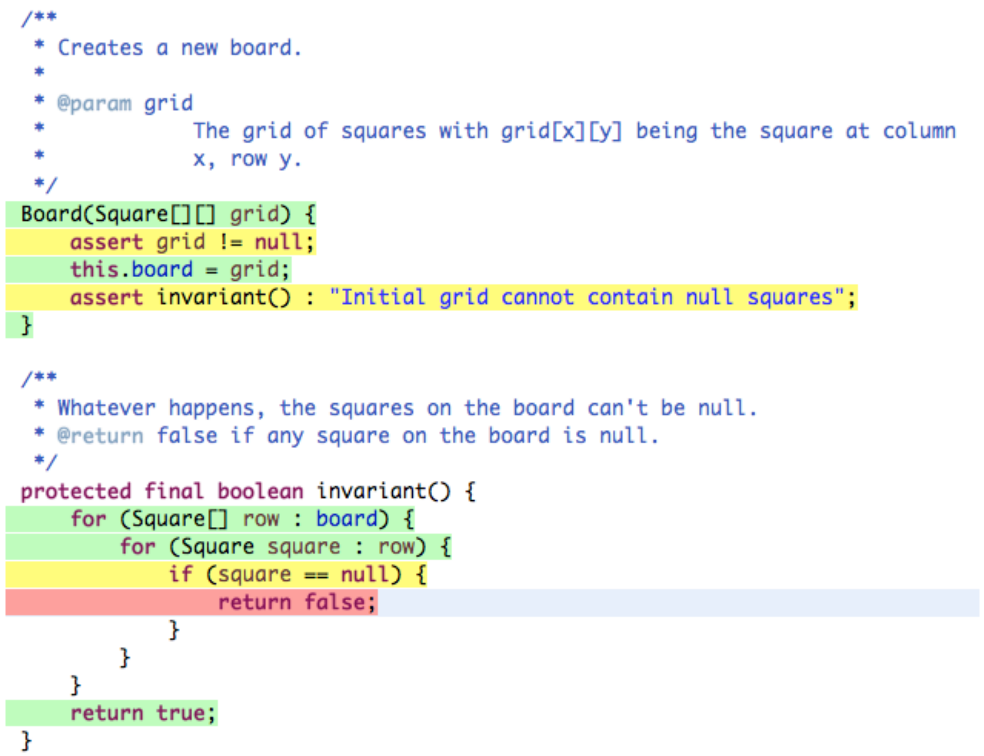
Runtime assertions enable `assert` statements to be run. Some functions which act as invariants are held within `assert` statements thus causing less coverage to be obtained when running without `-ea`.

An example is here in `/jpacman-framework/src/main/java/nl/tudelft/jpacman/board/Board.java`. This is the coverage when run without `-ea`.



```
/**
 * Creates a new board.
 *
 * @param grid
 *            The grid of squares with grid[x][y] being the square at column
 *            x, row y.
 */
Board(Square[][] grid) {
    assert grid != null;
    this.board = grid;
    assert invariant() : "Initial grid cannot contain null squares";
}

/**
 * Whatever happens, the squares on the board can't be null.
 * @return false if any square on the board is null.
 */
protected final boolean invariant() {
    for (Square[] row : board) {
        for (Square square : row) {
            if (square == null) {
                return false;
            }
        }
    }
    return true;
}
```

And this is the coverage when run with `-ea`.



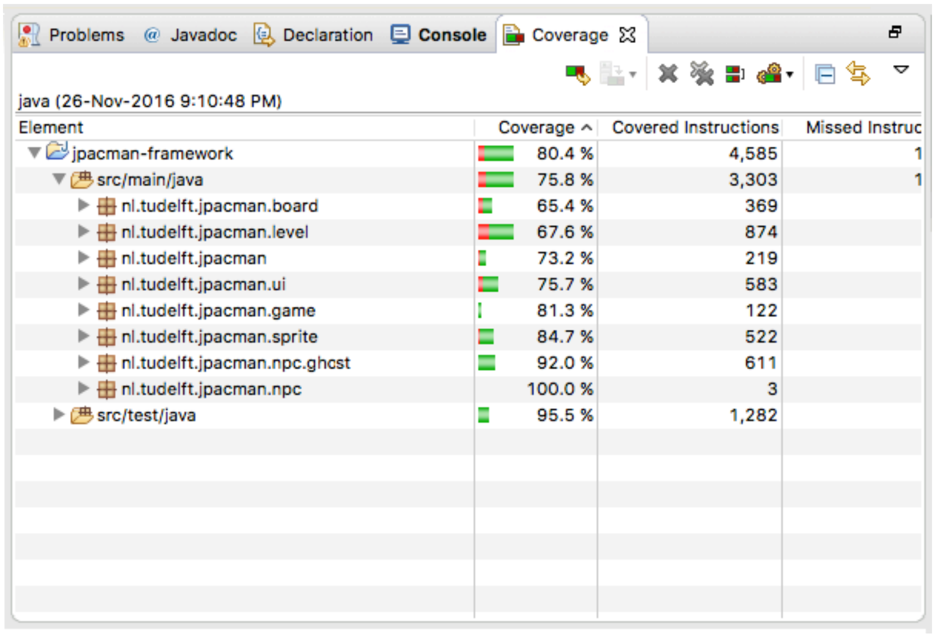
```
/**
 * Creates a new board.
 *
 * @param grid
 *            The grid of squares with grid[x][y] being the square at column
 *            x, row y.
 */
Board(Square[][] grid) {
    assert grid != null;
    this.board = grid;
    assert invariant() : "Initial grid cannot contain null squares";
}

/**
 * Whatever happens, the squares on the board can't be null.
 * @return false if any square on the board is null.
 */
protected final boolean invariant() {
    for (Square[] row : board) {
        for (Square square : row) {
            if (square == null) {
                return false;
            }
        }
    }
    return true;
}
```

As you can see the function inside the `assert invariant()` was run during the tests when Runtime Assertion was enabled. However all branch coverage does not exist still around the `assert` statements indicating a lack of test coverage still exists.

Part 2: Assess Quality of Tests using Mutation Testing

1. Run PIT with the default set of mutation operators on the existing test suite and measure mutation coverage. Report the results and compare them with line coverage results you got earlier. Explain what you see.

- Test coverage results:

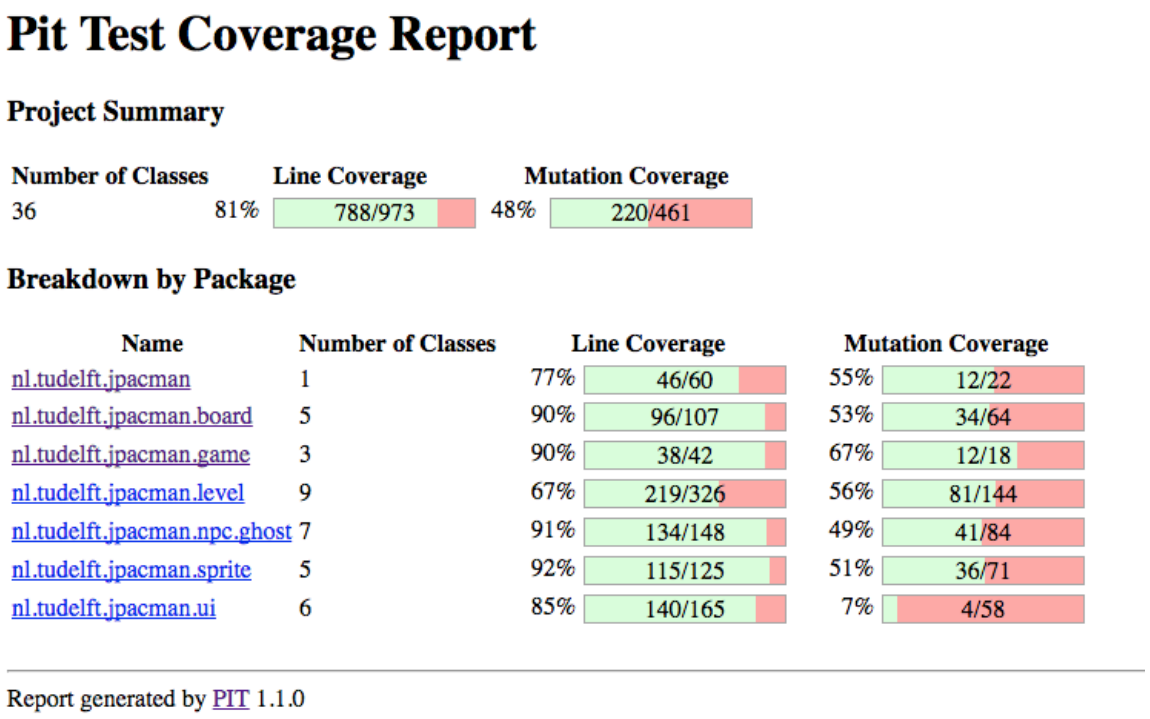


Problems | Javadoc | Declaration | Console | Coverage

java (26-Nov-2016 9:10:48 PM)

Element	Coverage	Covered Instructions	Missed Instruc
jpacman-framework	80.4 %	4,585	1
src/main/java	75.8 %	3,303	1
nl.tudelft.jpacman.board	65.4 %	369	
nl.tudelft.jpacman.level	67.6 %	874	
nl.tudelft.jpacman	73.2 %	219	
nl.tudelft.jpacman.ui	75.7 %	583	
nl.tudelft.jpacman.game	81.3 %	122	
nl.tudelft.jpacman.sprite	84.7 %	522	
nl.tudelft.jpacman.npc.ghost	92.0 %	611	
nl.tudelft.jpacman.npc	100.0 %	3	
src/test/java	95.5 %	1,282	

- PIT coverage results:



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
36	81%	788/973	48%	220/461

Breakdown by Package

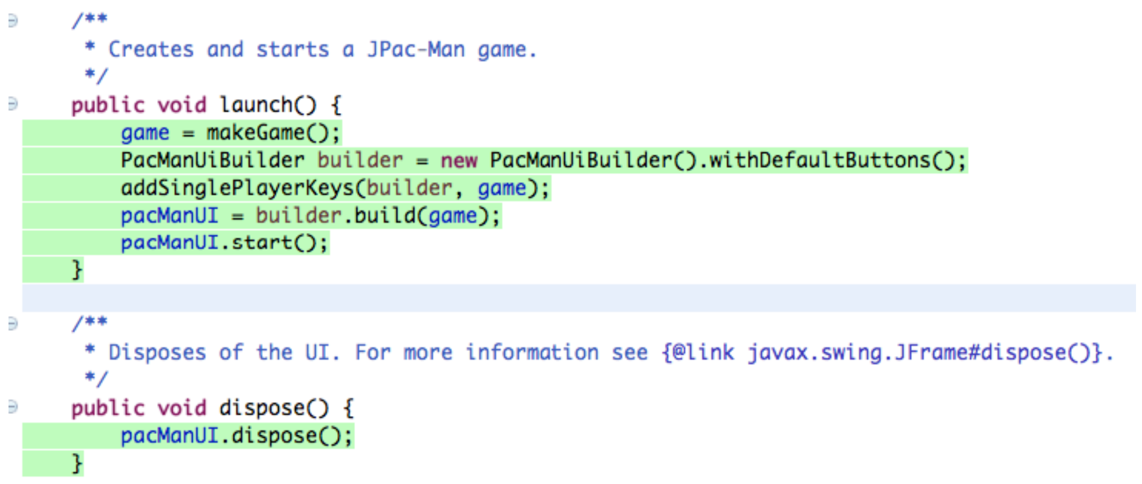
Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.jpacman	1	77%	46/60	55%	12/22
nl.tudelft.jpacman.board	5	90%	96/107	53%	34/64
nl.tudelft.jpacman.game	3	90%	38/42	67%	12/18
nl.tudelft.jpacman.level	9	67%	219/326	56%	81/144
nl.tudelft.jpacman.npc.ghost	7	91%	134/148	49%	41/84
nl.tudelft.jpacman.sprite	5	92%	115/125	51%	36/71
nl.tudelft.jpacman.ui	6	85%	140/165	7%	4/58

Report generated by PIT 1.1.0

As you can see line coverage seems to be very similar but mutation coverage has a large margin in difference.

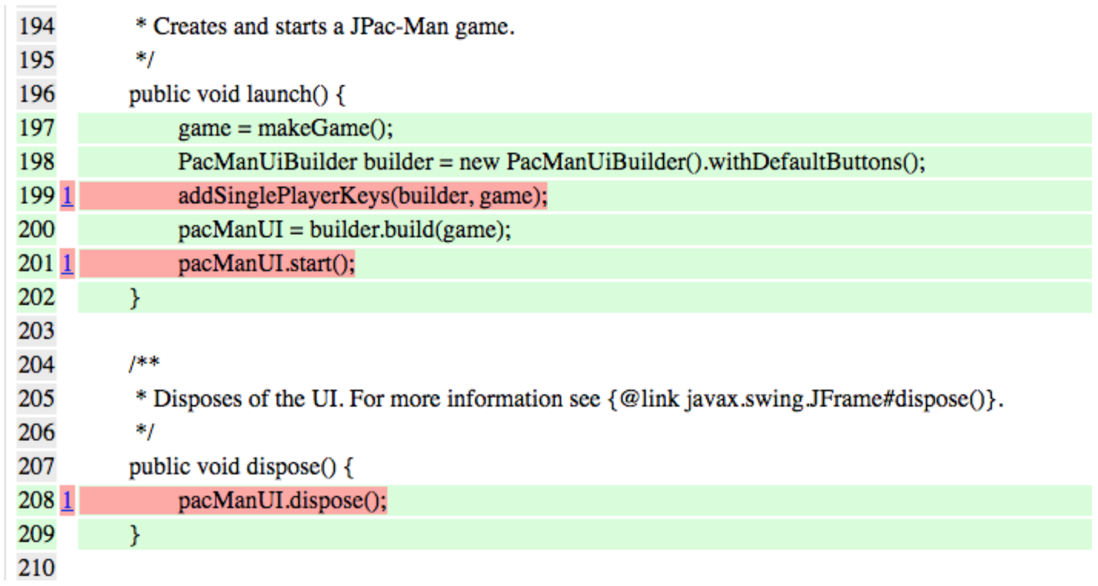
Lets look at an example where there is a large difference in line coverage and mutatation coverage.

As you can see in `/jpacman-framework/src/main/java/nl/tudelft/jpacman/Launcher.java` , 100% line coverage..



```
    /**
     * Creates and starts a JPac-Man game.
     */
    public void launch() {
        game = makeGame();
        PacManUiBuilder builder = new PacManUiBuilder().withDefaultButtons();
        addSinglePlayerKeys(builder, game);
        pacManUI = builder.build(game);
        pacManUI.start();
    }

    /**
     * Disposes of the UI. For more information see {@link javax.swing.JFrame#dispose()}.
     */
    public void dispose() {
        pacManUI.dispose();
    }
```

But in the same file we have marginally less mutation coverage:



```
194         * Creates and starts a JPac-Man game.
195         */
196        public void launch() {
197                game = makeGame();
198                PacManUiBuilder builder = new PacManUiBuilder().withDefaultButtons();
199 1              addSinglePlayerKeys(builder, game);
200                pacManUI = builder.build(game);
201 1              pacManUI.start();
202        }
203
204        /**
205         * Disposes of the UI. For more information see {@link javax.swing.JFrame#dispose()}.
206         */
207        public void dispose() {
208 1              pacManUI.dispose();
209        }
210
```

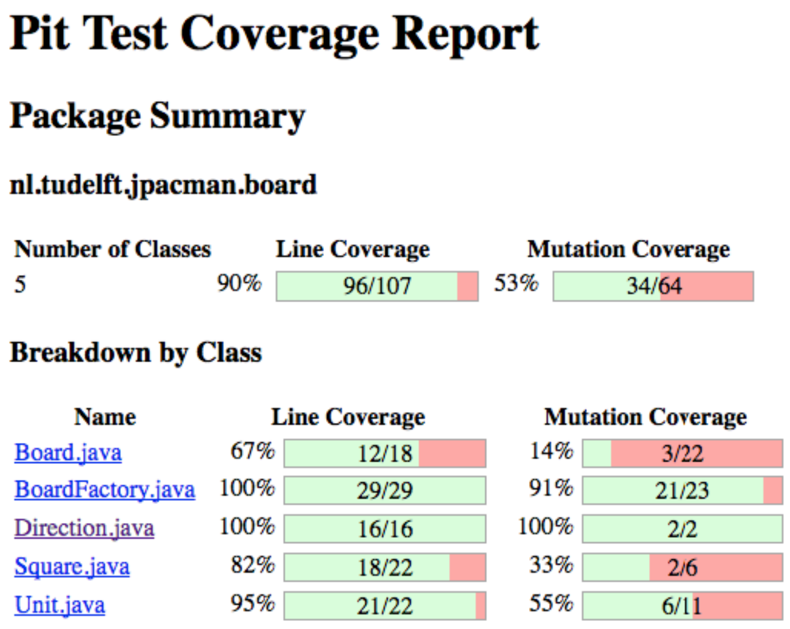
Mutation testing will run tests with some tests removed to see if they still pass. The red highlights here indicate that PIT ran the test suite with line 199 removed and the tests still passed. This indicates a `survived` mutation which it will not count into mutation coverage.

2. Run PIT with only “Conditionals Boundary Mutator”, “Increments Mutator”, and “Math Mutator”, respectively. Compare the results and explain.

The mutators specified are a subset of the default mutators used thus less mutators are used when running the test suite.

One example here is shown.

As you can see this is the summary of mutation coverage results for `nl.tudelft.jpacman.board` .



Pit Test Coverage Report

Package Summary

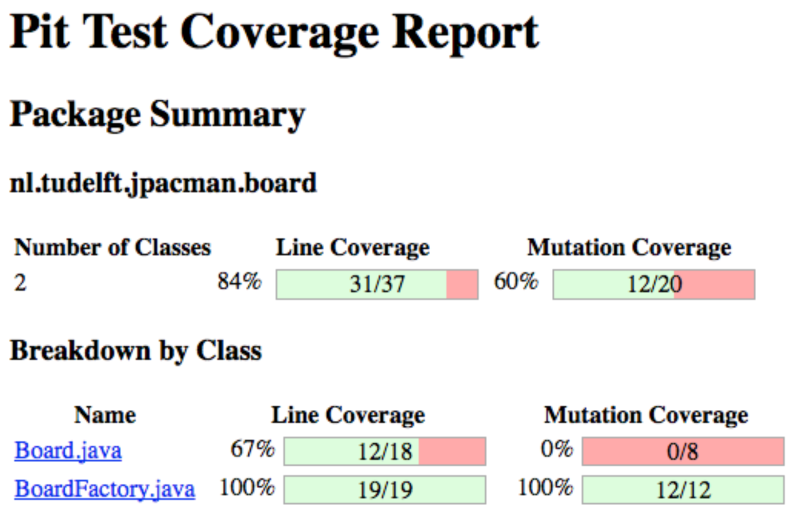
nl.tudelft.jpacman.board

Number of Classes	Line Coverage		Mutation Coverage	
5	90%	96/107	53%	34/64

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Board.java	67%	12/18	14%	3/22
BoardFactory.java	100%	29/29	91%	21/23
Direction.java	100%	16/16	100%	2/2
Square.java	82%	18/22	33%	2/6
Unit.java	95%	21/22	55%	6/11

This is the same summary but with the specified mutators.



Pit Test Coverage Report

Package Summary

nl.tudelft.jpacman.board

Number of Classes	Line Coverage		Mutation Coverage	
2	84%	31/37	60%	12/20

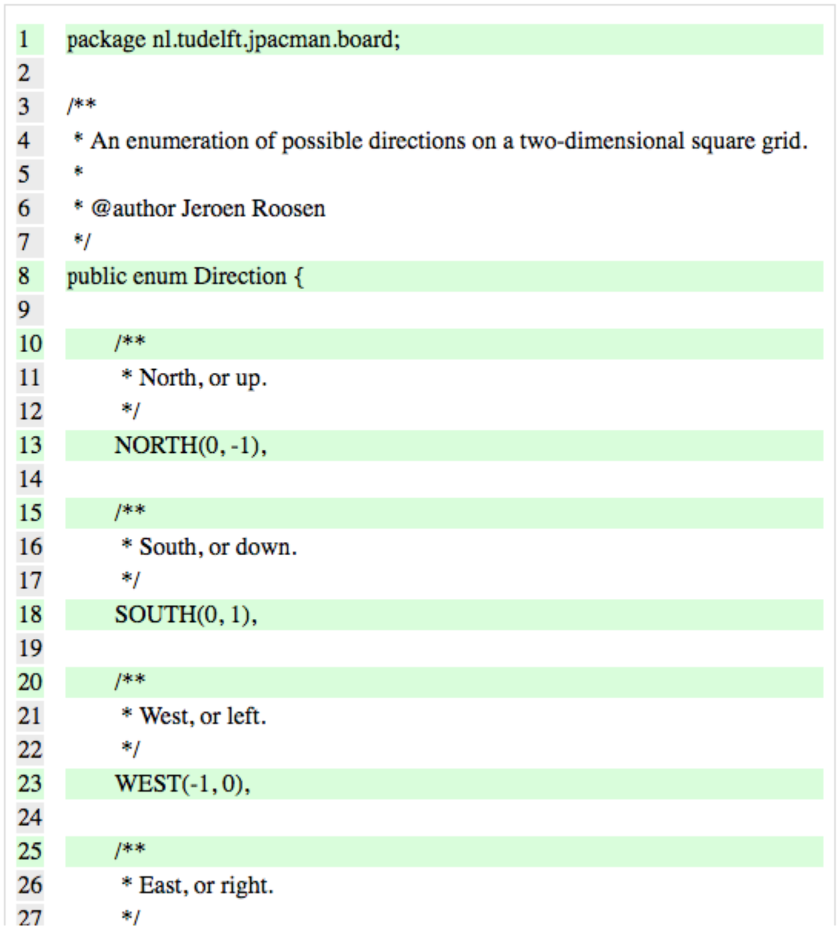
Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Board.java	67%	12/18	0%	0/8
BoardFactory.java	100%	19/19	100%	12/12

As you can see, less classes were covered with mutators. This is because the classes which were excluded do not contain code which can pertain to the specific mutators chosen.

A more specific example is shown when looking at a class which did not get covered in our second PIT test coverage (the one with mutators specified).

Direction.java



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

/**
 * An enumeration of possible directions on a two-dimensional square grid.
 *
 * @author Jeroen Roosen
 */
public enum Direction {

	/**
	 * North, or up.
	 */
	NORTH(0, -1),

	/**
	 * South, or down.
	 */
	SOUTH(0, 1),

	/**
	 * West, or left.
	 */
	WEST(-1, 0),

	/**
	 * East, or right.
	 */
```

The rest of the file shows only green lines as well. This file was not covered in the test run where we specified mutators because the file does not contain lines with:

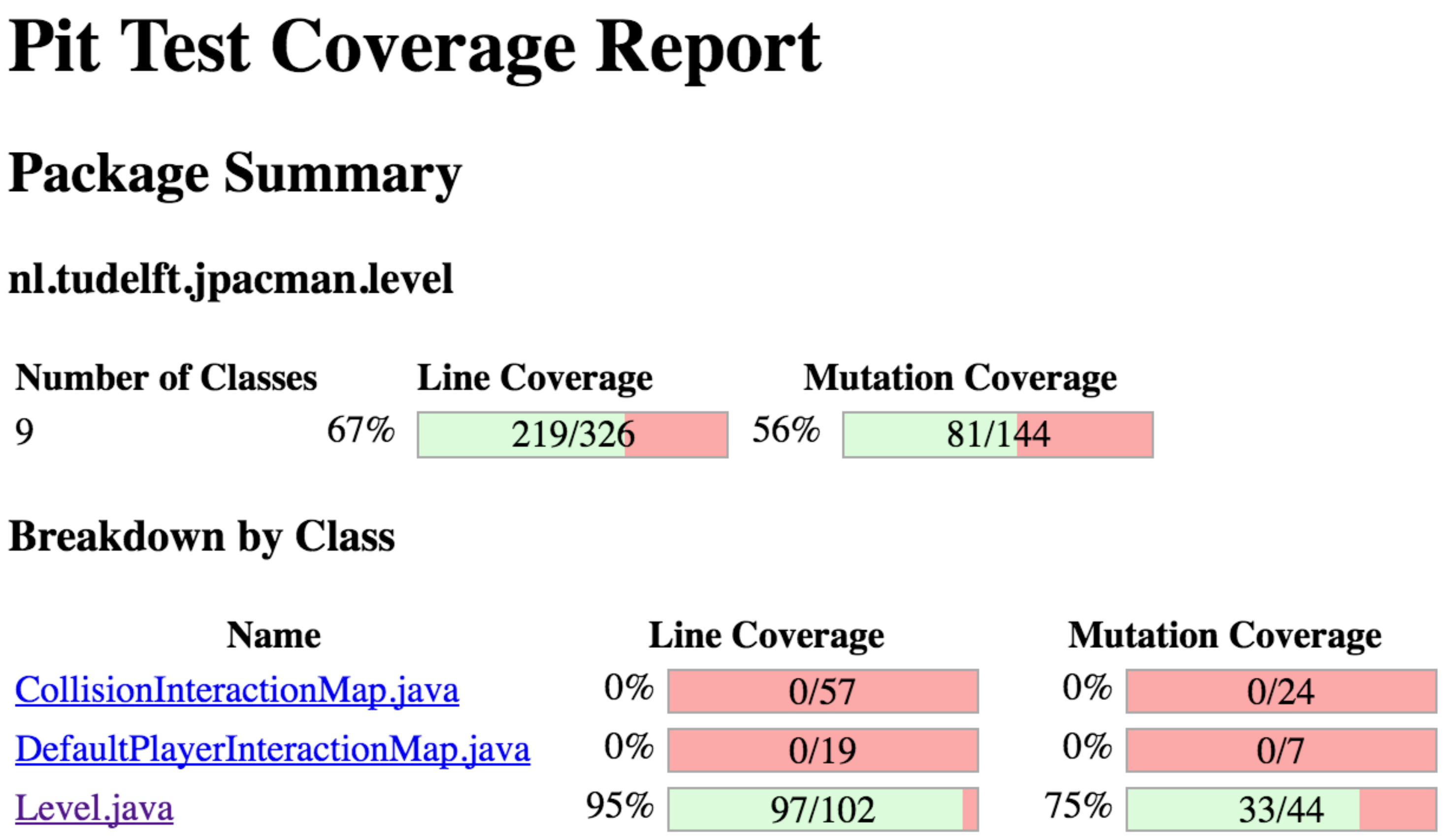
- Conditional boundaries: `<, <=, >, >=`
- Increments: `++, --`
- Mathematical operators: `+, -, ...`

3. Add one Junit test case to an appropriate package (e.g., in “ src/test/java/... ”) so that with it more mutants can be killed using default PIT configurations (i.e., the mutation score increases). Include your test case in the report and explain.

Found in `/jpacman-framework/src/main/java/nl/tudelft/jpacman/level/Level.java` starting on line 274.

```java
/**
 * Returns <code>true</code> iff at least one of the players in this level
 * is alive.
 *
 * @return <code>true</code> if at least one of the registered players is
 *         alive.
 */
public boolean isAnyPlayerAlive() {
    for (Player p : players) {
        if (p.isAlive()) {
            return true;
        }
    }
    return false;
}
```

Before:



Pit Test Coverage Report

Package Summary

nl.tudelft.jpacman.level

Number of Classes	Line Coverage		Mutation Coverage	
9	67%	219/326	56%	81/144

Breakdown by Class

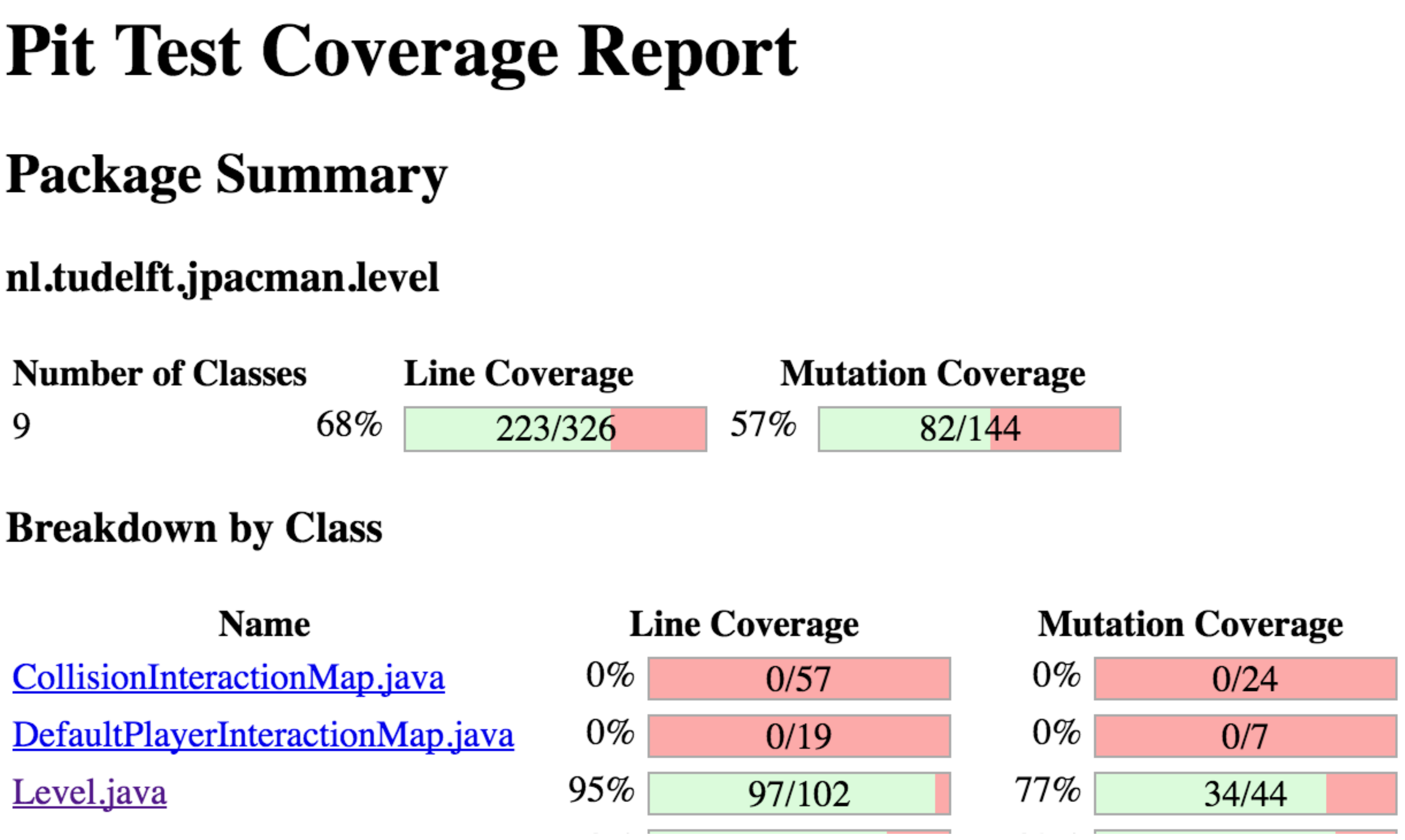
Name	Line Coverage		Mutation Coverage	
CollisionInteractionMap.java	0%	0/57	0%	0/24
DefaultPlayerInteractionMap.java	0%	0/19	0%	0/7
Level.java	95%	97/102	75%	33/44

PIT reported this for the return line for `isAnyPlayerAlive`:

```
1.1 Location : isAnyPlayerAlive Killed by : none replaced return of integer sized value with (x == 0 ? 1 : 0) → SURVIVED
```

This simply means that PIT tried running the test suite with returning True instead of False and they all succeeded. This means that the return value is not being asserted anywhere in the test suite.

After:



Pit Test Coverage Report

Package Summary

nl.tudelft.jpacman.level

Number of Classes	Line Coverage		Mutation Coverage	
9	68%	223/326	57%	82/144

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
CollisionInteractionMap.java	0%	0/57	0%	0/24
DefaultPlayerInteractionMap.java	0%	0/19	0%	0/7
Level.java	95%	97/102	77%	34/44

Test added:

```java
/**
 * Verifies if no players are in the game, no one is alive.
 */
@Test
@SuppressWarnings("PMD.JUnitTestsShouldIncludeAssert")
public void isAnyPlayerAlive() {
    assertFalse(level.isAnyPlayerAlive());
}
```

With this test, the return value of `isAnyPlayerAlive` is checked, thus the mutation mentioned previously will be killed.

Part 3: Extend Test Suite with Symbolic Execution