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`.

   ```java
   /**
    * Adds a button with the caption {@value #STOP_CAPTION} that stops the
    * game.
    *
    * @param game
    *            The game to stop.
    */
   private void addStopButton(final Game game) {
       assert game != null;

       buttons.put(STOP_CAPTION, new Action() {
           @Override
           public void doAction() {
               game.stop();
           }
       });
   }
   ```

   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
   /**
    * Moves the unit into the given direction if possible and handles all
    * collisions.
    *
    * @param unit
    *            The unit to move.
    * @param direction
    *            The direction to move the unit in.
    */
   public void move(Unit unit, Direction direction) {
       assert unit != null;
       assert direction != null;

       if (!isInProgress()) {
           return;
       }

       synchronized (moveLock) {
           ...
       }
   }
   ```

   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
1  @Override
2  public void keyPressed(KeyEvent e) {
3      assert e != null;
4      Action action = mappings.get(e.getKeyCode());
5      if (action != null) {
6          action.doAction();
7      }
8  }
```

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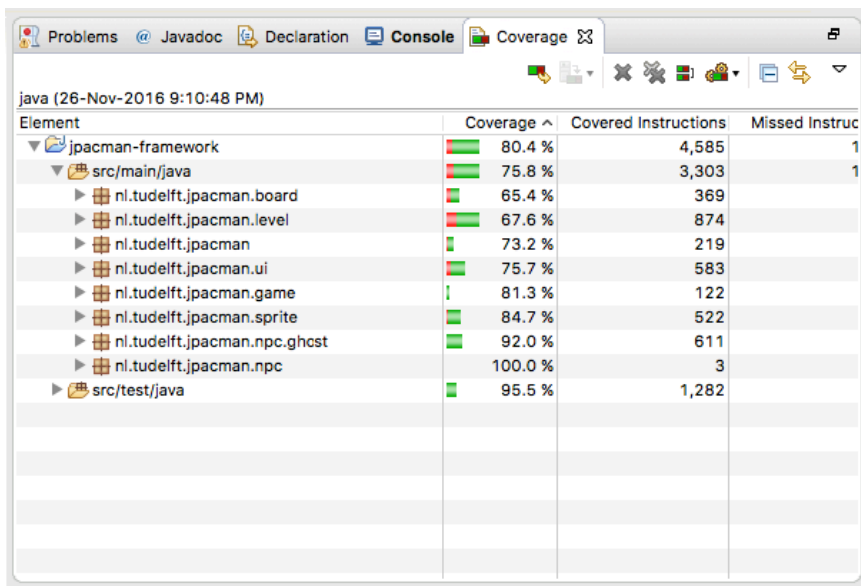
2. Report the coverage percentage. Identify the three least covered application classes. Identify the three least covered application classes. Explain why the tests for them are adequate or how they can be improved.

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`

- With `-ea`

```
Problems  @ Javadoc  Declaration  Console  Coverage

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`.

```java
/**
 * 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`.

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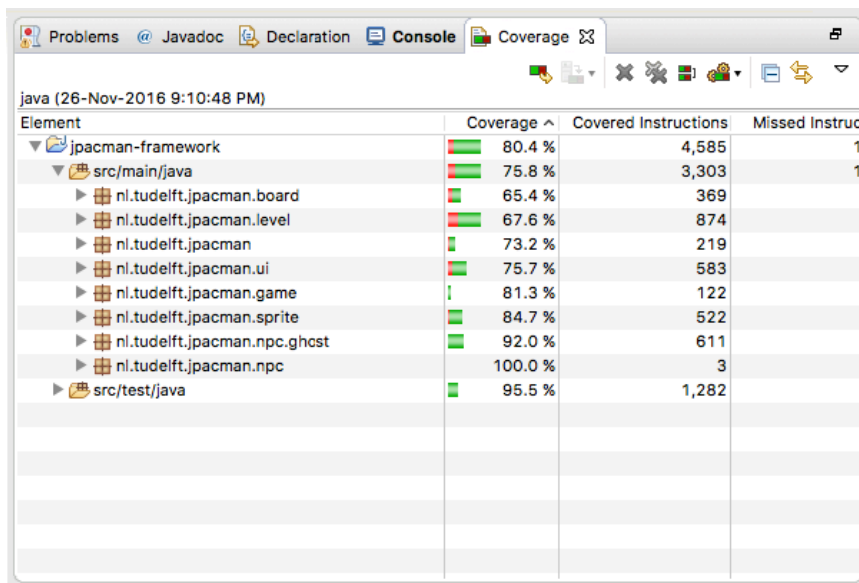
As you can see the function inside the `assert invariant()` was run during the tests when Runtime Assertion was enabled. However all branch coverage does not exist still around the `assert` statements indicating a lack of test coverage still exists.

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.

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	

- Test 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

- PIT coverage results:

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..

```java
/**
 * 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.

Code which needs to appear for the mutator to be applied to the test related test suite.

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

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

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

Increments Mutator

This is the same summary but with the `Increments Mutator`.

Pit Test Coverage Report

Package Summary

nl.tudelft.jpacman.board

Number of Classes	Line Coverage		Mutation Coverage	
2	49%	32/65	60%	3/5

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Board.java	28%	13/46	0%	0/2
BoardFactory.java	100%	19/19	100%	3/3

As you can see here, the mutation coverage technically rises! The reason for this is that code which is excluded from the report (`Direction.java` for example) did not include increments. PIT was smart enough to not run those tests as mutations with this setting does not apply to it.

Math Mutator

This is the same summary but with the `Math Mutator` .

Pit Test Coverage Report

Package Summary

nl.tudelft.jpacman.board

Number of Classes	Line Coverage		Mutation Coverage	
1	100%	19/19	100%	6/6

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
BoardFactory.java	100%	19/19	100%	6/6

As you can see here, the mutation coverage technically rises! The reason for this is that code which is excluded from the report (`Direction.java` for example) did not include addition, subtracting, modulus, etc. PIT was smart enough to not run those tests as mutations with this setting does not apply to it.

Condition Boundaries

This is the same summary but with the `Math Mutator` .

Pit Test Coverage Report

Package Summary

nl.tudelft.jpacman.board

Number of Classes	Line Coverage		Mutation Coverage	
2	49%	32/65	56%	5/9

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Board.java	28%	13/46	33%	2/6
BoardFactory.java	100%	19/19	100%	3/3

As you can see here, the mutation coverage technically rises! The reason for this is that code which is excluded from the report (`Direction.java` for example) did not include conditional operators such as `>` and `<=`, etc. PIT was smart enough to not run those tests as mutations with this setting does not apply to it.

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.

```
1   /**
2    * Returns <code>true</code> iff at least one of the players in this level
3    * is alive.
4    *
5    * @return <code>true</code> if at least one of the registered players is
6    *          alive.
7    */
8   public boolean isAnyPlayerAlive() {
9       for (Player p : players) {
10          if (p.isAlive()) {
11              return true;
12          }
13      }
14      return false;
15  }
```

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:

```
1  /**
2   * Verifies if no players are in the game, no one is alive.
3   */
4  @Test
5  @SuppressWarnings("PMD.JUnitTestsShouldIncludeAssert")
6  public void isAnyPlayerAlive() {
7      assertFalse(level.isAnyPlayerAlive());
8  }
```

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

1.

Part 1: Note that in order to apply SPF on a non-static method, you need to write a driver method ' static void main(String[]) ' which instantiates the target class and calls the target methods.

```
1   // added to Board.java
2   static void main(String[] args){
3       Square x0y0 = new BasicSquare();
4       Square x0y1 = new BasicSquare();
5       Square x0y2 = new BasicSquare();
6       Square x1y0 = new BasicSquare();
7       Square x1y1 = new BasicSquare();
8       Square x1y2 = new BasicSquare();
9       Unit occupant = new BasicUnit();
10      x0y0.put(occupant);
11      Unit occupant1 = new BasicUnit();
12      x0y1.put(occupant1);
13      Unit occupant2 = new BasicUnit();
14      x0y2.put(occupant2);
15      Unit occupant3 = new BasicUnit();
16      x1y0.put(occupant3);
17      Unit occupant4 = new BasicUnit();
18      x1y1.put(occupant4);
19      Unit occupant5 = new BasicUnit();
20      x1y2.put(occupant5);
21      Square[][] grid = new Square[2][3];
22      grid[0][0] = x0y0;
23      grid[0][1] = x0y1;
24      grid[0][2] = x0y2;
25      grid[1][0] = x1y0;
26      grid[1][1] = x1y1;
27      grid[1][2] = x1y2;
28      Board board = new Board(grid);
29      board.withinBorders(1, 2);
30  }
```

When running

```
java -jar /Users/alecbrunelle/Github/jpf-core/build/RunJPF.jar +shell.port=4242 /Users/alecbrunelle/School/CSC410-A2/q3part1.jpf ...
```

```
 1   Executing command: java -jar /Users/alecbrunelle/Github/jpf-core/build/RunJPF.jar +shell.port=4242 /Users/alecbrunelle/School/CSC4
 2   Running Symbolic PathFinder ...
 3   symbolic.dp=choco
 4   symbolic.string_dp_timeout_ms=0
 5   symbolic.string_dp=none
 6   symbolic.choco_time_bound=30000
 7   symbolic.max_pc_length=2147483647
 8   symbolic.max_pc_msec=0
 9   symbolic.min_int=-1000000
10   symbolic.max_int=1000000
11   symbolic.min_double=-8.0
12   symbolic.max_double=7.0
13   JavaPathfinder core system v8.0 (rev ${version}) - (C) 2005-2014 United States Government. All rights reserved.
14
15
16   ==================================================== system under test
17   nl.tudelft.jpacman.board.Board.main()
18
19   ==================================================== search started: 01/12/16 2:12 PM
20
21   ==================================================== Method Summaries
22   Inputs: x_1_SYMINT,y_2_SYMINT
23
24   nl.tudelft.jpacman.board.Board.withinBorders(0,0)  --> Return Value: 1
25   nl.tudelft.jpacman.board.Board.withinBorders(0,3)  --> Return Value: 0
26   nl.tudelft.jpacman.board.Board.withinBorders(0,-1000000)  --> Return Value: 0
27   nl.tudelft.jpacman.board.Board.withinBorders(2,-2147483648(don't care))  --> Return Value: 0
28   nl.tudelft.jpacman.board.Board.withinBorders(-1000000,-2147483648(don't care))  --> Return Value: 0
29
30   ==================================================== Method Summaries (HTML)
31   <h1>Test Cases Generated by Symbolic JavaPath Finder for nl.tudelft.jpacman.board.Board.withinBorders (Path Coverage) </h1>
32   <table border=1>
33   <tr><td>x_1_SYMINT</td><td>y_2_SYMINT</td><td>RETURN</td></tr>
34   <tr><td>0</td><td>0</td><td>Return Value: 1</td></tr>
35   <tr><td>0</td><td>3</td><td>Return Value: 0</td></tr>
36   <tr><td>0</td><td>-1000000</td><td>Return Value: 0</td></tr>
37   <tr><td>2</td><td>-2147483648(don't care)</td><td>Return Value: 0</td></tr>
38   <tr><td>-1000000</td><td>-2147483648(don't care)</td><td>Return Value: 0</td></tr>
39   </table>
40
41   ==================================================== results
42   no errors detected
43
44   ==================================================== statistics
45   elapsed time:        00:00:00
46   states:              new=9,visited=0,backtracked=9,end=5
47   search:              maxDepth=5,constraints=0
48   choice generators:   thread=1 (signal=0,lock=1,sharedRef=0,threadApi=0,reschedule=0), data=4
49   heap:                new=484,released=283,maxLive=476,gcCycles=8
50   instructions:        5296
51   max memory:          123MB
52   loaded code:         classes=77,methods=1681
53
54   ==================================================== search finished: 01/12/16 2:12 PM
```

**Part 2: Write a fully functional JUnit test class ' JPFBoardTest ' for the test cases generated in a new file
'/src/test/java/nl/tudelft/jpacman/board/JPFBoardTest.java '. The test class should have a test fixture (i.e., @Before) and several test cases (i.e., @Test). Include
your test class in the report as well.**

Test class `/jpacman-framework/src/test/java/nl/tudelft/jpacman/board/JPFBoardTest.java` .

```
1   package nl.tudelft.jpacman.board;
2
3   import static org.junit.Assert.assertFalse;
4   import static org.junit.Assert.assertTrue;
```

```java
import static org.mockito.Mockito.mock;

import org.junit.Before;
import org.junit.Test;

/*
 * Test suite for JPF related Board class tests.
 */
public class JPFBoardTest {
    private Board board;

    private final Square x0y0 = mock(Square.class);
    private final Square x0y1 = mock(Square.class);
    private final Square x0y2 = mock(Square.class);
    private final Square x1y0 = mock(Square.class);
    private final Square x1y1 = mock(Square.class);
    private final Square x1y2 = mock(Square.class);

    private static final int MAX_WIDTH = 2;
    private static final int MAX_HEIGHT = 3;

    /**
     * Setup a board that can be used for testing.
     */
    @Before
    public void setUp() {
        Square[][] grid = new Square[MAX_WIDTH][MAX_HEIGHT];
        grid[0][0] = x0y0;
        grid[0][1] = x0y1;
        grid[0][2] = x0y2;
        grid[1][0] = x1y0;
        grid[1][1] = x1y1;
        grid[1][2] = x1y2;
        board = new Board(grid);
    }

    /**
     * Verifies that given a position on the board, returns True.
     */
    @Test
    public void verifyWithinBorders() {
        assertTrue(board.withinBorders(0, 0));
    }

    /**
     * Verifies that given a position not on the board, returns False.
     */
    @Test
    public void verifyWithinBordersOutsideBounds() {
        assertFalse(board.withinBorders(0, 3));
    }

    /**
     * Verifies that given a position not on the board, returns False.
     */
    @Test
    public void verifyWithinBordersOutsideBounds2() {
        assertFalse(board.withinBorders(0, -1000000));
    }

    /**
     * Verifies that given a position not on the board, returns False.
     */
    @Test
    public void verifyWithinBordersOutsideBounds3() {
        assertFalse(board.withinBorders(2, -2147483648));
    }
```

```
73        /**
74         * Verifies that given a position not on the board, returns False.
75         */
76        @Test
77        public void verifyWithinBordersOutsideBounds4() {
78            assertFalse(board.withinBorders(-1000000, -2147483648));
79        }
80    }
```

2

SPF did not generate test cases to cover all boundary values. `x` and `y` are bound by integers. This means valid values are found within a range of integers. Tests were generated for all `on` and `in` cases but not `off`. The table below explains which ones were there and which ones were missing.

Valid range for `x` : x >= 0, x < 2

Valid range for `y` : y >= 0, y < 3

Type	Range Value	Variable	Test case	Generated from SPF
on	0	x	`withinBorders(0,-1000000)`	Yes
on	2	x	`withinBorders(2,-2147483648(don't care))`	Yes
on	0	y	`withinBorders(0,0)`	Yes
on	3	y	`withinBorders(0,3)`	Yes
off	1 or 3	x	see below	No
off	-1 or 1	x	see below	No
off	2 or 4	y	see below	No
off	-1 or 1	y	see below	No
in	0 or 1	x and y	`withinBorders(0,0)`	Yes

```
 1  /**
 2   * Verifies that given a position not on the board, returns False.
 3   */
 4  @Test
 5  public void verifyWithinBordersOffXUpperBound() {
 6      assertFalse(board.withinBorders(3, 1));
 7  }
 8
 9  /**
10   * Verifies that given a position not on the board, returns False.
11   */
12  @Test
13  public void verifyWithinBordersOffXLowerBound() {
14      assertFalse(board.withinBorders(-1, 1));
15  }
16
17  /**
18   * Verifies that given a position not on the board, returns False.
19   */
20  @Test
21  public void verifyWithinBordersOffYUpperBound() {
22      assertFalse(board.withinBorders(0, 4));
23  }
24
25  /**
26   * Verifies that given a position not on the board, returns False.
27   */
28  @Test
29  public void verifyWithinBordersOffYLowerBound() {
30      assertFalse(board.withinBorders(0, -1));
31  }
```

Part 4

1:

A couple of strategies learned during lectures which can be used to make a good test suite:

Black Box

- Manual Testing
  - After implementing the Freeze (Unfreeze) button, we launched the game.
    - We saw that the button appeared, we can conclude that the code added to PacManUIBuilder displayed the desire bdehavior.
    - Pressing start, the game began, after a few moves, we clicked on the freeze button, and saw that the NPCs did in fact freeze, while everything else remained the same, once again displaying desired behavior.
    - Pressing Freeze again, the NPC's unfroze. This again was an exhibition of desired behavior.
    - We started a new game and pressed start, and after a few moves pressed freeze, we then pressed start to see if the NPCs would start again, but they did not, this was the desired behavior.
    - We started a new game and then pressed start, then freeze, then stop, then start again to see if during a freeze, we could start/stop the game. It was in fact possible, this was not desired behavior, so we had to modify the function.
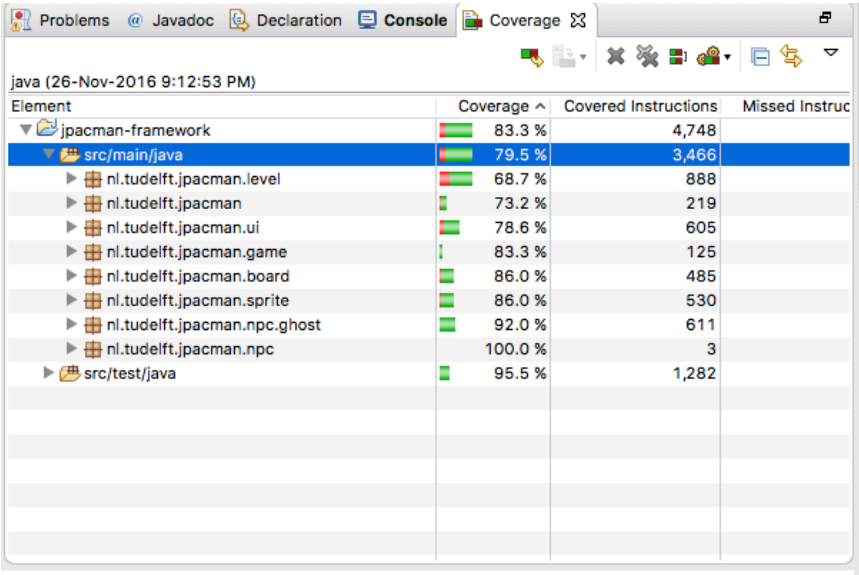
White Box

Code Coverage

Code coverage was done iteratively through the implementation of the feature. After each test was written, tests were made to attain full line and branch coverage. For reference, these tests can be found in: * `/src/test/java/nl/tudelft/jpacman/ui/PacManUIBuilderTest.java` * The entire file is new * Tests were written only for code explicitly added to the class PacManUIBuilder * `/src/test/java/nl/tudelft/jpacman/level/LevelTest.java` * Starting at line 70 * `/src/test/java/nd/tudelft/jpacman/game/GameTest.java` * The entire file is new * Tests were written only for code explicitly added to the class Game

We also wanted to make sure code coverage stayed the same/increase and did not decrease.

Code coverage report before implementation of feature:

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	

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

Problems @ Javadoc Declaration Console Coverage ⊠

Code coverage report after implementation of feature:

Element		Coverage ^	Covered Instructions	Missed Instructions	Total Instructions
▼ 📂 jpacman-framework		84.9 %	5,300	940	6,240
▼ 📂 src/main/java		79.9 %	3,547	893	4,440
▶ ⊞ nl.tudelft.jpacman.level		69.3 %	913	404	1,317
▶ ⊞ nl.tudelft.jpacman		73.2 %	219	80	299
▶ ⊞ nl.tudelft.jpacman.ui		79.5 %	641	165	806
▶ ⊞ nl.tudelft.jpacman.game		83.6 %	143	28	171
▶ ⊞ nl.tudelft.jpacman.sprite		86.0 %	530	86	616
▶ ⊞ nl.tudelft.jpacman.board		86.3 %	487	77	564
▶ ⊞ nl.tudelft.jpacman.npc.ghost		92.0 %	611	53	664
▶ ⊞ nl.tudelft.jpacman.npc		100.0 %	3	0	3

As you can see it increases overall so we are confident in our test coverage.

Mutation

Mutation coverage was run after initial tests pertaining to code coverage were written. We analyzed the mutation coverage pertaining just to the code. For some reason PIT was timing out when trying to run the entire code base through PIT. These results are from specifying the classes individually in pom.xml.

- PacManUIBuilder mutation results

```java
61        /**
62         * Creates a new Pac-Man UI with the set keys and buttons.
63         *
64         * @param game
65         *          The game to build the UI for.
66         * @return A new Pac-Man UI with the set keys and buttons.
67         */
68        public PacManUI build(final Game game) {
69            assert game != null;
70
71   1        if (defaultButtons) {
72   1            addStartButton(game);
73   1            addStopButton(game);
74   1            addFreezeButton(game);
75        }
76   1        return new PacManUI(game, buttons, keyMappings, scoreFormatter);
77    }
78
115       /**
116        * Adds a button with the caption {@value #FREEZE_CAPTION} that FREEZES/UNFREEZES the
117        * game.
118        *
119        * @param game
120        *          The game to freeze/unfreeze.
121        */
122       // easier to test if public
123       // TODO: change to private and change tests
124       public void addFreezeButton(final Game game) {
125           buttons.put(FREEZE_CAPTION, new Action() {
126               @Override
127               public void doAction() {
128   1              game.freeze();
129               }
130           });
131       }
```

```
168    /**
169     * Adds a start, stop and freeze button to the UI. The actual actions for these
170     * buttons will be added upon building the UI.
171     *
172     * @return The builder.
173     */
174    public PacManUiBuilder withDefaultButtons() {
175        defaultButtons = true;
176        buttons.put(START_CAPTION, null);
177        buttons.put(STOP_CAPTION, null);
178        buttons.put(FREEZE_CAPTION, null);
179 1    return this;
```

- Level mutation results

```
97      */
98     public Level(Board b, List<NPC> ghosts, List<Square> startPositions,
99             CollisionMap collisionMap) {
100        assert b != null;
101        assert ghosts != null;
102        assert startPositions != null;
103
104        this.board = b;
105        this.inProgress = false;
106        this.frozen = false;
107        this.npcs = new HashMap<>();
```

```
231    /**
232     * Freezes this level, just stopping NPCs from moving but everything else is the same.
233     *
234     * If already frozen, unfreezes the NPCs.
235     */
236    public void freeze(){
237 1        if (!isInProgress()) {
238            return;
239        }
240 1        if (!frozen){
241 1            stopNPCs();
242            frozen = true;
243            return;
244        }
245 1        startNPCs();
246        frozen = false;
247    }
```

- Game mutation results

```
64     /**
65      * Freezes the game. Player's action, pellets, score board, etc., are not be affected.
66      */
67    public void freeze() {
68        synchronized (progressLock) {
69 1            if (!isInProgress()) {
70                return;
71            }
72 1            getLevel().freeze();
73        }
74    }
```

SPF and Symbolic Execution

To make sure we covered cases we did not think of we thought about using SPF for class methods we wrote. After deeper analysis, we came to the conclusion that our methods are quite simple and do not involve ranges of any kind. All permutations are covered via unit tests already and no class methods accept integers or complex objects of any kind.

2:

As mentioned in 4.1, When manual testing the prototype, we unconvered unexpected behavior with the use of start and stop while the game was frozen, this was considered a bug, and we fixed the bug by adding a check to see if the game was frozen. This was the only bug we came accross through our tests.

The feature we added was quite simple, unit tests, mutation tests and code coverage all returned expected results.

3: What features have you tested? What approaches have you used to improve your tests? The marks will be given based on both your understanding as well as applications of different test strategies and the quality of your implementation and tests.

Features to be tested:

We are only testing one feature, it is the feature we added, the freeze (unfreeze button).

Features not to be tested:

The rest of the program has already been tested, so no new tests will be created for the rest of the program.

Approach:

Regression testing: Our first goal was to ensure that unchanged functionality remained the same as it was before. We re-ran all the previous tests, from parts 1, 2 and 3 to ensure that the unchagned parts of the program remained the same. Once we were sure that the addition of the code for the new feature (freeze/unfreeze) did not introduce any unexpected behavior elsewhere in the program we continued to unit testing.

Unit testing: We decided to ensure that the methods that make up the program each behaved as expected, in accordance to a specification. Having made small modifications to `start` and `stops we added additional tests to ensure these updated functions maintained their desired behavior.

Line/Branch Coverage: The purpose for testing line and branch coverage was that we wanted to make sure our tests cases executed the methods completely, and that no line of code, or branch was left unexecuted by our tests, as this increases the likelyhood of undiscovered bugs, it also implies our tests are inadequate.

Mutation testing: We decided to employ mutation testing techniques to further verify the adequacy of our tests.

Pass/Fail criteria:

If the previous tests pass after the implementation of the new feature, we know that the new feature has not created bugs in other parts of the codebase. This is considered a pass.

Following regression testing, if our unit tests return expected results, we could consider this a pass.

Line and Branch coverage were employed to verify adequacy of our tests, to ensure there weren't any lines of code, or branches of execution that were missed by our test suites. If we achieve 100% line and branch coverage, this is considered a pass.

Mutation Testing was also employed to ensure everything was tested correctly. If we achieved 100% mutation coverage on the new lines we implemented, this would be considered a pass.

Risks and contingencies:

When adding a new feature to an existing code base, care must be taken to ensure that previously tested parts of the codebase continue to behave in accordance with the specification, and continue to pass the tests created for them. When adding the Freeze/Unfreeze button, we needed to ensure that the functionality of Start and Stop did not interfere with Freeze/Unfreeze, and that Freeze/Unfreeze did not interfere with start and stop. Initially, modifications were made to start and stop to check the state of the game, to verify whether it was frozen or not.

Deliverables:

Refer to 4.1 for delieverables.

Staff and responsibilities:

The team consisted of Alec Brunelle and Mohammad Durrani, the required skills were familiarity with Java, Eclipse, Maven, PIT, SPF and Jacoco. We tackled each phase of testing together, the responsibilities were not divided, as we saw it best to have two pairs of eyes, and a diverse set of ideas at each stage in the testing process.

Environmental needs:

Software Needs: Eclipse, Maven, JPF, PIT, Jacoco