

## Problem 1

### 1.1

1. Started the game then stopped the game to see if all sprites stopped
  - a. Test Case: startStop() in levelTest.java
  - b. Result: all sprites stopped as expected
2. Tested to see whether player can control pacman when the game has not been started.
  - a. Test Case: noStart() in levelTest.java
  - b. Result: It didn't allow the player to control the pacman
3. Tested to see if two ghosts can overlap
  - a. Test Case: testReoccupy in OccupantTest.java
  - b. Result: The ghosts were able to overlap and didn't disappear after

### 1.2

src/test/java: 80.5%

CollisionInterctionMap.java 0%

DefaultPlaterInterationMap.java 0%

PacmanConfigurtionException.java 0%

The test cases for these 3 classes are clearly inadequate since for all 3 of these classes 0% of the cases were covered. Instead, they should test all the paths and possible outcomes to try to reach maximum coverage of the classes.

### 1.3

83.5%

we see a slight increase in the percentage of covered cases from 80.5% to 83.5%. We assume that it is because we enabled more tests which in turn covers more test cases of the program.

## Problem 2

### 2.1

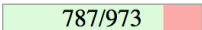
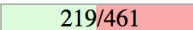
line coverage test - 80.5%

mutation test – noc 36, 48% 219, line coverage 81%

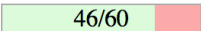
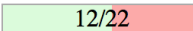
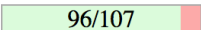
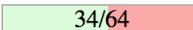
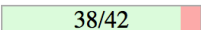
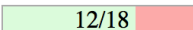
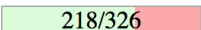
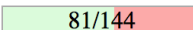
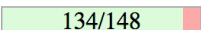
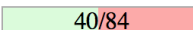
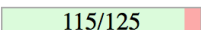
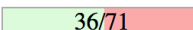
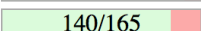
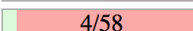
as we can see there is a slight increase of 0.5% coverage for the mutation test as oppose to the line coverage test we did in part 1.

## Pit Test Coverage Report

### Project Summary

Number of Classes	Line Coverage	Mutation Coverage
36	81% 	48% 

### Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">nl.tudelft.jpacman</a>	1	77% 	55% 
<a href="#">nl.tudelft.jpacman.board</a>	5	90% 	53% 
<a href="#">nl.tudelft.jpacman.game</a>	3	90% 	67% 
<a href="#">nl.tudelft.jpacman.level</a>	9	67% 	56% 
<a href="#">nl.tudelft.jpacman.npc.ghost</a>	7	91% 	48% 
<a href="#">nl.tudelft.jpacman.sprite</a>	5	92% 	51% 
<a href="#">nl.tudelft.jpacman.ui</a>	6	85% 	7% 

### 2.2

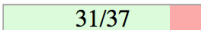
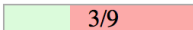
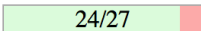
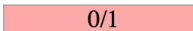
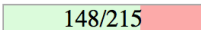
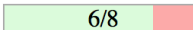
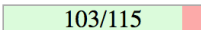
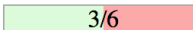
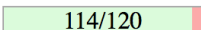
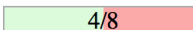
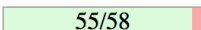
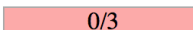
Conditional Boundary Mutator– number of classes 17, 83% and 46%

## Pit Test Coverage Report

### Project Summary

Number of Classes	Line Coverage	Mutation Coverage
17	83% 	46% 

### Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">nl.tudelft.jpacman.board</a>	2	84% 	33% 
<a href="#">nl.tudelft.jpacman.game</a>	1	89% 	0% 
<a href="#">nl.tudelft.jpacman.level</a>	3	69% 	75% 
<a href="#">nl.tudelft.jpacman.npc.ghost</a>	5	90% 	50% 
<a href="#">nl.tudelft.jpacman.sprite</a>	4	95% 	50% 
<a href="#">nl.tudelft.jpacman.ui</a>	2	95% 	0% 

Increments Methods – noc 13, 81% 67%

# Pit Test Coverage Report

## Project Summary

Number of Classes	Line Coverage	Mutation Coverage
13	81% <div><div>372/459</div></div>	67% <div><div>16/24</div></div>

## Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">nl.tudelft.jpacman.board</a>	2	84% <div><div>31/37</div></div>	60% <div><div>3/5</div></div>
<a href="#">nl.tudelft.jpacman.level</a>	3	69% <div><div>148/215</div></div>	78% <div><div>7/9</div></div>
<a href="#">nl.tudelft.jpacman.npc.ghost</a>	4	90% <div><div>85/94</div></div>	60% <div><div>3/5</div></div>
<a href="#">nl.tudelft.jpacman.sprite</a>	2	96% <div><div>53/55</div></div>	100% <div><div>2/2</div></div>
<a href="#">nl.tudelft.jpacman.ui</a>	2	95% <div><div>55/58</div></div>	33% <div><div>1/3</div></div>

Math Mutator – noc 10, 95%, 47%

# Pit Test Coverage Report

## Project Summary

Number of Classes	Line Coverage	Mutation Coverage
10	95% <div><div>299/314</div></div>	47% <div><div>16/34</div></div>

## Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">nl.tudelft.jpacman.board</a>	1	100% <div><div>19/19</div></div>	100% <div><div>6/6</div></div>
<a href="#">nl.tudelft.jpacman.level</a>	3	94% <div><div>120/128</div></div>	83% <div><div>5/6</div></div>
<a href="#">nl.tudelft.jpacman.npc.ghost</a>	1	94% <div><div>15/16</div></div>	0% <div><div>0/1</div></div>
<a href="#">nl.tudelft.jpacman.sprite</a>	4	95% <div><div>114/120</div></div>	33% <div><div>5/15</div></div>
<a href="#">nl.tudelft.jpacman.ui</a>	1	100% <div><div>31/31</div></div>	0% <div><div>0/6</div></div>

Compare and Explain

As we can see, Math mutator has the highest percentage of 95% and Increments mutator has the highest mutation coverage of 67%.

2.3

Test added:

```
@Test
public void aliveTest() {
    assertFalse(level.isAnyPlayerAlive());
}
```

81%, 48% 220

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
36	81% <div><div>792/973</div></div>	48% <div><div>220/461</div></div>

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">nl.tudelft.jpacman</a>	1	77% <div><div>46/60</div></div>	55% <div><div>12/22</div></div>
<a href="#">nl.tudelft.jpacman.board</a>	5	90% <div><div>96/107</div></div>	53% <div><div>34/64</div></div>
<a href="#">nl.tudelft.jpacman.game</a>	3	90% <div><div>38/42</div></div>	67% <div><div>12/18</div></div>
<a href="#">nl.tudelft.jpacman.level</a>	9	68% <div><div>223/326</div></div>	57% <div><div>82/144</div></div>
<a href="#">nl.tudelft.jpacman.npc.ghost</a>	7	91% <div><div>134/148</div></div>	48% <div><div>40/84</div></div>
<a href="#">nl.tudelft.jpacman.sprite</a>	5	92% <div><div>115/125</div></div>	51% <div><div>36/71</div></div>
<a href="#">nl.tudelft.jpacman.ui</a>	6	85% <div><div>140/165</div></div>	7% <div><div>4/58</div></div>

```
public boolean isAnyPlayerAlive() {
    for (Player p : players) {
        if (p.isAlive()) {
            return true;
        }
    }
    return false;
}
```

covered the case where it isAnyPlayerAlive() method returns false;

## Problem 3

### 3.1

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

import static org.junit.Assert.*;
import static org.mockito.Mockito.mock;

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

public class JPFBoardTest {

    private Board board;

    private static final int MAX_X = 4;
    private static final int MAX_Y = 4;
    private static final int MIN_X = 0;
    private static final int MIN_Y = 0;
    private static final int MID_X = (MIN_X + MAX_X) / 2;
    private static final int MID_Y = (MIN_Y + MAX_Y) / 2;

    @Before
    public void setUp() {
        Square[][] grid = new Square[MAX_X - MIN_X + 1][MAX_Y -
MIN_Y + 1];

        for (int i = MIN_X; i <= MAX_X; i++) {
            for (int j = MIN_Y; j <= MAX_Y; j++) {
                grid[i][j] = mock(Square.class);
            }
        }

        board = new Board(grid);
    }

    @Test
    public void WithinBoardTest() {
        assertTrue(board.withinBorders(MID_X, MID_Y));
    }

    @Test
    public void OnMinXTest() {
        assertTrue(board.withinBorders(MIN_X, MID_Y));
    }

    @Test
    public void OneLessThanMinXTest() {
        assertFalse(board.withinBorders(MIN_X - 1, MID_Y));
    }

    @Test
    public void OneMoreThanMinXTest() {
        assertTrue(board.withinBorders(MIN_X + 1, MID_Y));
    }
}
```

```

    }

    @Test
    public void OnMaxXTest() {
        assertTrue(board.withinBorders(MAX_X, MID_Y));
    }

    @Test
    public void OneLessThanMaxXTest() {
        assertTrue(board.withinBorders(MAX_X - 1, MID_Y));
    }

    @Test
    public void OneMoreThanMaxXTest() {
        assertFalse(board.withinBorders(MAX_X + 1, MID_Y));
    }
// -----
    @Test
    public void OnMinYTest() {
        assertTrue(board.withinBorders(MID_X, MIN_Y));
    }

    @Test
    public void OneLessThanMinYTest() {
        assertFalse(board.withinBorders(MID_X, MIN_Y - 1));
    }

    @Test
    public void OneMoreThanMinYTest() {
        assertTrue(board.withinBorders(MIN_X + 1, MIN_Y + 1));
    }

    @Test
    public void OnMaxYTest() {
        assertTrue(board.withinBorders(MID_X, MAX_Y));
    }

    @Test
    public void OneLessThanMaxYTest() {
        assertTrue(board.withinBorders(MID_X, MAX_Y - 1));
    }

    @Test
    public void OneMoreThanMaxYTest() {
        assertFalse(board.withinBorders(MID_X, MAX_Y + 1));
    }

//-----left top corner

    //topleftcorner centre
    @Test
    public void leftTopCornerTest() {
        assertTrue(board.withinBorders(MIN_X, MAX_Y));
    }

```

```

//topleft North West corner
@Test
public void leftTopCornerNorthWestTest() {
    assertFalse(board.withinBorders(MIN_X - 1, MAX_Y + 1));
}
//topleft North corner
@Test
public void leftTopCornerNorthTest() {
    assertFalse(board.withinBorders(MIN_X, MAX_Y + 1));
}
//topleft North East corner
@Test
public void leftTopCornerNorthEastTest() {
    assertFalse(board.withinBorders(MIN_X + 1, MAX_Y + 1));
}
//topleft West corner
@Test
public void leftTopCornerWestTest() {
    assertFalse(board.withinBorders(MIN_X - 1, MAX_Y));
}
//topleft East corner
@Test
public void leftTopCornerEastTest() {
    assertTrue(board.withinBorders(MIN_X + 1, MAX_Y));
}
//topleftcorner South West
@Test
public void leftTopCornerSouthWestTest() {
    assertFalse(board.withinBorders(MIN_X - 1, MAX_Y - 1));
}
//topleftcorner South
@Test
public void leftTopCornerSouthTest() {
    assertTrue(board.withinBorders(MIN_X, MAX_Y - 1));
}
//topleftcorner South East
@Test
public void leftTopCornerSouthEastTest() {
    assertTrue(board.withinBorders(MIN_X + 1, MAX_Y - 1));
}

//-----Right top corner

//toprightcorner centre
@Test
public void rightTopCornerTest() {
    assertTrue(board.withinBorders(MAX_X, MAX_Y));
}
//topright North West corner
@Test
public void rightTopCornerNorthWestTest() {
    assertFalse(board.withinBorders(MAX_X - 1, MAX_Y + 1));
}
//topright North corner

```

```

@Test
public void rightTopCornerNorthTest() {
    assertFalse(board.withinBorders(MAX_X, MAX_Y + 1));
}
//topright North East corner
@Test
public void rightTopCornerNorthEastTest() {
    assertFalse(board.withinBorders(MAX_X + 1, MAX_Y + 1));
}
//topright West corner
@Test
public void rightTopCornerWestTest() {
    assertTrue(board.withinBorders(MAX_X - 1, MAX_Y));
}
//topright East corner
@Test
public void rightTopCornerEastTest() {
    assertFalse(board.withinBorders(MAX_X + 1, MAX_Y));
}
//toprightcorner South West
@Test
public void rightTopCornerSouthWestTest() {
    assertTrue(board.withinBorders(MAX_X - 1, MAX_Y - 1));
}
//toprightcorner South
@Test
public void rightTopCornerSouthTest() {
    assertTrue(board.withinBorders(MAX_X, MAX_Y - 1));
}
//toprightcorner South East
@Test
public void rightTopCornerSouthEastTest() {
    assertFalse(board.withinBorders(MAX_X + 1, MAX_Y - 1));
}

//-----left bottom corner

//bottomleftcorner centre
@Test
public void leftBottomCornerTest() {
    assertTrue(board.withinBorders(MIN_X, MIN_Y));
}
//bottomleft North West corner
@Test
public void leftBottomCornerNorthWestTest() {
    assertFalse(board.withinBorders(MIN_X - 1, MIN_Y + 1));
}
//bottomleft North corner
@Test
public void leftBottomCornerNorthTest() {
    assertTrue(board.withinBorders(MIN_X, MIN_Y + 1));
}
//bottomleft North East corner
@Test

```



```

public void leftBottomCornerNorthEastTest() {
    assertTrue(board.withinBorders(MIN_X + 1, MIN_Y + 1));
}
//bottomleft West corner
@Test
public void leftBottomCornerWestTest() {
    assertFalse(board.withinBorders(MIN_X - 1, MIN_Y));
}
//bottomleft East corner
@Test
public void leftBottomCornerEastTest() {
    assertTrue(board.withinBorders(MIN_X + 1, MIN_Y));
}
//bottomleftcorner South West
@Test
public void leftBottomCornerSouthWestTest() {
    assertFalse(board.withinBorders(MIN_X - 1, MIN_Y - 1));
}
//bottomleftcorner South
@Test
public void leftBottomCornerSouthTest() {
    assertFalse(board.withinBorders(MIN_X, MIN_Y - 1));
}
//bottomleftcorner South East
@Test
public void leftBottomCornerSouthEastTest() {
    assertFalse(board.withinBorders(MIN_X + 1, MIN_Y - 1));
}

//-----right bottom corner

//bottomrightcorner centre
@Test
public void rightBottomCornerTest() {
    assertTrue(board.withinBorders(MAX_X, MIN_Y));
}
//bottomright North West corner
@Test
public void rightBottomCornerNorthWestTest() {
    assertTrue(board.withinBorders(MAX_X - 1, MIN_Y + 1));
}
//bottomright North corner
@Test
public void rightBottomCornerNorthTest() {
    assertTrue(board.withinBorders(MAX_X, MIN_Y + 1));
}
//bottomright North East corner
@Test
public void rightBottomCornerNorthEastTest() {
    assertFalse(board.withinBorders(MAX_X + 1, MIN_Y + 1));
}
//bottomright West corner
@Test
public void rightBottomCornerWestTest() {

```

```

    assertTrue(board.withinBorders(MAX_X - 1, MIN_Y));
}
//bottomright East corner
@Test
public void rightBottomCornerEastTest() {
    assertFalse(board.withinBorders(MAX_X + 1, MIN_Y));
}
//bottomrightcorner South West
@Test
public void rightBottomCornerSouthWestTest() {
    assertFalse(board.withinBorders(MAX_X - 1, MIN_Y - 1));
}
//bottomrightcorner South
@Test
public void rightBottomCornerSouthTest() {
    assertFalse(board.withinBorders(MAX_X, MIN_Y - 1));
}
//bottomrightcorner South East
@Test
public void rightBottomCornerSouthEastTest() {
    assertFalse(board.withinBorders(MAX_X + 1, MIN_Y - 1));
}
}

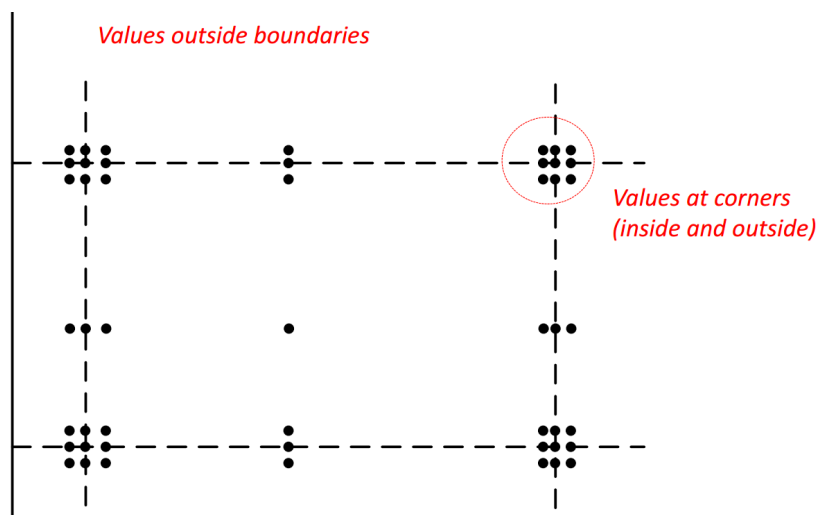
```

### 3.2

We used the Robust worst-case boundary-value test cases which includes every corner and the 8 squares surrounding it. We also tested within boundary min y and max y and the squares above and bottom of them. And min x and max x and the squares on the right and on the left of it. In total we had 49 test cases which covers all boundary values. Here is a picture of the cases we tested.

After we ran JPF, it generated cases (0,0), (0,1), (0, -100000), (1, -2147483648) and (-1000000, -2147483648). These cases only cover extreme scenarios.

Result = no errors detected. The test suite that was generated by JPF does not cover all boundary values because the above 5 cases were the only cases covered by JPF.



## Problem 4

### Test Suites

freezeTest.java

freezeGameTest.java

### 4.2

Yes while we ran line coverage our test suites for the function freeze, we realised that there were NullPointerExceptions which prevented us from freezing the game before the game starts. Hence we had to catch the Exceptions in the test. Since our freeze button is not designed to be used with stop and start, therefore we have to unfreeze before using the functions start and stop. We realised that the sequence of start, freeze, stop, start, freeze causes the NPCs to move abnormally quick which is a clear bug in the program. Another bug we discovered is when we press freeze after we stopped the game and the start and then freeze again, the NPCs start to move abnormally fast. Another bug is that after we press freeze, the only way to unfreeze the game is to press the freeze button again, however while the game state is in freeze, we were able to press stop which causes the player not being able to move as well the NPCs which are still in freeze state. The start and stop function should not be functional when we are in the freeze state. We also realised from debugging that the NPCs moved exceptional fast because the start button was pressed more than once which caused them to move at a faster speed. This is also a bug which should be fixed since after pressing start, pressing start again causes the NPCs to move acceptional quicker.

### 4.3

#### **Test Plan**

##### **Analysis and test items:**

Jpacman game

Freeze fucntion at these Locations:

Assignment2/as2-project/src/main/java/nl/tudelft/jpacman/game/freeze()

Assignment2/as2-project/src/main/java/nl/tudelft/jpacman/level/freeze()

Test suite Locations:

Assignment2/as2-project/src/test/java/nl/tudelft/jpacman/freeze

##### **Features to be tested:**

Freeze function

Unfreeze function

##### **Features not to be tested:**

Everything besides freeze

Direction, win/lose, etc

##### **Approach:**

First we did manual testing. We reviewed to code and started implementing our freeze function. After implenting the function we launched the game and tested out our freeze function on the ui. At first, we started off simple by just pressing

freeze and then freeze again to unfreeze. Then we tested most of the permutations of Start, stop and freeze up to 4 consecutive moves. This is considered black box testing.

Afterwards, we starting doing test scripting. We starting writing our own test suites, they are freezeTest.java freezeGameTest.java to test freezing in the level layer and the game layer, respectively. Which is considered white box testing. Our test suites contains all the possible permutations and outcomes of the freeze start and stop buttons.

We then assessed the quality of our tests using line coverage. We utilised EcEmma to produce a coverage report of our test suites and the freeze function, and based on that we improved our test cases accordingly until we reached full coverage for the freeze function. The coverage report shows that we covered 15 instructions and 0 were missed which computes 100% coverage on the function freeze().

Subsequently, we assess the quality of our testing using mutation testing. We ran the pit test on our new test suites and it showed 100% mutation coverage on our function freeze() which means that all the mutants from freeze() were killed. Which further secures our test suites.

Afterwards, we extended our test suite using symbolic execution. We generated cases that tackle specific path in the function freeze. We symbolically analysed the inputs we need to execute every single possible path in the freeze function.

### **Pass/Fail Criteria**

**Pass Criteria:** The freeze button successfully freezes the NPC but not the user. The user will be allowed to move and around to get points, however, if the user runs into an NPC they will still die. The only way to reverse the freeze action is to press the freeze button again. While start and stop should not interfere with the freeze state.

**Fail Criteria:** If the freeze button fails to freeze the NPC or it freezes the user in any way. If pressing buttons besides the freeze button again can unfreeze the NPCs. If pressing other buttons besides the freeze button can freeze the NPCs. If the NPCs can move again without pressing the freeze button again to unfreeze them.

### **Suspension and resumption criteria**

#### **Suspension Criteria:**

If the freeze function has too many defects: (since the unit tests would not run if the function doesn't work)

If the freeze function fails to freeze the NPCs

If the freeze function freezes the user

If pressing the freeze button does not unfreeze the NPCs

#### **Resumption Criteria:**

Testing is resumed when the functions works well enough to have most of the test cases to pass.

If the freeze button manages to freeze the NPCs while allowing the user to move and pressing the freeze button again unfreezes the NPCs

### **Risks and contingencies**

Risks :

Writing test before code is completed or before code works

Contingency plan:

Write mock codes, or wait until code has been developped.

No access on computer

Contingency plan:

Get access to a computer . ie. CDF comps.

Computer not able to run tests on the code

Contingency plan:

Use another computer that is capable of running the tests. Ie.CDF comps.

### **Deliverables:**

Line coverage report

Pit Reports – mutation report

JPF

Junit test results

### **Task and Schedule:**

November 25<sup>th</sup> : familiarize with the tools and technique – all members

November 27<sup>th</sup>: Part1: assessing quality of tests using line coverage – ECLemma and Jacoco – both members

Nobember 30<sup>th</sup>: Assess quality of tests using mutation coverage – pit test – both members

December 3<sup>rd</sup>: Extend test suite using symbolic execution – JPF – both members

December 4<sup>th</sup>: Implement freeze funciton – both members

December 5<sup>th</sup>: Write test suite for freeze function and test the test suite using methods we learned in class– both members

December 6<sup>th</sup>: finish off report – both members

### **Staff and Responsibility:**

We use the driver/Navigator approach for this task. Both the members are responsible for all the parts while exchanging their position as driver and navigator. Required skill for both members are basic knowledge of java and logic.

**Environmental needs:**

Computer that runs Eclipse, Jacoco, ECL emma, PIT, SPF, maven, java and github.