CSC410, Fall 2016 - Assignment 2

Name: Jinnan Lu

Student Number: 997698807

Lecture: Monday

Due: Tuesday, 6th December 2016 at 23:59

Part1: Line Coverage Test

1. Exploratory Testing

Scenario1: Test "Start" and "Stop" button works as expected.

1. click "Start" button, result: NPCs move
2. click "Start" button, result: NPCs still move
3. click "Stop" button, result: NPCs stop
4. click "Stop" button, result: NPCs keep still

Scenario2: Test "Score" is correct.

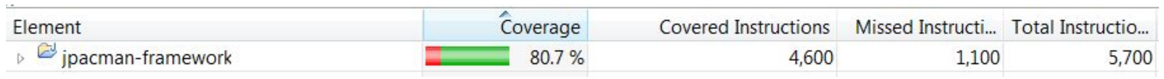
1. click "Start" button, result: NPCs move, score=0
2. press "left" key then click "Stop", result:player moves left by 1 unit, score=10
3. press "left" key then click "Stop", result: player moves left by 1 unit, score=20

Scenario3: Test player can not move to forbidden region

1. click "Start" button, press "up" key twice, result: player change its face up, but does not move
2. press "left key" seven times, result: player moves left by 6 units then keeps still

2 Measure Code Coverage

Following is the code coverage result for jpacman-framework. According to this report, coverage percentage of jpacman-framework is 80.7%.



Element	Coverage	Covered Instructions	Missed Instructi...	Total Instructio...
jpacman-framework	80.7 %	4,600	1,100	5,700

When looking at detail level report, we found three least covered application classes (0 for each of them):

1. nl.tudelft.jpacman.level.CollisionInteractionMap

Test for this class need to be improved. We should add test for the following function

```
public <C1 extends Unit, C2 extends Unit> void onCollision(
                    Class<C1> collider, Class<C2> collidee,
                    CollisionHandler<C1, C2> handler)
```

We need to test that after setting handler can be added, and the added handlers are symmetric.

We also should add test for the following function

```
public <C1 extends Unit, C2 extends Unit> void onCollision(
                    Class<C1> collider, Class<C2> collidee, boolean symetric,
                    CollisionHandler<C1, C2> handler)
```

We need to test that handlers can be added with this function, and the symmetric works as expected.

We also should add test for the following function

```
public <C1 extends Unit, C2 extends Unit> void collide(C1 collider,
                    C2 collidee)
```

We can first add mocks for the handler map, then test that correct handlers are invoked by this function.

2. nl.tudelft.jpacman.level.DefaultPlayerInteractionMap

Test for this class need to be improved. We can add test to the following function by passing different pairs of mocked Units to it. And verify that by calling this function, expected action on those Units are invoked.

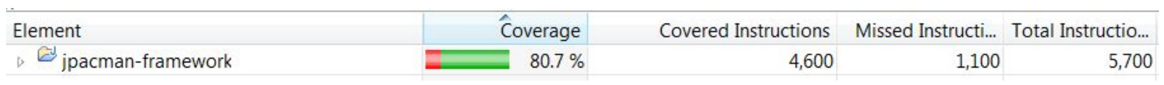
```
public void collide(Unit mover, Unit movedInto)
```

3. nl.tudelft.jpacman.PacmanConfigurationException

This is adequate since the class is just a subclass of RuntimeException, no extra logic in it.

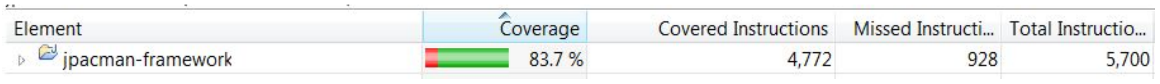
Compare results with/without –ea option

Following is coverage result without –ea option.



Element	Coverage	Covered Instructions	Missed Instructi...	Total Instructio...
jpacman-framework	80.7 %	4,600	1,100	5,700

Following is coverage result with –ea option.

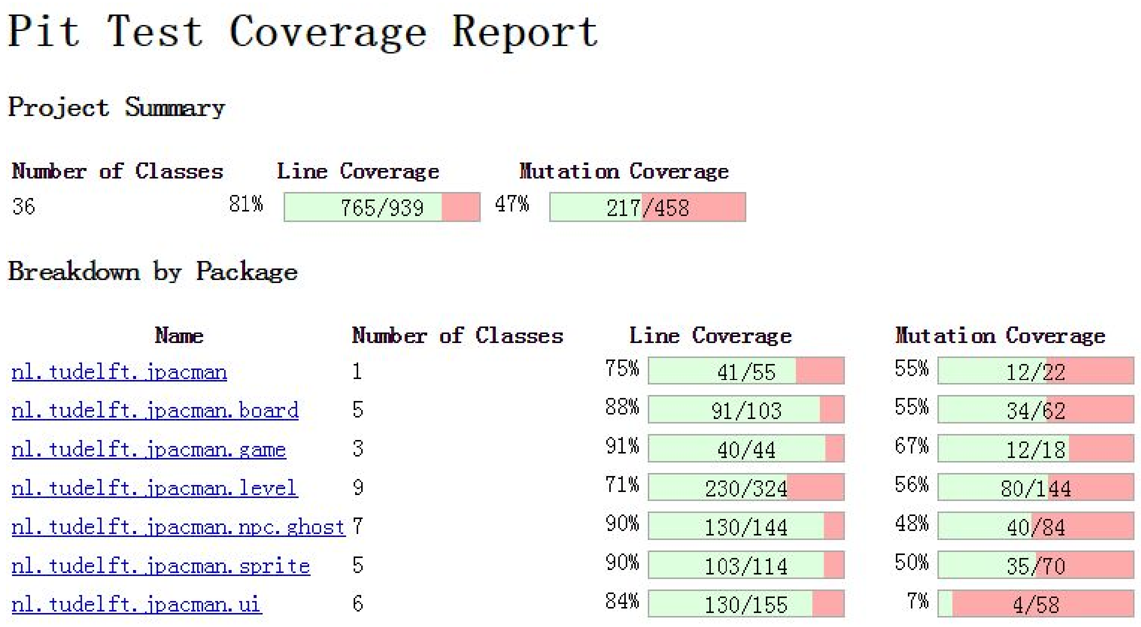


Element	Coverage	Covered Instructions	Missed Instructi...	Total Instructio...
jpacman-framework	83.7 %	4,772	928	5,700

With -ea option, assertions are enabled, so the number covered instructions is increased, and so does the coverage percentage.

Part2: Mutation Test

Following is mutation test result for jpacman-framework.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
36	81% 765/939	47% 217/458

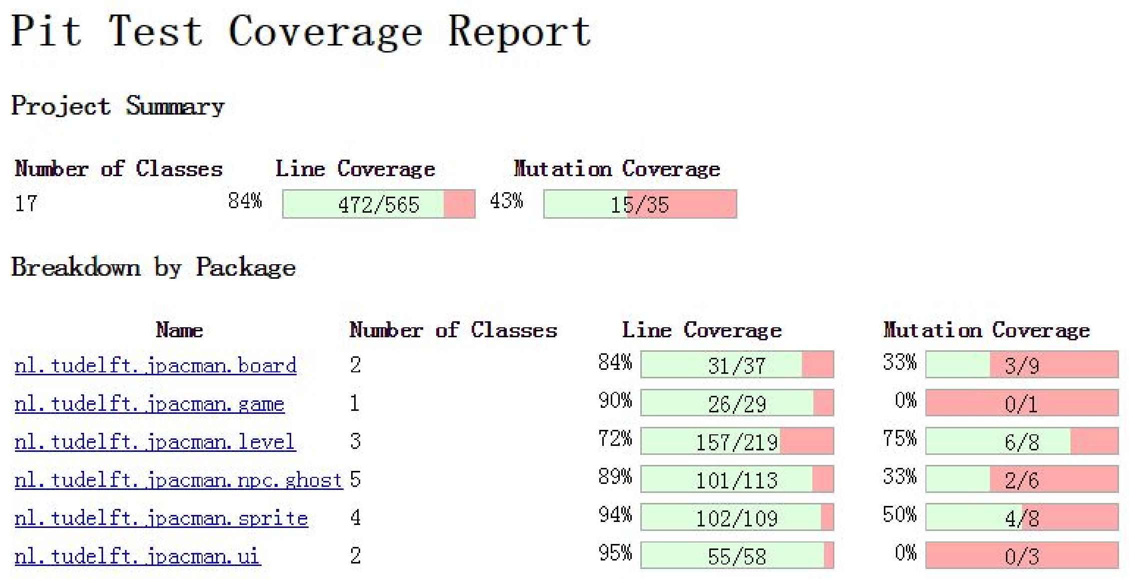
Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
nl.tudelft.jpacman	1	75% 41/55	55% 12/22
nl.tudelft.jpacman.board	5	88% 91/103	55% 34/62
nl.tudelft.jpacman.game	3	91% 40/44	67% 12/18
nl.tudelft.jpacman.level	9	71% 230/324	56% 80/144
nl.tudelft.jpacman.npc.ghost	7	90% 130/144	48% 40/84
nl.tudelft.jpacman.sprite	5	90% 103/114	50% 35/70
nl.tudelft.jpacman.ui	6	84% 130/155	7% 4/58

The line coverage percentage is very close to that given by EclEmma/Jacoco. This is because before running the tests, PIT will perform a traditional line coverage analysis for the tests, then use this data along with the timings of the tests to pick a set of test cases targeted at the mutated code.

Following are test results for several mutators:

With only Conditionals Boundary Mutator



Pit Test Coverage Report

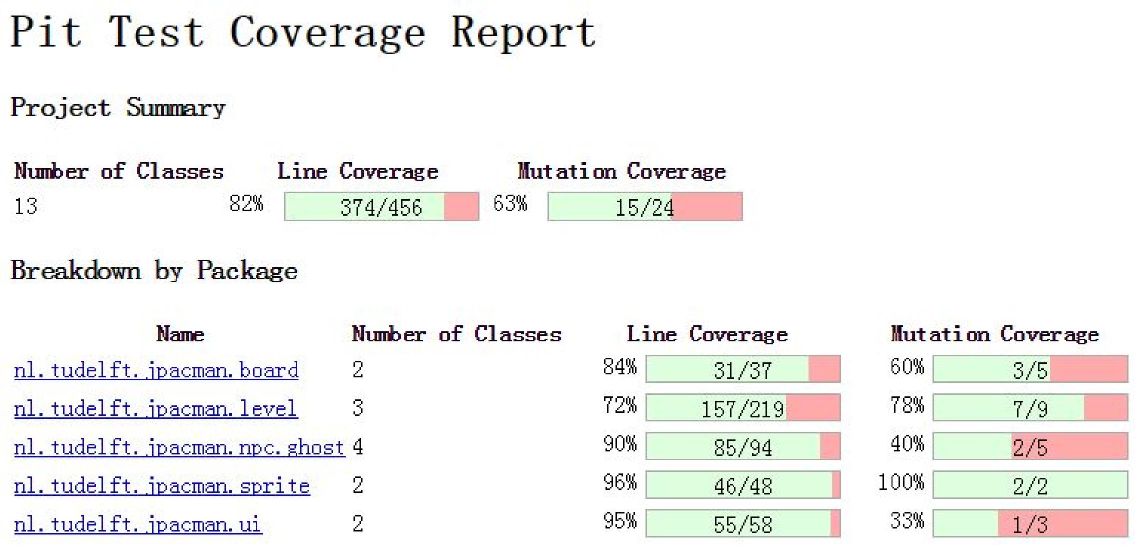
Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
17	84%	472/565	43%	15/35

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.jpacman.board	2	84%	31/37	33%	3/9
nl.tudelft.jpacman.game	1	90%	26/29	0%	0/1
nl.tudelft.jpacman.level	3	72%	157/219	75%	6/8
nl.tudelft.jpacman.npc.ghost	5	89%	101/113	33%	2/6
nl.tudelft.jpacman.sprite	4	94%	102/109	50%	4/8
nl.tudelft.jpacman.ui	2	95%	55/58	0%	0/3

With only Increments Mutator



Pit Test Coverage Report

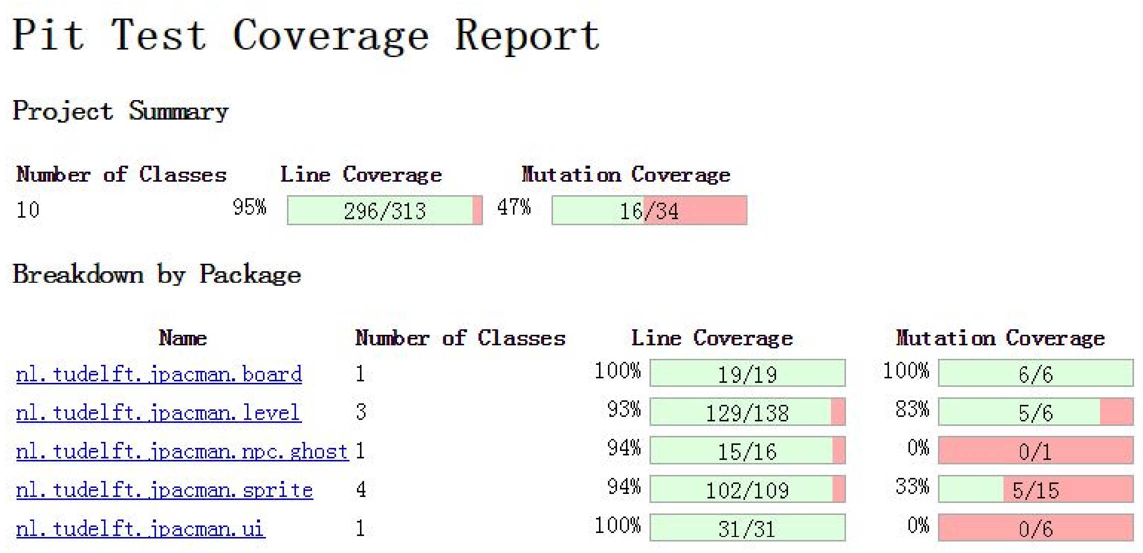
Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
13	82%	374/456	63%	15/24

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.jpacman.board	2	84%	31/37	60%	3/5
nl.tudelft.jpacman.level	3	72%	157/219	78%	7/9
nl.tudelft.jpacman.npc.ghost	4	90%	85/94	40%	2/5
nl.tudelft.jpacman.sprite	2	96%	46/48	100%	2/2
nl.tudelft.jpacman.ui	2	95%	55/58	33%	1/3

With only Math Mutator


Pit Test Coverage Report

Project Summary

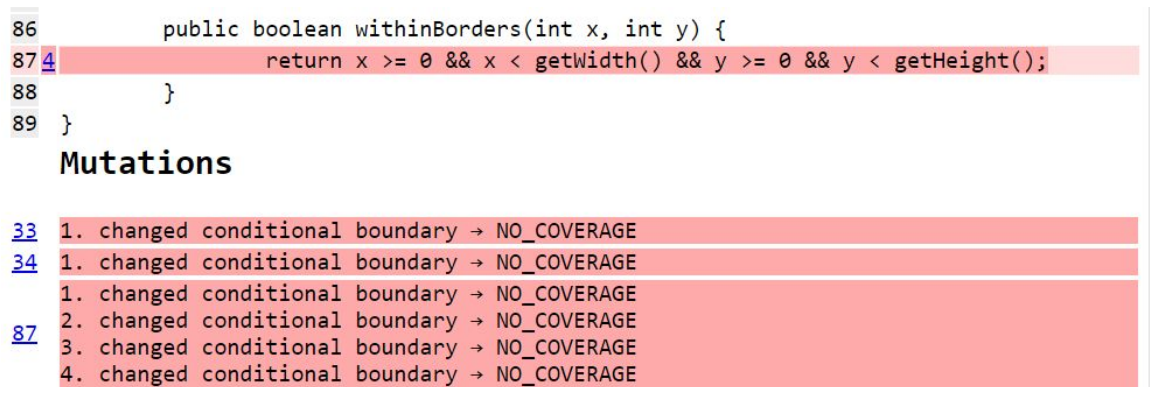
Number of Classes	Line Coverage		Mutation Coverage	
10	95%	296/313	47%	16/34

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.jpacman.board	1	100%	19/19	100%	6/6
nl.tudelft.jpacman.level	3	93%	129/138	83%	5/6
nl.tudelft.jpacman.npc.ghost	1	94%	15/16	0%	0/1
nl.tudelft.jpacman.sprite	4	94%	102/109	33%	5/15
nl.tudelft.jpacman.ui	1	100%	31/31	0%	0/6

From the above results, we can see that PIT will select the classes and lines according to the given mutators, and then apply the mutation. In this project, tests for all of the 3 kinds of mutations need to be improved.

In the detailed report, we can find the following that survived mutation test:


```
86          public boolean withinBorders(int x, int y) {
87 4                return x >= 0 && x < getWidth() && y >= 0 && y < getHeight();
88          }
89  }
```

Mutations

```
33  1. changed conditional boundary → NO_COVERAGE
34  1. changed conditional boundary → NO_COVERAGE
    1. changed conditional boundary → NO_COVERAGE
87  2. changed conditional boundary → NO_COVERAGE
    3. changed conditional boundary → NO_COVERAGE
    4. changed conditional boundary → NO_COVERAGE
```

From above report, we can see that conditional boundary mutation for withinBorders function survives. We can add the following test cases to improve the coverage:

The corner points (0,0), (0, Y_MAX), (X_MAX, 0), (X_MAX, Y_MAX) are within borders

The points (0, Y_MAX+1), (X_MAX+1,0), (-1,0), (0, -1) are out of borders.

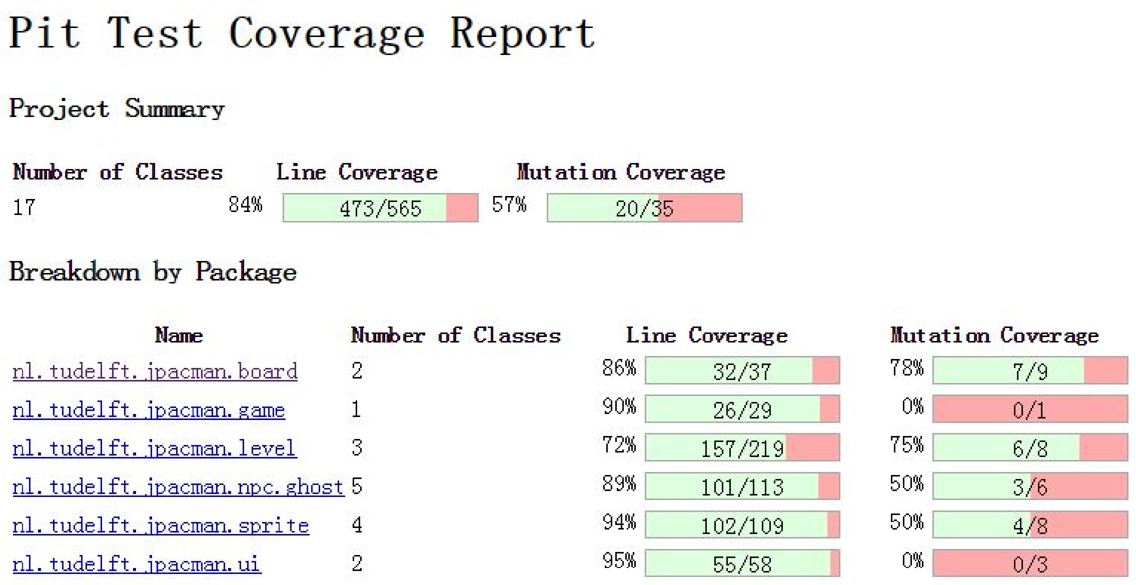
Follow are test cases for the above.

```
    /**
     * Verifies the square at x0y0 is indeed within the borders.
     */
    @Test
    public void verifyX0Y0WithinBorders() {
```

```java
            assertTrue(board.withinBorders(0, 0));
    }
   /**
     * Verifies the square at x0y2 is indeed within the borders.
     */
    @Test
    public void verifyX0Y2WithinBorders() {
            assertTrue(board.withinBorders(0, 2));
    }

    /**
     * Verifies the square at x1y0 is indeed within the borders.
     */
    @Test
    public void verifyX1Y0WithinBorders() {
            assertTrue(board.withinBorders(1, 0));
    }

    /**
     * Verifies the square at x1y2 is indeed within the borders.
     */
    @Test
    public void verifyX1Y2WithinBorders() {
            assertTrue(board.withinBorders(1, 2));
    }

    /**
     * Verifies the square at x0y3 is indeed out of the borders.
     */
    @Test
    public void verifyX0Y3OutOfBorders() {
            assertFalse(board.withinBorders(0, 3));
    }

    /**
     * Verifies the square at x2y0 is indeed out of the borders.
     */
    @Test
    public void verifyX2Y0OutOfBorders() {
            assertFalse(board.withinBorders(2, 0));
    }

    /**
     * Verifies the square at x(-1)y0 is indeed out of the borders.
     */
    @Test
    public void verifyX_1Y0OutOfBorders() {
            assertFalse(board.withinBorders(-1, 0));
    }

    /**
     * Verifies the square at x0y(-1) is indeed out of the borders.
     */
```

```
        @Test
        public void verifyX0Y_1OutOfBorders() {
                assertFalse(board.withinBorders(0, -1));
        }
```

With the above change, the coverage increases (see below).



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
17	84%	473/565	57%	20/35

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.jpacman.board	2	86%	32/37	78%	7/9
nl.tudelft.jpacman.game	1	90%	26/29	0%	0/1
nl.tudelft.jpacman.level	3	72%	157/219	75%	6/8
nl.tudelft.jpacman.npc.ghost	5	89%	101/113	50%	3/6
nl.tudelft.jpacman.sprite	4	94%	102/109	50%	4/8
nl.tudelft.jpacman.ui	2	95%	55/58	0%	0/3

Part 3: Extend Test Suite with Symbolic Execution

Following is jpf file for withinBorders function.

```
# here write your own classpath and un-comment
target = nl.tudelft.jpacman.board.Board
classpath=

symbolic.method= nl.tudelft.jpacman.board.Board.withinBorders(sym#sym)
# listener to print information: PCs, test cases
listener = gov.nasa.jpf.symbc.SymbolicListener
# The following JPF options are usually used for SPF as well:
# no state matching
vm.storage.class=nil
# do not stop at first error
search.multiple_errors=true
```

Following is test cases for withinBorders function. It test points within borders, including corner points, as well as points outside borders.

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

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
```

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

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

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 the square at x0y0 is indeed within the borders.
	 */
	@Test
	public void verifyX0Y0WithinBorders() {
		assertTrue(board.withinBorders(0, 0));
	}

	/**
	 * Verifies the square at x0y1 is indeed within the borders.
	 */
	@Test
	public void verifyX0Y1WithinBorders() {
		assertTrue(board.withinBorders(0, 1));
	}

	/**
	 * Verifies the square at x0y2 is indeed within the borders.
	 */
	@Test
	public void verifyX0Y2WithinBorders() {
		assertTrue(board.withinBorders(0, 2));
	}

	/**
	 * Verifies the square at x1y0 is indeed within the borders.
	 */
	@Test
	public void verifyX1Y0WithinBorders() {
		assertTrue(board.withinBorders(1, 0));
	}

	/**
	 * Verifies the square at x1y1 is indeed within the borders.
	 */
	@Test
	public void verifyX1Y1WithinBorders() {
		assertTrue(board.withinBorders(1, 1));
	}

	/**
	 * Verifies the square at x1y2 is indeed within the borders.
	 */
	@Test
	public void verifyX1Y2WithinBorders() {
		assertTrue(board.withinBorders(1, 2));
	}

	/**
	 * Verifies the square at x0y3 is indeed out of the borders.
	 */
	@Test
	public void verifyX0Y3OutOfBorders() {
		assertFalse(board.withinBorders(0, 3));
	}

	/**
	 * Verifies the square at x2y0 is indeed out of the borders.
	 */
	@Test
	public void verifyX2Y0OutOfBorders() {
		assertFalse(board.withinBorders(2, 0));
	}

	/**
	 * Verifies the square at x(-1)y0 is indeed out of the borders.
	 */
	@Test
	public void verifyX_1Y0OutOfBorders() {
		assertFalse(board.withinBorders(-1, 0));
	}
	/**
	 * Verifies the square at x0y(-1) is indeed out of the borders.
	 */
	@Test
	public void verifyX0Y_1OutOfBorders() {
		assertFalse(board.withinBorders(0, -1));
	}
}
```

Part 4: Freeze/Unfreeze Feature

We implemented this freeze/unfreeze feature by adding a state variable isFreeze to Level class. And before doing the move operation, we will check the isFreeze. If it is true, and the Unit object is a NPC, then we finish the move operation by doing nothing.

To test this new feature, we first verify that the freeze/unfreeze can change the state of Level correctly, and does not interfere with start/stop.

```
/**
	 * Validates the state of the level when it isn't started yet.
	 */
	@Test
	public void noFreeze() {
		assertFalse(level.isFreeze());
	}

	/**
	 * Validates the state of the level when it is unfrozen without freeze.
	 */
	@Test
	public void unfreeze() {
		level.unfreeze();
		assertFalse(level.isFreeze());
	}

	/**
	 * Validates the state of the level when it is frozen.
	 */
	@Test
	public void freeze() {
		level.freeze();
		assertTrue(level.isFreeze());
	}

	/**
	 * Validates the state of the level when it is frozen then unfrozen.
	 */
	@Test
	public void freezeUnfreeze() {
        level.freeze();
        level.unfreeze();
        assertFalse(level.isFreeze());
    }

    /**
     * Validate the freeze feature does not interfere with start/stop
     */
    @Test
    public void freezeAndStart() {
        assertFalse(level.isInProgress());
        assertFalse(level.isFreeze());
        level.start();
        assertTrue(level.isInProgress());
        assertFalse(level.isFreeze());
        level.freeze();
        assertTrue(level.isInProgress());
        assertTrue(level.isFreeze());
        level.unfreeze();
        assertTrue(level.isInProgress());
        assertFalse(level.isFreeze());
        level.stop();
        assertFalse(level.isInProgress());
        assertFalse(level.isFreeze());
    }
```

Then we verify that 1. when in freeze state, NPCs can not move, 2. when in unfreeze state, NPCs can move, 3. when in freeze state, player can move.

```
    /**
     * Verify that when freezed, NPC can not move
     */
    @Test
    public void NpcCannotMoveWhenFreeze() {
        level.start();
        level.freeze();
        NPC npc = mock(NPC.class);
        Square loc = mock(Square.class);
        Square dest = mock(Square.class);
        Mockito.doReturn(loc).when(npc).getSquare();
        Mockito.doReturn(dest).when(loc).getSquareAt(Mockito.any());
        Mockito.doReturn(true).when(dest).isAccessibleTo(Mockito.any());
        Mockito.doReturn(new ArrayList<Unit>()).when(dest).getOccupants();
        Mockito.doNothing().when(npc).occupy(Mockito.any());
        level.move(npc, Direction.NORTH);
        Mockito.verify(npc, Mockito.times(0)).occupy(dest);
    }

    /**
     * Verify that when unfreezed, NPC can move
     */
    @Test
```

```java
	public void NpcCanMoveWhenUnfreeze() {
		level.start();
		level.freeze();
		level.unfreeze();
		NPC npc = mock(NPC.class);
		Square loc = mock(Square.class);
		Square dest = mock(Square.class);
		Mockito.doReturn(loc).when(npc).getSquare();
		Mockito.doReturn(dest).when(loc).getSquareAt(Mockito.any());
		Mockito.doReturn(true).when(dest).isAccessibleTo(Mockito.any());
		Mockito.doReturn(new ArrayList<Unit>()).when(dest).getOccupants();
		Mockito.doNothing().when(npc).occupy(Mockito.any());
		level.move(npc, Direction.NORTH);
		Mockito.verify(npc, Mockito.times(1)).occupy(dest);
	}

	/**
	 * Verify that when freezed, player can move
	 */
	@Test
	public void PlayerCanMoveWhenFreeze() {
		level.start();
		level.freeze();
		Player player = mock(Player.class);
		Square loc = mock(Square.class);
		Square dest = mock(Square.class);
		Mockito.doReturn(loc).when(player).getSquare();
		Mockito.doReturn(dest).when(loc).getSquareAt(Mockito.any());
		Mockito.doReturn(true).when(dest).isAccessibleTo(Mockito.any());
		Mockito.doReturn(new ArrayList<Unit>()).when(dest).getOccupants();
		Mockito.doNothing().when(player).occupy(Mockito.any());
		level.move(player, Direction.NORTH);
		Mockito.verify(player, Mockito.times(1)).occupy(dest);
	}
```