Zachary Morgan
CS472

Github Link: https://github.com/zmorgancs/jpacman.git
Task 2.1

Method Tested	Fully Qualified Method Name
createInky()	src/main/java/nl/tudelft/jpacman/npc/ghost/GhostFactory.createInky
isInProgress()	src/main/java/nl/tudelft/jpacman/game/Game.isInProgress
createGround()	src/main/java/nl/tudelft/jpacman/board/BoardFactory.createGround

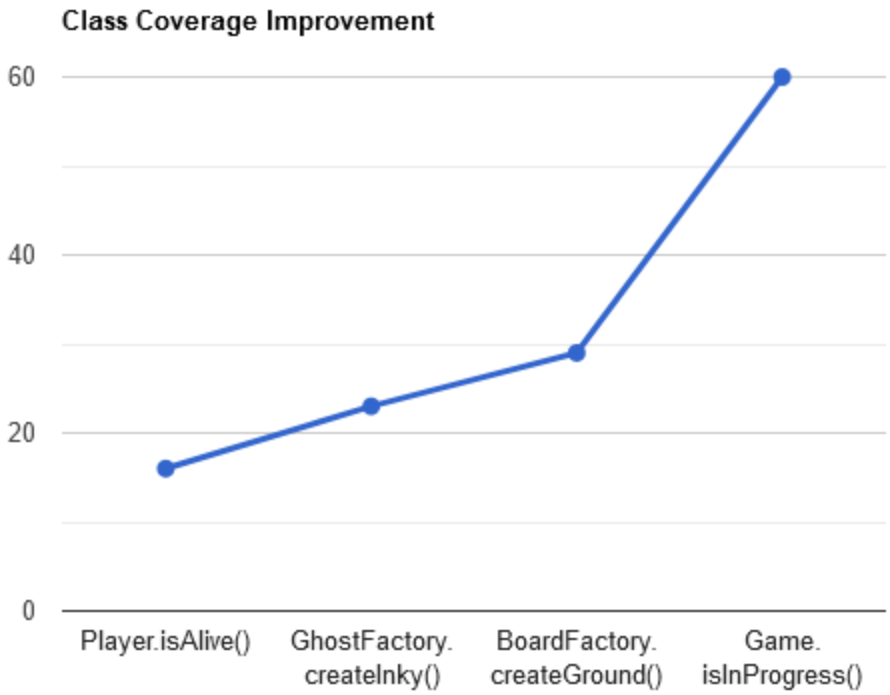
Coverage Scores:
Initial

After adding test for Player.isAlive()

After adding test for GhostFactory.createInky()

After adding test for BoardFactory.createGround()

After adding test for Game.isInProgress()



As the graph shows, I wasn't able to improve the class coverage very much by implementing test cases for the create methods, however the class coverage gains for a method like isInProgress were insanely good in comparison.

Task 3

- **Are the coverage results from JaCoCo similar to the ones you wrote from Intellij in the last task? Why so or why not?**
  The results from both were similar, but certain coverage results were much different which I have to assume is from JaCoCo having more considerations in how it determines the coverage score, resulting in lower percentages in some areas.
- **Did you find helpful the source code visualization from JaCoCo on uncovered branches?**
  Yes, I found it helpful that the visuals succinctly convey which branches/instructions were missed so I can determine whether my test cases need to cover more ground.
- **Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?**
  I felt more comfortable navigating the IntelliJ coverage report than the JaCoCo report while also enjoying the feature set built into IntelliJ. Being able to access and edit source files easily via the result window just makes working on test cases much quicker than having to navigate the JaCoCo result screen.

Code Snippets

GhostFactory.createInky()

```
package nl.tudelft.jpacman.ghost;
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
no usages
public class InkyTest {
    1 usage
    private static GhostFactory Factory = new GhostFactory(new PacManSprites());
    no usages
    @Test
    void makeAnInkyGhost(){
        assertThat( actual: Factory.createInky() != null);
    }
}
```

Game.isInProgress()

```
package nl.tudelft.jpacman.game;

import nl.tudelft.jpacman.Launcher;

import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

no usages
public class gameProgress {
    1 usage
    private static Launcher makeNewGame = new Launcher();
    2 usages
    private Game currentGame = makeNewGame.makeGame();

    no usages
    @Test
    void isGameStillRunning(){
        currentGame.start();
        assertThat(currentGame.isInProgress()).isEqualTo( expected: true);
    }
}
```

BoardFactory.createGround()

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

import nl.tudelft.jpacman.board.BoardFactory;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

no usages
public class makeGround {
    1 usage
    private static BoardFactory Factory = new BoardFactory(new PacManSprites());

    no usages
    @Test
    void canMakeGround(){
        assertThat( actual: Factory.createGround() != null);
    }
}
```

Player.isAlive()

```java
package nl.tudelft.jpacman.level;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;
no usages
public class PlayerTest {
    1 usage
    private static PlayerFactory Factory = new PlayerFactory(new PacManSprites());
    1 usage
    private Player ourPlayer = Factory.createPacMan();
    no usages
    @Test
    void testIfAlive(){
        assertThat(ourPlayer.isAlive()).isEqualTo( expected: true);
    }
}
```