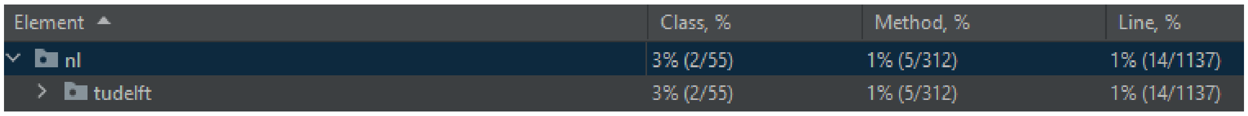
Personal Github Repository: https://github.com/zmorgancs/Group8

Task 2.1

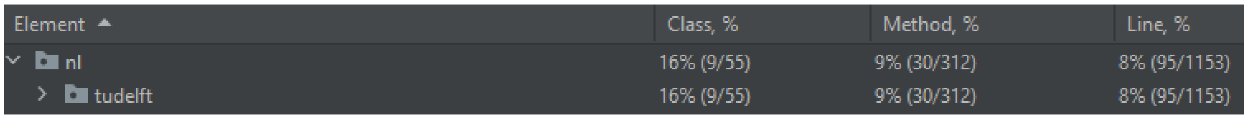
The methods I chose to implement test coverage for were createPinky(), stop() and isInProgress().

Initial Coverage:



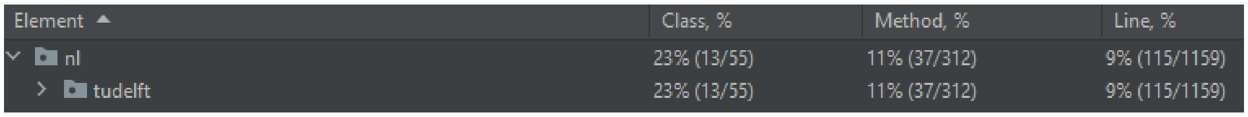
Element	Class, %	Method, %	Line, %
nl	3% (2/55)	1% (5/312)	1% (14/1137)
tudelft	3% (2/55)	1% (5/312)	1% (14/1137)

Coverage After Implementing Player.isAlive() Test:



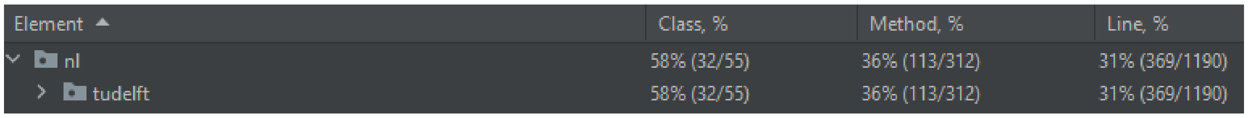
Element	Class, %	Method, %	Line, %
nl	16% (9/55)	9% (30/312)	8% (95/1153)
tudelft	16% (9/55)	9% (30/312)	8% (95/1153)

Coverage After Implementing GhostFactory.createPinky() Test:



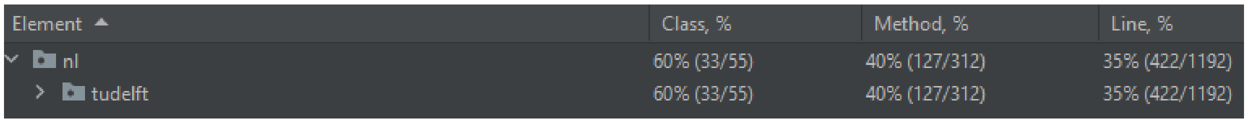
Element	Class, %	Method, %	Line, %
nl	23% (13/55)	11% (37/312)	9% (115/1159)
tudelft	23% (13/55)	11% (37/312)	9% (115/1159)

Coverage After Implementing Game.stop() Test:

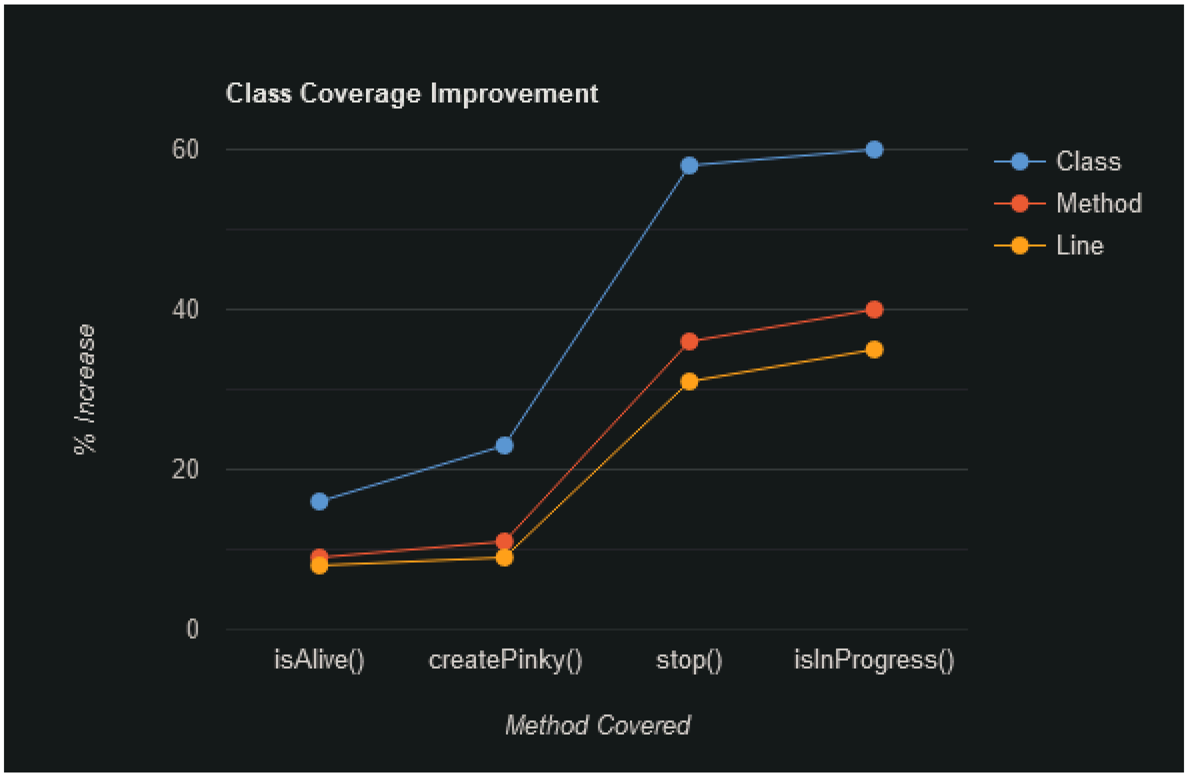


Element	Class, %	Method, %	Line, %
nl	58% (32/55)	36% (113/312)	31% (369/1190)
tudelft	58% (32/55)	36% (113/312)	31% (369/1190)

Coverage After Implementing Game.isInProgress() Test:



Element	Class, %	Method, %	Line, %
nl	60% (33/55)	40% (127/312)	35% (422/1192)
tudelft	60% (33/55)	40% (127/312)	35% (422/1192)

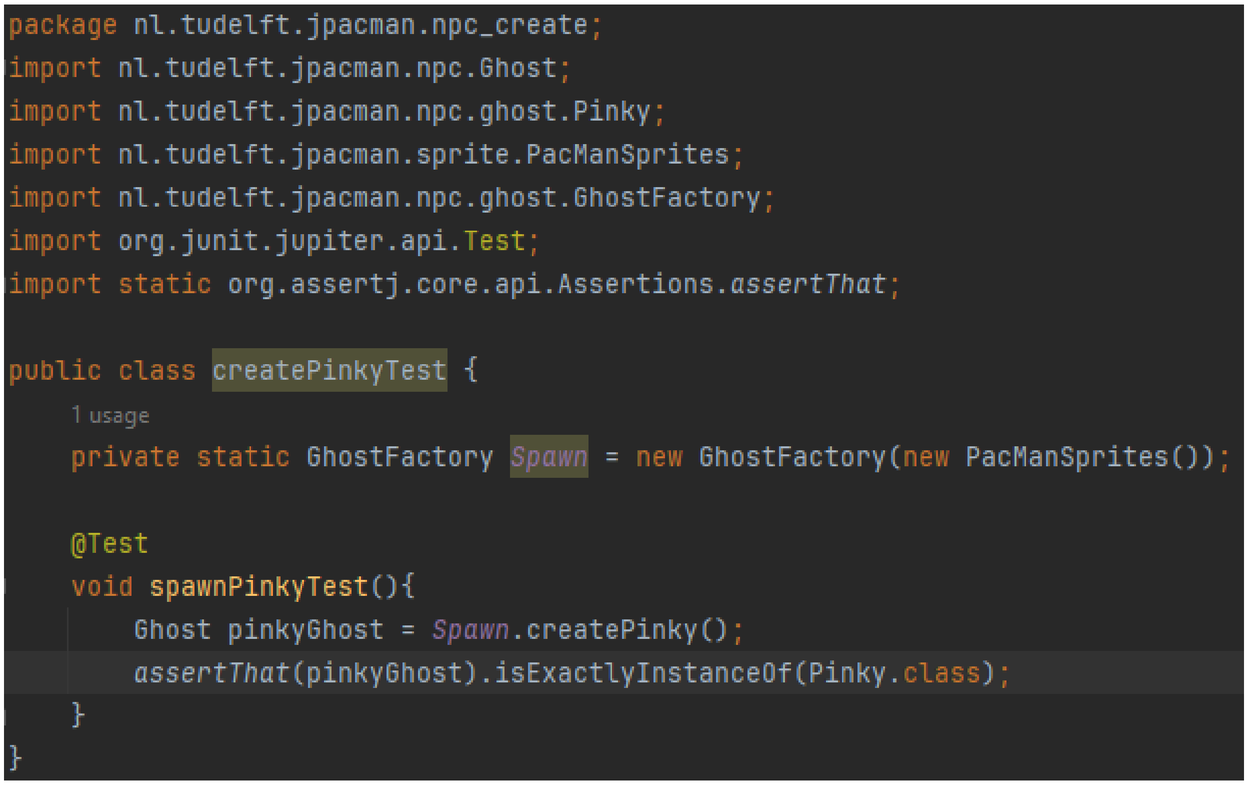


As the graph shows, the bulk of my coverage was obtained with the stop() method unit test, whereas the other methods produced minor coverage gains in comparison.

Task 3

1. **Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?**
   At first I thought the results were very similar, but in analyzing certain coverage results I noted that they could be very different on JaCoCo than on IntelliJ which may be because JaCoCo takes different or more strict considerations in how it determines its overall coverage profile, leading to lower coverage scores in some areas.
2. **Did you find helpful the source code visualization from JaCoCo on uncovered branches?**
   They were helpful because they clearly conveyed where my unit testing was lacking in terms of which branches and instructions I missed, allowing me to determine whether or not I need to add more coverage to my test cases.
3. **Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?**
   I found navigating the JaCoCo report to be more of a hassle than using IntelliJ due to IntelliJ having a better feature set that allowed me to access my source files quickly when I needed to make edits while displaying my results with a coverage panel built right into the IDE, making things more efficient and smooth overall.

Jpacman Code Snippets:
createPinky()

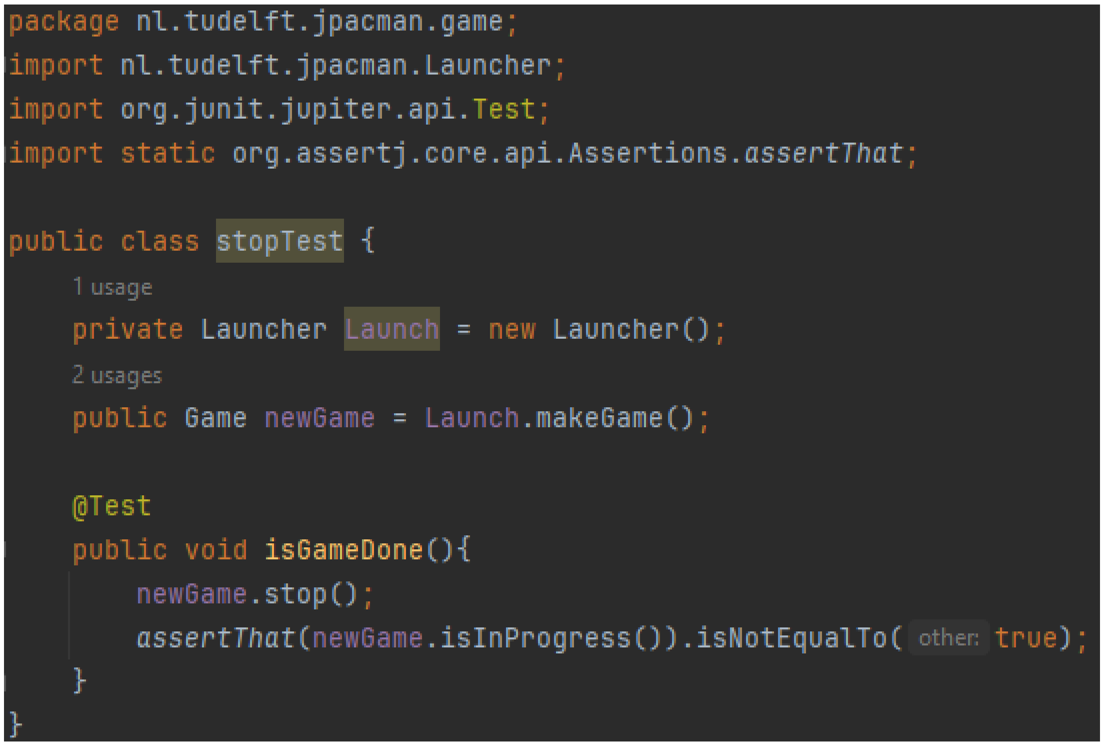


```java
package nl.tudelft.jpacman.npc_create;
import nl.tudelft.jpacman.npc.Ghost;
import nl.tudelft.jpacman.npc.ghost.Pinky;
import nl.tudelft.jpacman.sprite.PacManSprites;
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

public class createPinkyTest {
    1 usage
    private static GhostFactory Spawn = new GhostFactory(new PacManSprites());

    @Test
    void spawnPinkyTest(){
        Ghost pinkyGhost = Spawn.createPinky();
        assertThat(pinkyGhost).isExactlyInstanceOf(Pinky.class);
    }
}
```

stop()

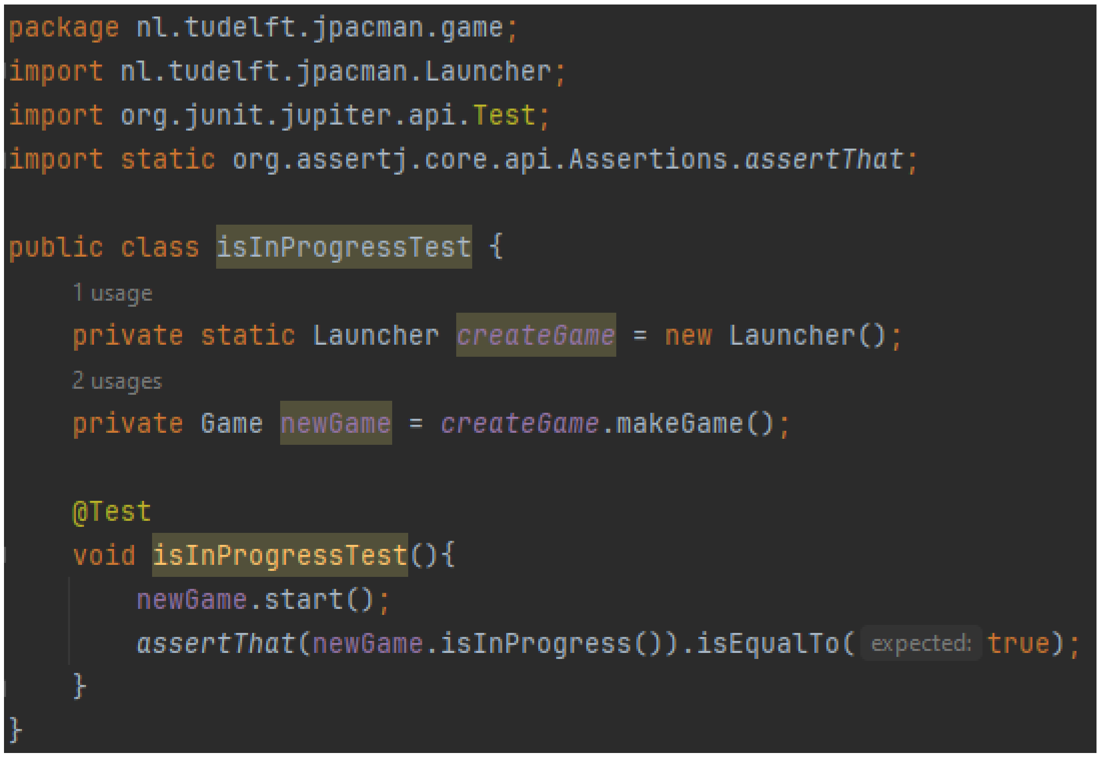


```java
package nl.tudelft.jpacman.game;
import nl.tudelft.jpacman.Launcher;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

public class stopTest {
    1 usage
    private Launcher Launch = new Launcher();
    2 usages
    public Game newGame = Launch.makeGame();

    @Test
    public void isGameDone(){
        newGame.stop();
        assertThat(newGame.isInProgress()).isNotEqualTo( other: true);
    }
}
```

isInProgress()



```java
package nl.tudelft.jpacman.game;
import nl.tudelft.jpacman.Launcher;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

public class isInProgressTest {
    1 usage
    private static Launcher createGame = new Launcher();
    2 usages
    private Game newGame = createGame.makeGame();

    @Test
    void isInProgressTest(){
        newGame.start();
        assertThat(newGame.isInProgress()).isEqualTo( expected: true);
    }
}
```