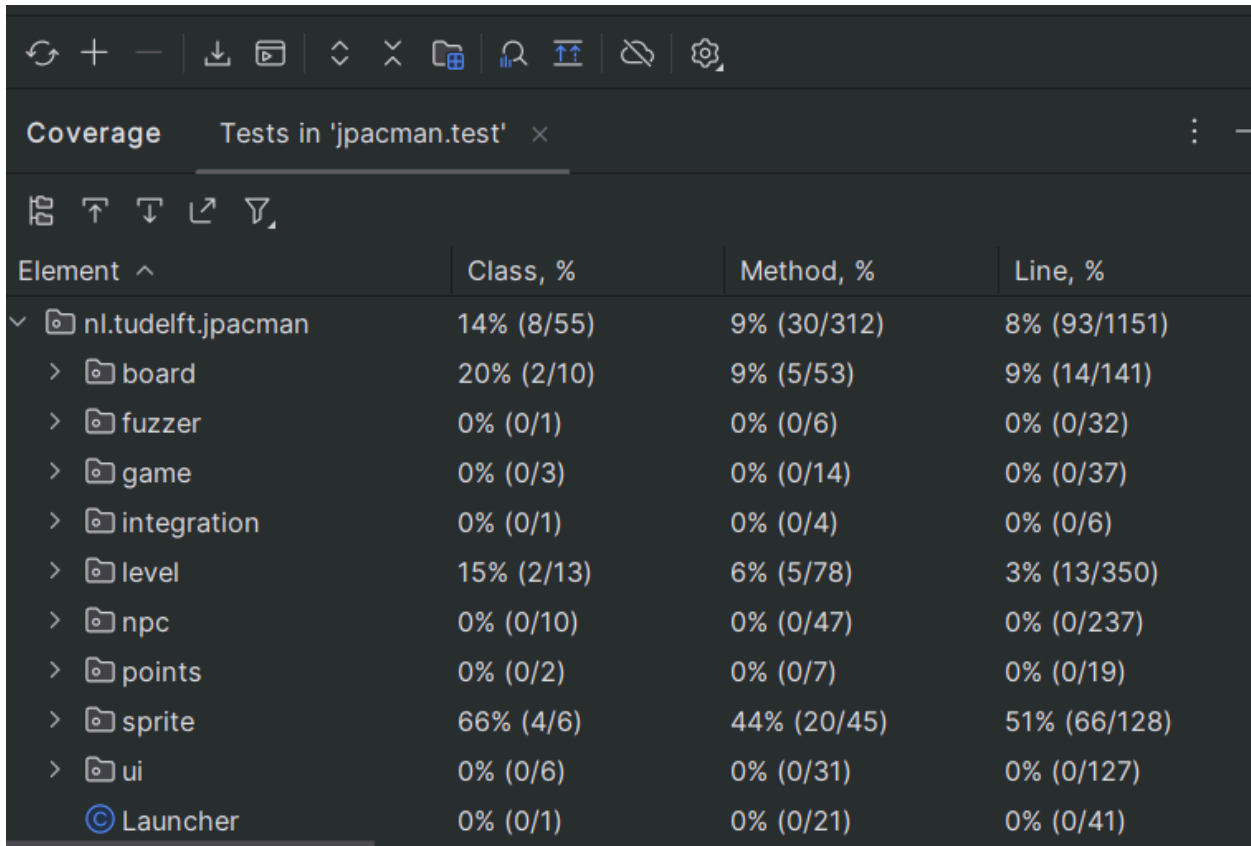


When first taking a look at the Coverage provided by the base code with the `IsAlive()` Test provided, I saw that for the whole repository, it had a Class % of 14%, a Method % of 9%, and a Line % of 8% (Percentages of total coverage).



The screenshot shows a coverage tool interface with a toolbar at the top and a table of results below. The table has four columns: 'Element', 'Class, %', 'Method, %', and 'Line, %'. The 'Element' column lists various packages and classes, including 'nl.tudelft.jpacman', 'board', 'fuzzer', 'game', 'integration', 'level', 'npc', 'points', 'sprite', 'ui', and 'Launcher'. The 'Class, %' column shows coverage percentages and counts (e.g., '14% (8/55)'). The 'Method, %' column shows coverage percentages and counts (e.g., '9% (30/312)'). The 'Line, %' column shows coverage percentages and counts (e.g., '8% (93/1151)'). The 'Launcher' class is highlighted with a blue circle icon.

Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (93/1151)
> board	20% (2/10)	9% (5/53)	9% (14/141)
> fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> game	0% (0/3)	0% (0/14)	0% (0/37)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
> level	15% (2/13)	6% (5/78)	3% (13/350)
> npc	0% (0/10)	0% (0/47)	0% (0/237)
> points	0% (0/2)	0% (0/7)	0% (0/19)
> sprite	66% (4/6)	44% (20/45)	51% (66/128)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
Ⓢ Launcher	0% (0/1)	0% (0/21)	0% (0/41)

I first implemented a test that Checked the use of the method `getKiller()`, however it was a bit more loaded than I thought, due to the fact that I had to create a player and a ghost, for this example I chose Pinky, and I had to have them collide which required me to get the `PlayerCollision` class imported. This significantly raised some of the percentages which can be seen here. Included is a code snippet as well.

```
public class PlayerTestGetKiller{
    /**
     * I prefer to save the instances for this test in particular
     * because it is really a pain to instantiate Player, and I
     * will want to test other methods of Player in here.
     */
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    private Player ThePlayer = Factory.createPacMan();

    GhostFactory Ghostfactory = new GhostFactory(SPRITE_STORE);

    Ghost Pinky = Ghostfactory.createPinky();
    private DefaultPointCalculator point = new DefaultPointCalculator();
    PlayerCollisions playerCollisions = new PlayerCollisions(point);

    @Test
```

```

void testGetKiller(){
    // Lets double check to ensure the Player is alive
    assertThat(ThePlayer.isAlive()).isEqualTo(true);
    ThePlayer.addPoints(1);
    playerCollisions.collide(ThePlayer, Pinky);
    // Now lets Check they have died
    assertThat(ThePlayer.isAlive()).isNotEqualTo(true);
    // Now lets ensure that their killer was pinky

    assertThat(ThePlayer.getKiller().getSprite()).isEqualTo(Pinky.getSprite());

}
}

```

Coverage Tests in 'jpacman.test' x			
Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	25% (14/55)	15% (49/312)	12% (144/1166)
> board	20% (2/10)	11% (6/53)	10% (15/141)
> fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> game	0% (0/3)	0% (0/14)	0% (0/37)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
> level	23% (3/13)	16% (13/78)	10% (37/358)
> npc	40% (4/10)	14% (7/47)	7% (18/243)
> points	50% (1/2)	14% (1/7)	5% (1/20)
> sprite	66% (4/6)	48% (22/45)	57% (73/128)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)

Next was a test Called SetPlayerAlive, which was to test the SetAlive method. This one had a minimal impact and only increased for the line percentage.

```

public class PlayerTestSetAlive{
    /*
     * I prefer to save the instances for this test in particular
     * because it is really a pain to instantiate Player, and I
     * will want to test other methods of Player in here.
     */
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
}

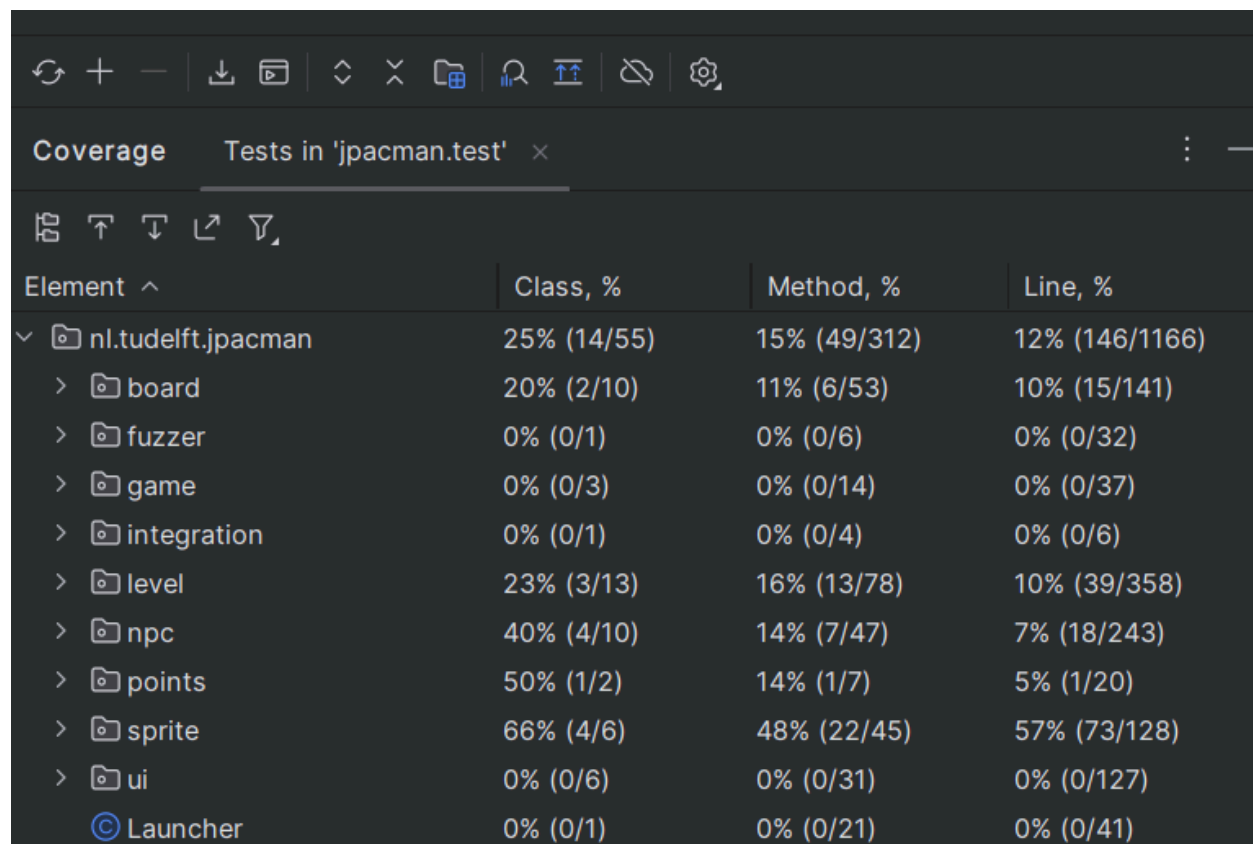
```

```

private PlayerFactory Factory = new PlayerFactory( SPRITE_STORE );
private Player ThePlayer = Factory.createPacMan();

@Test
void testDeath() {
    ThePlayer.setAlive( false );
    assertThat( ThePlayer.isAlive() ).isEqualTo( false );
    assertThat( ThePlayer.isAlive() ).isNotEqualTo( true );
    ThePlayer.setAlive( true );
    assertThat( ThePlayer.isAlive() ).isEqualTo( true );
}
}

```



The screenshot shows the Coverage tool in IntelliJ IDEA, displaying test results for 'jpacman.test'. The table lists various elements and their coverage percentages for Class, Method, and Line.

Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	25% (14/55)	15% (49/312)	12% (146/1166)
> board	20% (2/10)	11% (6/53)	10% (15/141)
> fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> game	0% (0/3)	0% (0/14)	0% (0/37)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
> level	23% (3/13)	16% (13/78)	10% (39/358)
> npc	40% (4/10)	14% (7/47)	7% (18/243)
> points	50% (1/2)	14% (1/7)	5% (1/20)
> sprite	66% (4/6)	48% (22/45)	57% (73/128)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
© Launcher	0% (0/1)	0% (0/21)	0% (0/41)

Lastly included is the AddPointTest where I added points to the players score using the addPoints method then I used the getScore method to ensure the player only had 1 point. This allowed us to check off another method.

```

public class AddPointsTest{
    @Test
    void checkPoints() {
        PacManSprites SPRITE_STORE = new PacManSprites();
        PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
        Player ThePlayer = Factory.createPacMan();

        ThePlayer.addPoints(1);
    }
}

```

```

    assertThat(ThePlayer.getScore()).isEqualTo(1);
}
}

```

Coverage Tests in 'jpacman.test' x			
Element v	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	25% (14/55)	16% (50/312)	12% (147/1166)
⚡ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)
Ⓢ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
Ⓢ Launcher	0% (0/1)	0% (0/21)	0% (0/41)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
> sprite	66% (4/6)	48% (22/45)	57% (73/128)
> points	50% (1/2)	14% (1/7)	5% (1/20)
> npc	40% (4/10)	14% (7/47)	7% (18/243)
> level	23% (3/13)	17% (14/78)	11% (40/358)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
> game	0% (0/3)	0% (0/14)	0% (0/37)

The coverage results from JaCoCo are sorta similar to the ones in IntelliJ, however, they give much more positive results than IntelliJ. However, it helps a lot more when attempting to get information about branch coverage, this is because when I went to find out why `getSprite()` in `level/player` had a 50% branch coverage, I found out there was a test that covered what sprite to get if the player had died, and because we never tested that, we never got to test the death sprite. It was incredibly helpful to be able to see uncovered branches with JaCoCo so that in a later test, I can make sure those branches are covered. I prefer JaCoCo Visually because it allows me to check off many more options and for me personally, it's easier to follow and understand in comparison to IntelliJ.