

Initial coverage











Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
> 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	0% (0/13)	0% (0/78)	0% (0/345)
> npc	0% (0/10)	0% (0/47)	0% (0/237)
> points	0% (0/2)	0% (0/7)	0% (0/19)
> sprite	0% (0/6)	0% (0/45)	0% (0/119)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
Ⓢ Launcher	0% (0/1)	0% (0/21)	0% (0/41)
Ⓢ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
Ⓢ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

After PlayerTest example

Coverage Tests in 'jpacman.test' ×			
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)
Ⓢ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
⚡ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)











After testSouth()

```
new *
@Test
void testSouth() {
    Direction south = Direction.valueOf(name: "SOUTH");
    assertThat(south.getDeltaY()).isEqualTo(expected: 1);
}
```

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)
Ⓢ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
⚡ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

After testAddPoints()

```
@Test
void testAddPoints(){
    //test to see if the addPoints function is correctly adding points
    int pointValue = 1;
    ThePlayer.addPoints(pointValue);
    assertThat(ThePlayer.getScore()).isNotEqualTo( other: 0);
}
```

Element ^	Class, %	Method, %	Line, %
✓  nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (94/1129)
>  board	20% (2/10)	7% (4/53)	9% (13/136)
>  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)	7% (6/78)	4% (15/334)
>  npc	0% (0/10)	0% (0/47)	0% (0/236)
>  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)
⦿ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
⦿ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

After testPlayerVersusGhost()

```
//create a collision object
1 usage
private final DefaultPointCalculator ptCalc = new DefaultPointCalculator();
1 usage
private final PlayerCollisions collision = new PlayerCollisions(ptCalc);

//create a ghost object
//use Blinky as the type
1 usage
private final GhostFactory gFactory = new GhostFactory(SPRITE_STORE);
2 usages
private final Ghost TheGhost = gFactory.createBlinky();
```

```
@Test
void testPlayerVersusGhost(){
    collision.playerVersusGhost(ThePlayer, TheGhost);

    //test to see that the player is correctly dead
    assertThat(ThePlayer.isAlive()).isEqualTo(expected: false);

    //test to see if the killer is correctly being set
    assertThat(ThePlayer.getKiller()).isEqualTo(TheGhost);
}
```

Element ^	Class, %	Method, %	Line, %
nl.tudelft.jpacman	25% (14/55)	13% (43/312)	11% (132/1144)
> board	20% (2/10)	7% (4/53)	9% (13/136)
> 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)	12% (10/78)	8% (28/341)
> npc	40% (4/10)	12% (6/47)	6% (17/243)
> 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)
© LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
© PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)











After testPelletSprite()

```
public class PelletTest {
    3 usages
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private final DefaultPointCalculator ptCalc = new DefaultPointCalculator();
    1 usage
    private final GhostFactory gFactory = new GhostFactory(SPRITE_STORE);
    1 usage
    private final LevelFactory Factory = new LevelFactory(SPRITE_STORE, gFactory, ptCalc);
    no usages
    private final Pellet ThePellet = Factory.createPellet();

    @Test
    void testPelletSprite(){
        //create the pellet
        Sprite pelletSprite = SPRITE_STORE.getPelletSprite();
        int pointVal = 10;
        Pellet testPellet = new Pellet(pointVal, pelletSprite);

        //test that the sprite is correctly being set
        assertThat(testPellet.getSprite()).isNotEqualTo( other: null);

        //test that the point values are correctly being set
        assertThat(testPellet.getValue()).isNotEqualTo( other: 0);
    }
}
```

Element ^	Class, %	Method, %	Line, %
✓  nl.tudelft.jpacman	29% (16/55)	16% (52/309)	12% (150/1165)
>  board	20% (2/10)	10% (5/50)	10% (14/138)
>  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	38% (5/13)	21% (17/78)	12% (44/360)
>  npc	40% (4/10)	12% (6/47)	6% (17/243)
>  points	50% (1/2)	14% (1/7)	5% (1/20)
>  sprite	66% (4/6)	51% (23/45)	57% (74/128)
>  ui	0% (0/6)	0% (0/31)	0% (0/127)
© Launcher	0% (0/1)	0% (0/21)	0% (0/41)
© LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
© PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

JaCoCo testing report

jpacman

Sessions

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		68%		58%	72	155	102	344	20	69	4	12
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.ui		77%		47%	54	86	21	144	7	31	0	6
default		0%		0%	12	12	21	21	5	5	1	1
nl.tudelft.jpacman.board		86%		58%	44	93	2	110	0	40	0	7
nl.tudelft.jpacman.sprite		88%		62%	29	70	10	113	5	38	0	5
nl.tudelft.jpacman		69%		25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.points		60%		75%	1	11	5	21	0	9	0	2
nl.tudelft.jpacman.game		87%		60%	10	24	4	45	2	14	0	3
nl.tudelft.jpacman.npc		100%		n/a	0	4	0	8	0	4	0	1
Total	1,201 of 4,694	74%	290 of 637	54%	290	590	226	1,039	50	268	6	47

Created with JaCoCo 0.8.3.201901230119

Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task?

Why so or why not?

- The coverage results are similar to the ones that the Gradle Coverage shows, but the total amount of items that the JaCoCo looks for is less. But, the percentages are relatively the same. The Class% is lower on JaCoCo, but both the Method% and Line% are higher on JaCoCo. This is most likely a result of different classifications of what is considered a line(whitespace, etc.) if interfaces are considered classes, or if the test functions are considered methods.

Did you find helpful the source code visualization from JaCoCo on uncovered branches?

- The source code visualization helps to identify directly which lines/methods/classes are uncovered. This saves a lot of time trying to identify which branches of code are uncovered, and which lines/methods/classes need coverage.

Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?

- I prefer the IntelliJ's coverage window, because it directly integrates with the platform that I am coding on, so I am able to quickly implement a unit test if I see that there is an uncovered line/method/class in the code. However, both do show the percentages and have easy to understand visualizations of the uncovered items or branches.