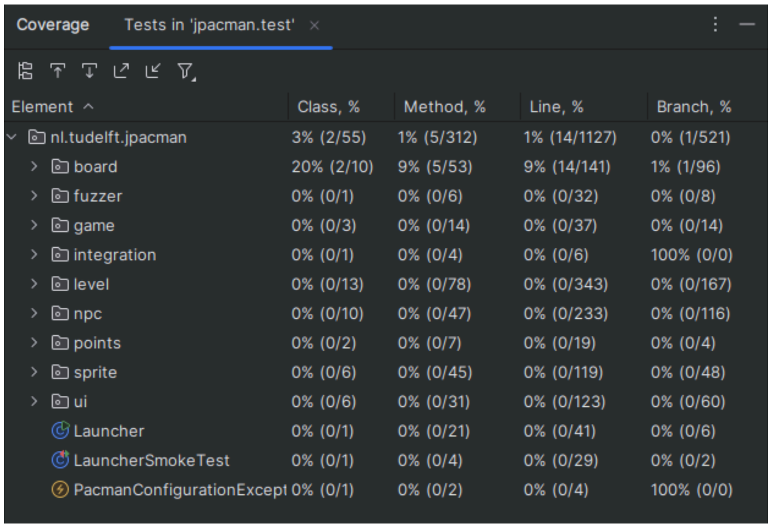
Dynamic Analysis Lab

Kyle Meyer

CS472-1001

September 14, 2024

Task 1 – JPacman Test Coverage



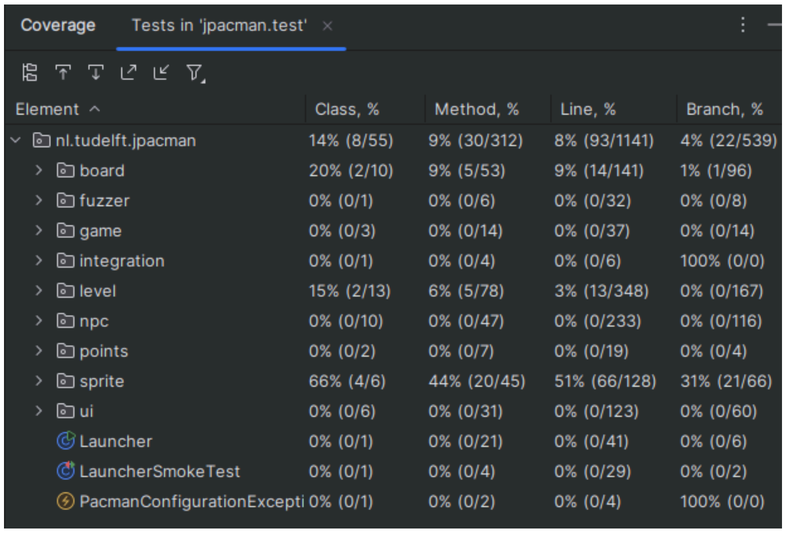
Coverage   Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1127)	0% (1/521)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	0% (0/13)	0% (0/78)	0% (0/343)	0% (0/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	0% (0/6)	0% (0/45)	0% (0/119)	0% (0/48)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationExcept	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

Question: Is the test coverage good enough?

No, the test coverage is not good enough. Most of the code has no test coverage at all. The average amount of coverage is less than 10%. For the test coverage to be good, the amount should exceed 80%.

Task 2 – Increasing Coverage on JPacman



Coverage   Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (93/1141)	4% (22/539)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	15% (2/13)	6% (5/78)	3% (13/348)	0% (0/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	66% (4/6)	44% (20/45)	51% (66/128)	31% (21/66)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationExcepti	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

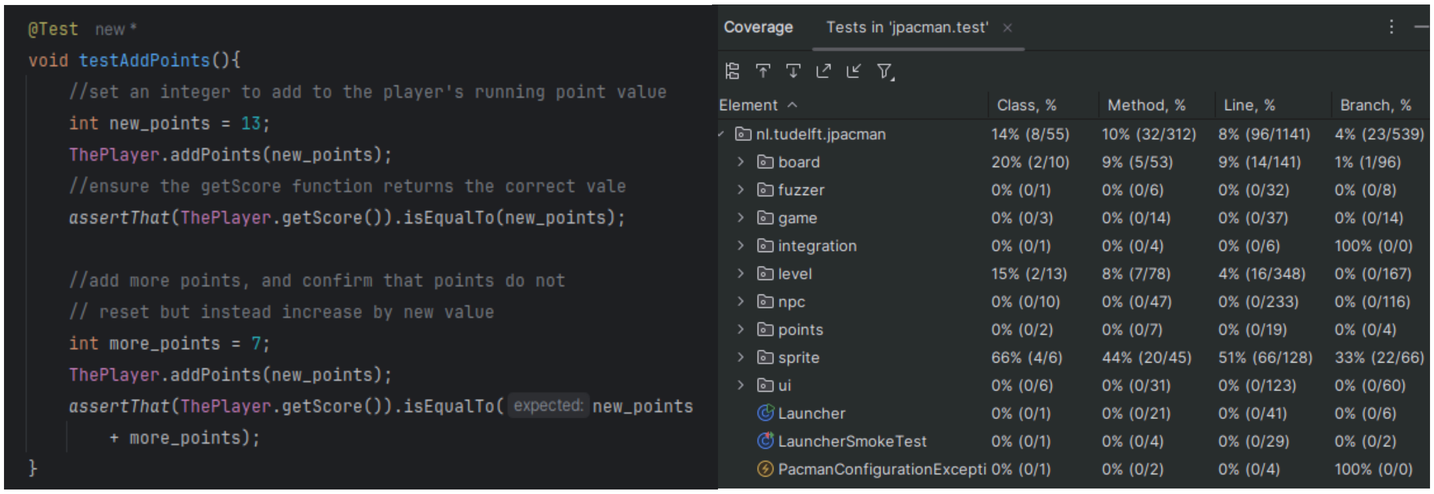
<- Note that after the suggested test implementations, the test coverage of all lines jumped from 0% to 6%.

Task 2.1 – Adding Unit Tests for JPacman

The following are the three methods that I chose to expand test coverage of:

1	src/main/java/nl.tudelft.jpacman/level/Player/Player.addPoints
2	src/main/java/nl.tudelft.jpacman/level/PlayerCollisions/PlayerCollisions.playerVersusGhost
3	src/main/java/nl.tudelft.jpacman/level/PlayerCollisions/PlayerCollisions.playerVersusPellet

The following screenshots show my test code (left) and test coverage analysis (right), in order, for the above methods.

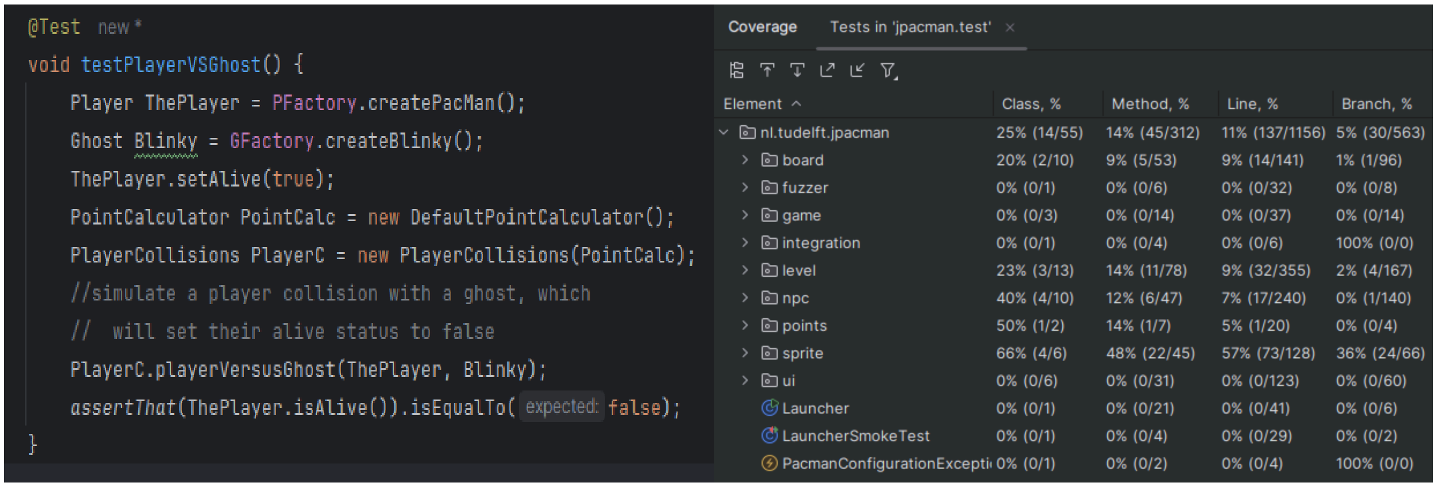


```
@Test  new *
void testAddPoints(){
    //set an integer to add to the player's running point value
    int new_points = 13;
    ThePlayer.addPoints(new_points);
    //ensure the getScore function returns the correct vale
    assertThat(ThePlayer.getScore()).isEqualTo(new_points);

    //add more points, and confirm that points do not
    // reset but instead increase by new value
    int more_points = 7;
    ThePlayer.addPoints(new_points);
    assertThat(ThePlayer.getScore()).isEqualTo( expected: new_points
        + more_points);
}
```

Coverage   Tests in 'jpacman.test'

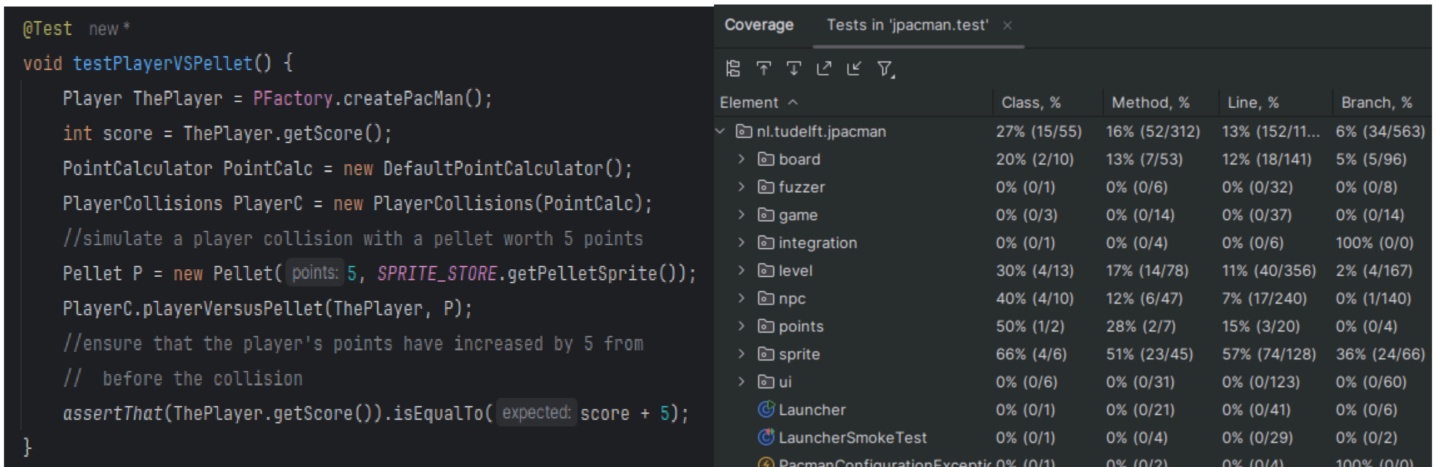
Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	14% (8/55)	10% (32/312)	8% (96/1141)	4% (23/539)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	15% (2/13)	8% (7/78)	4% (16/348)	0% (0/167)
npc	0% (0/10)	0% (0/47)	0% (0/233)	0% (0/116)
points	0% (0/2)	0% (0/7)	0% (0/19)	0% (0/4)
sprite	66% (4/6)	44% (20/45)	51% (66/128)	33% (22/66)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationExcepti	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)



```
@Test  new *
void testPlayerVSGhost() {
    Player ThePlayer = PFactory.createPacMan();
    Ghost Blinky = GFactory.createBlinky();
    ThePlayer.setAlive(true);
    PointCalculator PointCalc = new DefaultPointCalculator();
    PlayerCollisions PlayerC = new PlayerCollisions(PointCalc);
    //simulate a player collision with a ghost, which
    //  will set their alive status to false
    PlayerC.playerVersusGhost(ThePlayer, Blinky);
    assertThat(ThePlayer.isAlive()).isEqualTo( expected: false);
}
```

Coverage   Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	25% (14/55)	14% (45/312)	11% (137/1156)	5% (30/563)
board	20% (2/10)	9% (5/53)	9% (14/141)	1% (1/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	23% (3/13)	14% (11/78)	9% (32/355)	2% (4/167)
npc	40% (4/10)	12% (6/47)	7% (17/240)	0% (1/140)
points	50% (1/2)	14% (1/7)	5% (1/20)	0% (0/4)
sprite	66% (4/6)	48% (22/45)	57% (73/128)	36% (24/66)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationExcepti	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)



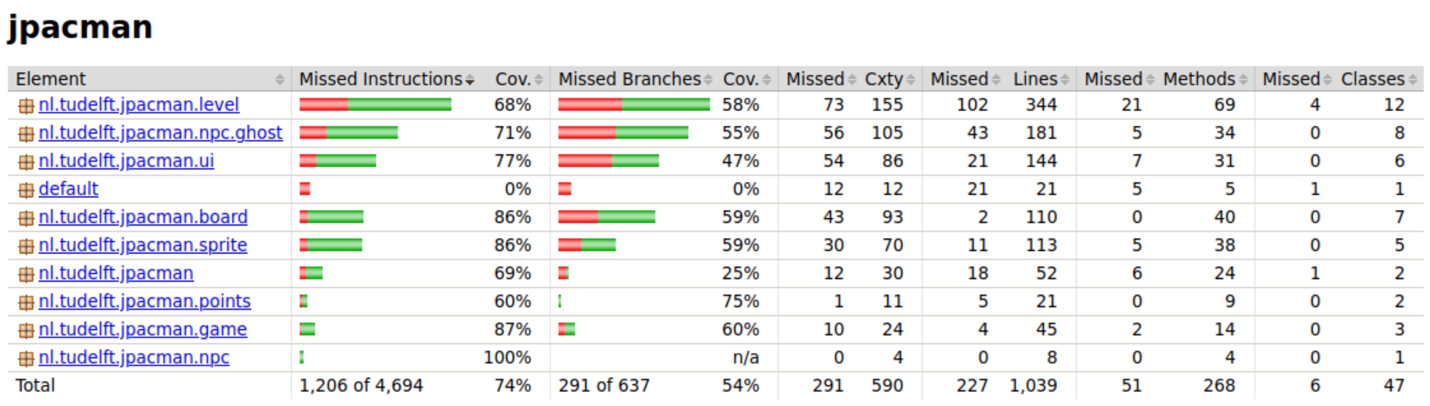
```
@Test  new *
void testPlayerVSPellet() {
    Player ThePlayer = PFactory.createPacMan();
    int score = ThePlayer.getScore();
    PointCalculator PointCalc = new DefaultPointCalculator();
    PlayerCollisions PlayerC = new PlayerCollisions(PointCalc);
    //simulate a player collision with a pellet worth 5 points
    Pellet P = new Pellet( points: 5, SPRITE_STORE.getPelletSprite());
    PlayerC.playerVersusPellet(ThePlayer, P);
    //ensure that the player's points have increased by 5 from
    //  before the collision
    assertThat(ThePlayer.getScore()).isEqualTo( expected: score + 5);
}
```

Coverage   Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %	Branch, %
nl.tudelft.jpacman	27% (15/55)	16% (52/312)	13% (152/11...	6% (34/563)
board	20% (2/10)	13% (7/53)	12% (18/141)	5% (5/96)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)	0% (0/8)
game	0% (0/3)	0% (0/14)	0% (0/37)	0% (0/14)
integration	0% (0/1)	0% (0/4)	0% (0/6)	100% (0/0)
level	30% (4/13)	17% (14/78)	11% (40/356)	2% (4/167)
npc	40% (4/10)	12% (6/47)	7% (17/240)	0% (1/140)
points	50% (1/2)	28% (2/7)	15% (3/20)	0% (0/4)
sprite	66% (4/6)	51% (23/45)	57% (74/128)	36% (24/66)
ui	0% (0/6)	0% (0/31)	0% (0/123)	0% (0/60)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)	0% (0/6)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)	0% (0/2)
PacmanConfigurationExceptic	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

Task 3 – JaCoCo Report on JPacman

JaCoCo Test Coverage Report:



jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		68%		58%	73	155	102	344	21	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%		59%	43	93	2	110	0	40	0	7
nl.tudelft.jpacman.sprite		86%		59%	30	70	11	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	1,206 of 4,694	74%	291 of 637	54%	291	590	227	1,039	51	268	6	47

Question: Are the coverage results from JaCoCo similar to the ones you got from IntelliJ?

No, JaCoCo shows a much higher percentage of coverage than IntelliJ. This is likely because JaCoCo has a different process of determining whether code has been tested, or requires testing at all.

Question: Did you find helpful the source code visualization from JaCoCo?

Yes, JaCoCo's visualization was extremely helpful for discovering uncovered code branches. It is very easy to click the directories, files, and classes to see which code needs more coverage, as it is indicated by a red and green bar diagram.

Question: Which visualization did you prefer and why?

I much preferred the JaCoCo report, because the visual and interactive nature of it was extremely easy to use for analysis of code test coverage. It was very simple to see a mostly-red bar, click the file, and realize which methods need expanded test coverage.