LAB REPORT

In this lab, I conducted unit tests on four separate methods created and used in the jpacman repository. Those unit tests include Player.isAlive(), Ghost.getSprite(), AnimatedSprite.getHeight(), and PacManUiBuilder.build(). For these four methods, I created individual test files that simulated the creation of the class, usage of the class method, and tested the return value against the expected value output.

Player.isAlive()

Before Test

Element	Class, %	Method, %	Line, %
nl	3% (4/110)	1% (10/624)	1% (28/2274)
tudelft	3% (4/110)	1% (10/624)	1% (28/2274)
jpacman	3% (4/110)	1% (10/624)	1% (28/2274)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	0% (0/26)	0% (0/156)	0% (0/690)

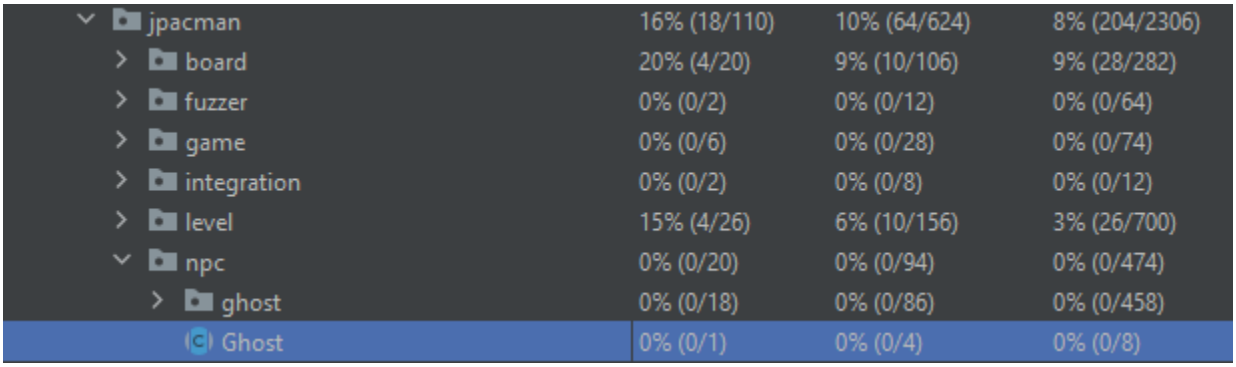
After Test

level	15% (4/26)	6% (10/156)	3% (26/700)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	0% (0/2)	0% (0/17)	0% (0/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	100% (1/1)	25% (2/8)	33% (8/24)

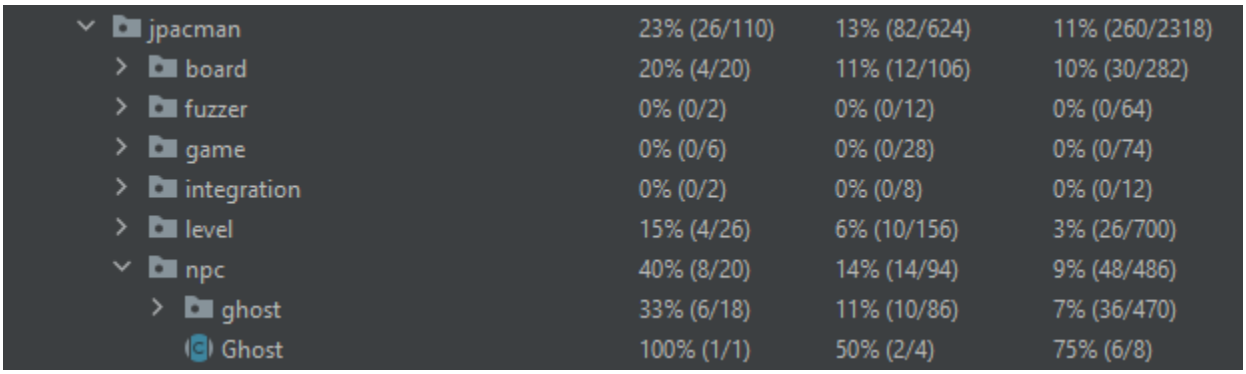
As you can see in the first test. The level package had a coverage of 0% as no class in the package was being tested on. After conducting the test, the level package coverage became 3% as the coverage for class Player was 33%.

Ghost.getSprite()

Before Test



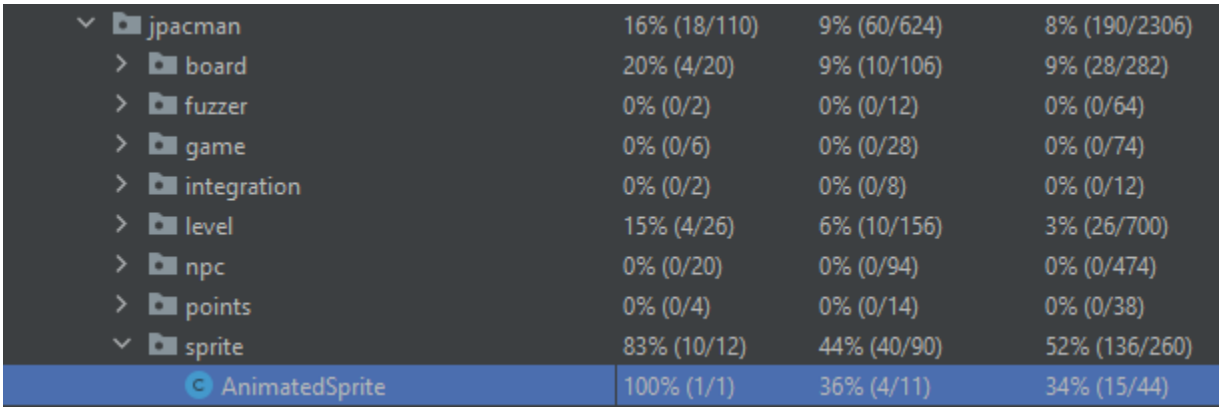
After Test



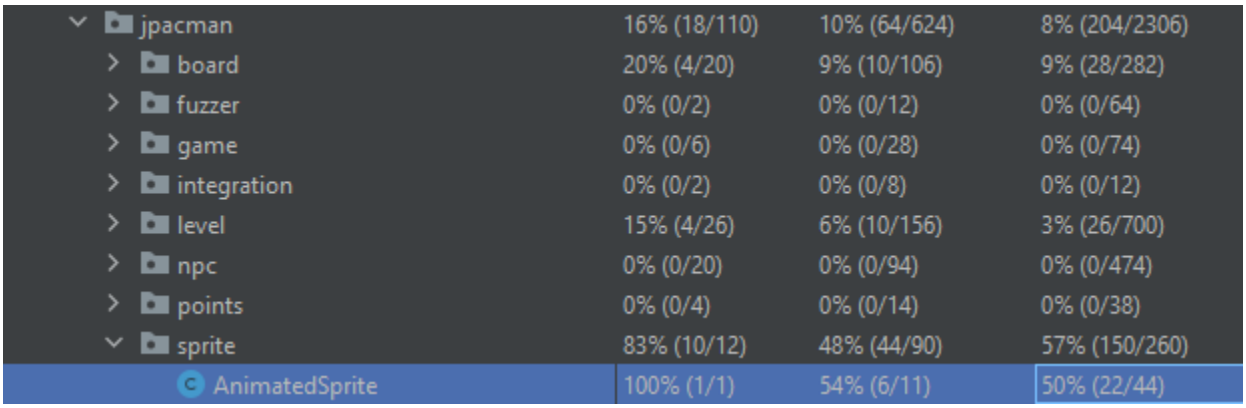
In this example, we tested the getSprite method in the Ghost class. We tested to see if the value returned from the method was an instance of the abstract class Sprite. Before the test our coverage of the Ghost class was 0% as there was no other test being conducted. After the test method, we gained a coverage of 75% effectively increasing the coverage by 3 quarters.

AnimatedSprite.getHeight()

Before Test



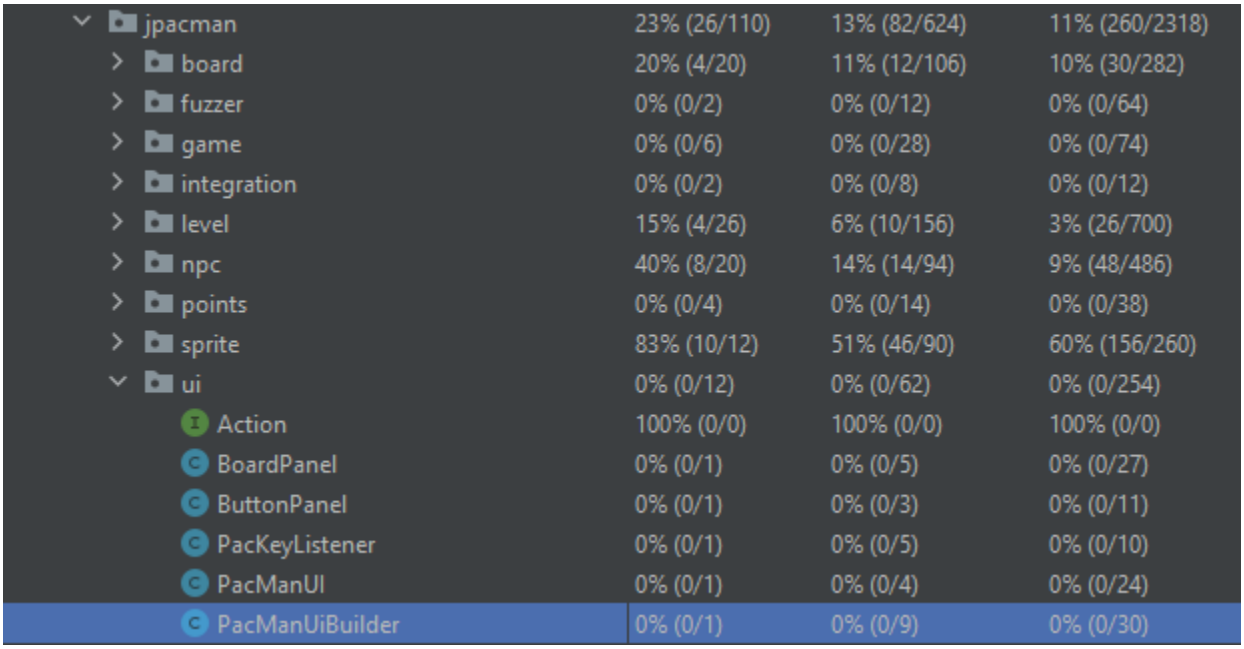
After Test



For the getHeight function in the AnimatedSprite class, you can see that the coverage before the test was 34% as highlighted above. After the test, the coverage increased to 50%. A 16% increase of coverage was made after creating our test class.

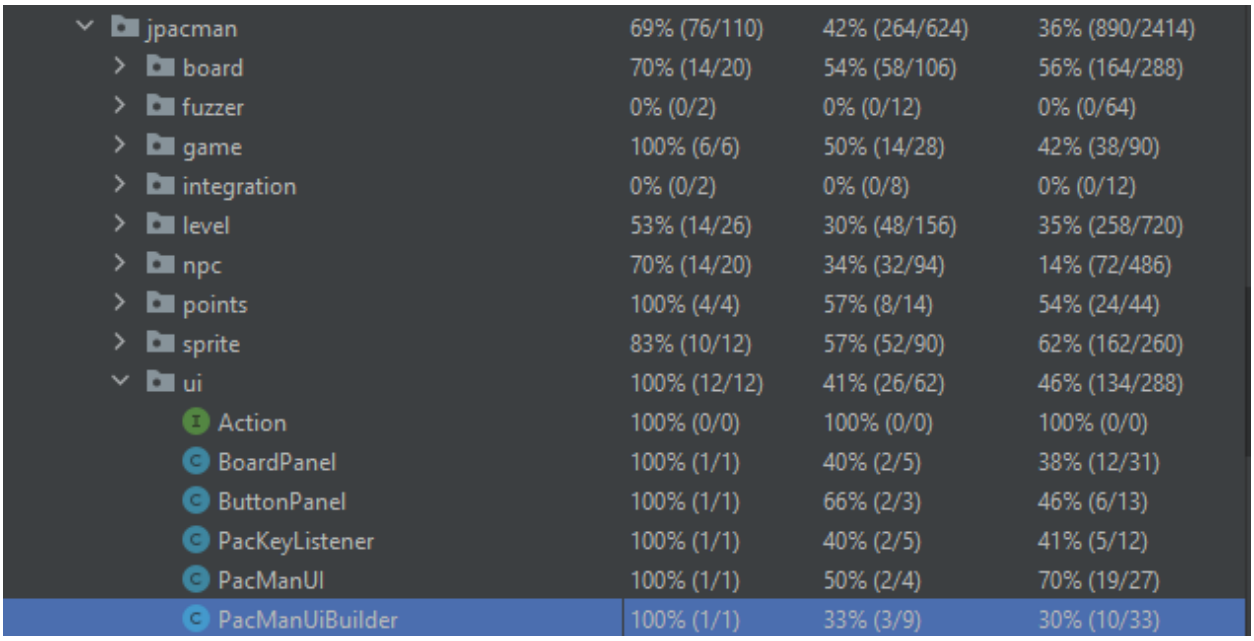
PacManUiBuilder.build()

Before Test



jpacman	23% (26/110)	13% (82/624)	11% (260/2318)
board	20% (4/20)	11% (12/106)	10% (30/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	15% (4/26)	6% (10/156)	3% (26/700)
npc	40% (8/20)	14% (14/94)	9% (48/486)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	51% (46/90)	60% (156/260)
ui	0% (0/12)	0% (0/62)	0% (0/254)
Action	100% (0/0)	100% (0/0)	100% (0/0)
BoardPanel	0% (0/1)	0% (0/5)	0% (0/27)
ButtonPanel	0% (0/1)	0% (0/3)	0% (0/11)
PacKeyListener	0% (0/1)	0% (0/5)	0% (0/10)
PacManUI	0% (0/1)	0% (0/4)	0% (0/24)
PacManUiBuilder	0% (0/1)	0% (0/9)	0% (0/30)

After Test

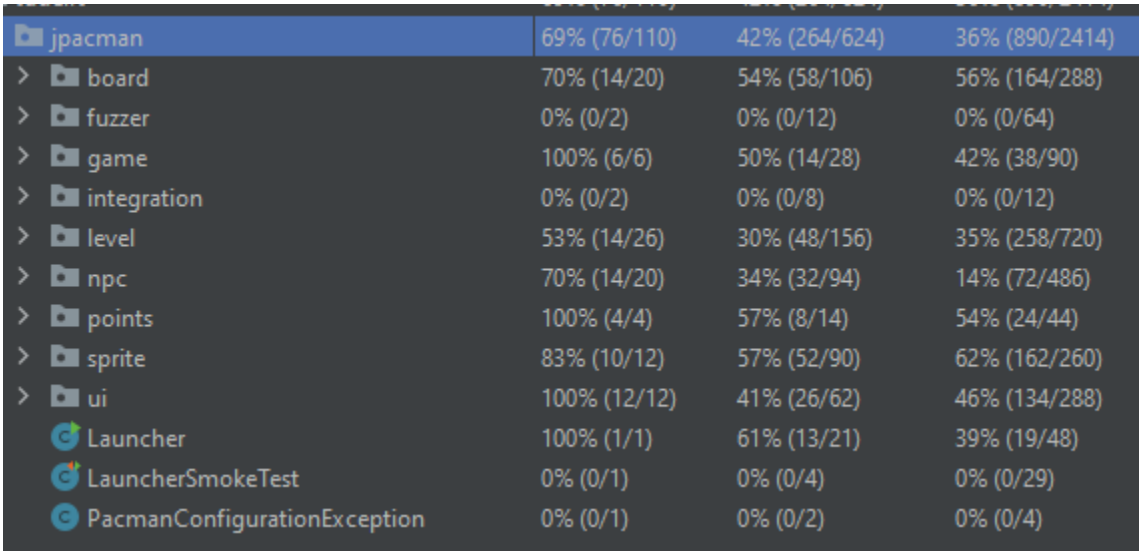


jpacman	69% (76/110)	42% (264/624)	36% (890/2414)
board	70% (14/20)	54% (58/106)	56% (164/288)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	100% (6/6)	50% (14/28)	42% (38/90)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	53% (14/26)	30% (48/156)	35% (258/720)
npc	70% (14/20)	34% (32/94)	14% (72/486)
points	100% (4/4)	57% (8/14)	54% (24/44)
sprite	83% (10/12)	57% (52/90)	62% (162/260)
ui	100% (12/12)	41% (26/62)	46% (134/288)
Action	100% (0/0)	100% (0/0)	100% (0/0)
BoardPanel	100% (1/1)	40% (2/5)	38% (12/31)
ButtonPanel	100% (1/1)	66% (2/3)	46% (6/13)
PacKeyListener	100% (1/1)	40% (2/5)	41% (5/12)
PacManUI	100% (1/1)	50% (2/4)	70% (19/27)
PacManUiBuilder	100% (1/1)	33% (3/9)	30% (10/33)

Our last unit test was conducted on the PacManUiBuilder.build method. We checked to make sure the return value was of the PacManUi class. From the highlighted values in the pictures above. Our coverage went from 0% to an effective 30%.

After all of these unit tests, you can see that our coverage was increased effectively from these four methods. In our JaCoCo results we see that our coverage looks like this

Element	Missed Instructions	Cov.	Missed Branches	Cov.
nl.tudelft.jpacman.level		67%		57%
nl.tudelft.jpacman.npc.ghost		71%		55%
nl.tudelft.jpacman.ui		77%		48%
default		0%		0%
nl.tudelft.jpacman.board		86%		58%
nl.tudelft.jpacman.sprite		86%		59%
nl.tudelft.jpacman		69%		25%
nl.tudelft.jpacman.points		60%		75%
nl.tudelft.jpacman.game		87%		60%
nl.tudelft.jpacman.npc		100%		n/a
Total	1,213 of 4,694	74%	292 of 637	54%



Comparing our Intellij coverage results with the JaCoCo report. We see that both reports show different results. This difference could be due to different code being used to test the methods as one of the reports could have tested different methods. I found that using the IntelliJ report was more effective at identifying uncovered branches as I could see the difference when I built my test module with and without each test method. I personally appeal to the IntelliJ visualization as it was more clean, and more simple to understand the coverage of the unit tests.

link to forked repository: [GarettPF/472-2023-G3 (github.com)]