

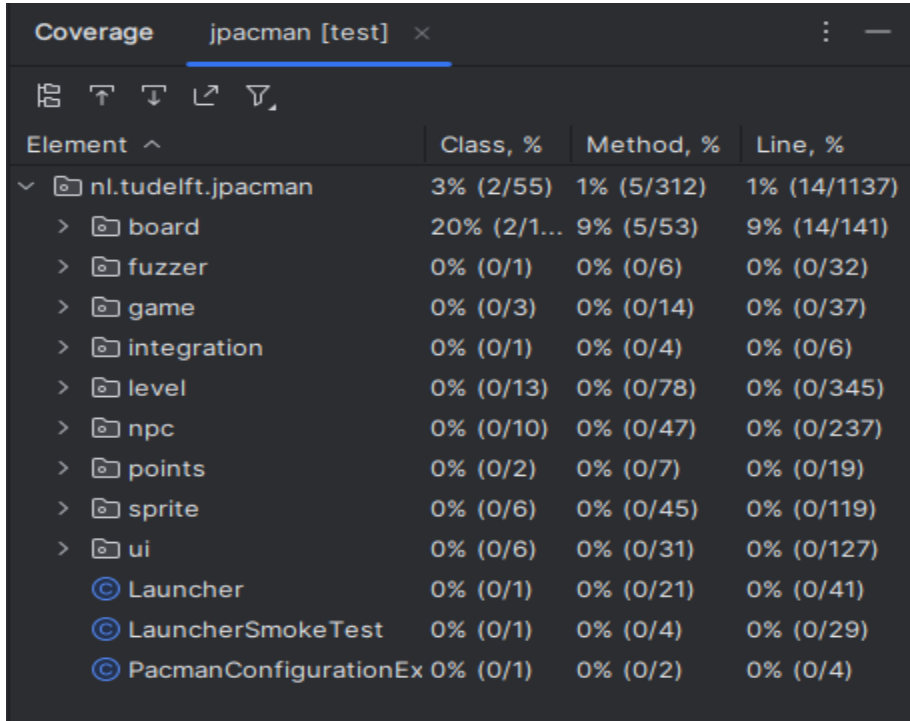
Testing Lab

CS 472 - 1001

Justin Vence Llamas

Link to fork repo - <https://github.com/jusasin/cs472-group4>

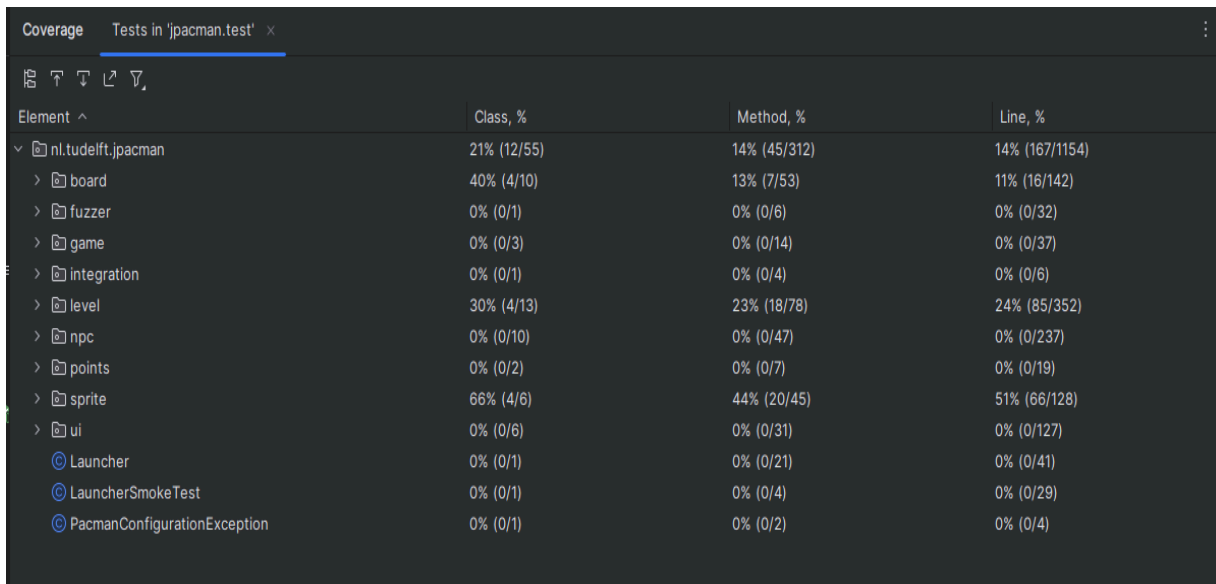
Task 1 - JPacman Test Coverage (Can be used as a before picture reference)



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)

- The coverage is not good enough, majority of the coverage is pretty low. Most of them being 0%. In the next task it seems like we will be trying to better the coverage.

Task 2.1 - After unit test picture



Element ^	Class, %	Method, %	Line, %
nl.tudelft.jpacman	21% (12/55)	14% (45/312)	14% (167/1154)
board	40% (4/10)	13% (7/53)	11% (16/142)
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	30% (4/13)	23% (18/78)	24% (85/352)
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)

- Test coverage after completing unit tests for three methods. The three methods that I decided to work on was in Level.java: start, stop, registerPlayer methods. There were actually more methods that were affected by my unit test in Level but those were the three I mainly focused on.

```
@Test
public void testingPlayerRegister() throws NoSuchFieldException, IllegalAccessException{
    //create a new player
    Player player = mock(Player.class);

    //create starting squares and run a mock
    CollisionMap collisionMap = mock(CollisionMap.class);
    Square sq1 = mock(Square.class);
    Square sq2 = mock(Square.class);
    List<Square> startSquares = List.of(sq1, sq2);
    Level level = new Level(level.getBoard(), new ArrayList<>(), startSquares, collisionMap);

    //Testing when the player is not in the list and squares are not empty
    level.registerPlayer(player);

    //had to search up what reflection was to access the private 'players' field
    Field playersField = Level.class.getDeclaredField(name: "players");
    playersField.setAccessible(true);
    List<Player> players = (List<Player>) playersField.get(level);

    //Check to see if the player was added and is in the game
    assertTrue(players.contains(player));
    Mockito.verify(player).occupy(sq1);

    //Try to add the same player to the list if they are already there
    level.registerPlayer(player);

    //Did the player count change
    assertEquals(expected: 1, players.size());
}
```

- Code Snippet of RegisterPlayer: The above code will test if the player is already on the list for the level, giving them a starting position, and checking if they are already listed once, listing them again should not cause an issue.

```

//Test to check that the game will stop correctly in progress
@Test
public void testStopInProgress() throws NoSuchFieldException, IllegalAccessException {
    //create mock variables
    ScheduledExecutorService scheduledExecutorService = mock(ScheduledExecutorService.class);
    Ghost ghost = mock(Ghost.class);

    //Have to use reflection again to set npcs
    Field npcsField = Level.class.getDeclaredField(name: "npcs");
    npcsField.setAccessible(true);
    Map<Ghost, ScheduledExecutorService> npcs = (Map<Ghost, ScheduledExecutorService>) npcsField.get(level);

    //input the ghost into npc
    npcs.put(ghost, scheduledExecutorService);

    //begin the game
    level.start();
    assertTrue(level.isInProgress());

    //stop the level
    level.stop();

    //check that the level was stopped
    assertFalse(level.isInProgress());

    //had to search up a different way to check instead of using verify that no interaction happened
    Mockito.verifyZeroInteractions(scheduledExecutorService);
}

```

```

//Test the function of the game where it should not stop when not in progress
//Same layout but dont start the game
@Test
public void testStopNotInProgress() throws NoSuchFieldException, IllegalAccessException {
    //create mock variables
    ScheduledExecutorService executorService = mock(ScheduledExecutorService.class);
    Ghost ghost = mock(Ghost.class);

    //Have to use reflection again to set npcs
    Field npcsField = Level.class.getDeclaredField(name: "npcs");
    npcsField.setAccessible(true);
    Map<Ghost, ScheduledExecutorService> npcs = (Map<Ghost, ScheduledExecutorService>) npcsField.get(level);

    //input the ghost into npc
    npcs.put(ghost, executorService);

    //stop the level
    level.stop();

    //check scheduledexecutorservice was not called
    verify(executorService, never()).shutdownNow();

    //check that the level is still not in progress
    assertFalse(level.isInProgress());
}

```

- Code Snippet for Stop and Start (Effects both based on coverage): The first image with the function called “testStopInProgress” is starting a game then stopping it then checking for if that level was stopped properly. No movement was being made and all NPCs came

to a halt. Had to verify that the `shutdownNow()` function was called on. The second image with the function called “`testStopNotInProgress`” does the same thing but instead it does not start the level, it only tries to stop it. I used verify to check if `shutdownNow()` was not called on and the game was still not in progress.

Task 3 - JaCoCo Report on JPacman

nl.tudelft.jpacman.level

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Level		86%		70%	25	55	6	105	1	15	0	1
CollisionInteractionMap		0%		0%	19	19	46	46	7	7	1	1
MapParser		87%		78%	7	26	7	69	1	10	0	1
PlayerCollisions		75%		57%	5	14	6	28	1	7	0	1
LevelFactory.RandomGhost		0%		0%	6	6	12	12	3	3	1	1
Player		85%		66%	3	11	3	24	1	8	0	1
LevelFactory		89%		80%	1	8	1	17	0	4	0	1
Level.NpcMoveTask		100%		100%	0	3	0	10	0	2	0	1
DefaultPlayerInteractionMap		0%		n/a	5	5	17	17	5	5	1	1
CollisionInteractionMap.InverseCollisionHandler		0%		n/a	2	2	5	5	2	2	1	1
PlayerFactory		100%		n/a	0	3	0	5	0	3	0	1
Pellet		100%		n/a	0	3	0	6	0	3	0	1
Total	438 of 1,365	67%	69 of 165	58%	73	155	103	344	21	69	4	12

- The coverage results from JaCoCo and the coverage result from IntelliJ are completely different. After doing the Unit Tests for the three methods, most of the coverage percentages from IntelliJ were either 0 or had very little coverage. While JaCoCo’s coverage results are showing what instructions were missed and what branches. It even color codes it to show specific lines for the coverage.
- I did find it helpful that the source code visualization from JaCoCo was color coded and neatly presented on uncovered branches.
- I prefer looking at JaCoCo’s report in terms of visualization because it gives me a clear picture of what lines are covered, which ones aren’t and I am not quite sure what highlighted yellow lines mean. IntelliJ isn’t bad either because its already incorporated in the framing of my IDE so I don’t have to open anything else like a website like JaCoCo has.