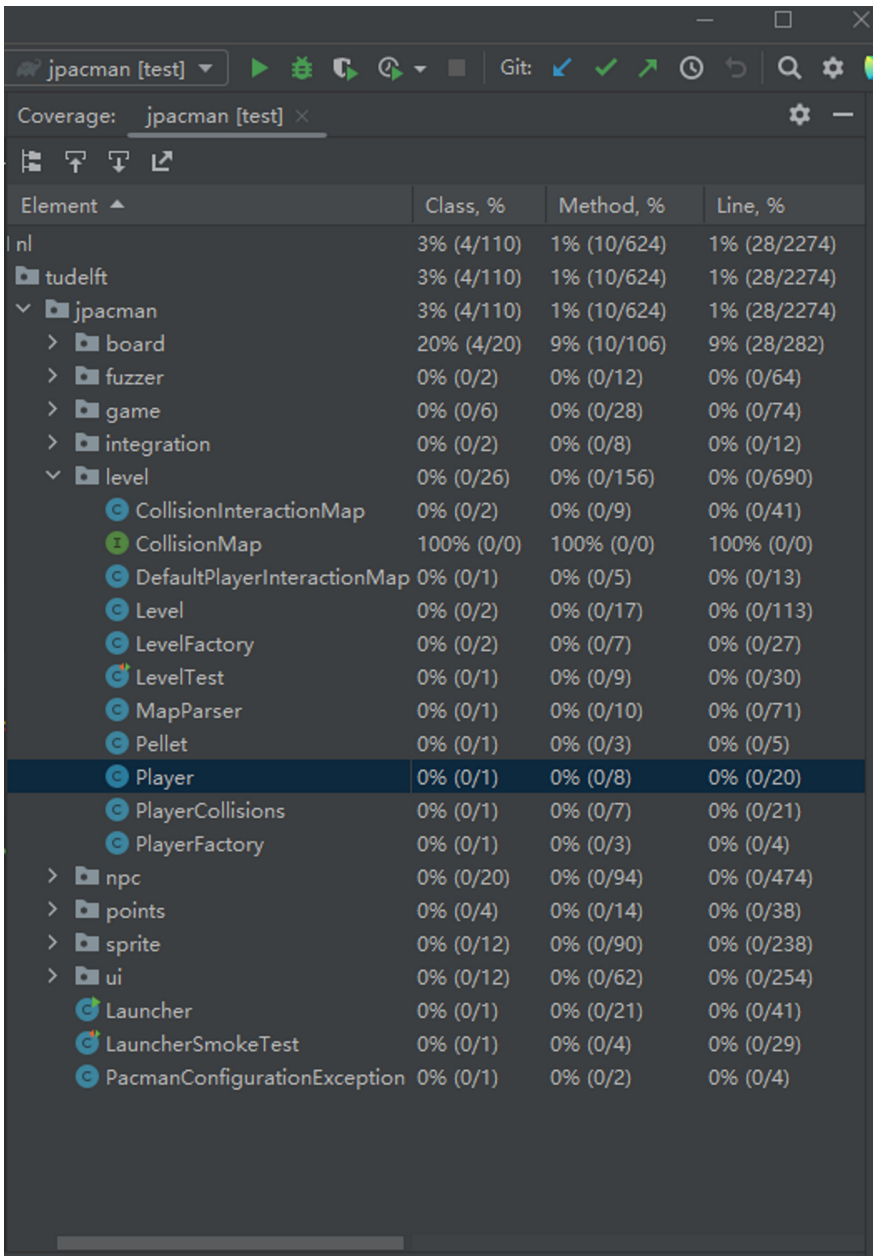
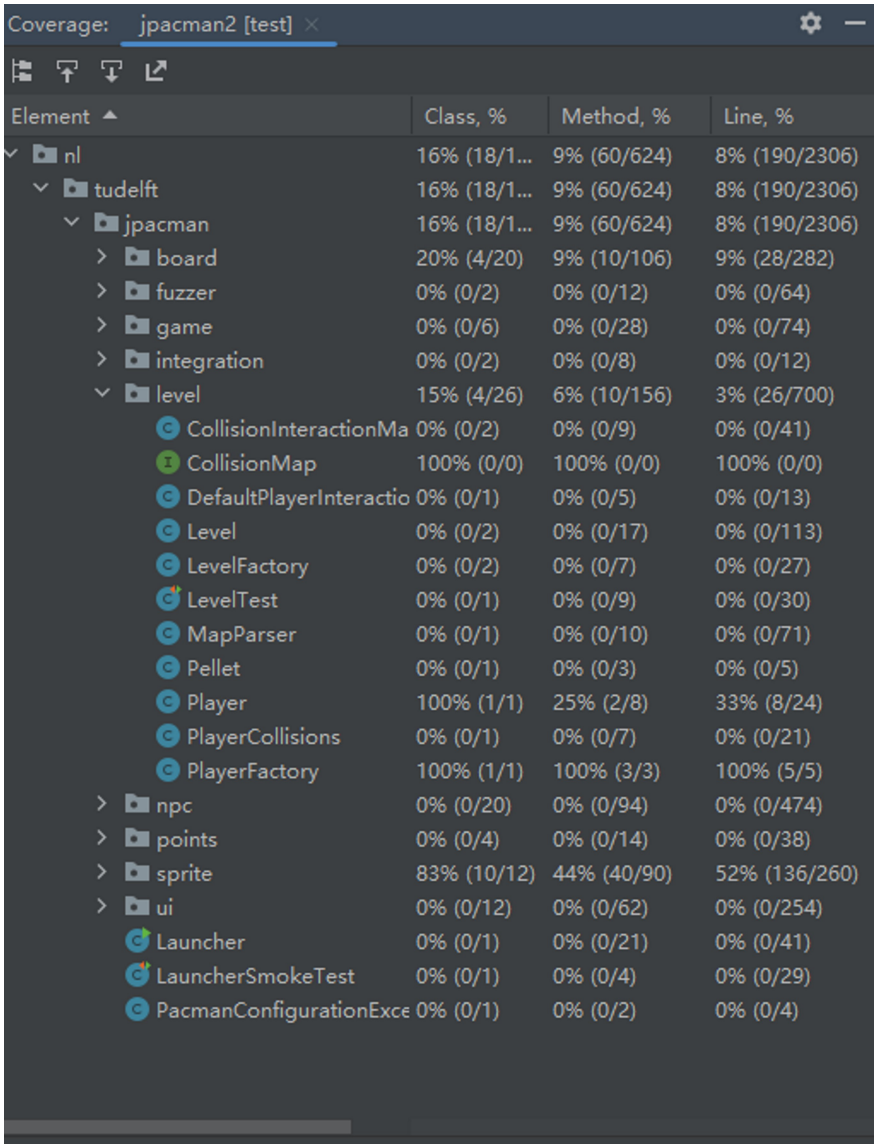
Testing Lab Report

2023年1月29日 16:37

This is code coverage before the test.



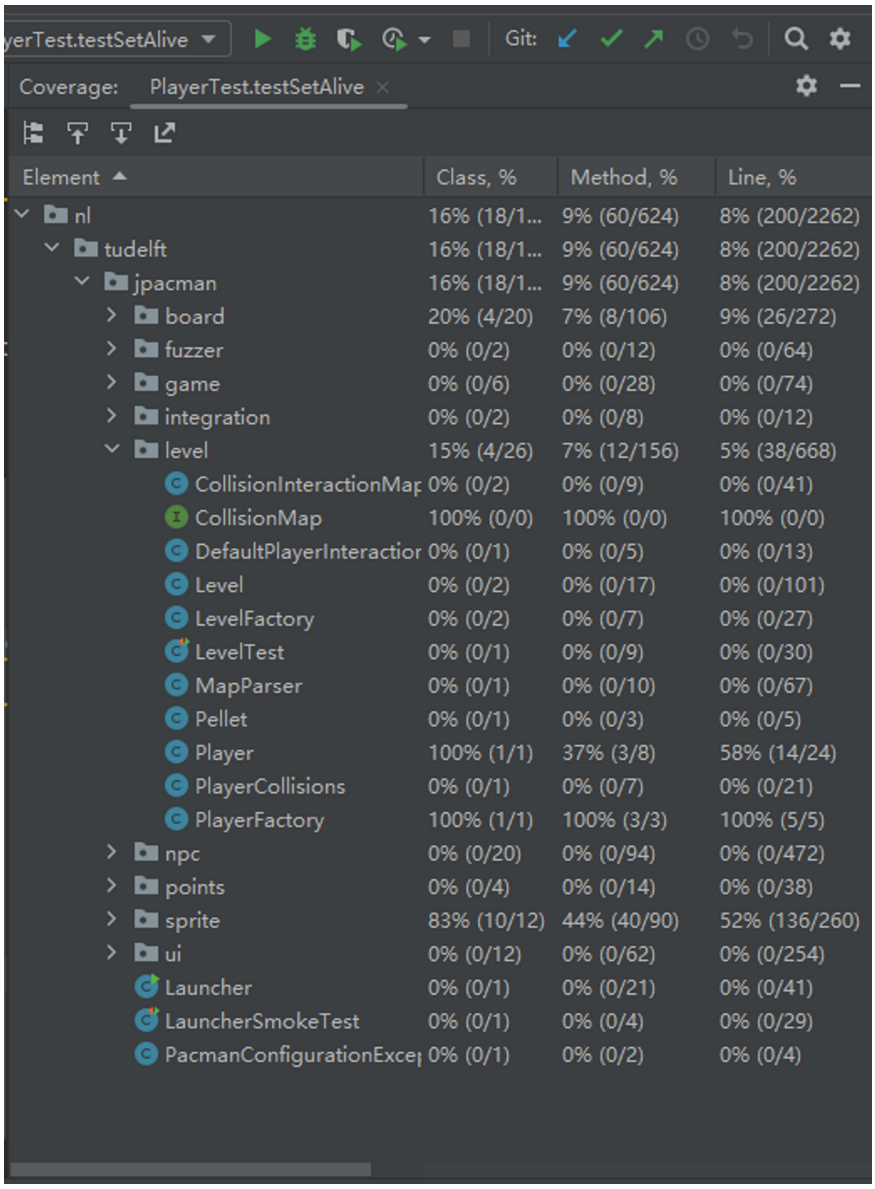
This is the codecovergae after adding test for isAlive().



This is the code coverage after adding test for setAlive(), which covers all the ranches.

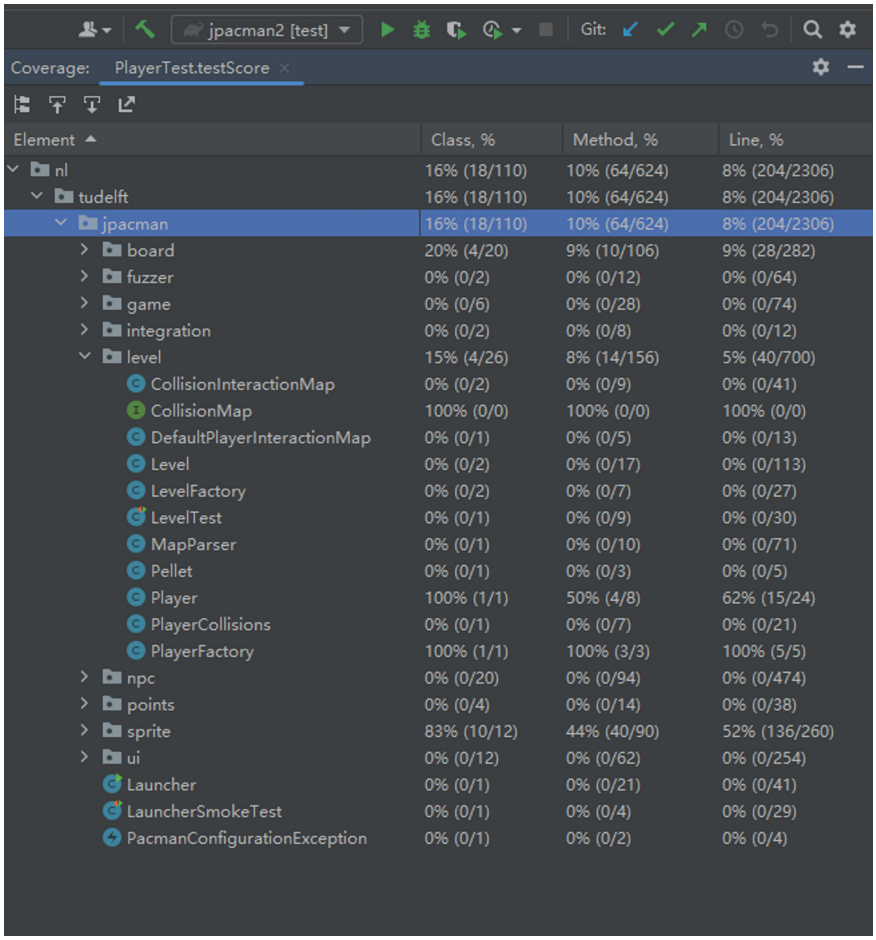
```
no usages
@Test
void testSetAlive_TForALiveFForDead(){
    ThePlayer.setAlive(true);
    assertThat(ThePlayer.isAlive()).isEqualTo( expected: true);
    ThePlayer.setAlive(false);
    assertThat(ThePlayer.isAlive()).isEqualTo( expected: false);

}
```



This is the code coverage after adding test for setKiller(), which makes sure the killer is set.

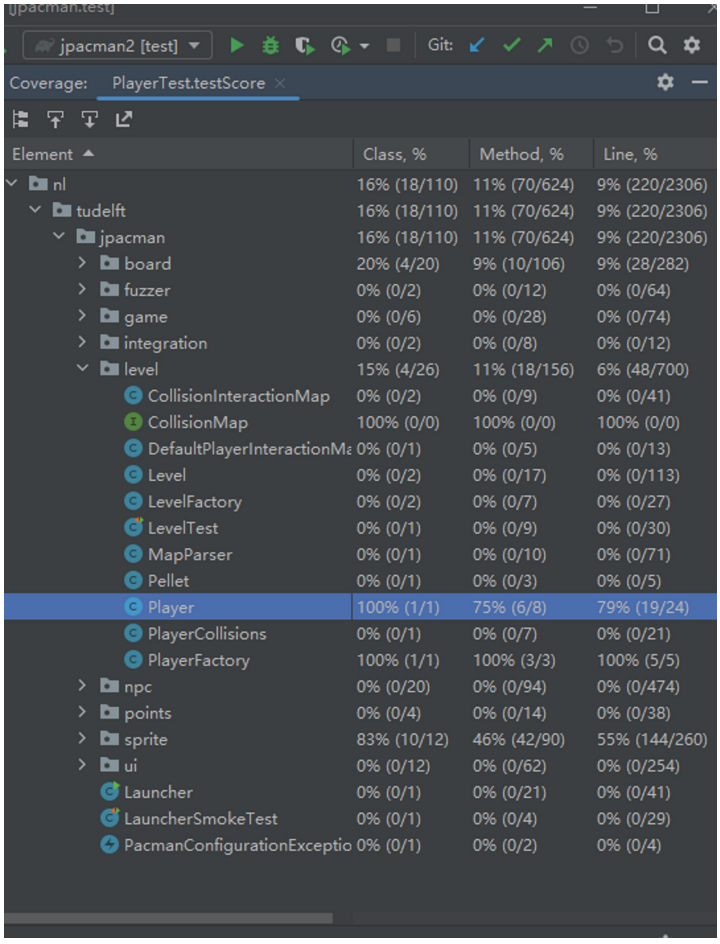
```
@Test
void testSetKiller(){
    Unit killer;
    killer = ThePlayer.getKiller();
    assertThat( actual: killer == ThePlayer.getKiller());
}
```

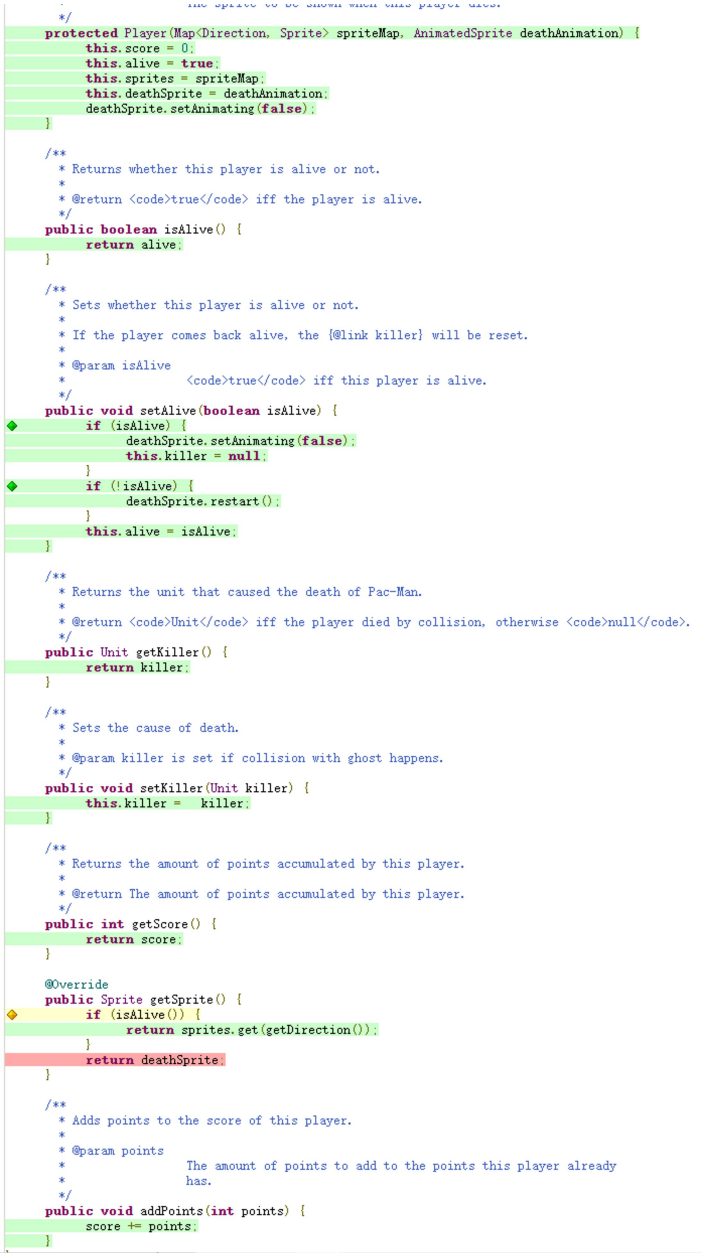


Element	Class, %	Method, %	Line, %
nl	16% (18/110)	10% (64/624)	8% (204/2306)
tudelft	16% (18/110)	10% (64/624)	8% (204/2306)
jpacman	16% (18/110)	10% (64/624)	8% (204/2306)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	15% (4/26)	8% (14/156)	5% (40/700)
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)	50% (4/8)	62% (15/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	44% (40/90)	52% (136/260)
ui	0% (0/12)	0% (0/62)	0% (0/254)
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)

This is the code covergae after adding test for addPoints(), which makes sure that correct points is added to score.

```
no usages
@Test
void testAddPoints(){
    int before = ThePlayer.getScore();
    ThePlayer.addPoints(5);
    int after = before + 5;
    assertThat( actual: after == before);
}
```





```
     */
    protected Player(Map<Direction, Sprite> spriteMap, AnimatedSprite deathAnimation) {
        this.score = 0;
        this.alive = true;
        this.sprites = spriteMap;
        this.deathSprite = deathAnimation;
        deathSprite.setAnimating(false);
    }

    /**
     * Returns whether this player is alive or not.
     *
     * @return <code>true</code> iff the player is alive.
     */
    public boolean isAlive() {
        return alive;
    }

    /**
     * Sets whether this player is alive or not.
     *
     * If the player comes back alive, the {@link killer} will be reset.
     *
     * @param isAlive
     *                <code>true</code> iff this player is alive.
     */
    public void setAlive(boolean isAlive) {
        if (isAlive) {
            deathSprite.setAnimating(false);
            this.killer = null;
        }
        if (!isAlive) {
            deathSprite.restart();
        }
        this.alive = isAlive;
    }

    /**
     * Returns the unit that caused the death of Pac-Man.
     *
     * @return <code>Unit</code> iff the player died by collision, otherwise <code>null</code>.
     */
    public Unit getKiller() {
        return killer;
    }

    /**
     * Sets the cause of death.
     *
     * @param killer is set if collision with ghost happens.
     */
    public void setKiller(Unit killer) {
        this.killer =   killer;
    }

    /**
     * Returns the amount of points accumulated by this player.
     *
     * @return The amount of points accumulated by this player.
     */
    public int getScore() {
        return score;
    }

    @Override
    public Sprite getSprite() {
        if (isAlive()) {
            return sprites.get(getDirection());
        }
        return deathSprite;
    }

    /**
     * Adds points to the score of this player.
     *
     * @param points
     *               The amount of points to add to the points this player already
     *               has.
     */
    public void addPoints(int points) {
        score += points;
    }
```

It looks similar to what was on intellij from last task, but it also shows coverage of related methods in other classes. The branch coverage is useful as it suggests the branches of getSprite() are not fully covered. I would prefer JaCoCo's report, since it shows more specifics on statement coverage and branch coverage.