Task 1 Question(s):

- Is the coverage good enough?

The coverage report indicates that only a small portion of the code has been tested and covered. Out of the total number of Classes (110), only 4 of them are covered, which is a mere 3% coverage. In terms of Methods, only 1% (10 out of 624) have been covered. The coverage of Lines is even lower, with only 1% (28 out of 2274) being covered.

This low coverage may indicate that the existing test cases are inadequate and more tests need to be added to ensure that the code is thoroughly tested and functioning as expected.

Task 3 Question(s):

- Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?

IntelliJ's coverage report provides more comprehensive metrics, including information on the number of executed lines, missed lines, and missed branches. It also integrates the coverage information directly within the IDE, making it easier for developers to access and interpret the coverage results.

JaCoCo, on the other hand, provides a more basic coverage metric, measuring the percentage of code that has been executed by test cases. JaCoCo provides its coverage information through its API.

In conclusion, both IntelliJ's coverage report and JaCoCo provide code coverage metrics, but IntelliJ provides more detailed information that is easily accessible within the IDE, while JaCoCo is a more basic tool that can be integrated with other tools.

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

The source code visualization from JaCoCo on uncovered branches provides valuable insights into the quality of software testing. By visualizing the uncovered branches, we can easily identify areas of the code that need to be tested and focus their efforts on improving the coverage.

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

IntelliJ's coverage report provides detailed information about code coverage, including the number of executed lines, missed lines, missed branches, etc. This information is easily accessible within the IDE, which makes it convenient for developers to review and analyze the results.

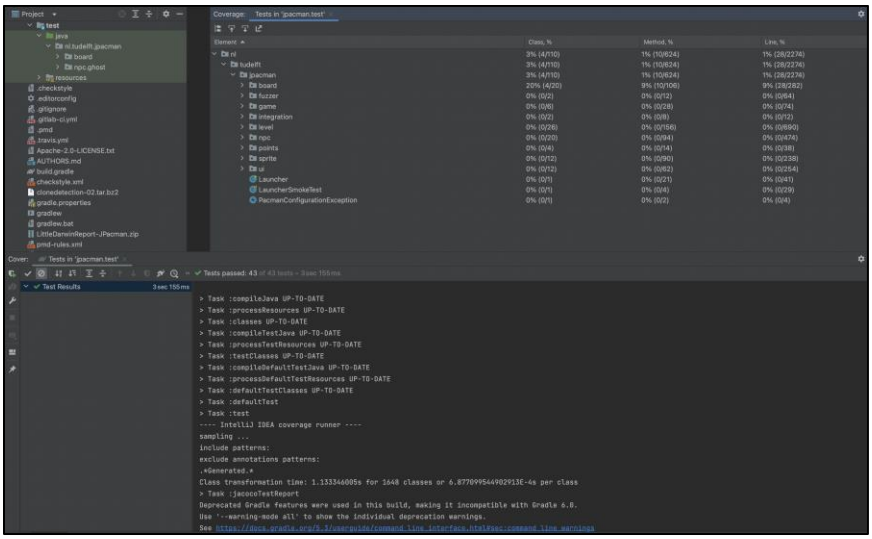
In contrast, JaCoCo is a standalone tool that provides basic coverage metrics such as the percentage of code that has been executed by test cases. While JaCoCo can be integrated with

other tools such as Jenkins, Gradle, etc, it does not provide the same level of detail and accessibility as IntelliJ's coverage report.

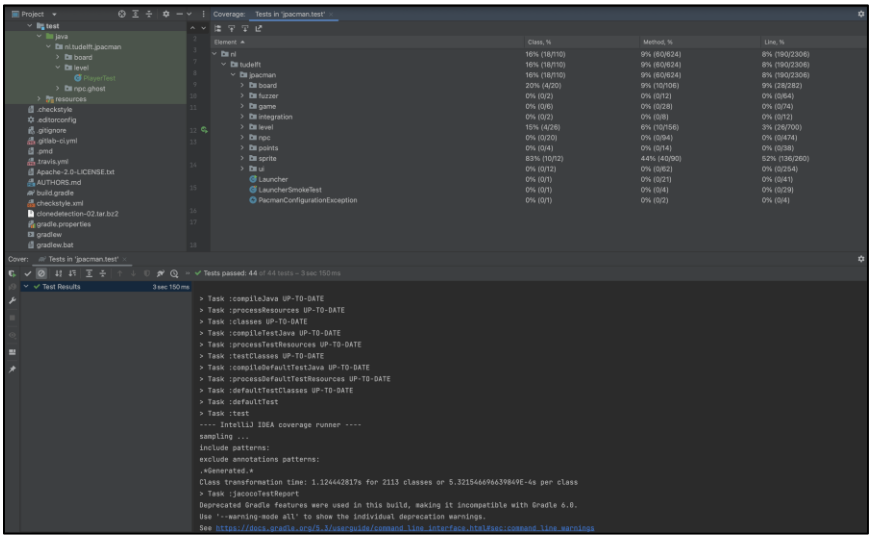
Another advantage of IntelliJ's coverage report is that it provides a visual representation of the code, which can help developers understand the structure and flow of the code. This can be particularly helpful in identifying areas that are difficult to test, or areas where tests are failing.

In summary, the coverage report from IntelliJ is considered better than JaCoCo due to its integration within the IDE, the additional metrics it provides, and the visual representation of the code coverage, which can help developers improve the quality of their testing.

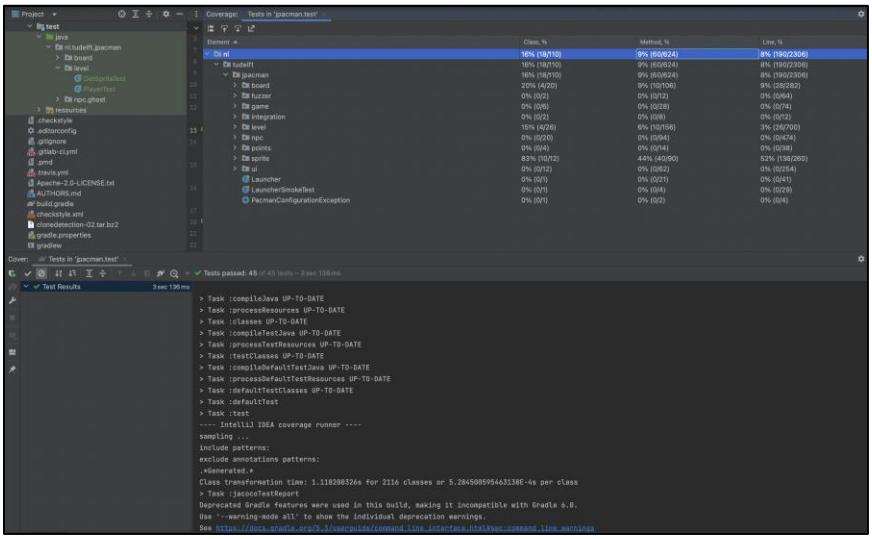
IntelliJ Test Report Task 1:



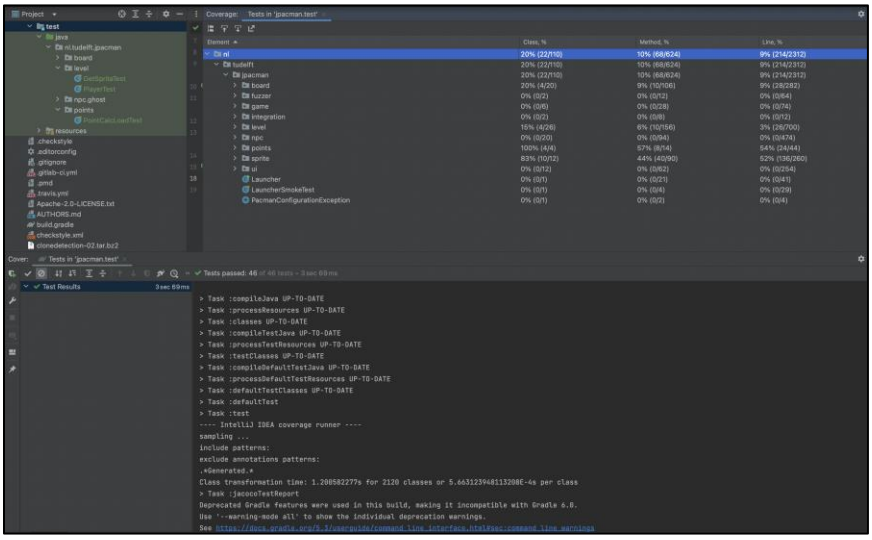
IntelliJ Test Report Tast 2:



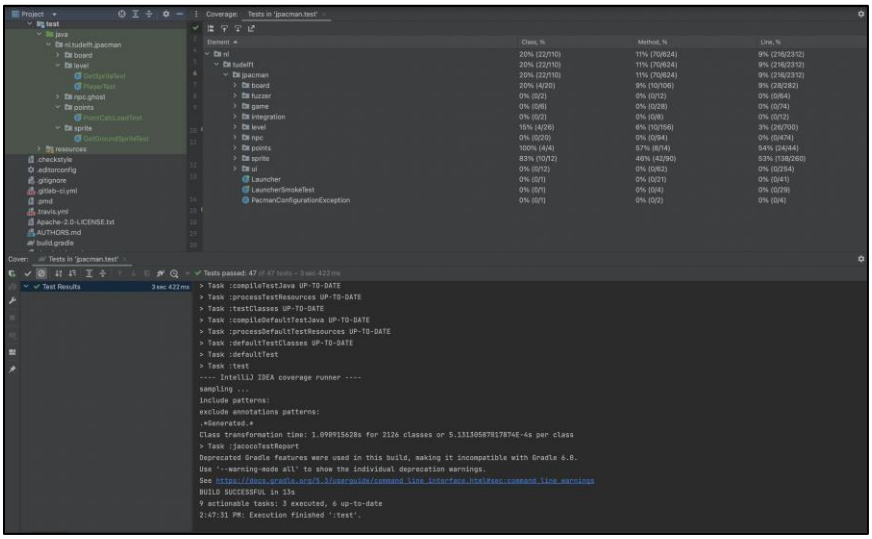
IntelliJ Test Report Task 2.1 (Method 1):



IntelliJ Test Report Task 2.1 (Method 2):



IntelliJ Test Report Task 2.1 (Method 3):



Fork Repository: https://github.com/devyngilliam/jpacman