CS -587 Project Final Report

Student Names: 1) Asbin Dahal (https://github.com/asbin4747)

2) Astghik Hovhannisyan (https://github.com/AstghikHov)

Proposed System Under Test (SUT): Washer-chess (Java)

For the sake of demonstrating what we did, we have not merged our pull requests into our master branch. The latest pull request should consist of all the tests written so far. Only one of the pull requests was merged into the master branch unintentionally: tests for Bishop, Board, Node,and MoveList.

1. **Unit Testing with Sufficient Coverage**

   As mentioned in our proposal report we were able to write unit tests for most of the classes. We have successfully written unit tests for methods BoardArrayFactory, Bishop, BoardEntry, Board, King, Knight, MoveList, Node, Pawn, PieceListFactory, PieceList, PlyTable, Ply, Queen, Rook, Side, Tree and WasherEngine classes. We originally intended to include the jacoco and pittest for measuring our test reports. Unfortunately, our SUT did not consist of a pom.xml file so we had difficulty running the maven commands. Instead, we used the built in code coverage report that is provided by Intellij. The coverage screenshot is provided along with this report.

   There were a lot of unit tests written. We have implemented unit tests for methods of all the classes mentioned below:

   - BishopTest: link
   - BoardTest: list
   - BoardEntryTest: link
   - KnightTest: link
   - MoveListTest: link
   - NodeTest: list
   - PawnTest: link
   - PieceListTest: link
   - PlyTableTest: link
   - PlyTest: link
   - BoardArrayFactory: link
   - PieceListFactory: link
   - QueenTest: link

- ➢ RookTest: link
- ➢ SideTest: link
- ➢ TreeTest: Link
- ➢ WasherEnfineTest: Link

2. **Automated Testing**
   As mentioned in our proposal we have also included automated testing in our SUT. We have used loops to generate inputs and test them in BoardTest, KingTest, NodeTest, and WasherEngineTest. We have successfully checked for the behaviours of each piece at different positions as mentioned in our proposal.

   - BoardTest (`TestGetLocation, testGetFile, testGetFile1, testGetFile1 : from line 56`) Link
   - KingTest (`checkBooleanReturnWithArray` : line 75) link
   - NodeTest(`getNumberOfChildTest, getChildTest,getChildrenTest: from line 12`) link
   - Washer engine (`TestIndent` : line 27) Link

3. **Bugs Found**
   Our test cases helped us to find 5 bugs in the SUT. The tests we wrote did not pass as it should have.
   - Tests `addChildAt01Test() and addChildAt_IndexOutOfBound_Test` Helped to find bugs in method getChild of Node. We found that there it fails when we add a child in the index that is out of boundary link `(lines 52, 71)`
   - Test `TestToListNull` helped to find that toList method of Tree doesn't work correctly for an empty tree Link `(line 30)`
   - `TestGetMoveAtNull` test for MoveList sows that getMove method of MoveList doesn't fails when the index is out of boundary of the list Link `(line 35)`
   - `test_isDraw` for BoardTest should have returned true but instead showed errors.(`line 101`) link

4. **Refactoring**
   We did some refactoring to increase the testability. To test some of the private methods we created public methods that call the private ones and return the same output. Also,if it was possible, we created methods that return the output of void methods.
   - Created `findDirectionalMovesPublic` which returns the seoult of private method `findDirectionalMovesPublic` Link
   - Created some public methods to test private methods, and methods to test void methods (explained in comments) link
   - Refactoring of void and private methods Link
   - Refactored private method Link
   - Modified private method Link

5. **Lessons Learned**
   - There were a number of things we did learn from this project. We did choose this project to focus more on unit testing, but later found out that testing for GUI applications is challenging. Therefore we decided to focus only on unit tests and automated tests. We could not cover the GUI testing and if we had done this differently we probably would have had a game plan for GUI testing.
   - We did learn about Mockito which we have included in our test suite.
   - **Coverage Info** : We were only able to get 58% overall coverage. This happened because we were not able to test most of the methods and classes implemented in the SUT that consisted of abstract, static,void and dome of the private methods. It was challenging to refactor those methods as there were lots of dependencies among the methods. We did refactor some of the methods but found out that it really did not increase our code coverage. Considering nearly 4500 lines of code to test for we did cover more than half of the code.
   - **What we could have done better:** Better understanding of 0x88 representation of the Chess board could have increased the branch coverage and number of tests for several methods. Looking back we would have definitely first tried to understand a code better before writing the test, also we could have chosen SUT that had less abstract and private methods could have been a better option for us.

Coverage Screenshots and link

Coverage Report link :[link]

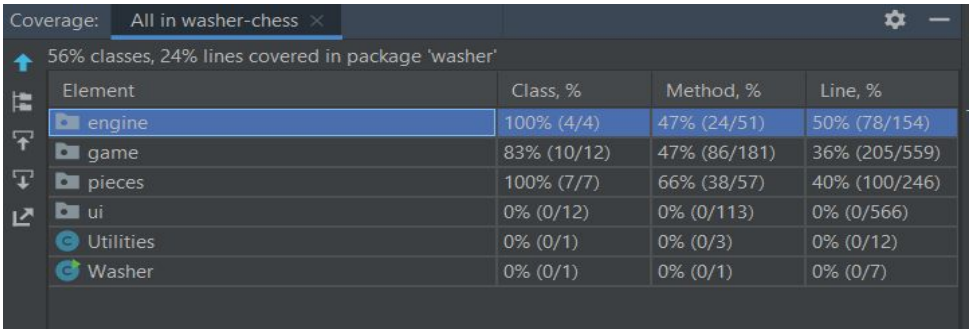

Fig: Overall coverage generated through Intellij which was 56%

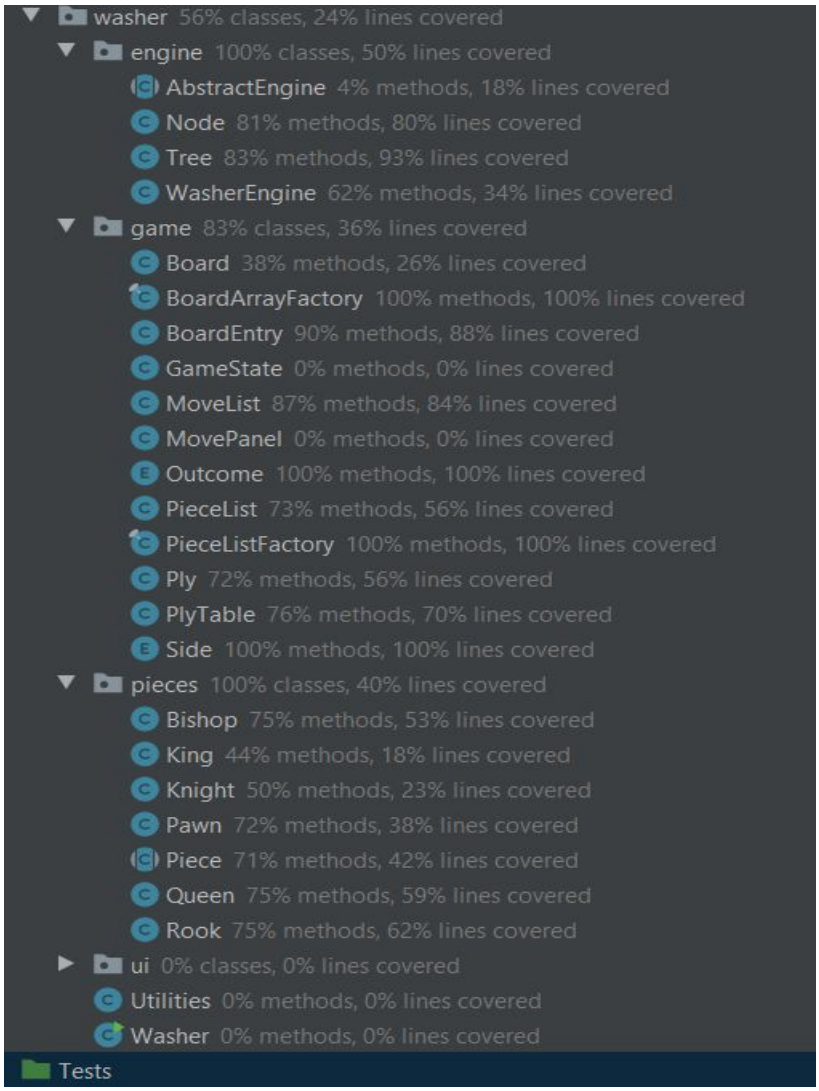

Fig : Breakdown by coverage for each class.