Ruslan Abilkaiyrov

Mikael Kuyumchyan

Daniel Navarro

Comp 587

Testing BlackWidow-Chess

Unit Testing

BlackWidow-Chess was really hard to test because originally it was using ANT-testing in order to test it. So, we had to change and modify a lot of files because ANT does not allow us to get jacoco:report for some reason, that's why we changed to something that we know and it was maven. We made a lot of changes, and because of that we had to cut out a lot of lines of code. So, we basically combined original files and modified it so that it would work when we will use maven.

Link to github: https://github.com/csun-comp587-s20/BlackWidow-Chess

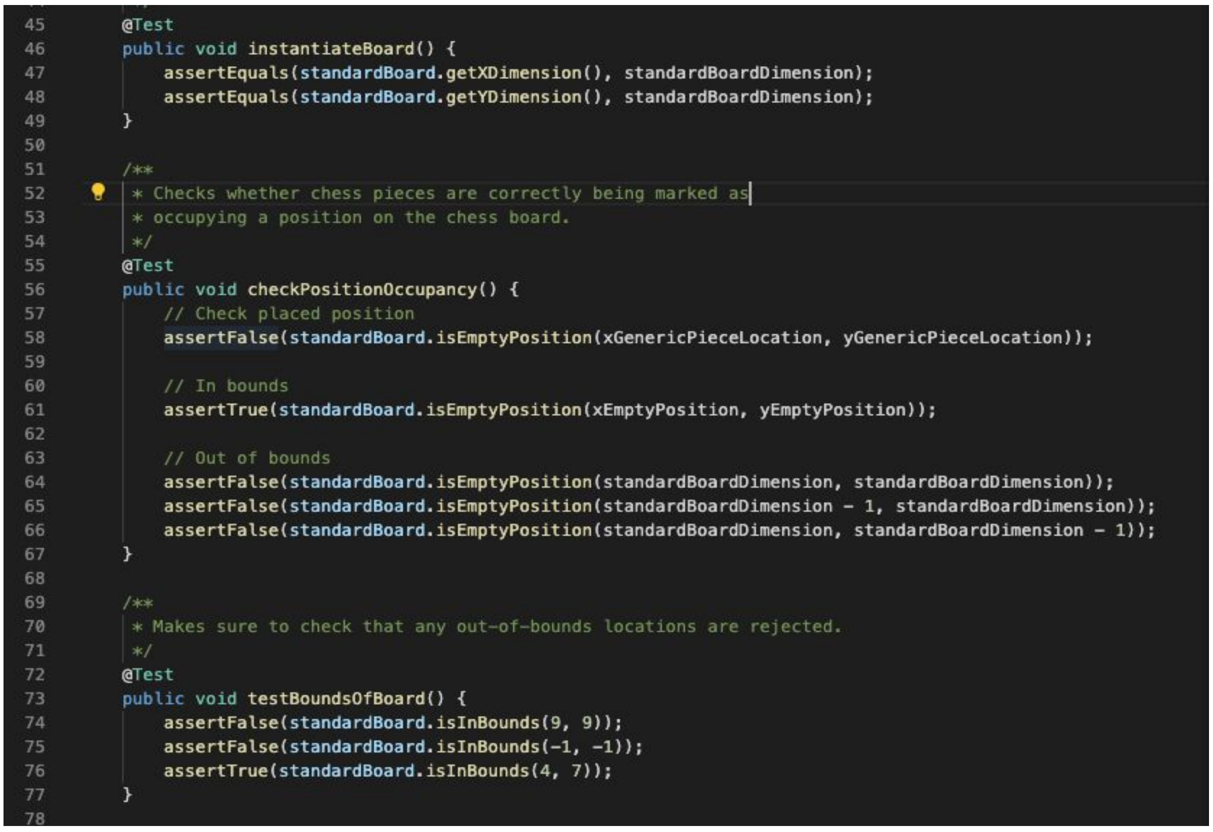
Some example of unit tests:

So when you start the game, it should first create the dimension of x and y coordinates on the board. It also checks the position and boundary of the board.

Testfile link:

https://github.com/csun-comp587-s20/BlackWidow-Chess/blob/master/BlackWidow-Chess/src/test/java/arithmetic/ChessTest.java

Test coverage 47%.

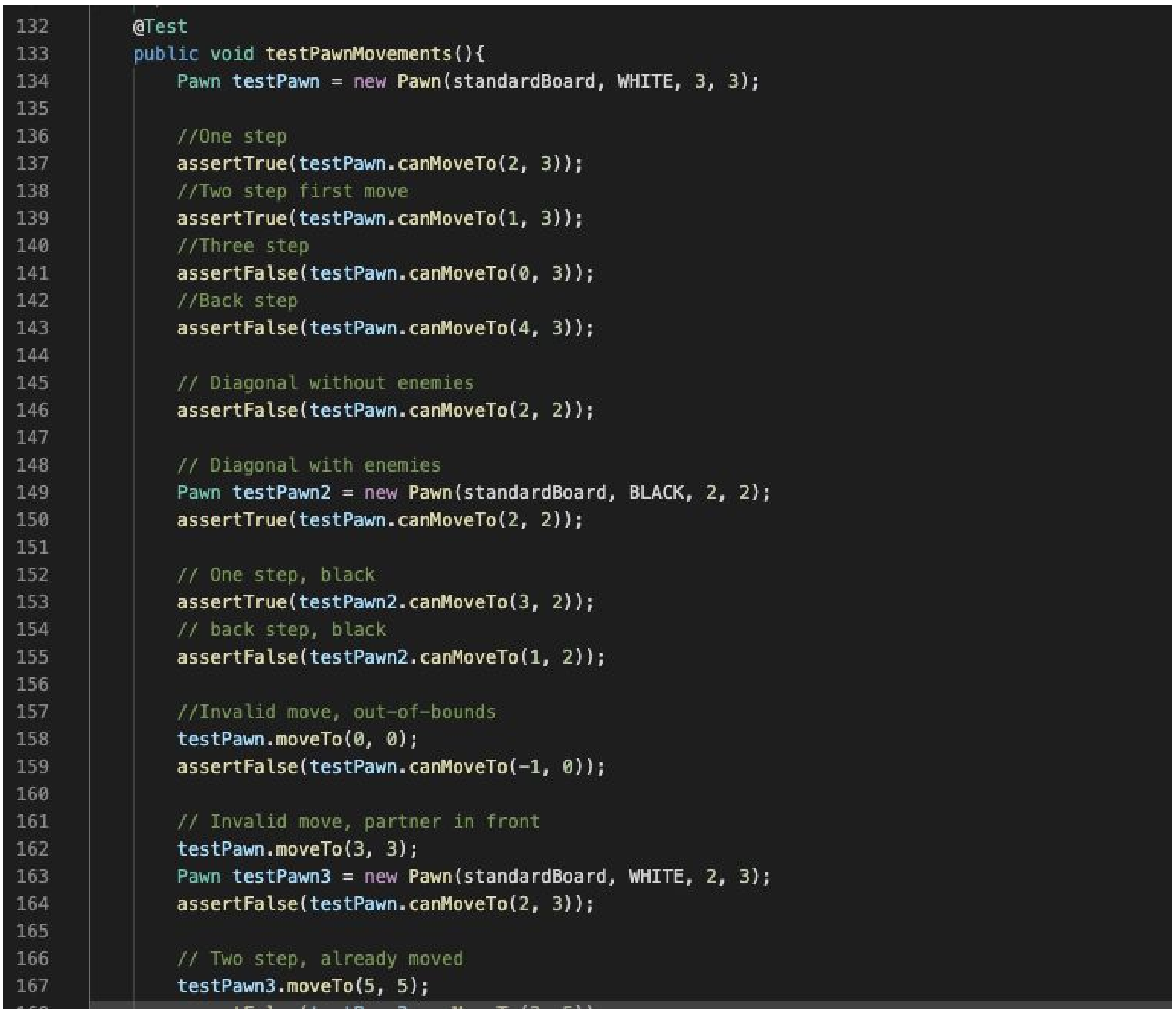


```
    @Test
    public void instantiateBoard() {
        assertEquals(standardBoard.getXDimension(), standardBoardDimension);
        assertEquals(standardBoard.getYDimension(), standardBoardDimension);
    }

    /**
     * Checks whether chess pieces are correctly being marked as
     * occupying a position on the chess board.
     */
    @Test
    public void checkPositionOccupancy() {
        // Check placed position
        assertFalse(standardBoard.isEmptyPosition(xGenericPieceLocation, yGenericPieceLocation));

        // In bounds
        assertTrue(standardBoard.isEmptyPosition(xEmptyPosition, yEmptyPosition));

        // Out of bounds
        assertFalse(standardBoard.isEmptyPosition(standardBoardDimension, standardBoardDimension));
        assertFalse(standardBoard.isEmptyPosition(standardBoardDimension - 1, standardBoardDimension));
        assertFalse(standardBoard.isEmptyPosition(standardBoardDimension, standardBoardDimension - 1));
    }

    /**
     * Makes sure to check that any out-of-bounds locations are rejected.
     */
    @Test
    public void testBoundsOfBoard() {
        assertFalse(standardBoard.isInBounds(9, 9));
        assertFalse(standardBoard.isInBounds(-1, -1));
        assertTrue(standardBoard.isInBounds(4, 7));
    }
```

Another example of unit test:

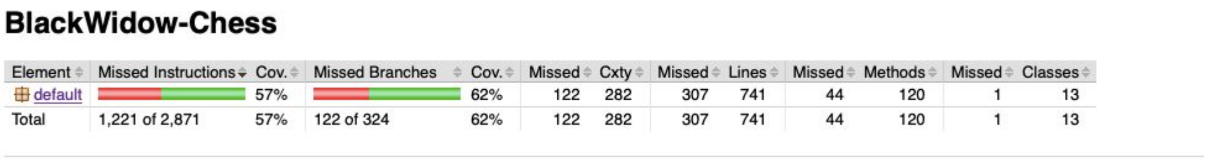
This is an example of testing the right movement of a pawn piece. It was not that hard to write unit tests for checking because as long as you know how to make a move and the board itself. So, it checks every move and if it is true it allows you to move if not you can’t make that move. It checks for the white player and balck player because you have to check every move from horizontal and vertical location, and you only have the option to make a move, and it moves two blocks straightforward.

The code coverage for the testing Pawn was 81%.



```java
    @Test
    public void testPawnMovements(){
        Pawn testPawn = new Pawn(standardBoard, WHITE, 3, 3);

        //One step
        assertTrue(testPawn.canMoveTo(2, 3));
        //Two step first move
        assertTrue(testPawn.canMoveTo(1, 3));
        //Three step
        assertFalse(testPawn.canMoveTo(0, 3));
        //Back step
        assertFalse(testPawn.canMoveTo(4, 3));

        // Diagonal without enemies
        assertFalse(testPawn.canMoveTo(2, 2));

        // Diagonal with enemies
        Pawn testPawn2 = new Pawn(standardBoard, BLACK, 2, 2);
        assertTrue(testPawn.canMoveTo(2, 2));

        // One step, black
        assertTrue(testPawn2.canMoveTo(3, 2));
        // back step, black
        assertFalse(testPawn2.canMoveTo(1, 2));

        //Invalid move, out-of-bounds
        testPawn.moveTo(0, 0);
        assertFalse(testPawn.canMoveTo(-1, 0));

        // Invalid move, partner in front
        testPawn.moveTo(3, 3);
        Pawn testPawn3 = new Pawn(standardBoard, WHITE, 2, 3);
        assertFalse(testPawn.canMoveTo(2, 3));

        // Two step, already moved
        testPawn3.moveTo(5, 5);
```

Test coverage for our project is 62%



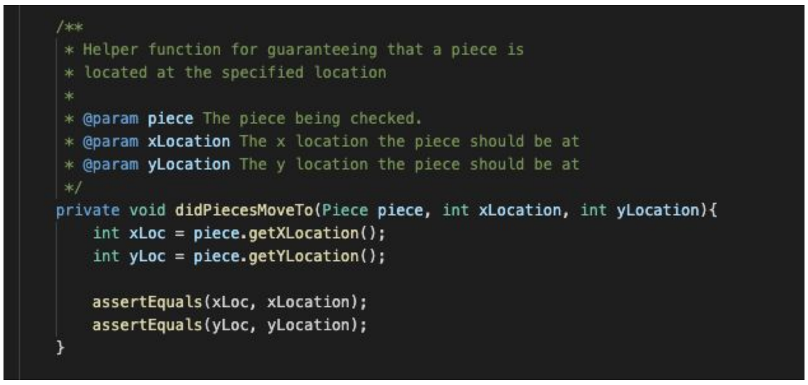
BlackWidow-Chess

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default	🟥🟩	57%	🟥🟩	62%	122	282	307	741	44	120	1	13
Total	1,221 of 2,871	57%	122 of 324	62%	122	282	307	741	44	120	1	13

We specifically changed the test cases for each of the chess pieces, this includes: pawn, knight, bishop, castle, Queen, King. We tested the movement of each piece, making sure that it cannot move outside of the board, or onto any other pieces that are already on the board. Each piece has its unique way of moving, and they all got their corresponding movement type tested. We made sure to test that white pieces don’t eat other white pieces, and black pieces don’t eat other black pieces.

Automated Testing

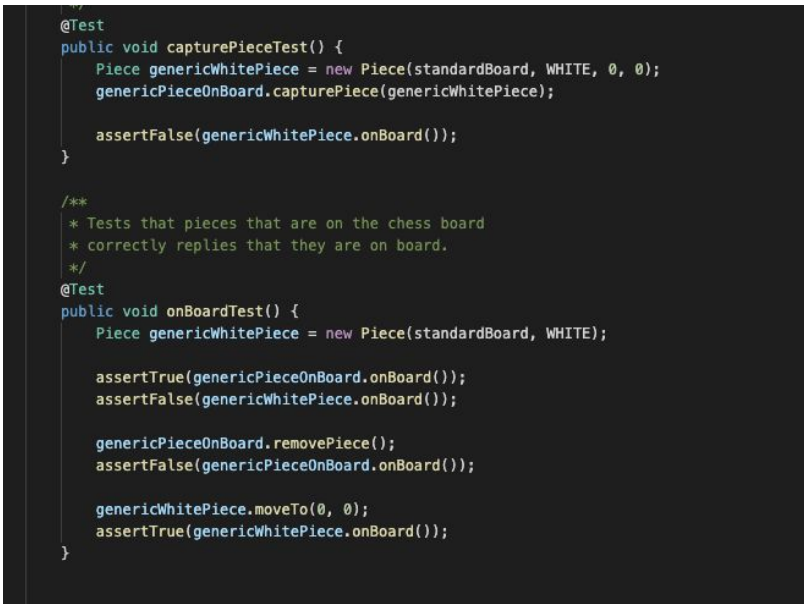
We were able to apply an automated test case generator on the ‘did pieces move to’ function and see the same function being tested on different inputs to get their location on the board. This gave us testing the exact place of pieces on the x and y coordinates and then checking if it is on the right spot and if it is, it gives us true and it is placed on that spot. After this test is done it goes to check to capturePieceTest() and onBoardTest() and only then it gives you output.



```
/**
 * Helper function for guaranteeing that a piece is
 * located at the specified location
 *
 * @param piece The piece being checked.
 * @param xLocation The x location the piece should be at
 * @param yLocation The y location the piece should be at
 */
private void didPiecesMoveTo(Piece piece, int xLocation, int yLocation){
    int xLoc = piece.getXLocation();
    int yLoc = piece.getYLocation();

    assertEquals(xLoc, xLocation);
    assertEquals(yLoc, yLocation);
}
```

Another example of Automated testing:

So those two tests are automated. capturePieceTest() tests if a piece is in the right place and if each piece stays on its spot. onBoardTest() tests the boundary of the board and checks if there are no extra pieces on the board. It was quite easy to do capturePieceTest() since we know the rules, but in order to do onBoardTest(), it was hard to make it work because checking boundary was something logic concern, especially we spent a lot of time to figure out what to do with extra pieces, and then we only needed to remove them from the board.



```
    @Test
    public void capturePieceTest() {
        Piece genericWhitePiece = new Piece(standardBoard, WHITE, 0, 0);
        genericPieceOnBoard.capturePiece(genericWhitePiece);

        assertFalse(genericWhitePiece.onBoard());
    }

    /**
     * Tests that pieces that are on the chess board
     * correctly replies that they are on board.
     */
    @Test
    public void onBoardTest() {
        Piece genericWhitePiece = new Piece(standardBoard, WHITE);

        assertTrue(genericPieceOnBoard.onBoard());
        assertFalse(genericWhitePiece.onBoard());

        genericPieceOnBoard.removePiece();
        assertFalse(genericPieceOnBoard.onBoard());

        genericWhitePiece.moveTo(0, 0);
        assertTrue(genericWhitePiece.onBoard());
    }
```

Lessons Learned

We have learned a lot about testing throughout the completion of the project. With the many bumps along the way, we improved in testing the methods we sought to test. If we were to do the project all over again, we would spend more time on planning which parts exactly should be tested, and how they are going to be tested. We would also improve on the time managing skills, diving up the work to make it flow smoother and faster. For many of us, Github was also a new skill we had to learn for this class, so there was a big learning curve in that aspect as many of the merge conflicts took hours to resolve. Getting maven to work on our computers was also a hassle, as we had to pool some fancy computer science techniques to get through the problem.