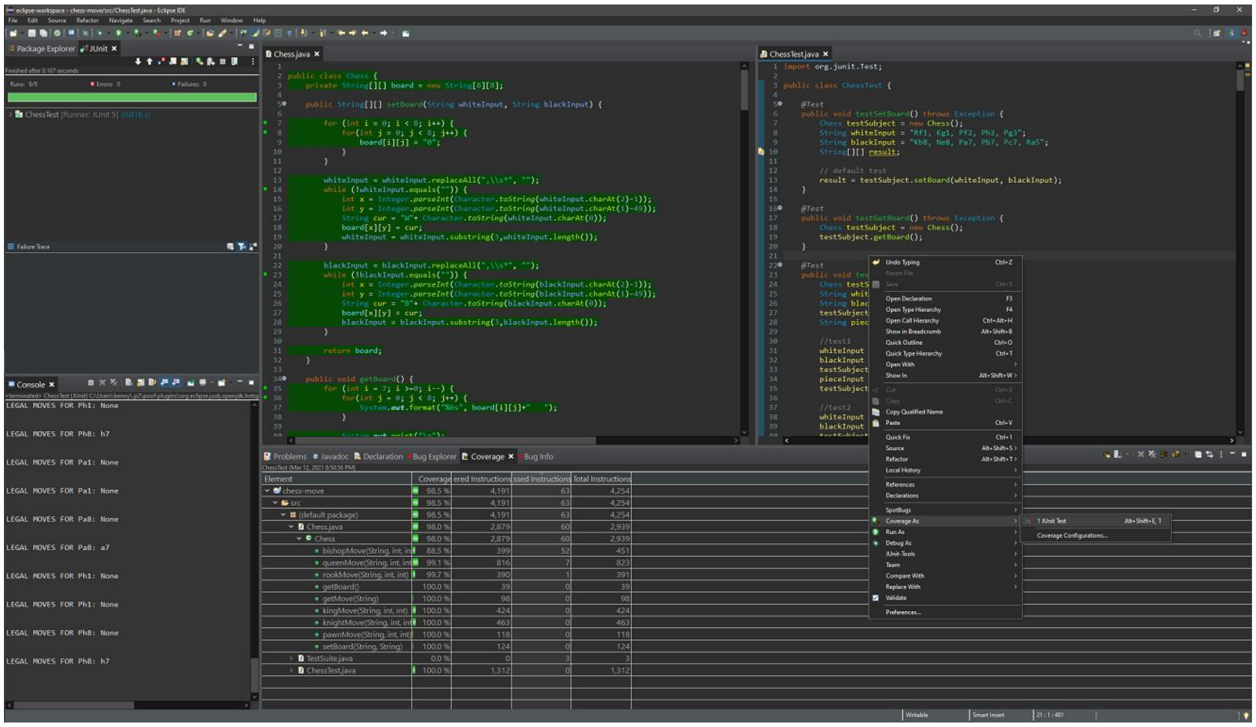
Instructions and Static analysis tool and code coverage Report

Yiyun Zhang

3/12/2021

Manual:

To run this program, go to Eclipse and open **Chess.java**, **ChessTest.java** and **Main.java** under the same package. Go go ChessTest.java and right click the file and choose **Coverage as -> JUnit Test** as Figure 1 shows, in the Console window you will see the output for each test case, in Coverage you will see the code coverage information. You can run **Main.java** as a Java Application as well, it runs the example in the assignment: it takes input and output the result as the instructions in the assignment document. The *whiteInput*, *blackInput* and *pieceInput* String variables take board configuration from the user input. The *setBoard()*, *getBoard()* and *getMove()* methods perform outputs and calculations.



```
public class Chess {
    private String[][] board = new String[8][8];

    public String[][] setBoard(String whiteInput, String blackInput) {

        for (int i = 0; i < 8; i++) {
            for(int j = 0; j < 8; j++) {
                board[i][j] = "0";
            }
        }

        whiteInput = whiteInput.replaceAll(",\\s*", "");
        while (!whiteInput.equals("")) {
            int x = Integer.parseInt(Character.toString(whiteInput.charAt(2)-1));
            int y = Integer.parseInt(Character.toString(whiteInput.charAt(1)-49));
            String cur = "W"+ Character.toString(whiteInput.charAt(0));
            board[x][y] = cur;
            whiteInput = whiteInput.substring(3,whiteInput.length());
        }

        blackInput = blackInput.replaceAll(",\\s*", "");
        while (!blackInput.equals("")) {
            int x = Integer.parseInt(Character.toString(blackInput.charAt(2)-1));
            int y = Integer.parseInt(Character.toString(blackInput.charAt(1)-49));
            String cur = "B"+ Character.toString(blackInput.charAt(0));
            board[x][y] = cur;
            blackInput = blackInput.substring(3,blackInput.length());
        }

        return board;
    }

    public void getBoard() {
        for (int i = 7; i >=0; i--) {
            for(int j = 0; j < 8; j++) {
                System.out.format("%6s", board[i][j]+"   ");
            }

```

```
import org.junit.Test;

public class ChessTest {

    @Test
    public void testSetBoard() throws Exception {
        Chess testSubject = new Chess();
        String whiteInput = "Rf1, Kg1, Pf2, Ph2, Pg3";
        String blackInput = "Kb8, Ne8, Pa7, Pb7, Pc7, Ra5";
        String[][] result;

        // default test
        result = testSubject.setBoard(whiteInput, blackInput);
    }

    @Test
    public void testGetBoard() throws Exception {
        Chess testSubject = new Chess();
        testSubject.getBoard();
    }

    @Test
    public void tes
        Chess testS
        String whit
        String blac
        testSubject
        String piec

        //test1
        whiteInput
        blackInput
        testSubject
        pieceInput
        testSubject

        //test2
        whiteInput
        blackInput
```

```
<terminated> ChessTest [JUnit] C:\Users\benny\.p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
LEGAL MOVES FOR Ph1: None


LEGAL MOVES FOR Ph8: h7


LEGAL MOVES FOR Pa1: None


LEGAL MOVES FOR Pa1: None


LEGAL MOVES FOR Pa8: None


LEGAL MOVES FOR Pa8: a7


LEGAL MOVES FOR Ph1: None


LEGAL MOVES FOR Ph1: None


LEGAL MOVES FOR Ph8: None


LEGAL MOVES FOR Ph8: h7
```

ChessTest (Mar 12, 2021 8:50:56 PM)

Element	Coverage	ered Instructions	ssed Instructions	Total Instructions
chess-move	98.5 %	4,191	63	4,254
src	98.5 %	4,191	63	4,254
(default package)	98.5 %	4,191	63	4,254
Chess.java	98.0 %	2,879	60	2,939
Chess	98.0 %	2,879	60	2,939
bishopMove(String, int, in	88.5 %	399	52	451
queenMove(String, int, int	99.1 %	816	7	823
rookMove(String, int, int)	99.7 %	390	1	391
getBoard()	100.0 %	39	0	39
getMove(String)	100.0 %	98	0	98
kingMove(String, int, int)	100.0 %	424	0	424
knightMove(String, int, int	100.0 %	463	0	463
pawnMove(String, int, int)	100.0 %	118	0	118
setBoard(String, String)	100.0 %	124	0	124
TestSuite.java	0.0 %	0	3	3
ChessTest.java	100.0 %	1,312	0	1,312

Figure 1

Test cases:

testSetBoard(): The **testSetBoard()** method tests the board's setup function by accepting configuration from the user.

testGetBoard(): The **testGetBoard()** method tests board's print function.

testGetMove(): The **testGetMove()** method tests a piece's possible move output.

testKingMove(): The **testKingMove()** method tests all situations of King piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.

testQueenMove(): The **testQueenMove()** method tests all situations of Queen piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.

testRookMove(): The **testRookMove()** method tests all situations of Rook piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.
testKnightMove(): The **testKnightMove()** method tests all situations of Knight piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.
testBishopMove(): The **testBishopMove()** method tests all situations of Bishop piece's move, including: move to all directions, partially blocked by friendly pieces, fully blocked, move with take.
testPawnMove(): The **testPawnMove()** method tests all situations of Pawn piece's move, including: move to all directions as white, partially blocked by friendly pieces as white, fully blocked as white, move with take as white, move to all directions as black, partially blocked by friendly pieces as black, fully blocked as black, move with take as black

Static analysis tool improvements:
Since FindBugs is not working with the Eclipse Maven plugin, I use SpotBugs instead.
Before I run my program, the SpotBugs shows 8 bugs as Figure 2 shows.

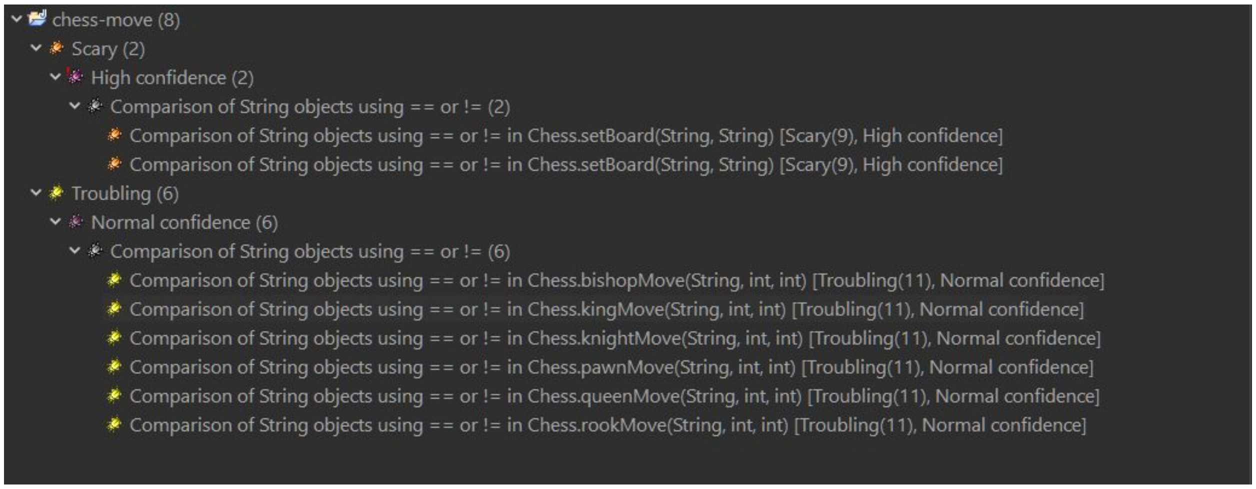

```
chess-move (8)
  Scary (2)
    High confidence (2)
      Comparison of String objects using == or != (2)
        Comparison of String objects using == or != in Chess.setBoard(String, String) [Scary(9), High confidence]
        Comparison of String objects using == or != in Chess.setBoard(String, String) [Scary(9), High confidence]
  Troubling (6)
    Normal confidence (6)
      Comparison of String objects using == or != (6)
        Comparison of String objects using == or != in Chess.bishopMove(String, int, int) [Troubling(11), Normal confidence]
        Comparison of String objects using == or != in Chess.kingMove(String, int, int) [Troubling(11), Normal confidence]
        Comparison of String objects using == or != in Chess.knightMove(String, int, int) [Troubling(11), Normal confidence]
        Comparison of String objects using == or != in Chess.pawnMove(String, int, int) [Troubling(11), Normal confidence]
        Comparison of String objects using == or != in Chess.queenMove(String, int, int) [Troubling(11), Normal confidence]
        Comparison of String objects using == or != in Chess.rookMove(String, int, int) [Troubling(11), Normal confidence]
```

Figure 2

I was using == and != to check and compare actual contents of String objects, which is a mistake in Java programming as Figure 3 and 4 shows.

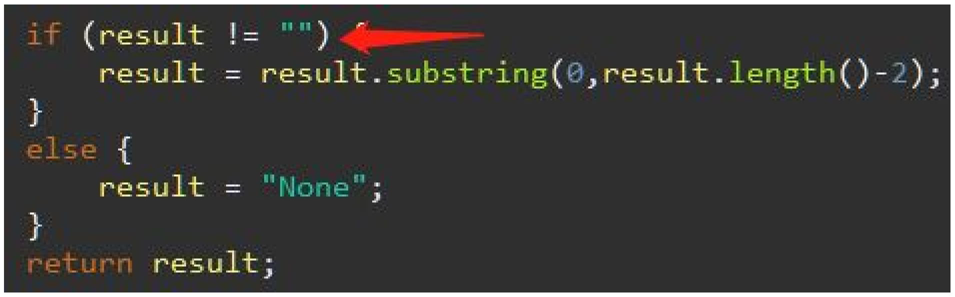
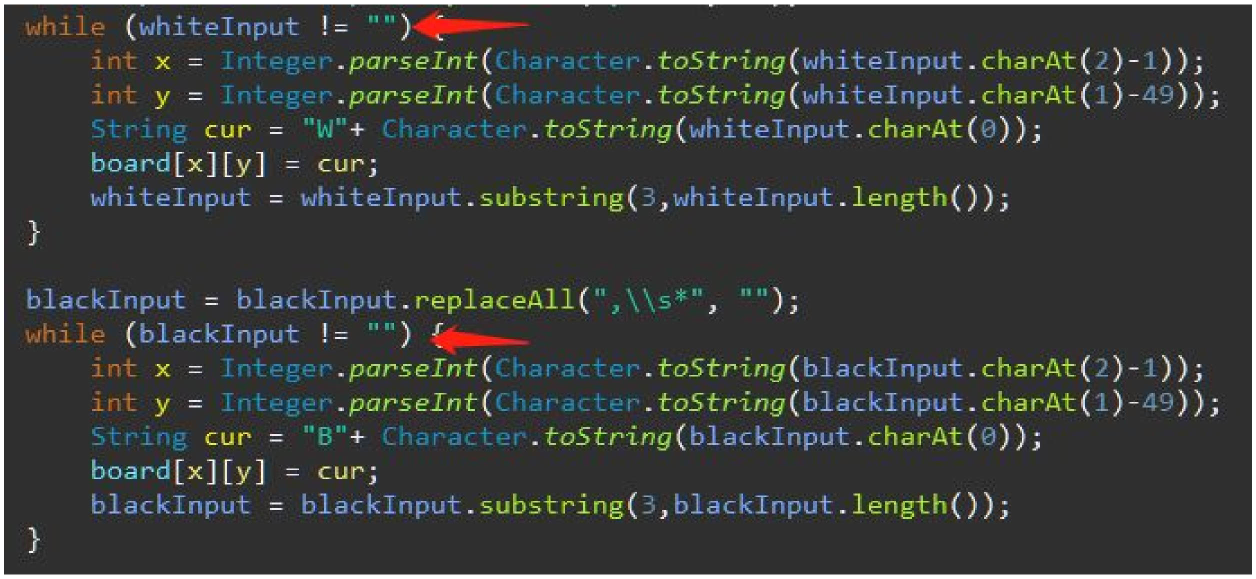

```
if (result != "") {
    result = result.substring(0,result.length()-2);
}
else {
    result = "None";
}
return result;
```

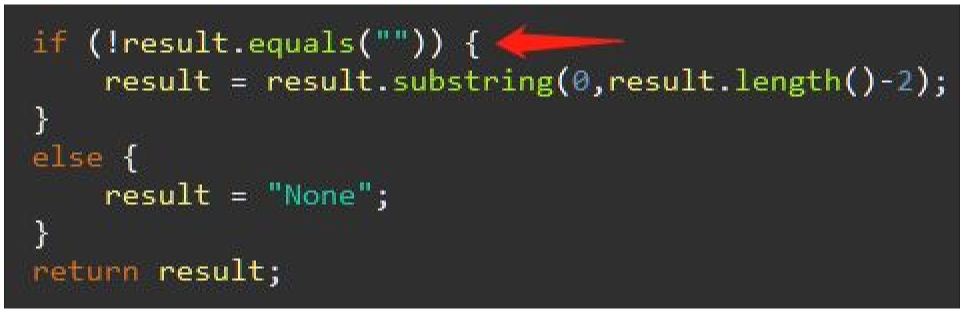
Figure 3



```
while (whiteInput != "") {
    int x = Integer.parseInt(Character.toString(whiteInput.charAt(2)-1));
    int y = Integer.parseInt(Character.toString(whiteInput.charAt(1)-49));
    String cur = "W"+ Character.toString(whiteInput.charAt(0));
    board[x][y] = cur;
    whiteInput = whiteInput.substring(3,whiteInput.length());
}

blackInput = blackInput.replaceAll(",\\s*", "");
while (blackInput != "") {
    int x = Integer.parseInt(Character.toString(blackInput.charAt(2)-1));
    int y = Integer.parseInt(Character.toString(blackInput.charAt(1)-49));
    String cur = "B"+ Character.toString(blackInput.charAt(0));
    board[x][y] = cur;
    blackInput = blackInput.substring(3,blackInput.length());
}
```

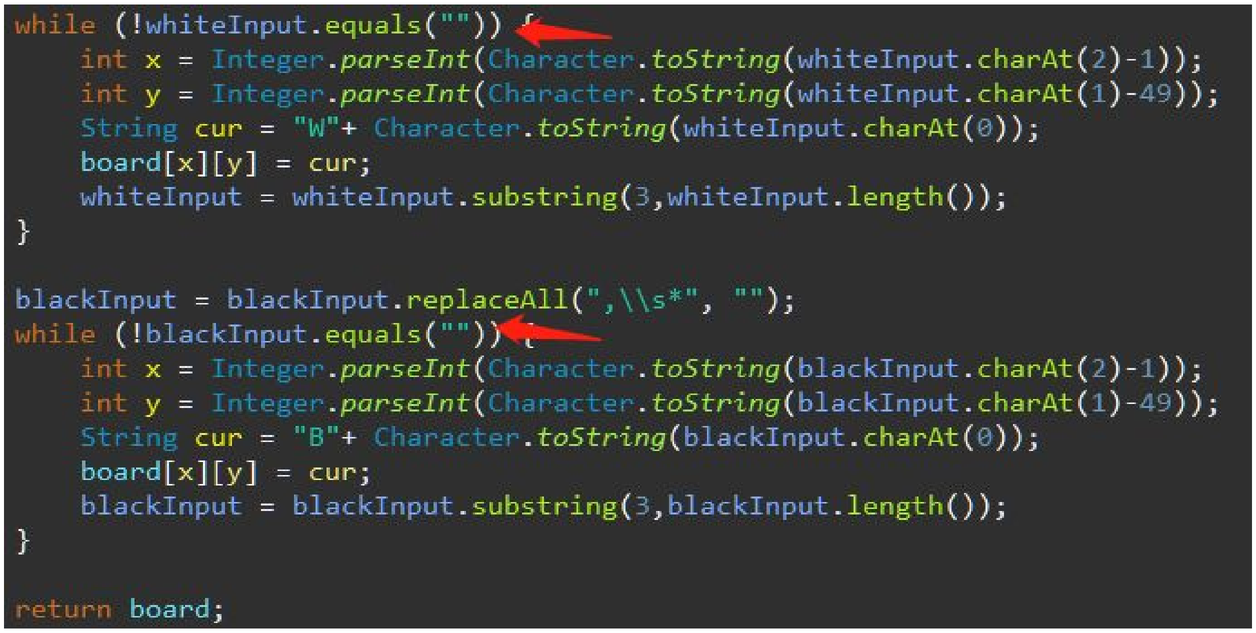
Figure 4

Therefore, I changed the comparison method to an appropriate String function: .equals() as Figure 5 and 6 shows.



```
if (!result.equals("")) {
    result = result.substring(0,result.length()-2);
}
else {
    result = "None";
}
return result;
```

Figure 5

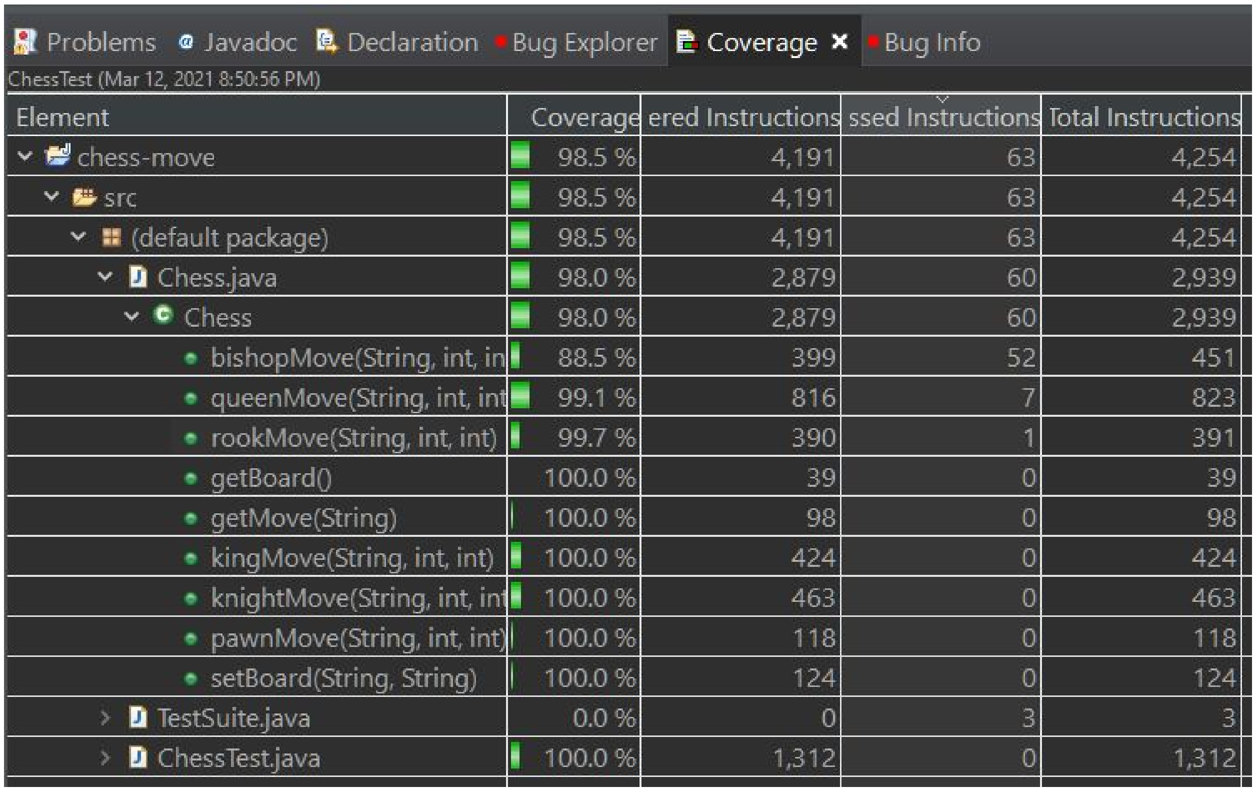


```
while (!whiteInput.equals("")) {
    int x = Integer.parseInt(Character.toString(whiteInput.charAt(2)-1));
    int y = Integer.parseInt(Character.toString(whiteInput.charAt(1)-49));
    String cur = "W"+ Character.toString(whiteInput.charAt(0));
    board[x][y] = cur;
    whiteInput = whiteInput.substring(3,whiteInput.length());
}

blackInput = blackInput.replaceAll(",\\s*", "");
while (!blackInput.equals("")) {
    int x = Integer.parseInt(Character.toString(blackInput.charAt(2)-1));
    int y = Integer.parseInt(Character.toString(blackInput.charAt(1)-49));
    String cur = "B"+ Character.toString(blackInput.charAt(0));
    board[x][y] = cur;
    blackInput = blackInput.substring(3,blackInput.length());
}

return board;
```

Figure 6

After making this modification, the SpotBugs no longer shows anything. Therefore I assume there is no bug in my program.

Final code coverage report:
The final code coverage is 98.5% shown as Figure 7. The missing 1.5% is due to the nested **if** statements in move methods of pieces. Although I added all possible situations in test cases, there are conflicts between situations that make certain conditions cannot be triggered while other conditions are satisfied. Therefore the nested **if** statements cannot be covered 100%.



ChessTest (Mar 12, 2021 8:50:56 PM)

Element	Coverage	ered Instructions	ssed Instructions	Total Instructions
chess-move	98.5 %	4,191	63	4,254
src	98.5 %	4,191	63	4,254
(default package)	98.5 %	4,191	63	4,254
Chess.java	98.0 %	2,879	60	2,939
Chess	98.0 %	2,879	60	2,939
bishopMove(String, int, in	88.5 %	399	52	451
queenMove(String, int, int	99.1 %	816	7	823
rookMove(String, int, int)	99.7 %	390	1	391
getBoard()	100.0 %	39	0	39
getMove(String)	100.0 %	98	0	98
kingMove(String, int, int)	100.0 %	424	0	424
knightMove(String, int, int	100.0 %	463	0	463
pawnMove(String, int, int)	100.0 %	118	0	118
setBoard(String, String)	100.0 %	124	0	124
TestSuite.java	0.0 %	0	3	3
ChessTest.java	100.0 %	1,312	0	1,312

Figure 7