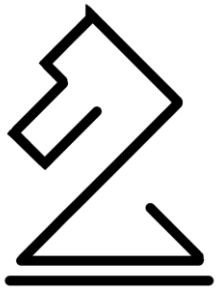
Java Chess J-Unit Descriptions



Submitted by: Tutorial 10, Group 01
April 10, 2019
Computer Science 233 – Lecture 02

Chess J-Unit Tests

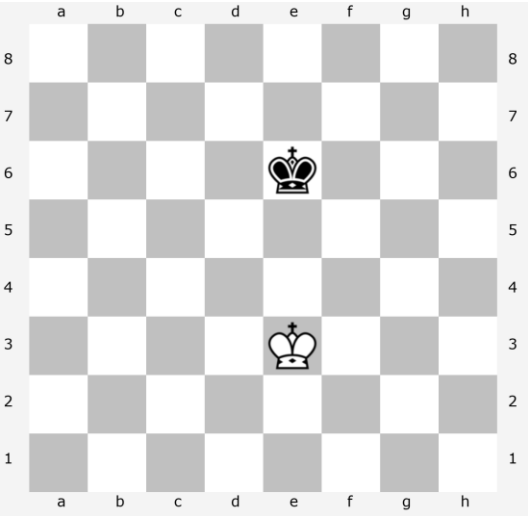
This document describes the tests used for piece movement using j-units. Each test comes with a copy of the code for the test, and a visual representation of the board for the test being conducted. A short explanation is provided of the test, as well as the expected outcome.

These tests were meant to comprehensively test the movement rules that have been implemented for each piece in this version of chess. Though it seems like there are few tests for general movement, each piece is tested. The majority of the tests are meant to test the more complicated rules, specifically: en passant, castling, and promotion. This means that both the Piece class is tested, and through it the individual pieces which are child classes of Piece.

Most of the tests work by testing the number of possible moves associate with a position on the board. The game is able to get a list of all moves that each piece is able to make. Because the class for each piece only evaluates if a given move is legal, the game checks the number of possible legal moves that each piece is able to make to determine if the pieces are evaluating possible legal moves correctly. One test produces two errors in the console printout that say “Error: Can’t put self in check.” This is printed by the game, but is the proper response to the test and so is a pass for that particular test.

Test 1

```java
@Test
public void testKing(){
    game.importGame("8/8/4k3/8/8/4K3/8/8 w - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 8; //the king
    assertEquals("King test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.changeTurn();
    assertEquals("King test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



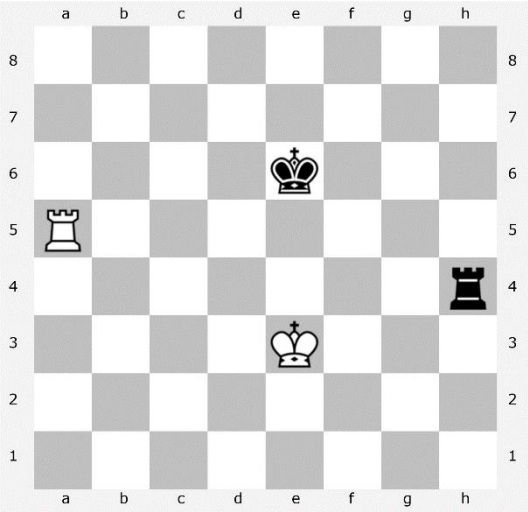
Player Turn: White

Test: Basic movement. Checks to make sure that the King can move one space in each direction.

Expected Moves: 8 (King)

Test 2

```
@Test
public void testKingSuicide(){
    game.importGame("8/8/4k3/R7/7r/4K3/8/8 w - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 5; //the king
    numOfLegalMoves += 14;// rook
    assertEquals("King suicide test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.changeTurn();
    assertEquals("King suicide test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



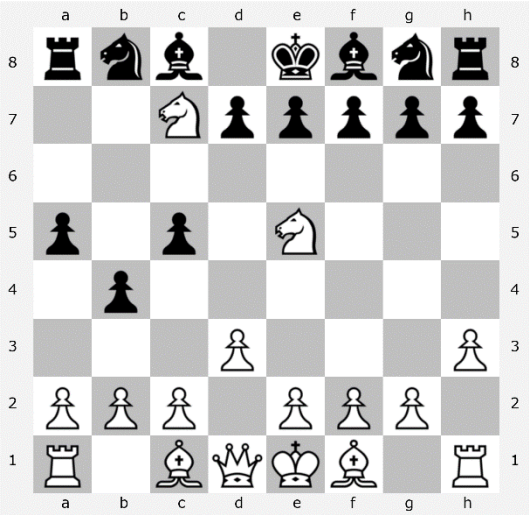
Player Turn: White

Test: Uses the rooks to make sure the King cannot move into a position it can be captured (or “suicide”). If Kings are working properly, they should only be able to move to the side and backward.

Expected Moves: 5 (King)
14 (Rook)

Test 3

```
@Test
public void testCheckmate(){
    game.importGame("rnb1kbnr/2Nppppp/8/p1p1N3/1p6/3P3P/PPP1PPP1/R1BQKB1R b KQkq - 1 7");

    int numOfLegalMoves = 1;// escape check
    assertEquals("Checkmate test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



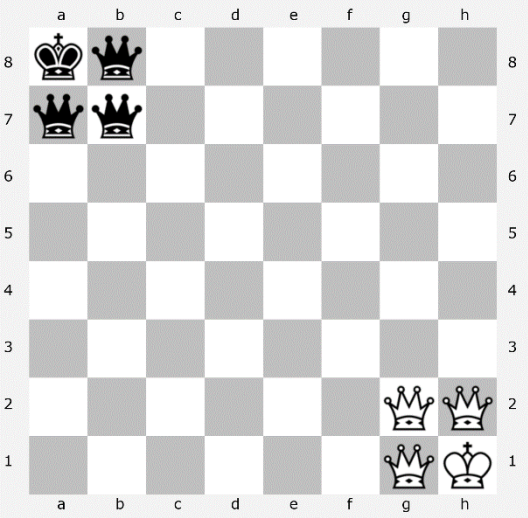
Player Turn: Black

Test: To test if the game recognizes the state of check. If the game is properly programmed, it should recognize that despite many pieces with many possible moves for Black, the only possible move is to move the Black King out of check.

Expected Moves: 1 (King)

Test 4 & 5

```java
@Test
public void testQueenWhite(){
    game.importGame("kq6/qq6/8/8/8/8/6QQ/6QK w - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 12;//othagnal moves
    numOfLegalMoves += 12 + 5;//diagnal moves
    assertEquals("Queen suicide test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



Player Turn: White

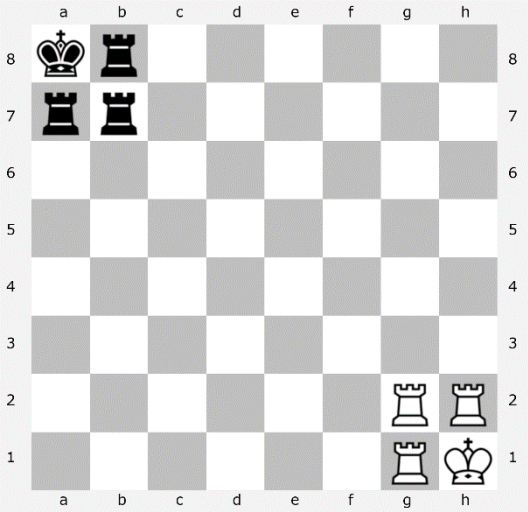
Test: The first test is for the movement of the queen. The Queen at g1 can move horizontally and diagonally, and the Queen at h2 can both move vertically and diagonally. The Queen at g2 can *only* move diagonally because if it moved vertically or diagonally it would expose the King to capture. The Queen at g2 is therefore pinned in place. If the game is working properly, it should only be able to move diagonally to keep protecting the King

Expected Moves: 29 (Queens)

Test 5 is the same test, except it being Black's turn.

Test 6

```java
@Test
public void testRook(){
    game.importGame("kr6/rr6/8/8/8/8/6RR/6RK w - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 24; //the rooks
    assertEquals("Rook test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.changeTurn();
    assertEquals("Rook test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



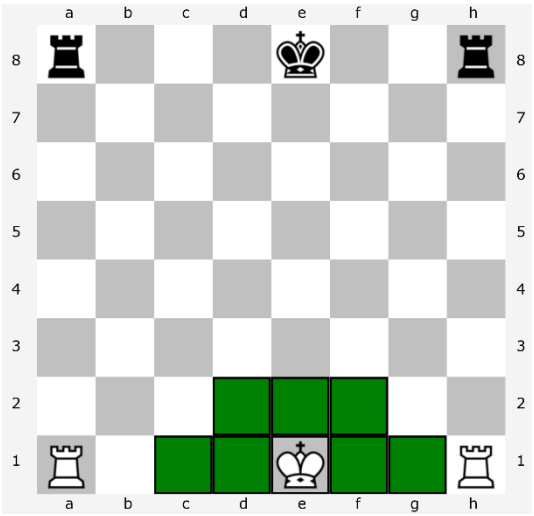
Player Turn: White

Test: The movement of Rooks. The Rook at g1 can only move horizontally. The Rook at h2 can only move vertically. The rook at g2 can move both horizontally and vertically.

Expected Moves: 24 (Rooks)

Test 7 & 8

```java
@Test
public void testBasicWhiteCastle(){
    game.importGame("r3k2r/8/8/8/8/8/8/R3K2R w KQkq - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 14; //rooks
    numOfLegalMoves += 5; //rooks
    numOfLegalMoves += 5; //the king
    numOfLegalMoves += 2; //2 castle moves
    assertEquals("White Castle Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



Player Turn: White

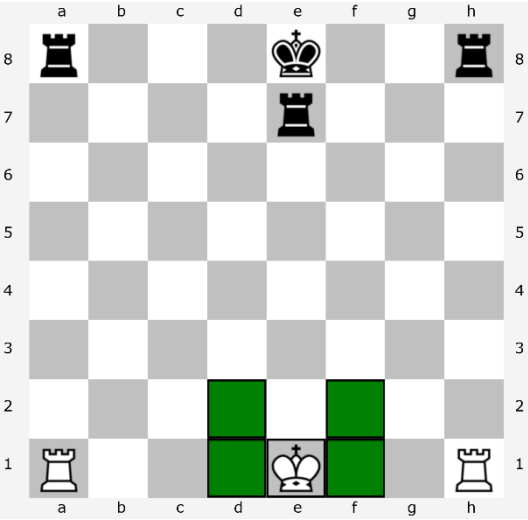
Test: The castling ability of the King. In this position the King should be able to move to the locations indicated in the picture.

Expected Moves: 7 (King)
19 (Rooks)

Test 8 is the same test, except it being Black's turn.

Test 9 & 10

```java
@Test
public void testInCheckWhiteCastle(){
    game.importGame("r3k2r/4r3/8/8/8/8/8/R3K2R w KQkq - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 0; //rooks
    numOfLegalMoves += 0; //rooks
    numOfLegalMoves += 4; //the king
    numOfLegalMoves += 0; //2 castle moves
    assertEquals("White Castle Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



Player Turn: White

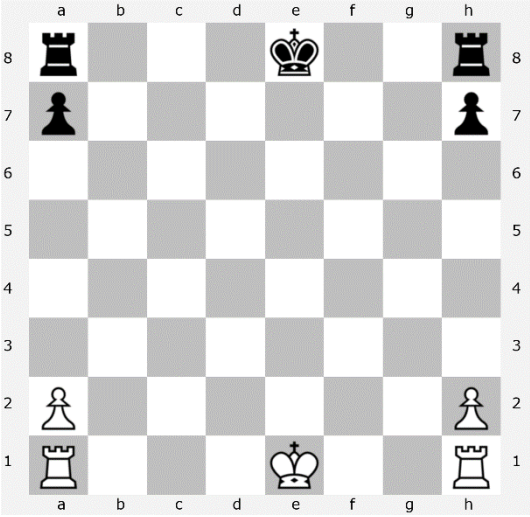
Test: The castling ability of the King. If programmed properly, the King should *not* be able to castle on either side because it is in check by the Black Rook at e7. The King can therefore only move itself out of check, and neither of the White Rooks can move to block, therefore the only possible moves should be the ones displayed in the picture to the left.

Expected Moves: 4 (King)

Test 10 is the same test, except the positions are reversed and it is Black's turn.

Test 11, 12, 13, 14

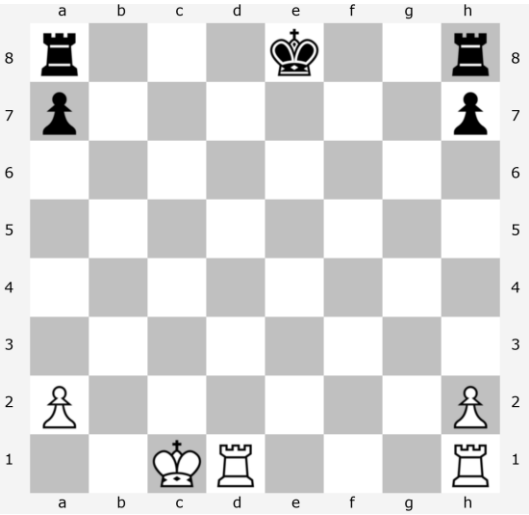
```java
@Test
public void testWhiteQueenCastle(){
    game.importGame("r3k2r/p6p/8/8/8/8/P6P/R3K2R w KQkq - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 5; //rooks
    numOfLegalMoves += 5; //the king
    numOfLegalMoves += 2; //2 castle moves
    numOfLegalMoves += 4; //pawns
    assertEquals("White Castle Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.move(new Move(new Cord(0,4), new Cord(0,2)));
    game.changeTurn();

    numOfLegalMoves = 0;
    numOfLegalMoves += 6 + 7; //rooks
    numOfLegalMoves += 4; //the king
    numOfLegalMoves += 4; //pawns
    assertEquals("White Castle Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



Player Turn: White

Test: The King's ability to castle to it's left (called the "Queen side"). This test checks the number of moves at the initial setup of the board (left) and then again after the King has castled (left, lower).

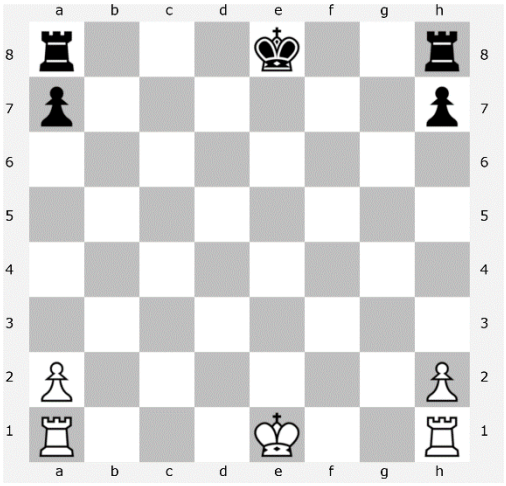
Expected Moves: 16 (before castling)
21 (after castling)



Test 12 is the same test, except for Black King castling on Queen side. Test 13 and 14 are the same except for castling on the opposite side ("King side").

Test 15, 16, 17, 18

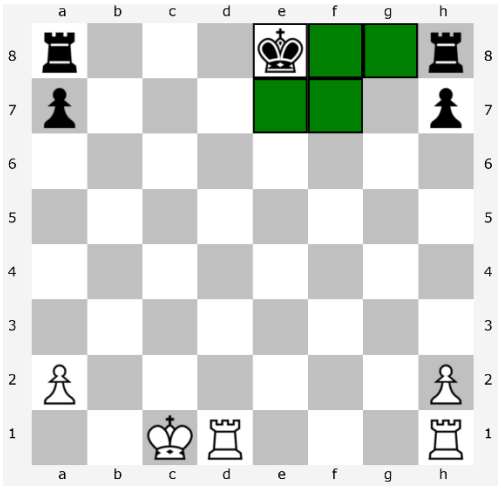
```
@Test
public void testWhiteQueenCastleCheck(){
    game.importGame("r3k2r/p6p/8/8/8/8/P6P/R3K2R w KQkq - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 5; //rooks
    numOfLegalMoves += 5; //the king
    numOfLegalMoves += 2; //2 castle moves
    numOfLegalMoves += 4; //pawns
    assertEquals("White Castle Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.move(new Move(new Cord(0,4), new Cord(0,2)));

    numOfLegalMoves = 0;
    numOfLegalMoves += 5; //rooks
    numOfLegalMoves += 3; //the king
    numOfLegalMoves += 1; //2 castle moves
    numOfLegalMoves += 4; //pawns
    assertEquals("White Castle Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



Player Turn: White

Test: Tests to make sure that the King cannot castle by moving through a square that is under attack. The rules of castling require that the spaces between the King and the Rook are clear of pieces, and that none of the squares that the King will move through are under attack. This test castles the White King to the left (Queen side, picture left) and then tests the number of moves available to the Black player (picture lower left). The Black King should not be able to castle to it's left side as d8 is under threat by the Rook as d1, but can still castle King side.

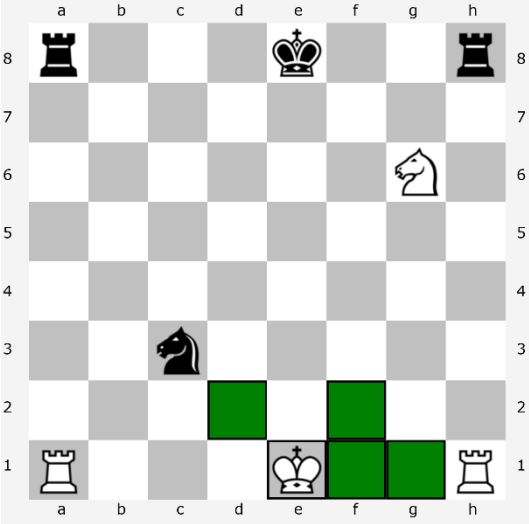
Expected Moves: 13 (cannot castle Queen side)



Test 16 is the same, except testing White's ability to castle when a Black Rook is attacking Queen side. Tests 17 and 18 test the same mechanic except for castling to the King side when a Rook has one of those squares under attack.

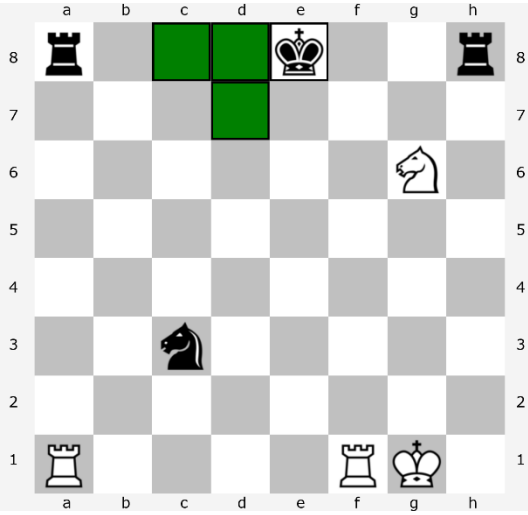
Test 19

```java
@Test
public void testCastleGeneral(){
    game.importGame("r3k2r/8/6N1/8/8/2n5/8/R3K2R w KQkq - 0 1");

    tester.move("e1d1");//should fail
    tester.move("e1c1");//should fail
    tester.move("e1g1");//should work

    tester.move("e8g8");//should fail
    tester.move("e8f8");//should fail
    tester.move("e8c8");//should work

    assertEquals("Castle General failed", "2kr3r/8/6N1/8/8/2n5/8/R4RK1 w - - 2 2", GameInfo.toFEN(game));
}
```



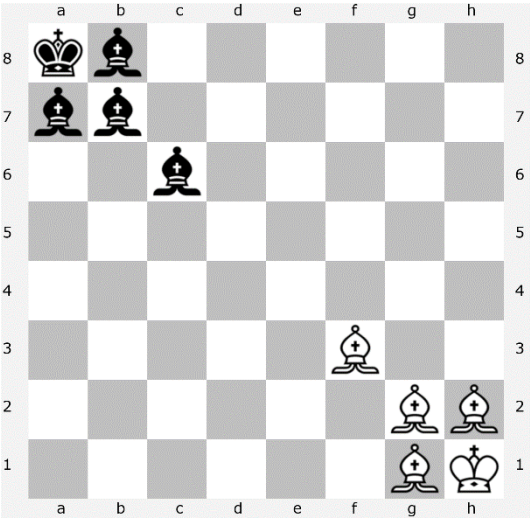
Player Turn: White

Test: The ablity of the King to move to or through either of the castling squares to the left (Queen side) while one of those squares is under attack. In this case the White King attempts to move to c1 and d1 but should not be able to because the Black Knight at c3 is threatening d1. If neither of those moves was made, the White King should then castle to g1 (picture left). The Black King then attempts to make two illegal castling moves King side, and then should castle Queen side to c8 if those moves were not made (pictured lower left). The test then checks the state of the board through the FEN to make sure it matches the expected end state.



Test 20

```java
@Test
public void testBishop(){
    game.importGame("kb6/bb6/2b5/8/8/5B2/6BB/6BK b - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 9 + 12;
    assertEquals("Bishop test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.changeTurn();
    assertEquals("Bishop test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



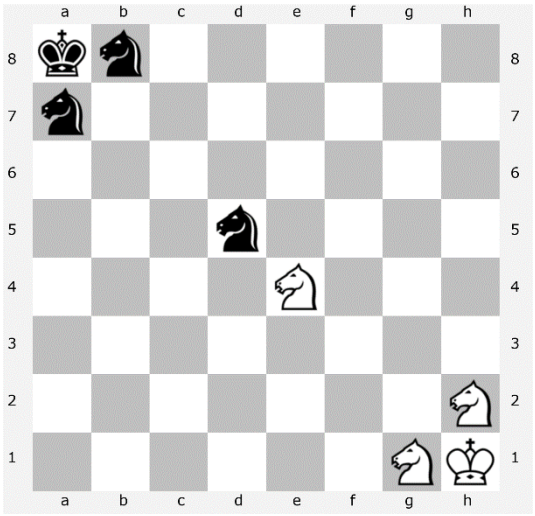
Player Turn: Black

Test: General movement rules of the Bishop. Takes the placement of the pieces seen on the left and tests the number of legal moves for the Bishops, first on the Black side and then changes the turn to White and runs the same test.

Expected Moves: 21 (Bishops)

Test 21

```java
@Test
public void testKnight(){
    game.importGame("kn6/n7/8/3n4/4N3/8/7N/6NK w - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 6; //knights corner
    numOfLegalMoves += 8; //center knight
    numOfLegalMoves += 1; //the king
    assertEquals("Knight test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.changeTurn();
    assertEquals("knight test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



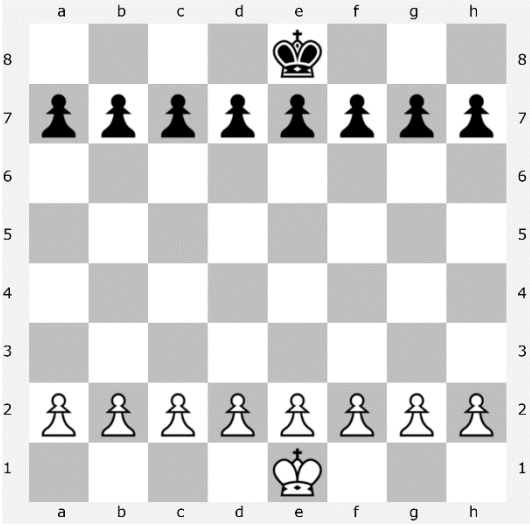
Player Turn: White

Turn: General movement rules of Rooks. Checks a Rook near the middle of the board that should have every move available, and also checks two Rooks in a corner to check for edge cases (that the Rook cannot jump off the edge). Then changes turns and runs the same test for Black.

Expected Moves: 14 (Knights)
1 (King)

Test 22

```java
@Test
public void testPawnAdvance(){
    game.importGame("4k3/pppppppp/8/8/8/8/PPPPPPPP/4K3 w - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 16; //pawns
    numOfLegalMoves += 2; //the king
    assertEquals("Pawn Advance test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.changeTurn();
    assertEquals("Pawn Advance test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



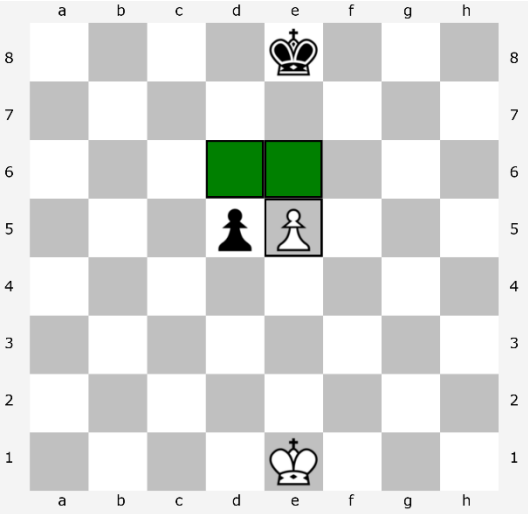
Player Turn: White

Test: Test of basic Pawn movement from their starting positions. If functioning properly, each Pawn should be able to move two spaces forward. Then changes turns and makes sure that Black Pawns have the same number of moves.

Expected Moves: 16 (Pawns)
2 (King)

Test 23 and 24

```
@Test
public void testWhiteEnPassant(){
    game.importGame("4k3/8/8/3pP3/8/8/8/4K3 w - d6 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 2; //pawn
    numOfLegalMoves += 5; //the king
    assertEquals("White En Passant test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());

    game.makeMove(new Move(new Cord(4, 4), new Cord(5, 3)));

    numOfLegalMoves = 0;
    numOfLegalMoves += 4; //king
    assertEquals("White En Passant test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



Player Turn: White

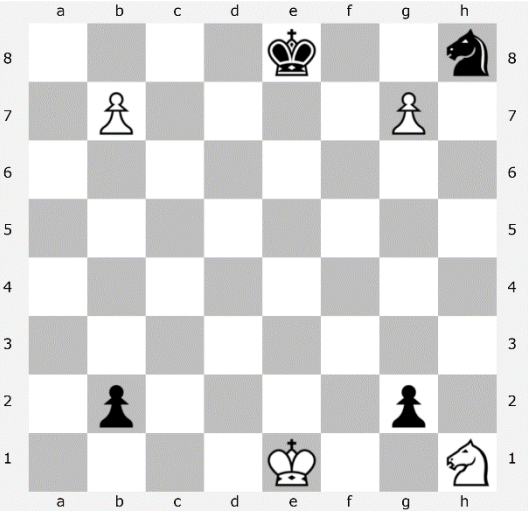
Test: En passant ability of Pawns. Pawns have a unique ability to capture an opposing Pawn that moves two spaces forward on it's first move. This test simulates the Black Pawn moving two spaces from d7 to d5. This means that the White Pawn would be able to capture the Black Pawn by moving to d6 (pictured left), but only on it's next move. If a player does not immediately take the opportunity to capture en passant, they loose the ability to capture.

Expected Moves: 2 (Pawn)
5 (King)

Test 24 is the same test, just reversed to test the Black Pawn's ability to capture en passant.

Test 25 & 26

```java
@Test
public void testWhitePromotion(){
    game.importGame("4k2n/1P4P1/8/8/8/8/1p4p1/4K2N w - - 0 1");

    int numOfLegalMoves = 0;
    numOfLegalMoves += 3 * 4; // knight
    numOfLegalMoves += 2; //knight
    numOfLegalMoves += 4; //king
    assertEquals("White promotion test Failed", numOfLegalMoves, GameHelper.allLegalMoves(game).size());
}
```



Player Turn: White

Test: Promotion ability for Pawns once they reach the opposite side of the board. The White Pawns have three possible moves: b8, g8, and the capture of the Black Rook at h8. All of these moves will result in the White Pawn getting promoted, and the game treats the four possible promotions (Knight, Rook, Bishop, Queen) as distinct move possiblities. Three possible moves that result in four possible promotions means the two pawns have 12 possible moves (or outcomes from moving).

Expected Moves: 12 (Pawns)
2 (Knight)
4 (King)

Test 26 is the same test for Black Pawn promotion.