Project 3 Test Report

In this project, most of the test will be in the Junit, and the method that involves the display will be tested by myself, and if it is necessary, I will provide screen shots to proof that it works correctly.

ChessPiece Class

In this class, here are all the method:

1. getSide
2. getChessBoard
3. getLabel

The three getter methods will be tested in the same trial. Since there will be only one possible output, the test is simple.

Tester name: testGetter

In the test, the element input that is in the constructor should be the same as the getter method gets.

1. getIcon

Since it returns null, no test is provided with this method

1. setLocation
2. getRow
3. getColumn

Since method 5,6 can verify method 4, so they are tested as a whole

Tester name: testLocation

When testing the getter method, the element input in the constructor should be the same as the getter method results.

When testing the setter method, the input of setter should be the same as the getter methods get

1. moveDone

Since it moves the target piece to new location and also remove the old piece, and does not involves with any condition, only one test is provided, which is to move one piece to another location.

1. isLegalMove
2. isLegalNonCaptureMove
3. isLegalCaptureMove

Method 9,10, 11 will be inherited to each individual chess piece and they will act differently, so they will not be tested in this class.

Now, all the methods in the ChessPiece class is tested and they all work functionally

Since nearly all the methods are inherited, there are only isLegalMove, isLegalNonCaptureMove isLegalCaptureMove three methods that various in each ChessPiece. I use isLegalMoveDiagnoal, and isLegalMoveStraight two methods to solve most of the isLegalMove methods in each chess piece.

Since isLegalMoveStraight is identical to Rook’s isLegalMove, and isLegalMoveDiagnoal is identical to Bishop’s isLegalMove, and

Queen = isLegalMoveDiagnoal + isLegalMoveStraight If one of them returns true, it can move. I will only test Bishop and Rook, and only simple Queen tests to show that it works.

Since isLegalNonCaptureMove default situation is to return the result of isLegalMoveStraight and all five pieces(except for king) shares the same method, it is meaningless to have a separate tester for it. In some piece test, there will be two assertEquals to verify that the output of isLegalMoveStraight and output of isLegalNonCaptureMove are the same.

Since isLegalCaptureMove only verifies that there is an enemy piece at the destination, and the piece can move to there, and all five pieces(except for king) share the same method, it will only be tested twice as the first isLegalMove tries to move to the destination that has an enemy. The assertEquals will verify that the output of them are the same.

The method isLegalNotMove is called before every isLegalMove, which is to forbid not moving, it will only be tested at the first isLegalMove test, and when not moving, it returns false. (It is the test0 for all Move related method)

To better understand my location of each piece, here is a game design of each piece.

R56(black color) means: rook, row5, column 6, south side

R56(red color) means: rook, row5, column 6, north side

C means Captured

CF means Captured Failed

A means it will be add after all other spots is tested

The green n means cannot move to that location

The black y means can move to that location

The blue line just separates two different game situations.

Test Rook and Bishop (move straight and diagonal)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  | n |  |  |
| 1 |  |  |  |  |  | R-CF |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  | R-CF |  |  |  |  |
| 4 |  |  |  | n | n | y-R-C |  |  |
| 5 | n | R-CF | R-C-A | y | y | R | R-CF | n |
| 6 |  |  |  |  |  | y |  |  |
| 7 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  | B-C-A |  |  | B-CF |  |  |
| 3 |  |  |  | y |  |  | n |  |
| 4 | y |  | y |  | y | n | y-R-C |  |
| 5 | y | y |  |  |  | B |  |  |
| 6 | Q |  |  |  | y |  | B-CF |  |
| 7 |  |  |  | B-CF |  |  |  | n |
| 8 |  |  | n |  |  |  |  |  |

Since the logic behind them are the same, the test will be explained together.

They will first test that it can move one spot (test 1), then move many spot (test many)

Then test that it cannot move to other direction or with obstacle.

Then test that it can capture enemy when it can move there, then it cannot capture enemy on strange spot or capture friend piece.

Test Knight and King together for saving memory

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | y | y | y |  |  |
| 1 | k-cf | **k** | y |  |  |
| 2 | y |  | K-C | K-CF |  |
| 3 |  | n |  |  | n |
| 4 |  |  |  |  |  |
| 5 |  | y | Knight | y | N-cf |
| 6 | y | Knight | Knight | n | y-N-CF |
| 7 | n | Knight | **Knight** | N-cf |  |
| 8 | N-C |  |  | K-CF | y |

Here is the test for knight and king. For testing knight, since it can move anywhere regardless of the situation of its path, I will just test it with multiple move direction and test that it can only capture enemy piece on the spot that it can moves to.

The test of knight is only about the dark one. It will test that it can move all spots next to him and no spots further can be moved, also it can only capture the enemy on the spot that can move to.

Test Pawns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 |  |  |  |  |  | n |
| 1 | p | p | p | p |  |  |
| 2 |  |  |  |  |  | p |
| 3 |  |  |  |  |  |  |
| 4 | y | n |  |  | P-CF |  |
| 5 | y | P-n |  | P-C |  | P-CF |
| 6 | p | p |  | P-CF | p | n |
| 7 |  |  |  |  | n |  |

Regular move:

Pawn can move two spots in the first turn and then it is baned. Then since it involves multiple move, I test it manually, which is 0 moves (can move two spot), 1 moves (cannot) and 2 moves (cannot).

Then it will test the pawn can/cannot do regular move with/ without obstacle, and can/cannot capture enemy and cannot capture friend piece.

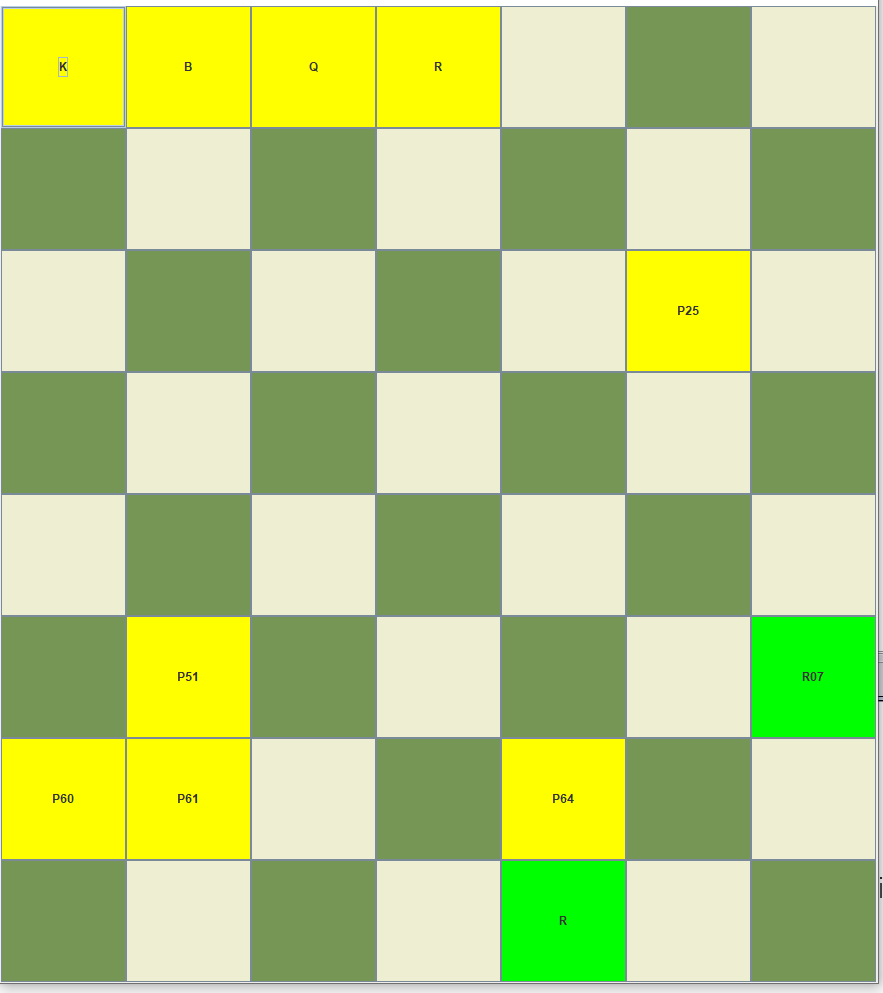
Special methods:

Pawn Upgrade

By using the same chess board as testing isLegalMove of Pawn, form P10 to P13 they will be moved to the front line and change to each upgradable piece and here is the screen shot after upgrading, for testing purpose.

The rook at the bottom is a pawn in the north side turns to rook.

Here is the screen shot of upgrade for pawn to four different pieces



The method works functionally.

Castle Move

Since it is the most complex method, I will first test that without any interruption, the two side two direction castle move will work functionally. Then I will test that with 1 or 2 threaten the castle move will fail, which corresponds to the red R-1 and R-2.

Then with an obstacle on the moving route, which is the red R-1 and R-2. Piece with -1 and -2 sign will be added to the test after the legal move without obstacle test trial, be added one by one.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | R | R-1 | R-1 |  | K |  |  | R |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  | R-1 | R-2 |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 | R |  |  |  | K |  |  | R |

Then I will test that after the King/Rook is moved that it will fail. It is tested on the main method by using the game situation provided above. Moving K04 and R77 forward and backward, then test (which should fail) then move R70 once and test(fail too.)

The method works functionally

EuropeanChess Class

1. legalPieceToPlay

It determines that I can move the method that I choose or not, and if I choose the other side’s method or want to move my side’s piece illegally, it will not allow me to move it. Thus, by move other side’s piece, move my piece to illegal location ( both failed) and move my piece to correct location (succeeded), the method is tested working functionally.

1. makeMove

After a person’s wish to move a piece it’s judged legal, this method is called. Thus, I made serval legal move and I can move the piece as I want. No display or setting wrong, it is tested functional.

1. canChangeSelection

At this point, this method is useless??? No test is provided.

Main Method

It should sets up the game correctly, and here is the screen shot of it.

