Project 5 Test Report

1. ChessBoard  
   Because it is an interface, I test their method by using the SwingChessBoard
2. getGameRules

Because it is impossible to compare two game rule variables, I create a method in the EuropeanChess called getGameName. Then I verify that the game name of the input and output are the same in the testGetRule method.

The rest of the methods are related with pieces, so they are placed in the testPieceStuff method.

1. addPiece
2. getPiece
3. hasPiece

Because add, get and has parts are the same, I test them together. \

Because it is impossible to compare two piece variables, I verify that the piece label of the input and output are the same to prove that the addPiece and getPiece methods works functionally. Then I use the hasPiece method to verify that the piece is there.

1. removePiece

I remove the piece that I added previously, then I use hadPiece method to verify that piece is not there anymore.

1. squareThreatened

I put an enemy rook in front of the king, and verify that the king is threated.

All of the function works correctly.

1. SwingChessBoard

Because it is identical to the code of project 3 and I just move some of the methods to the ChessBoard class for better hierarchy, so nothing is changed. After successfully run the tester of the project 3 and the EuropeanChess class can be played, nothing is tested.

1. SwingChessBoardDisplay

Since it is identical to the one of project 3 from professor except for some type name changed with a swing in front of the type, nothing is tested.

1. SwingEuropeanChessDisplay

Since it is identical to the one of project 3 from professor except for some type name changed with a swing in front of the type, nothing is tested.

1. ChessGame

Because all the methods are abstract, nothing is tested here.

1. EuropeanChess

There are three new methods:

1. getNumRows
2. getNumColumns

For the previous two methods, the tests are in the ChessGameTester, testAdditionalMethods methods and they just verify that it out put 8 and 8.

1. startGame

Since it only adds pieces to the game, I just use the start method of JavaFXChessBoard and use it to launch both JavaFX version of the game and the swing version of the game, and after all pieces pops up on the windows correctly, I know that it works functionally.

1. JavaFXChessBoard

Since the inherited methods of the FX version of chess board is identical to the swing version chess board, no test relates with the array of ChessPieces is provided here.

The other methods that relates with JavaFX is tested:

Since most of the addPiece and removePiece function is taken by the event handler, after adding and capturing some pieces, I know that these two functions works well.

The two new methods are the main and the start:

After typing combination of Xiangqi and JavaFX, Chess and Swing, Chess and JavaFX, and they all launch the correct version of the game and typing any other argument will not launch the start method, the main method is tested, and everything works.

After launch each version of the game and knowing that the rows and columns displayed correctly and each piece works functionally, the start method is tested and works well.

1. JavaFXChessBoardDisplay

Since it is an interface, nothing is provided here.

1. JavaFXEuropeanChessDisplay

There are three methods: displayEmptySquare, displayFilledSquare, and highlightSquare, after launch the game and seeing that the empty square, square with a piece and square that is selected are all display the wanted graph, these methods are tested, and all works functionally.

1. JavaFXXiangqiDisplay

Same as the previous one.

1. Xiangqi

The three old methods are the same as the one of European Chess class, and the test is included in the project 3 tester, no test is provided in project 5 tester.

There are three new methods:

1. getNumRows
2. getNumColumns
3. For the previous two methods, the tests are in the ChessGameTester, testAdditionalMethods methods and they just verify that it out put 9 and 10.
4. startGame

Since it only adds pieces to the game, I just use the start method of JavaFXChessBoard and use it to launch both JavaFX version of the game and the swing version of the game, and after all pieces pops up on the windows correctly, I know that it works functionally.

Since every piece can capture whatever enemy piece if it can move to there (except for Cannon, no isLegalCaptureMove is tested except for Cannon.

Also, because isLegalNonCaptureMove is always legal if it can move there, so no test is provided here.

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.

1. One special method: All scenario is tested in testKingFace, and everything works.

isKingFace: because in Xiang Qi, two king cannot face each other, so when a piece is moving, it cannot create a two king face each other scenario, so in every isLegalMove, this method will be called first, and if it creates a two kings face scenario, it is not a legal move.

Here is the test board.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | K |  | K | K | K |
| 1 | R | y | R |  |  |
| 2 | R | y | n | R |  |
| 3 | K |  | y | K | n |

In column0 test that when other piece covers two king, one piece can move to other location.

In column 2, it tests that one side king can move to the other column when a king is there with cover.

In column 3, when no other piece is in between, the only piece cannot move.

In column 4, it tests that one side king cannot move to the other column when a king is there with no cover (neither capture or no capture).

1. XiangqiKingPiece

Because in the previous test, it tested two king face situations, there are two scenarios: All function works correctly in testKingPiece.

1. Regular move

Testing it can forward straight and diagonally, and it can capture straight and forward diagonally, and since the is legal capture is inherited from project 3： here is the test board, and it is in the testKingPiece:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | y | y | y |
| 1 | y | K |  |

Because moving backward need two move, so it is tested by hand.

1. It cannot move out the 3\*3 grid.

Because this method is not hard coded, so it need to move twice to make it goes out of it bound, I tested it by hand and after two move, it cannot move away from its 3\*3 grid, then it is tested, and everything is working.

1. GuardPiece

There are two scenarios: All function works correctly in the testGuardPiece.

1. Regular move

It can capture enemy piece and cannot capture friend piece, and it can move diagonally one space. And it is using the standard board with one additional enemy piece, so no board is redesigned.

1. It cannot move out the 3\*3 grid.

Because a guard is always next to the king, so they have the same moving section, as the king’s 3\*3 move method, it is tested by the same way.

1. ElephantPiece

There are two scenarios: All function works correctly in the testElephantPiece.

1. Regular move

Because except for moving with other things in between and cannot move when it is not two square, other methods are the same as the one of project 3, so only the above feature is tested.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 | n |  |  |  |  |  |
| 7 |  | y |  |  |  | n |
| 8 |  |  | n |  | R |  |
| 9 |  |  |  | E |  |  |

1. It cannot go to the other center

Because it requires moving twice to go over its bond, it is tested manually that see whether it can to the other side or not.

1. HorsePiece

There are two scenarios: All function works correctly in the testHorsePiece.

1. Regular move

Since it is the same as the Knight piece of project 3, nothing is tested.

1. It cannot jump over pieces

Here is the board

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | n |  | n |
| 1 |  |  |  |
| 2 |  | R |  |
| 3 | R | H | R |
| 4 |  | R |  |
| 5 | n |  | n |

1. RookPiece

Because rook piece is identical to the one of European chess, the test is in the project 3 test report.

1. CannonPiece

There are two scenarios: All function works correctly in the testCannonPiece

1. Regular move

Without capture, it is the same as rook and method is inherited from the ChessPiece class from project 3, so no test is provided.

1. Capture another piece when something is in the middle

Here is the board

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 |  |  |  |
| 1 | R -CF-C-CF | R -C | R -C |
| 2 |  | R |  |
| 3 | R |  | R |
| 4 | R |  |  |
| 5 | C | C | C |
| 6 | C | C | R -C |
| 7 | R -C |  |  |

In row 1, first if will capture fail the enemy rook, then add a rook in between, the successfully capture the enemy, then add another one, then fail again. Then it tests that it can capture something behind it

In row 2, 3 it will capture an enemy when something is in between or near the enemy.

In column 6 and 8, it tests that it can capture something on the left or right enemy piece.

1. SoldierPiece

There are two scenarios: All function works correctly in the testSoldierPiece

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 | S |  |  |  |
| 4 |  | S | S |  |
| 5 |  | S | S |  |
| 6 | S |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |

(red is north side and black is south side)

1. Move toward enemy vertically in its own side

For S03, S06, S14, and S15, they all cannot go left or right, and they can go straight.

1. Move horizontally after cross the middle line

S42 and S52 are over the board line, so they can go left right freely.

Then testing that piece cannot go over one gird.

At this point, every method is tested, and all works functionally.