Project 5 Testing Document

Nobel Zhou (nxz157)

**6 December 2020**

Introduction

This the Project 5 testing document for all methods and classes. This chess program represents a playable game of chess, including promotion, check, checkmate, stalemate, draw by threefold repetition, and draw by 50 move rule. All moves are checked to make sure they don’t result in check. In addition, the chess board has been updated to add piece icons instead of just the symbol and the available moves are also highlighted when you click on the piece. There are a lot of methods that are unnecessary in the light of grading for this program but deemed necessary for the functionality and accuracy of creating a chess program. In addition, I will **not** be testing any methods that were pre-coded. I will be testing interface and abstract class methods through a concrete class because they are not available to be tested through their instances because they don’t have one. This testing document will be broken down and tackled within each class file. In addition, I have attached images containing the chess positions and the expected results of the resulting position if Junit is not feasible for testing those types of interactions. All chess board visualizations were taken using the analysis board at pychess.com/editor/xiangqi. Also, the testing uses the TestChessBoard class to test all methods related to the pieces and the game rules so the GUI doesn’t get in the way of things. I will also **not** be testing any methods that relate to Indo-European chess. I will **not** be Junit testing any GUI interface as well, but instead describe the visual tests and interaction pane code that I did to achieve the desired result. I will also not be testing SwingChessBoard, JavaFXChessBoardDisplay, RookPiece, and SwingChessBoardDisplay, as these were either covered in Project 3 or are interfaces that will be tested in another way (for JavaFXChessBoardDisplay, I will test the implementations of it). I will also **not** be testing any duplicate methods that also appear in Xiangqi.

XiangqiKingPiece, CanPalaceMove, CanFaceKingMove, and CanSingleStraightMove

## Introduction

This will cover all methods in the XiangqiKingPiece class as well as its implemented interfaces. I will first check instantiation of a XiangqiKingPiece, and then check its move methods.

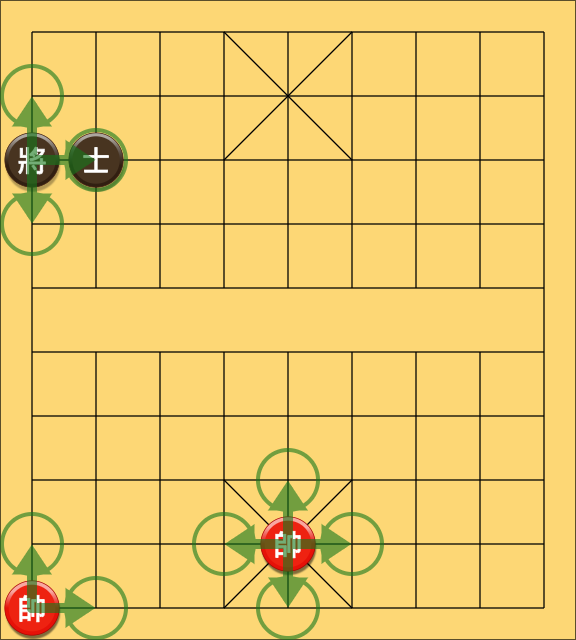
XiangqiKingPiece::XiangqiKingPiece()

For the first test, I will check the instantiation of a XiangqiKingPiece object by calling its constructor and then assigning it to a variable. Then I will call the getLabel method to make sure that the correct piece was created.

The test is grouped under testXiangqiKingPiece() in XiangqiChessTester.java::46-52. The test passed successfully.

CanSingleStraightMove::isValidSingleStraightMove()

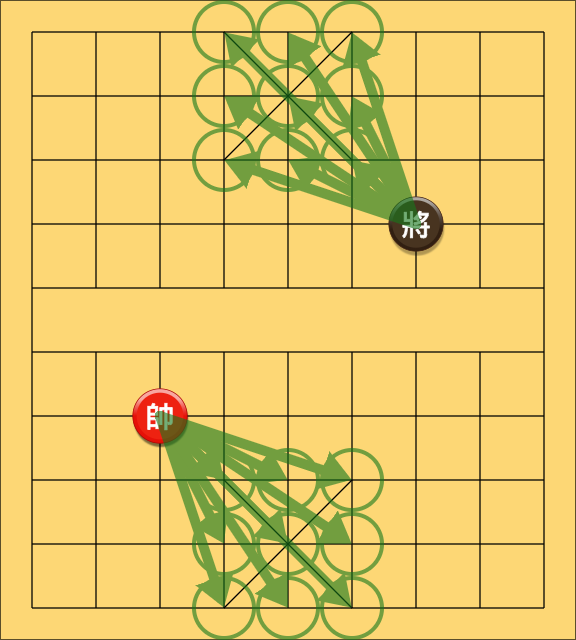
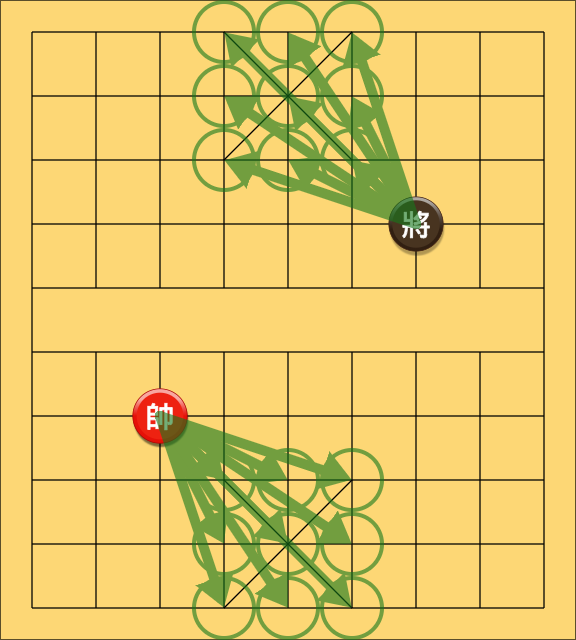
For the second test, I will check if the isValidSingleStraightMove method is working. To do this, I will add a XiangQiKingPiece to the board and test the corners, then in the middle, and then on the side. Note that this doesn’t test for if there is a piece on the move or not. These 4 pieces will be placed on the board in the following layout, and the moves are highlighted in green:



The test is grouped under testSingleStraightMove() in XiangqiChessTester.java::63-100. The test passed successfully.

CanPalaceMove::isValidPalaceMove()

For the 3rd test, I will check if the isValidPalaceMove method is working. To do this, I will add 4 XiangQiKingPieces, one from each side, to the board, and then test all the squares. This should return a 3x3 area called the palace. This will be the same test for the GuardPiece, as that also implements this interface. Note that this does not check for pieces at the location.The boards will be set up like the following:

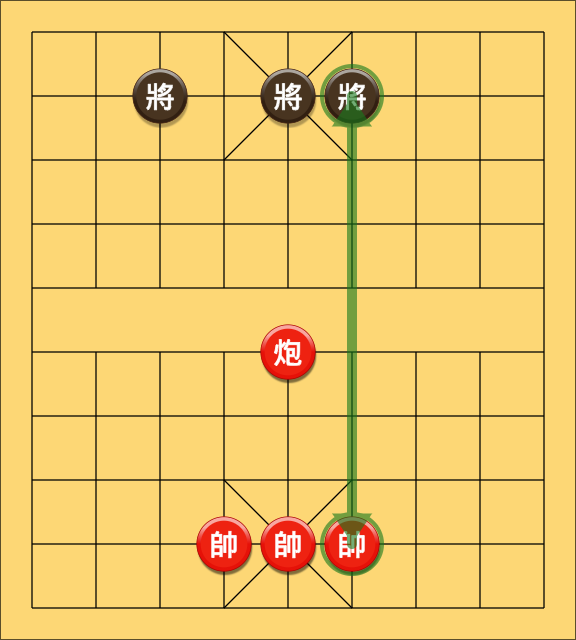
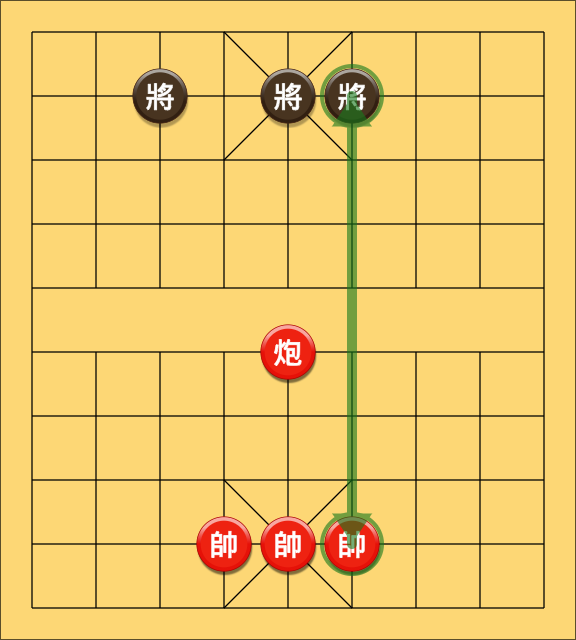


The left board represents a north-south orientation, while the right board represents the west-east orientation.

The test is grouped under testPalaceMove() in XiangqiChessTester.java::108-156. The test passed successfully.

CanFaceKingMove::isValidFaceKingMove()

For the 3rd test, I will check if the isValidFaceKingMove method is working. To do this, I will add 12 XiangQiKingPieces, 3 from each side, to the board, and then test all the squares. In the first scenario, the method will return false because they are not on the same row or column. In the second scenario, the method will also return false because there is a piece in between them. In the third scenario, the method will return true because they are on the same row/column and there are no pieces in between them. I will test both north-south and west-east boards. The pieces will be arranged like this (green circles represent moves that will return true):



The left board represents a north-south orientation, while the right board represents the west-east orientation.

The test is grouped under testFaceKingMove() in XiangqiChessTester.java::164-238. The test passed successfully.

XiangqiKingPiece::getOpposingKings()

For the 4th and final test, I will check if the getOpposingKings method is working. To do this, I will take the north-south scenario from above and run the method, which should return an array of 3 opposing kings. I will then make sure that the 3 opposing kings are in the array. I will only test this for the north-south chessboard because this does not depend on which chessboard it is.

The test is grouped under testOpposingKings() in XiangqiChessTester.java::246-278. The test passed successfully.

GuardPiece and CanSingleDiagonalMove

## Introduction

This will cover all methods in the GuardPiece class as well as its implemented interfaces. I will first check instantiation of a GuardPiece, and then check its move methods.

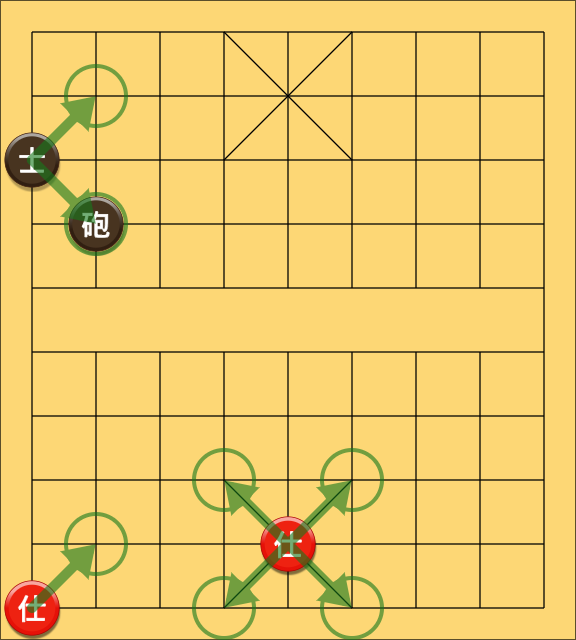
GuardPiece::GuardPiece()

For the first test, I will check the instantiation of a GuardPiece object by calling its constructor and then assigning it to a variable. Then I will call the getLabel method to make sure that the correct piece was created.

The test is grouped under testGuardPiece() in XiangqiChessTester.java::305-314. The test passed successfully.

CanSingleDiagonalMove::isValidSingleDiagonalMove()

For the second and final test, I will check if the isValidSingleDiagonalMove method is working. To do this, I will add a GuardPiece to the board and test the corners, then in the middle, and then on the side. Note that this doesn’t test for if there is a piece on the move or not. These 4 pieces will be placed on the board in the following layout, and the moves are highlighted in green:



The test is grouped under testSingleDiagonalMove() in XiangqiChessTester.java::322-359. The test passed successfully.

ElephantPiece and CanElephantMove

## Introduction

This will cover all methods in the ElephantPiece class as well as its implemented interfaces. I will first check instantiation of a ElephantPiece, and then check its move methods.

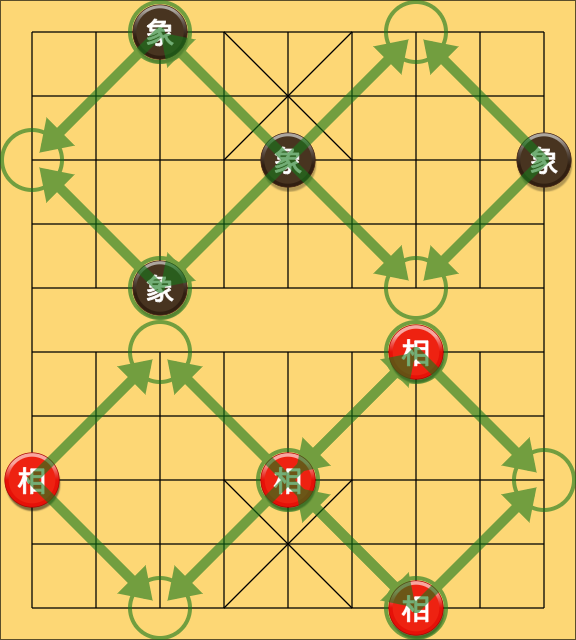
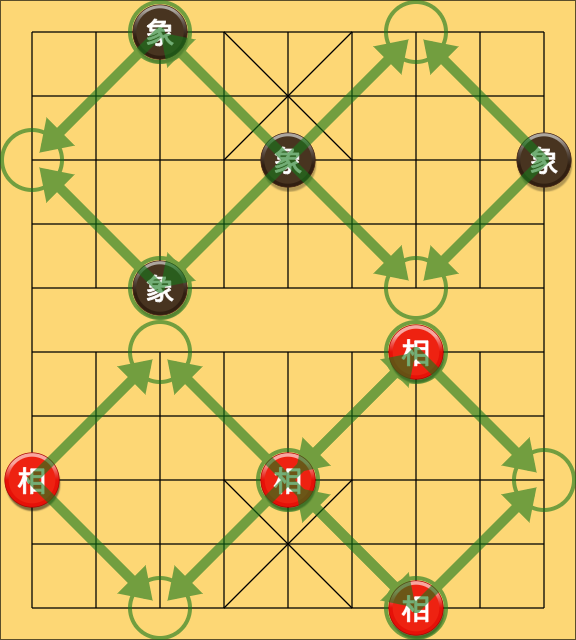
ElephantPiece::ElephantPiece()

For the first test, I will check the instantiation of a ElephantPiece object by calling its constructor and then assigning it to a variable. Then I will call the getLabel method to make sure that the correct piece was created.

The test is grouped under testElephantPiece() in XiangqiChessTester.java::367-376. The test passed successfully.

CanElephantMove::isValidElephantMove()

For the second and final test, I will check if the isValidElephantMove method is working. To do this, I will add an ElephantPiece to the board and test the corners, then in the middle, and then on the side. I will do this for each of the sides. Note that this doesn’t test for if there is a piece on the move or not. I will then repeat the same step except for west-east board orientation, since this method does have direction-sensitive logic. These 16 pieces will be placed on the board in the following layout, and the moves are highlighted in green:



The left board represents a north-south orientation, while the right board represents the west-east orientation.

The test is grouped under testElephantMove() in XiangqiChessTester.java::384-599. The test passed successfully.

SoldierPiece and CanSoldierMove

## Introduction

This will cover all methods in the SoldierPiece class as well as its implemented interfaces. I will first check instantiation of a SoldierPiece, and then check its move methods.

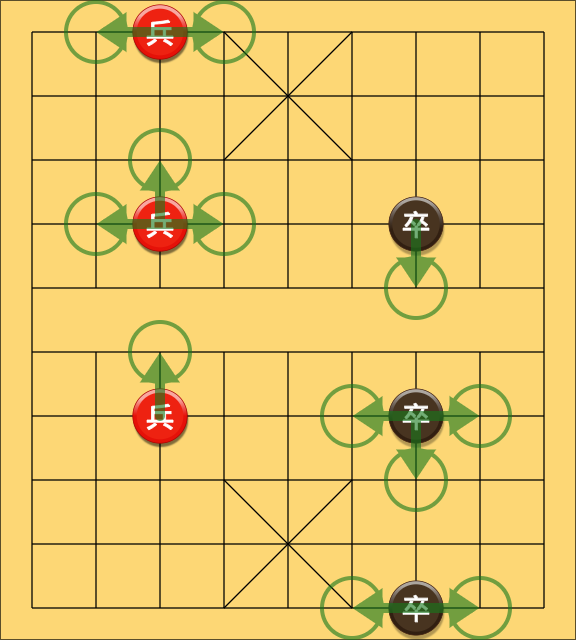
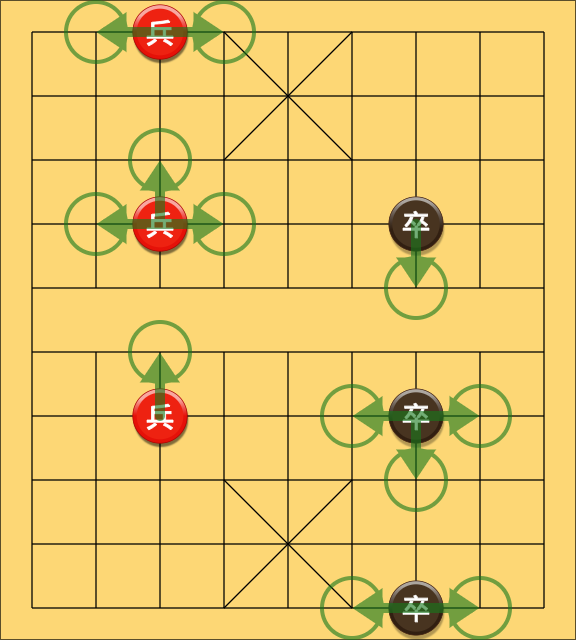
SoldierPiece::SoldierPiece()

For the first test, I will check the instantiation of a SoldierPiece object by calling its constructor and then assigning it to a variable. Then I will call the getLabel method to make sure that the correct piece was created.

The test is grouped under testSoldierPiece() in XiangqiChessTester.java::607-616. The test passed successfully.

CanSoldierMove::isValidSoldierMove()

For the second and final test, I will check if the isValidSoldierMove method is working. To do this, I will add a SoldierPiece to the board before the river, after the river, and then at the end of the board. I will do this for each of the sides. Before the river, the soldier should only be able to go forward; after the river, the soldier should be able to go forward and sideways, but never backwards. At the end of the board, the soldier should only be able to go sideways. Note that this doesn’t test for if there is a piece on the move or not. I will then repeat the same step except for west-east board orientation, since this method does have direction-sensitive logic. These 12 pieces will be placed on the board in the following layout, and the moves are highlighted in green:



The left board represents a north-south orientation, while the right board represents the west-east orientation.

The test is grouped under testSoldierMove() in XiangqiChessTester.java::624-767. The test passed successfully.

CannonPiece and CanCannonMove

## Introduction

This will cover all methods in the CannonPiece class as well as its implemented interfaces. I will first check instantiation of a CannonPiece, and then check its move methods. I will not be covering the CannonPiece’s non-capture move because these are analogous to a RookPiece’s straight moves, which have been covered in Project 3.

CannonPiece::CannonPiece()

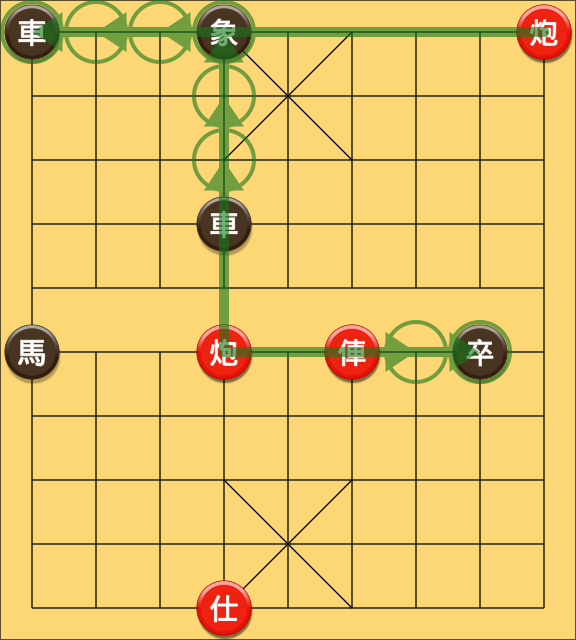
For the first test, I will check the instantiation of a CannonPiece object by calling its constructor and then assigning it to a variable. Then I will call the getLabel method to make sure that the correct piece was created.

The test is grouped under testCannonPiece() in XiangqiChessTester.java::775-784. The test passed successfully.

CanCannonMove::isValidCannonMove()

For the second and final test, I will check if the isValidCannonMove method is working. To do this, I will add a CannonPiece to the board, and then add the piece as shown in the below diagram. The cannon should only be able to capture by going over a piece, regardless of the side of that piece. Note that this method doesn’t actually check if the piece exists at the move.

I will only test this on the north-south chessboard because this does not depend on the chessboard. The pieces will be placed in the following layout, with the legal moves highlighted in green:



The test is grouped under testCannonMove() in XiangqiChessTester.java::792-826. The test passed successfully.

HorsePiece and CanHorseMove

## Introduction

This will cover all methods in the HorsePiece class as well as its implemented interfaces. I will first check instantiation of a HorsePiece, and then check its move methods.

HorsePiece::HorsePiece()

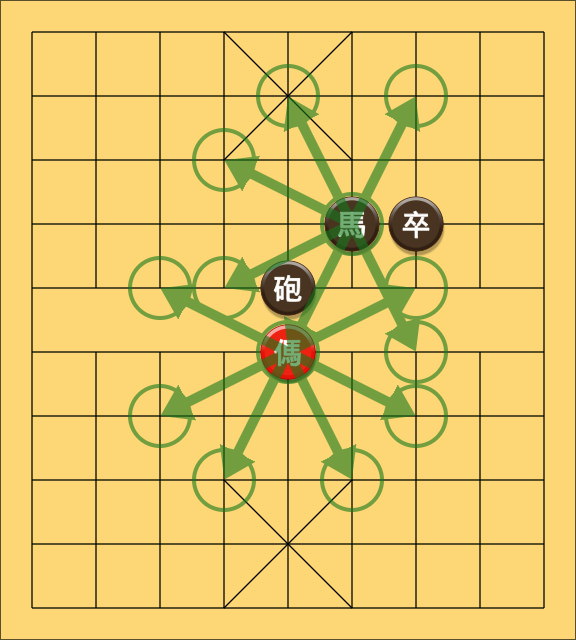
For the first test, I will check the instantiation of a HorsePiece object by calling its constructor and then assigning it to a variable. Then I will call the getLabel method to make sure that the correct piece was created.

The test is grouped under testHorsePiece() in XiangqiChessTester.java::834-842. The test passed successfully.

CanHorseMove::isValidHorseMove()

For the second and final test, I will check if the isValidHorseMove method is working. To do this, I will add 2 HorsePieces to the board, and then add the piece in between as shown in the below diagram. Only the top horse piece should be able to move to the bottom one, while the bottom one should not be able to move to the top because it is being blocked. Note that this method doesn’t actually check if the piece exists at the move.

I will only test this on the north-south chessboard because this does not depend on the chessboard. The pieces will be placed in the following layout, with the legal moves highlighted in green:



The test is grouped under testHorseMove() in XiangqiChessTester.java::852-881. The test passed successfully.

Xiangqi

## Introduction

This will cover only cover **new** methods in the Xiangqi class. The methods I will be covering are: makeMove, getNumRows, getNumColumns, and startGame. All other methods are either inherited from ChessGame or are exact replicas of the methods in EuropeanChess. I will also not be testing the constructor, because that is also the exact same as EuropeanChess.

Xiangqi::makeMove()

For the first test, I

The test is grouped under testMethod() in XiangqiChessTester.java::000-000. The test passed successfully.

XiangqiKingPiece, CanPalaceMove, CanFaceKingMove, and CanSingleStraightMove

## Introduction

This will cover all methods in the XiangqiKingPiece class as well

StraightMove::checkEmptyStraightMove()

For the first test, I

The test is grouped under testMethod() in XiangqiChessTester.java::000-000. The test passed successfully.