Computer Science Project 5 Testing Report

Mehlam Saifudeen

1. **JavaFXChessBoard Class:**
    a. void start(Stage primaryStage) - this method will help set the chessboard by creating buttons for the individual chess pieces. The scene makes a scene of a gridPane and the show method of the stage is used to display the chessboard. There are loops that add the piece to the button on the chessboard according to their row and column. There is also a choice to either run the Indo European chess which will display the 8 by 8 Indo European chess board and add the appropriate chess pieces of this game. Or the second option is to play the Chinese Xiangqi chess game which will display a 10 by 9 chess board and add the appropriate chess pieces on the board.

    b. ChessGame getGameRules() - is an overridden getter method of the interface ChessBoard and this method returns the chessGame being played.

    c. void setGameRules(ChessGame newRules) - a setter method to set which game is being played.

    d. final int numRows() - returns the number of rows on the chessboard. To test this method check if the number of rows passed are correct or not.

    e. final int Columns() - returns the number of columns on the chess board. To test this method check if the number of columns passed are correct or not.

    f. void addPiece(ChessPiece piece, int row, int column) - is an overridden method of the ChessBoard Interface and it adds the specific piece to its specific location on the chessboard. The parameters include the piece to be added and the row and column it needs to be added on. To test this method pass a chess piece, and a row and column to check if the piece has been added at that specific row and column and if it is correct or not.

    g. void removePiece(ChessPiece piece, int row, int column) - is an overridden method of the ChessBoard Interface and it removes the

specific piece to its specific location on the chessboard. The parameters include the piece to be removed and the row and column it needs to be added on. To test this method pass a chess piece, and a row and column to check if the piece has been removed at that specific row and column and if it is correct or not. This method also returns the piece that has been removed from the board.

h. void hasPiece(int row, int column) - a getter method that checks if there is a piece at a particular location on the board or not. To test the method, we pass a row and column on the board and it will return true if there is a piece at that location or false if there is no piece at that location.

i. void ChessPiece getPiece(int row, int column) - a getter method that returns the particular piece at a row or column. To test this method, we pass a row and column and it will return the piece that is present or null if there is no piece at that particular row or column

j. boolean squareThreathened(int row, int column, ChessPiece piece) - this method checks if a particular piece is threatened by a piece of another piece. It will return true if the piece is threatened by the piece of the opposite player or else will return false if that peace is not threatened. The parameters take a ChessPiece, row and column to check if that piece at specific row and column is threatened or not. To test this method we pass a piece, a row and a column and check if that is threatened or not.

k. void main(String [] args) - the main method that launches the application and the chessboard when run.

2. SwingChessBoardDisplay Class:

a. public SwingChessBoard(SwingChessBoardDisplay boardDisplay, ChessGame gameRules): This is a constructor that builds a board of the desired size, and display and the rules of the game. It sets up the squares of the board and adds them there.

b. ChessGame getGameRules() - is an overridden getter method of the interface ChessBoard and this method returns the chessGame being played.

a. void setGameRules(ChessGame newRules) - a setter method to set which game is being played.

b. final int numRows() - returns the number of rows on the chessboard. To test this method check if the number of rows passed are correct or not.

c. final int Columns() - returns the number of columns on the chess board. To test this method check if the number of columns passed are correct or not.

d. void addPiece(ChessPiece piece, int row, int col) - is an overridden method of the ChessBoard Interface and it adds the specific piece to its specific location on the chessboard. The parameters include the piece to be added and the row and column it needs to be added on. To test this method pass a chess piece, and a row and column to check if the piece has been added at that specific row and column and if it is correct or not.

e. void removePiece(ChessPiece piece, int row, int col) - is an overridden method of the ChessBoard Interface and it removes the specific piece to its specific location on the chessboard. The parameters include the piece to be removed and the row and column it needs to be added on. To test this method pass a chess piece, and a row and column to check if the piece has been removed at that specific row and column and if it is correct or not. This method also returns the piece that has been removed from the board.

f. void hasPiece(int row, int col) - a getter method that checks if there is a piece at a particular location on the board or not. To test the method, we pass a row and column on the board and it will return true if there is a piece at that location or false if there is no piece at that location.

g. void ChessPiece getPiece(int row, int col) - a getter method that returns the particular piece at a row or column. To test this method, we pass a row and column and it will return the piece that is present or null if there is no piece at that particular row or column.

h. boolean squareThreathened(int row, int col, ChessPiece piece) - this method checks if a particular piece is threatened by a piece of another piece. It will return true if the piece is threatened by the piece of the opposite player or else will return false if that peace is not threatened. The parameters take a ChessPiece, row and column to check if that piece at specific row and column is threatened or not. To test this method we pass a piece, a row and a column and check if that is threatened or not.

Testing in the interactions Pane for the SwingChessBoard Class:

```
// Making an instance of the chess board to launch the program
> SwingChessBoard board = new SwingChessBoard(new
SwingEuropeanChessDisplay(), new EuropeanChess())

// adding a new rook piece to 2 positions on the board
> board.addPiece(new RookPiece("R", 2, 3, ChessGame.Side.NORTH, board), 3, 4)
> board.removePiece(new RookPiece("R", 2, 3, ChessGame.Side.NORTH, board), 3,
4)

// removing the rook piece added in the previous statement
> board.removePiece(3, 4)
RookPiece@374cb5ef

Removing the second rook piece added previously
> board.removePiece(2, 3)
RookPiece@374cb5ef

// adding a queen piece to a specific position on the chess board
> board.addPiece(new QueenPiece("Q", 0,5,ChessGame.Side.SOUTH, board), 8, 4)

// using the hasPiece method to check if a position on the board has a piece or not
> board.hasPiece(0,5)
true
> board.hasPiece(0,7)
false

// using the getPiece method to get which piece is at a specific position
> board.getPiece(8,4)
QueenPiece@7d396887

// testing the method to get the number of columns on the board
> board.numColumns()
9

// testing the method to get the number of columns on the board
> board.numRows()
10
```

// method to add another rook piece to the board and checking it with the hasPiece and getPiece methods
> board.addPiece(new RookPiece("R", 0,0, ChessGame.Side.NORTH, board), 0, 7)
> board.hasPiece(0,0)
true
> board.getPiece(0,0)
RookPiece@4d465838

Testing the same methods of the Chess board class for the Xiangqi chess pieces
> board.addPiece(new GuardPiece("G", 0,3,ChessGame.Side.NORTH, board), 4, 5)
> board.addPiece(new ElephantPiece("E", 0,6,ChessGame.Side.NORTH, board), 9, 2)

> board.getPiece(9,2)
ElephantPiece@7225ba8c

> board.hasPiece(0, 6)
true

> board.addPiece(new HorsePiece("H", 0,1,ChessGame.Side.NORTH, board), 0, 7)

> board.addPiece(new HorsePiece("H", 9,1,ChessGame.Side.NORTH, board), 9, 7)

> board.removePiece(9,7)
HorsePiece@49242e78

> board.removePiece(0,7)
HorsePiece@d151a5b

> board.removePiece(9,1)
HorsePiece@49242e78

> board.removePiece(0,1)
HorsePiece@d151a5b

3. interface ChessBoard: This interface contains the following abstract methods that are overridden in the above JavaFXChessBoard and SwingChessBoard class

a. public ChessGame getGameRules()
b. public void addPiece(ChessPiece piece, int row, int column)
c. public ChessPiece removePiece(int row, int column)
d. public boolean hasPiece(int row, int column)
e. public ChessPiece getPiece(int row, int column)

4. interface SwingChessBoardDisplay: This is an interface contains abstract methods that are overridden in the SwingEuropeanChessDisplay and SwingXianqiDisplayClass

a. public void displayEmptySquare(JButton button, int row, int column)
b. public void displayFilledSquare(JButton button, int row, int column, ChessPiece piece)
c. public void highlightSquare(boolean highlight, JButton button, int row, int column, ChessPiece piece)

5. SwingEuropeanChessDisplay:

a. public void displayEmptySquare(JButton button, int row, int column): This method will display the colour of an empty square. It will alternate and make a square either red or black and in odd or even fashion.

b. public void displayFilledSquare(JButton button, int row, int column, ChessPiece piece): This method will display the color of a filled square. It will alternate between yellow and green depending on if it is an odd or even row and column.

c. public void highlightSquare(boolean highlight, JButton button, int row, int column, ChessPiece piece): This method will display the colour of a selected piece square as blue. There are if else cases to check if there is a piece or not using the 2 methods above and will not highlight the square if it has no piece. If the square has a piece it will highlight that square as blue.

6. interface ChessGame:

This is an interface that contains an enum for which side of the board or player a specific piece belongs to.

a. public enum Side {NORTH, SOUTH, EAST, WEST}

It also contains the following abstract methods that are overridden in the **EuropeanChess and Xiangqi classes.**

a. public boolean legalPieceToPlay(ChessPiece piece, int row, int column)
b. public boolean makeMove(ChessPiece piece, int toRow, int toColumn)
c. public default boolean canChangeSelection(ChessPiece piece, int row, int column)
d. public int getNumRows()
e. public int getNumColumns()
f. public void startGame(ChessBoard board)

7. interface JavaFXChessBoardDisplay: This interface contains abstract methods that are overridden by the JavaFXEuropeanChessDisplay and JavaFXXiangqiDisplay classes. The following are the methods of this class

a. public void displayEmptySquare(Button button, int row, int column)
b. public void displayFilledSquare(Button button, int row, int column, ChessPiece chessPiece)
c. public void highlightSquare(boolean highlight, Button button, int row, int column, ChessPiece chessPiece).

8. JavaFXEuropeanChessDisplay:

a. public void displayEmptySquare(Button button, int row, int column): This method will display the colour of an empty square. It will alternate and make a square either brown or bisque and in odd or even fashion.

b. public void displayFilledSquare(Button button, int row, int column, ChessPiece chessPiece): This method will display the color of a filled square. It will alternate between yellow and green depending on if it is an odd or even row and column.

c. public void highlightSquare(boolean highlight, Button button, int row, int column, ChessPiece chessPiece): This method will display the colour of a selected piece square as blue. There are if else cases to check if there is a piece or not using the 2 methods above and will not highlight the square if it has no piece. If the square has a piece it will highlight that square as blue.

9. JavaFXXianqiDisplay:

a. public void displayEmptySquare(Button button, int row, int column): This method will display the colour of an empty square. It will alternate and make a square Light Gray in colour.

b. public void displayFilledSquare(Button button, int row, int column, ChessPiece chessPiece): This method will display the color of a filled square. It will alternate between yellow and green depending on if it is an odd or even row and column.

c. public void highlightSquare(boolean highlight, Button button, int row, int column, ChessPiece chessPiece): This method will display the colour of a selected piece square as blue. There are if else cases to check if there is a piece or not using the 2 methods above and will not highlight the square if it has no piece. If the square has a piece it will highlight that square as blue.

10. EuropeanChess:This class is an implementation of the ChessGame interface and overrides the abstract methods of that interface

a. public boolean makeMove(ChessPiece piece, int toRow, int toColumn): This method checks if a particular piece can make a move or not. The parameters take the chess piece to be moved, and the row and column it wants to move to. If a piece can move to that specific row and column the method will return true, else it will return false. To test this method we can test it by making an instance of a piece and passing that piece with a row and column we want to move to.

b. public boolean legalPieceToPlay(ChessPiece piece, int row, int column): This method checks if a particular piece belongs to the player or not. If that piece belongs to that particular player it will return true, otherwise it will return false. The parameters of this method include the chess piece to be checked and the row and column it is at. To test this method, we make an instance of a piece and pass that piece with a row and column to check if it belongs to the NORTH or SOUTH side.

c. public ChessGame.Side getTurn(): A getter method that returns which player's turn it is. To test this method we make an instance of a piece and check if that piece can be selected or not.

d. public void setTurn(ChessGame.Side turn): A setter method that sets which player's turn it is.

e. public ChessGame getGame(): A getter method that returns which game is being played.

f. public void moveDone(): This method changes which player's turn it is. If it is the North player's turn the turn is changed so that the south player can now make a move or vice versa. To test this method, we check if the move of a piece has been completed or not.

g. public int getNumRows(): Returns the number of rows on the board

h. public int getNumColumns(): Returns the number of columns on the board.

10. Xianqi: This class is an implementation of the ChessGame interface and overrides the abstract methods of that interface

a. public boolean makeMove(ChessPiece piece, int toRow, int toColumn): This method checks if a particular piece can make a move or not. The parameters take the chess piece to be moved, and the row and column it wants to move to. If a piece can move to that specific row and column the method will return true, else it will return false. To test this method we can test it by making an instance of a piece and passing that piece with a row and column we want to move to.

b. public boolean legalPieceToPlay(ChessPiece piece, int row, int column): This method checks if a particular piece belongs to the player or not. If that piece belongs to that particular player it will return true, otherwise it will return false. The parameters of this method include the chess piece to be checked and the row and column it is at. To test this method, we make an instance of a piece and pass that piece with a row and column to check if it belongs to the NORTH or SOUTH side.

c. public ChessGame.Side getTurn(): A getter method that returns which player's turn it is. To test this method we make an instance of a piece and check if that piece can be selected or not.

d. public void setTurn(ChessGame.Side turn): A setter method that sets which player's turn it is.

e. public ChessGame getGame(): A getter method that returns which game is being played.

f. public void moveDone(): This method changes which player's turn it is. If it is the North player's turn the turn is changed so that the south player can now make a move or vice versa. To test this method, we check if the move of a piece has been completed or not.

g. public int getNumRows(): Returns the number of rows on the board

h. public int getNumColumns(): Returns the number of columns on the board.

11. Testing of all common methods for the pieces of the Xiangqi and Indo-European Chess Game: The pieces of this game include XiangqiKingPiece, GuardPiece, ElephantPiece, HorsePiece, RookPiece, CannonPiece, and SoldierPiece, PawnPiece, QueenPiece, KingPiece, BishopPiece and KnightPiece The following are the methods that are overridden from the ChessPiece class into these individual classes.

a. public XiangqiKingPiece(String label ,int toRow, int toColumn, ChessGame.Side side,SwingChessBoard chessBoard): A constructor (One for all the pieces) which inherits from the ChessPiece class and passes the label of the piece, the row at which it has to move to, the column it has to move to, the side to which it belongs and the chessBoard itself.

b. public String getLabel(): A getter method that returns the label of the piece. To test this method create an instance of the piece and test if the label of that piece is true or not.

c. public boolean isNonLegalCaptureMove(int row, int column): This method is an override of the abstract method from the ChessPiece class. It checks if the piece can make a move that only involves moving from one square to another on the chess board without capturing another piece. The parameters take the row and column the piece has to move to and will check accordingly. The method can be tested by creating a new piece and checking if the piece can move to a particular row and column on the board using this method. If the move if possible it returns true, otherwise it will return false.

Testing for the methods of the ChessPiece and Individual piece classes

// creating an instance of the chess board to run the program
> SwingChessBoard board = new SwingChessBoard(new SwingEuropeanChessDisplay(), new EuropeanChess())

// creating a new Rook piece on the chess board
> RookPiece r = new RookPiece("R", 0, 0, ChessGame.Side.NORTH, board)

//using the getter method to get the label of the piece
> r.getLabel()
"R"

// using the getter method to get the icon of the piece
> r.getIcon()
null

//using the get side method to get the side of the board to which the piece belongs
> r.getSide()
NORTH

// testing the isLegalNonCaptureMove the rook piece to see if it is allowed to move to a specific place
> r.isNonLegalCaptureMove(4,5)
false

// testing the isLegalCaptureMove to check if the rook can capture a piece of the other player
> r.isLegalCaptureMove(5,6)
false

// Testing the same methods as above for the pawin piece as well
> PawnPiece p = new PawnPiece("P", 1, 0, ChessGame.Side.NORTH, board)

> p.getLabel()
"P"

> p.getIcon()
Null

> p.getSide()
NORTH

> p.isNonLegalCaptureMove(2,0)
true

> p.isNonLegalCaptureMove(5,0)
true

> p.isLegalCaptureMove(2,3)
False

//Testing the same methods for the Elephant Piece of the Xiangqi Chess game
> ElephantPiece e = new ElephantPiece("E", 0, 2, ChessGame.Side.NORTH, board)
> ElephantPiece e1 = new ElephantPiece("E", 0, 6, ChessGame.Side.NORTH, board)
> ElephantPiece e2 = new ElephantPiece("E", 9,2, ChessGame.Side.SOUTH, board)
> ElephantPiece e3 = new ElephantPiece("E", 9,6, ChessGame.Side.SOUTH, board)

// Testing the regular getter methods of the Elephant Piece class
> e.getLabel()
"E"
> e.getIcon()
null
> e1.getLabel()
"E"

> e1.getSide()
NORTH

> e3.getSide()
SOUTH

// Testing the moves that the Elephant piece can make
> e.isNonLegalCaptureMove(4,2)
true

> e.isNonLegalCaptureMove(4,3)
false

> e3.isNonLegalCaptureMove(7,4)
False

//Testing the same methods for the Horse Piece of the Xiangqi Chess game
> HorsePiece h= new HorsePiece("H", 0, 1, ChessGame.Side.NORTH, board)
> HorsePiece h1 = new HorsePiece("H", 0, 7, ChessGame.Side.NORTH, board)
> HorsePiece h2 = new HorsePiece("H", 9,7, ChessGame.Side.SOUTH, board)
> HorsePiece h3 = new HorsePiece("H", 9,1, ChessGame.Side.SOUTH, board)

// Testing the regular getter methods of the Elephant Piece class
> e.getLabel()
"H"
> e.getIcon()
null
> e1.getLabel()
"H"

> h1.getSide()
NORTH

> h3.getSide()
SOUTH

// Testing the moves that the Elephant piece can make
> h.isNonLegalCaptureMove(3,2)
true

> h.isNonLegalCaptureMove(4,3)
false

> h3.isNonLegalCaptureMove(7,4)
True

> h.isLegalCaptureMove(4,3)
false

12. Main: This class creates an instance of the SwingChessBoard and the SwingChessBoardDisplay, and runs the Indo European Chess game with all the pieces on the board.

13. XiangqiMain: This class creates an instance of th =e JavaFXChessBoard, and JavaFXXiangqiDisplay and runs the Xiangqi Chess game with all the pieces on the board.

14. interface DiagnolMovement: This is an interface that checks if a piece can move diagonally on the chess board. If a diagonal move can be made by a piece it returns true otherwise it will return false. This interface is implemented in those chess pieces that are allowed to move diagonally for example the Queen, Elephant, Guard Bishop, King, etc.

15. interface VerticalMovement: This is an interface that checks if a piece can move vertically on the chessboard. If a vertical move can be made by a piece it returns true otherwise it will return false. This interface is implemented in those chess pieces that are allowed to move vertically for example the Queen, Pawn, Rook, Soldier, Cannon, King, etc.

15. interface HorizontalMovement: This is an interface that checks if a piece can move horizontally on the chessboard. If a horizontal move can be made by a piece it returns true otherwise it will return false. This interface is implemented in those chess pieces that are allowed to move horizontally for example the Queen, Rook, King, etc.