

# Manual Test Plan

Project Name: sp2020-cs242-assignment1

Version: 1.2

Plan Version:1.1

Date: 02/23/2020

Prepared by: Nikhil Paidipally

# Introduction:

## Project Overview:

The goal of this project is to produce a function, two-player, chess game using object-oriented design, MVC design patterns, and proper code style/structure. The first stage of the project should consist of a chess library that can act as a model within fully implemented game. The library should have features that represent the chess board, pieces, the pieces' behavior/movement, and game-ending conditions. When version 1.0 was complemented, most of these features were implemented as well as their corresponding tests. The main missing feature of the library was the ability to detect checkmate.

The second stage of the project involves polishing and extending the chess library to create something that more or less resembles an unfinished chess game following a MVC design pattern. Features such as a functional view for the chess board and extra pieces are implemented, along with checkmate detection which was missing in the first version. For the sake of simplicity and time constraints, a static terminal was implemented in place of a GUI.

The third stage of the project involves polishing and extending the previous stage to create a fully functional two-player, chess game. Features such as a Game class, acting as a controller, full game loop, score count, and undo functionality is implemented. In addition, copy constructors for the Board, Piece, and other actual pieces were implemented to support the undo function. The isInCheckmate, posInCheck, isInCheck, and isInStalemate functions had to be refactored/fixed.

## Project Plan (so far):

- 1.0: Implement Basic functionality:
  - Chess Board
  - Chess Pieces
  - Piece Movement
  - Putting King in Check
  - Stalemate
- 1.1: Polish and Extend functionality:
  - Implement Checkmate
  - Auto-Degenerate Documentation with Doxygen
  - Implemented Two Tairy Chess Pieces (Joker and HolyHorse)
  - Implement a Static Interface/View to visualize the Chess Board
  - Create a Test Plan (ie this document)
- 1.2: TBD:
  - Complete game loop with 2 player interactivity
  - Complete terminal UI with MVC design
  - Implemented full undo functionality

## Prerequisites:

- JDK 11
- Junit 5.4, 4
- IDE of some sort (Intellij preferred)

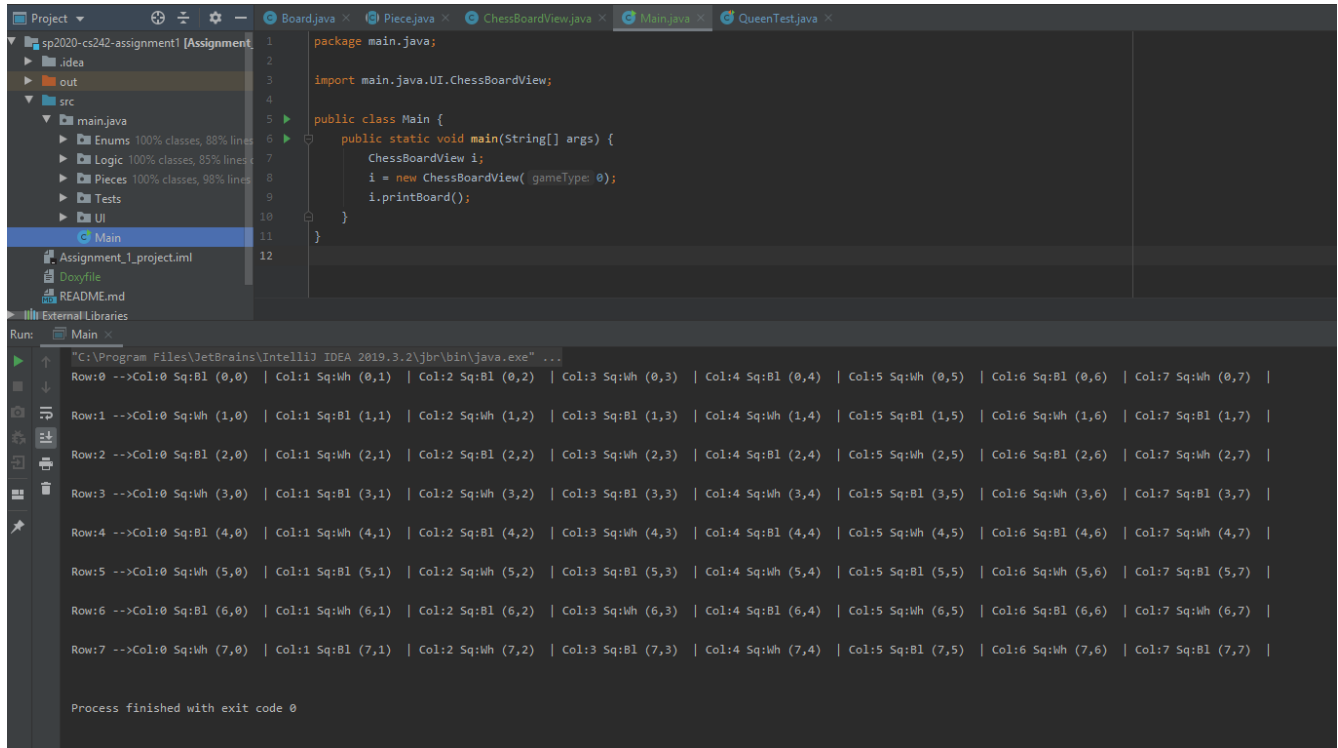
# Environment Setup and Configurations:

## Empty Board for Specific Configurations:

Import the UI package.

Create a new ChessBoardView object with parameter as 0

This result will be an empty board as shown below



The screenshot shows an IDE with a project named 'sp2020-cs242-assignment1'. The 'src' directory contains 'main.java' with packages 'Enums', 'Logic', 'Pieces', 'Tests', and 'UI'. The 'Main' class in 'main.java' is selected. The code in 'Main.java' is as follows:

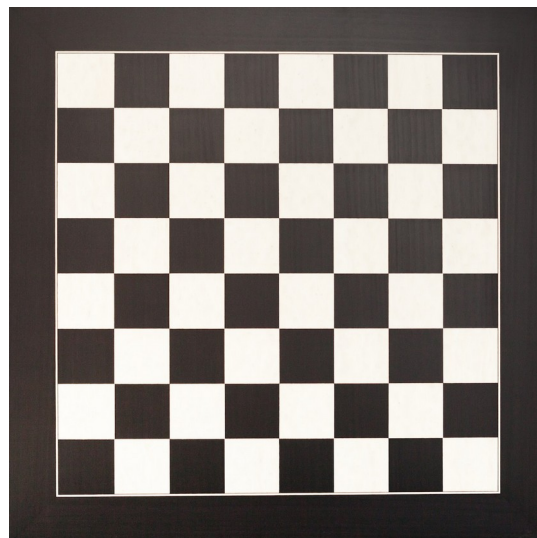
```
1 package main.java;
2
3 import main.java.UI.ChessBoardView;
4
5 public class Main {
6     public static void main(String[] args) {
7         ChessBoardView i;
8         i = new ChessBoardView( gameType: 0);
9         i.printBoard();
10    }
11 }
12
```

The terminal output shows the board representation:

```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.2\jbr\bin\java.exe" ...
Row:0 -->Col:0 Sq:B1 (0,0) | Col:1 Sq:W1 (0,1) | Col:2 Sq:B1 (0,2) | Col:3 Sq:W1 (0,3) | Col:4 Sq:B1 (0,4) | Col:5 Sq:W1 (0,5) | Col:6 Sq:B1 (0,6) | Col:7 Sq:W1 (0,7) |
Row:1 -->Col:0 Sq:W1 (1,0) | Col:1 Sq:B1 (1,1) | Col:2 Sq:W1 (1,2) | Col:3 Sq:B1 (1,3) | Col:4 Sq:W1 (1,4) | Col:5 Sq:B1 (1,5) | Col:6 Sq:W1 (1,6) | Col:7 Sq:B1 (1,7) |
Row:2 -->Col:0 Sq:B1 (2,0) | Col:1 Sq:W1 (2,1) | Col:2 Sq:B1 (2,2) | Col:3 Sq:W1 (2,3) | Col:4 Sq:B1 (2,4) | Col:5 Sq:W1 (2,5) | Col:6 Sq:B1 (2,6) | Col:7 Sq:W1 (2,7) |
Row:3 -->Col:0 Sq:W1 (3,0) | Col:1 Sq:B1 (3,1) | Col:2 Sq:W1 (3,2) | Col:3 Sq:B1 (3,3) | Col:4 Sq:W1 (3,4) | Col:5 Sq:B1 (3,5) | Col:6 Sq:W1 (3,6) | Col:7 Sq:B1 (3,7) |
Row:4 -->Col:0 Sq:B1 (4,0) | Col:1 Sq:W1 (4,1) | Col:2 Sq:B1 (4,2) | Col:3 Sq:W1 (4,3) | Col:4 Sq:B1 (4,4) | Col:5 Sq:W1 (4,5) | Col:6 Sq:B1 (4,6) | Col:7 Sq:W1 (4,7) |
Row:5 -->Col:0 Sq:W1 (5,0) | Col:1 Sq:B1 (5,1) | Col:2 Sq:W1 (5,2) | Col:3 Sq:B1 (5,3) | Col:4 Sq:W1 (5,4) | Col:5 Sq:B1 (5,5) | Col:6 Sq:W1 (5,6) | Col:7 Sq:B1 (5,7) |
Row:6 -->Col:0 Sq:B1 (6,0) | Col:1 Sq:W1 (6,1) | Col:2 Sq:B1 (6,2) | Col:3 Sq:W1 (6,3) | Col:4 Sq:B1 (6,4) | Col:5 Sq:W1 (6,5) | Col:6 Sq:B1 (6,6) | Col:7 Sq:W1 (6,7) |
Row:7 -->Col:0 Sq:W1 (7,0) | Col:1 Sq:B1 (7,1) | Col:2 Sq:W1 (7,2) | Col:3 Sq:B1 (7,3) | Col:4 Sq:W1 (7,4) | Col:5 Sq:B1 (7,5) | Col:6 Sq:W1 (7,6) | Col:7 Sq:B1 (7,7) |

Process finished with exit code 0
```

Ensure that an empty appears in the terminal output as shown above and/or it represents the board shown below:



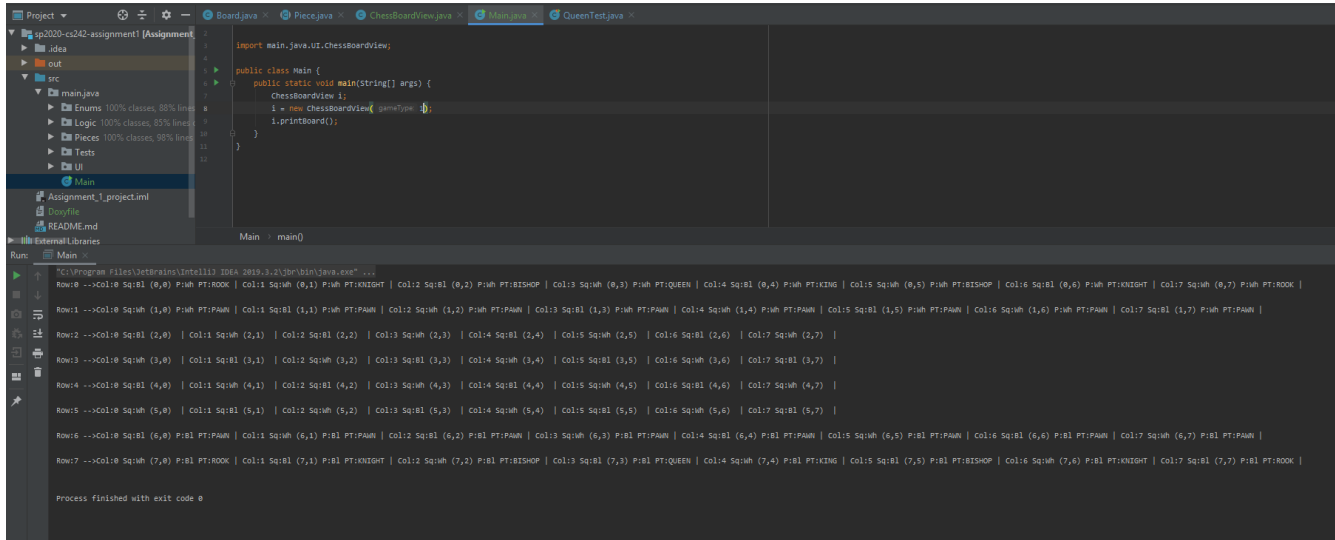
## Populated Board with Normal Chess Pieces and Configuration:

(board formatting has changed in 1.2 to a more readable format but piece coordinates will be the same as in the given figures)

Import the UI package.

Create a new ChessBoardView object with parameter as 1

This result will be a board with all of the normal pieces in the starting configuration as shown below



```
import main.java.UI.ChessBoardView;

public class Main {
    public static void main(String[] args) {
        ChessBoardView i =
            new ChessBoardView(1);
        i.printBoard();
    }
}

Main main()

Row0 -->Col:0 Sq:bl (0,0) P:wh PT:ROOK | Col:1 Sq:wh (0,1) P:wh PT:KNIGHT | Col:2 Sq:bl (0,2) P:wh PT:BISHOP | Col:3 Sq:wh (0,3) P:wh PT:QUEEN | Col:4 Sq:bl (0,4) P:wh PT:KING | Col:5 Sq:wh (0,5) P:wh PT:BISHOP | Col:6 Sq:bl (0,6) P:wh PT:KNIGHT | Col:7 Sq:wh (0,7) P:wh PT:ROOK |
Row1 -->Col:0 Sq:bl (1,0) P:wh PT:PAWN | Col:1 Sq:bl (1,1) P:wh PT:PAWN | Col:2 Sq:wh (1,2) P:wh PT:PAWN | Col:3 Sq:bl (1,3) P:wh PT:PAWN | Col:4 Sq:wh (1,4) P:wh PT:PAWN | Col:5 Sq:bl (1,5) P:wh PT:PAWN | Col:6 Sq:wh (1,6) P:wh PT:PAWN | Col:7 Sq:bl (1,7) P:wh PT:PAWN |
Row2 -->Col:0 Sq:bl (2,0) | Col:1 Sq:wh (2,1) | Col:2 Sq:bl (2,2) | Col:3 Sq:wh (2,3) | Col:4 Sq:bl (2,4) | Col:5 Sq:wh (2,5) | Col:6 Sq:bl (2,6) | Col:7 Sq:wh (2,7) |
Row3 -->Col:0 Sq:wh (3,0) | Col:1 Sq:bl (3,1) | Col:2 Sq:wh (3,2) | Col:3 Sq:bl (3,3) | Col:4 Sq:wh (3,4) | Col:5 Sq:bl (3,5) | Col:6 Sq:wh (3,6) | Col:7 Sq:bl (3,7) |
Row4 -->Col:0 Sq:bl (4,0) | Col:1 Sq:wh (4,1) | Col:2 Sq:bl (4,2) | Col:3 Sq:wh (4,3) | Col:4 Sq:bl (4,4) | Col:5 Sq:wh (4,5) | Col:6 Sq:bl (4,6) | Col:7 Sq:wh (4,7) |
Row5 -->Col:0 Sq:wh (5,0) | Col:1 Sq:bl (5,1) | Col:2 Sq:wh (5,2) | Col:3 Sq:bl (5,3) | Col:4 Sq:wh (5,4) | Col:5 Sq:bl (5,5) | Col:6 Sq:wh (5,6) | Col:7 Sq:bl (5,7) |
Row6 -->Col:0 Sq:bl (6,0) P:bl PT:PAWN | Col:1 Sq:wh (6,1) P:bl PT:PAWN | Col:2 Sq:bl (6,2) P:bl PT:PAWN | Col:3 Sq:wh (6,3) P:bl PT:PAWN | Col:4 Sq:bl (6,4) P:bl PT:PAWN | Col:5 Sq:wh (6,5) P:bl PT:PAWN | Col:6 Sq:bl (6,6) P:bl PT:PAWN | Col:7 Sq:wh (6,7) P:bl PT:PAWN |
Row7 -->Col:0 Sq:wh (7,0) P:bl PT:ROOK | Col:1 Sq:bl (7,1) P:bl PT:KNIGHT | Col:2 Sq:wh (7,2) P:bl PT:BISHOP | Col:3 Sq:bl (7,3) P:bl PT:QUEEN | Col:4 Sq:wh (7,4) P:bl PT:KING | Col:5 Sq:bl (7,5) P:bl PT:BISHOP | Col:6 Sq:wh (7,6) P:bl PT:KNIGHT | Col:7 Sq:bl (7,7) P:bl PT:ROOK |

Process finished with exit code 0
```

Ensure that a normal vanilla chess board is in the output terminal. (Ie should represent the board below)



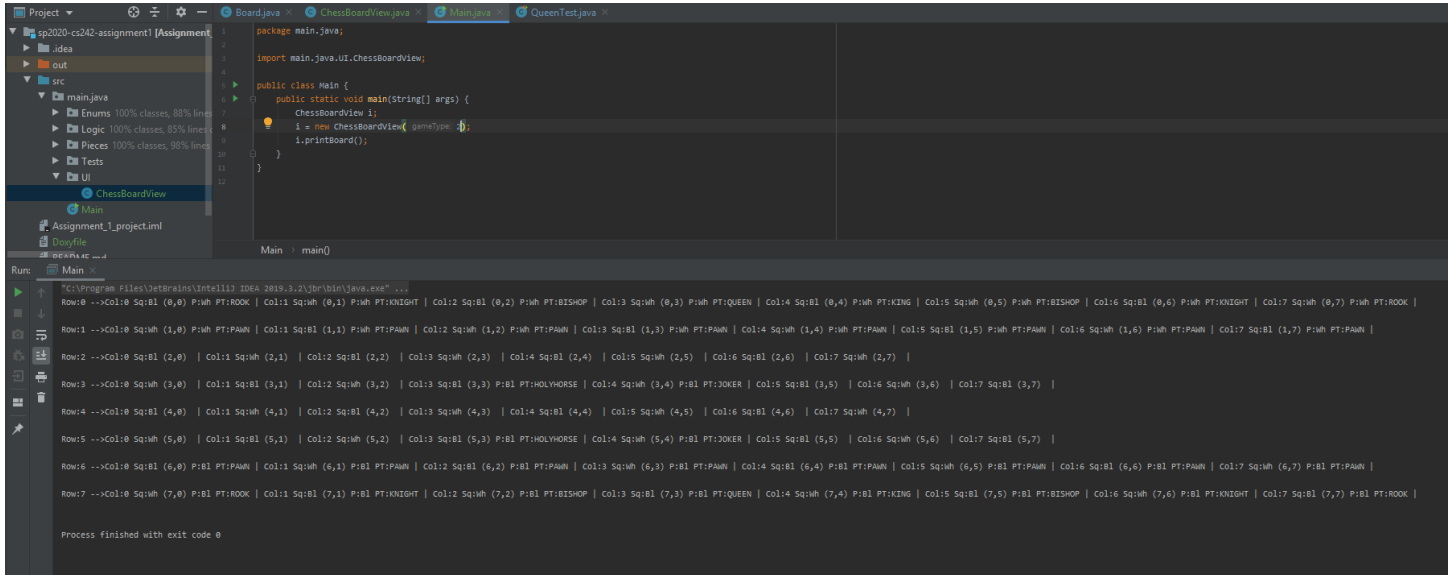
## Populated Board with Modded Chess Pieces and Configuration:

Import the UI package.

Create a new ChessBoardView object with parameter as 2

This result will be a board with all of the normal pieces in the starting configuration as shown below.

Ensure your terminal output matches the following:



```
package main.java;

import main.java.ui.ChessBoardView;

public class Main {

    public static void main(String[] args) {
        ChessBoardView i;
        i = new ChessBoardView(2);
        i.printBoard();
    }
}
```

Run: Main -> main()

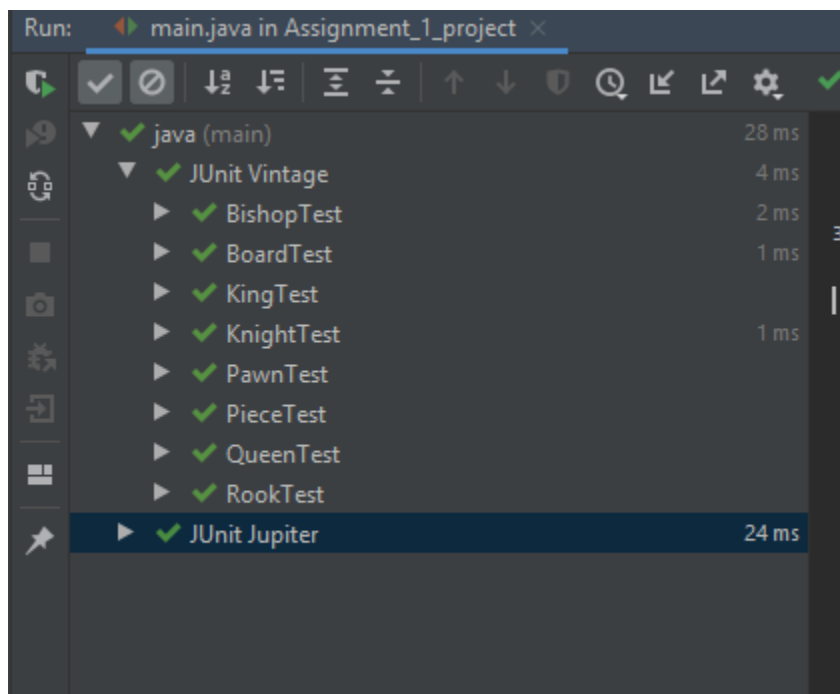
```
Row0 -->Col0 Sq:B1 (0,0) P:Wh PT:ROOK | Col1 Sq:Wh (0,1) P:Wh PT:KNIGHT | Col2 Sq:B1 (0,2) P:Wh PT:BISHOP | Col3 Sq:Wh (0,3) P:Wh PT:QUEEN | Col4 Sq:B1 (0,4) P:Wh PT:KING | Col5 Sq:Wh (0,5) P:Wh PT:BISHOP | Col6 Sq:B1 (0,6) P:Wh PT:KNIGHT | Col7 Sq:Wh (0,7) P:Wh PT:ROOK |
Row1 -->Col0 Sq:Wh (1,0) P:Wh PT:PAWN | Col1 Sq:B1 (1,1) P:Wh PT:PAWN | Col2 Sq:Wh (1,2) P:Wh PT:PAWN | Col3 Sq:B1 (1,3) P:Wh PT:PAWN | Col4 Sq:Wh (1,4) P:Wh PT:PAWN | Col5 Sq:B1 (1,5) P:Wh PT:PAWN | Col6 Sq:Wh (1,6) P:Wh PT:PAWN | Col7 Sq:B1 (1,7) P:Wh PT:PAWN |
Row2 -->Col0 Sq:B1 (2,0) | Col1 Sq:Wh (2,1) | Col2 Sq:B1 (2,2) | Col3 Sq:Wh (2,3) | Col4 Sq:B1 (2,4) | Col5 Sq:Wh (2,5) | Col6 Sq:B1 (2,6) | Col7 Sq:Wh (2,7) |
Row3 -->Col0 Sq:Wh (3,0) | Col1 Sq:B1 (3,1) | Col2 Sq:Wh (3,2) | Col3 Sq:B1 (3,3) P:B1 PT:HOLYHORSE | Col4 Sq:Wh (3,4) P:B1 PT:JOKER | Col5 Sq:B1 (3,5) | Col6 Sq:Wh (3,6) | Col7 Sq:B1 (3,7) |
Row4 -->Col0 Sq:B1 (4,0) | Col1 Sq:Wh (4,1) | Col2 Sq:B1 (4,2) | Col3 Sq:Wh (4,3) | Col4 Sq:B1 (4,4) | Col5 Sq:Wh (4,5) | Col6 Sq:B1 (4,6) | Col7 Sq:Wh (4,7) |
Row5 -->Col0 Sq:Wh (5,0) | Col1 Sq:B1 (5,1) | Col2 Sq:Wh (5,2) | Col3 Sq:B1 (5,3) P:B1 PT:HOLYHORSE | Col4 Sq:Wh (5,4) P:B1 PT:JOKER | Col5 Sq:B1 (5,5) | Col6 Sq:Wh (5,6) | Col7 Sq:B1 (5,7) |
Row6 -->Col0 Sq:B1 (6,0) P:B1 PT:PAWN | Col1 Sq:Wh (6,1) P:B1 PT:PAWN | Col2 Sq:B1 (6,2) P:B1 PT:PAWN | Col3 Sq:Wh (6,3) P:B1 PT:PAWN | Col4 Sq:B1 (6,4) P:B1 PT:PAWN | Col5 Sq:Wh (6,5) P:B1 PT:PAWN | Col6 Sq:B1 (6,6) P:B1 PT:PAWN | Col7 Sq:Wh (6,7) P:B1 PT:PAWN |
Row7 -->Col0 Sq:Wh (7,0) P:B1 PT:ROOK | Col1 Sq:B1 (7,1) P:B1 PT:KNIGHT | Col2 Sq:Wh (7,2) P:B1 PT:BISHOP | Col3 Sq:B1 (7,3) P:B1 PT:QUEEN | Col4 Sq:Wh (7,4) P:B1 PT:KING | Col5 Sq:B1 (7,5) P:B1 PT:BISHOP | Col6 Sq:Wh (7,6) P:B1 PT:KNIGHT | Col7 Sq:B1 (7,7) P:B1 PT:ROOK |

Process finished with exit code 0
```

## Tests/Operations and Results:

### Chess Library Tests:

In order to test the chess library and ensure that the functionality works as intended, you must run the tests in the Tests package and check if any test fails. You should see the following result if all tests pass:



```
Run: main.java in Assignment_1_project x
```

Test Class	Duration
java (main)	28 ms
JUnit Vintage	4 ms
BishopTest	2 ms
BoardTest	1 ms
KingTest	
KnightTest	1 ms
PawnTest	
PieceTest	
QueenTest	
RookTest	
JUnit Jupiter	24 ms

## Modification of Previous Configurations:

If modification of the previously shown configs are required, you must modify the following functions within the board class and ChessBoardView Constructor:

```
/**
 * This function sets the initial pieces on the board to start the game board
 * @param width width user selects for chess board
 * @param height height user selects for chess board
 * Helper function to help initialize active_pieces member variables
 */
public void setInitialPiecesRegular(int width, int height){...}

/**
 * This function sets the initial pieces on the board to start the game board
 * This function include the fairy pieces, Joker and HolyHorse
 * @param width width user selects for chess board
 * @param height height user selects for chess board
 * Helper function to help initialize active_pieces member variables
 */
public void setInitialPiecesModded(int width, int height){...}
```

```
/**
 * Constructor for the Game View
 * @param gameType 0 if normal, 1 if modded game, 2 if empty board
 */
public ChessBoardView(int gameType){
    if(gameType == 0)
        chessBoard = new Board( setWidth: 8, setHeight: 8, setPieces: false);
    else if (gameType == 1)
        chessBoard = new Board( setWidth: 8, setHeight: 8, setPieces: true);
    else if (gameType == 2) {
        chessBoard = new Board( setWidth: 8, setHeight: 8, setPieces: false);
        chessBoard.setInitialPiecesModded( width: 8, height: 8);
    }
}
```

## Game/UI Testing:

When testing the game loop and terminal UI, do the following and ensure the results are as expected:

- start the program
- access the main menu
- quit from the main menu
- display score from the main menu
- start a normal game from the main menu (use the show board option in the game menu to ensure board is correct)
  - forfeit a game from the game menu
  - move any piece a valid move location (ensure this by using the show board option and that the piece moved matched the one you wanted to move or that an error message appears)
  - make the other player move (check if the player turn changed to the opposite of what it was in the previous move)
  - undo a move(s) (ensure this by making multiple moves, noting the board at each state, and comparing the board to the corresponding state after undo)
  - ensure invalid moves cannot be made; ie within piece behavior and king cannot move into check
  - ensure game ends when checkmate is reached and the score for the wining player increases by 1 point
  - ensure game ends when stalemate is reached and the score is not updated for either player
- start a modded game from the main menu (use the show board option in the game menu to ensure board is correct)
  - forfeit a game from the game menu
  - move any piece a valid move location (ensure this by using the show board option and that the piece moved matched the one you wanted to move or that an error message appears)
  - make the other player move (check if the player turn changed to the opposite of what it was in the previous move)
  - undo a move(s) (ensure this by making multiple moves, noting the board at each state, and comparing the board to the corresponding state after undo)
  - ensure invalid moves cannot be made; ie within piece behavior and king cannot move into check
  - ensure game ends when checkmate is reached and the score for the wining player increases by 1 point
  - ensure game ends when stalemate is reached and the score is not updated for either player