**CIS 163**

**Project 2 – Chess Game**

**Group project**

**Groups can be 2 or 3 students per group**

**Due Date**

* At the beginning of the lab, see the schedule, last page of the syllabus.

**Before Starting the Project**

* Review Inheritance, Polymorphism, and Interfaces (Chapters 8 and 9 in the textbook)
* Read this entire project description before starting

**Learning Objectives**

After completing this project, you should be able to:

* design, implement, and test a small class hierarchy
* use two-dimensional arrays and enum types
* implement a GUI-based game
* work with a group on a project.

**Project Description**

Your assignment is to implement a simple chess game model and a simple GUI program that allows two humans to play the chess game. The design provided organizes the different pieces into a class hierarchy that utilizes polymorphism.

For information on objective of the chess game, board setup, how the chess pieces move, and to learn how to play the game, see: https://www.chess.com/learn-how-to-play-chess

**NOTE: The different steps could be divided among the group members.**

**Steps 2 – 5 and steps 9 - 10 (GUI) - (the ordering is only a suggestion, ]**

**Steps 6 – 8 must be completed in sequence. Do not start on Step 6 until steps 1 – 5 are completed.**

**Steps 11 and 12 are challenge activities. To be done when the other steps are completed.**

**Step 1: Using your favorite IDE, create a project called “ChessPrj”**

* Create a package named “chess”
* Include the following classes/interfaces in the chess package. These classes/interfaces are supplied to you. You **must** use these as provided, i.e., you are not allowed to make any changes to these classes/interfaces.
* IChessPiece
* IChessModel
* Player
* Move

**Step 2: Implement the ChessPiece class – initial code provided**

* The ChessPiece class implements the IChessPiece interface
* A player (black or white) owns a chess piece.
* type() method is abstract.
* isValidMove(Move move, IChessPiece[][] board)- this method should
  + Verify that the indexes (from and to) associated with the Move object are not out-of- bounce
  + Verify that the starting and ending locations are different.
  + Verify that this piece is located at [move.fromRow, move.fromColumn] on the board.
  + Verify that the board at location [move.toRow, move.toColumn] does not contain a piece belonging to the same player.

**Step 3: Implement the Pawn and Rook classes – initial code provided**

* Pawn and Rook classes extend the ChessPiece class
* Implement type() method
* Implement isValidMove()method. Make sure to invoke the isValidMove() method from the base ChessPiece class and add functionality specific to the piece.

**Step 4: Implement the King, Queen, Knight, and Bishop classes – initial code provided**

* King, Queen, Knight and Bishop classes also extend the ChessPiece class.
* Implement type() method.
* For now, make isValidMove() method return false. Full implementation of this method is not part of the base functionality (see Step 6).

**Step 5: Implement the ChessModel class – initial code provided**

* The ChessModel class implements the IChessModel interface.
* This class is responsible for storing the chessboard and implementing the game logic.
* Implement the methods from the IChessModel interface.
* For now, make inCheck() method return false. Full implementation of this method is part of the additional functionality (see Step 7).
* For now, make isComplete() method return false. Full implementation of this method is part of the extra/bonus functionality (see Step 8).

**Step 6: Complete the King, Queen, Knight, and Bishop classes**

* Fully implement isValidMove() method of King class.
* Fully implement isValidMove() method of Queen class.
* Fully implement isValidMove() method of Knight class.
* Fully implement isValidMove() method of Bishop class.
* Should only be able to move if it is a valid move.

**Step 7: Implement the inCheck() method of ChessModel class**

* Fully implement the inCheck()method of ChessModel class.
* Your program must display a message when the current player is in check using JOptionPane.showMessageDialog().
* Solid error checking.

**Step 8: CheckMate**

* Fully implement the isComplete() (i.e., checkmate)method of ChessModel class. For example: Check to see if the King is checkmated or can move out the way (i.e., uncheck itself) or another player can block the check. See the instructor for more details.
* Your program must display a message when the game is complete using JOptionPane.showMessageDialog()
* FULL Error checking! YES this means some JUnit testing. Please see the instructor for this step.

**Step 9: Implement the ChessPanel class – initial code provided**

* **Copy the chess pieces icons (white and black) from BlackBoard into the chess package created previously.**
* The ChessPanel class extends the JPanel class.
* This class is responsible for presenting the graphical user interface, responding to user actions, and updating the view.
* The game should implement a standard form of chess, white moves then black moves.
* Only allow valid moves.

**Step 10: Implement the ChessGUI class – initial code provided**

* The ChessGUI class contains the main method that creates and displays the chess game GUI.

**Challenge functionality**

**Step 11: Undo**

Fully implement an undo JBUTTON (Not a JMenuItem). Should allow multi-undos.

**Step 12: - More Functionality**

* Write a simple AI set of rules in the following order. This is difficult (do the best you can! Below are some suggested rules to follow)
  1. Check to see if you are in check.
     1. If so, get out of check by moving the king or placing a piece to block the check
  2. Attempt to put opponent into check (or checkmate, note: checkmating is difficult)
     1. Attempt to put opponent into check without losing your piece
     2. Perhaps you have won the game.
  3. Determine if any of your pieces are in danger, or you can take their piece.
     1. Take their piece OR.
     2. Attempt to protect your piece.
  4. Move a piece (pawns first) forward toward opponent king
     1. check to see if that piece is in danger of being removed, if so, move a different piece.

**Step 13: JUnits**

* Write JUnits test cases to get 100% coverage of the ChessModel.java class.

**Javadoc Commenting and Coding Style/Technique [10 points]**

* Use [Java Coding Style Guide | GVSU School of Computing](https://www.cis.gvsu.edu/java-coding-style-guide/) as a guide to document the source code in your project and observe good coding style practices.

**What/How to Turn in?**

* Sign up for project demo (sign-up sheet will be available to you later).
* Print out a copy of source code and have it ready for demonstration.
* Staple rubric below to front of print out.

**CIS 163 – Computer Science II**

**Project 2: Chess Game**

|  |  |
| --- | --- |
| Student Name |  |
| Due Date |  |

|  |  |  |
| --- | --- | --- |
| **Graded Item** | **Pts** | **Points Awarded** |
| Javadoc comments and coding style/technique  (<http://www.cis.gvsu.edu/studentsupport/javaguide>)   * Code Indentation (auto format source code in IDE) * Naming Conventions (see Java style guide) * Proper access modifiers for fields and methods * Use of helper (private) methods * Using good variable names * Header/class comments * Every method uses @param and @return * Every method uses a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* separator * Overall layout, readability, No text wrap * Using /\*\* … / for each Instance variable * Has many inner “inner” comments | 10 |  |
| Steps 1 – 5: Base Functionality   * Model/View separation * Functioning GUI * Initial chess board is set up correctly * Pawn and Rook pieces move correctly * King, Queen, Knight, and Bishop | 20 |  |
| Step 6 : Additional functionality   * King piece moves correctly * Queen piece moves correctly * Knight piece moves correctly * Bishop piece moves correctly | 10 |  |
| Step 7: Additional functionality   * inCheck() of ChessModel class | 10 |  |
| Step 8: CheckMate   * isComplete() of ChessModel class | 10 |  |
| Steps 9 – 10: GUI | 10 |  |
| Step 11: Undo | 10 |  |
| Step 12: AI | 10 |  |
| Step 12: JUNIT testing | 10 |  |
| **Total** | **100** |  |

**Comments:**