Project Proposal: Chess Game

My project will be a 2-player version of the game of chess.

A JavaFX GUI for this game will be implemented in a window using the MVC paradigm. The window will consist of an 8x8 grid of interactable components representing the squares of a chessboard. Each player will take turns interacting with the components to make their moves. The board will begin with images representing the standard pieces in chess organized in the standard layout. Upon clicking a piece, the piece will become highlighted, along with all the empty squares which the piece can legally move to on its turn. When one of the legal moves is selected, the piece will be moved to its square and the opposing player will be given their turn. When a piece is captured, it will be removed from the board and placed on a table representing all captured pieces by that player. Players will play until the game is over, and a message signaling the end of the game will be displayed, along with the option to play again. I don't know all of the rules of chess so it is likely that they will not all be implemented, but all core rules required to play chess (such as bishops move diagonally) will be implemented.

Implementation

This project will implement an Object-Oriented Programming design including classes with state and behaviour. This project will also take advantage of the Model-View-Controller design paradigm in its GUI using JavaFX.

Model

The model component of MVC will be responsible for all logical operations computed in the game of chess, along with the rules. The classes and methods implemented will include but not be limited to the following.

Point2D Class

This class will store x and y coordinates for other objects. I will likely copy the Point 2D class from past assignments which demonstrates reusability as one of the major advantages of OOP design.

Piece Class

The piece class will be an abstract parent class of all pieces in chess. It will implement a Point2D location state as well as a boolean captured state. It will also implement abstract behaviors such as an availableMoves method which will find all locations on the board which the piece can legally move to, a move method which will move the piece. A non-abstract isCaptured method which will remove the piece from the board and place it in a 'Captured' section elsewhere. This class will also inherit Comparable to be able to sort pieces (probably by importance) in the captured section.

Individual piece classes.

Each piece in chess (pawn, rook, knight, bishop, queen, and king) will have its own class which will extend the Piece class and inherit its state and behavior. Some pieces may have additional methods implemented such as a gameOver method in the King class which will end the game once it is captured. Each piece class will also include an Image of its piece included as a state.

Board Class

The board class will contain a 2D array of its pieces in their location on the board, and captured pieces. This class will have a display method that will update the locations of the pieces and return the 2D array.

View

The view component of MVC will be responsible for storing all JavaFX objects and displaying them to the user. The chessboard will use one JavaFX component and the controller class will use the mouse location on a click to determine the selected square in this class. It will have an update method which will have the model passed in and will update itself accordingly.

Controller

The controller component of MVC will be responsible for recording user interaction with the game and updating the model and view components using eventHandlers.

Course Concepts

Classes: Yes – As described above.

Abstraction: Yes – Piece will be an abstract class with abstract methods.

Encapsulation: Yes – Class state and behaviors will be adjusted accordingly.

Inheritance: Yes – Pieces will inherit from the Piece class.

Interfaces: Yes – Comparable will be implemented in the Piece class.

Polymorphism: Yes – The Board class will store an array of pieces as type Piece

Graphical User Interfaces: Yes – Implemented with the MVC paradigm.

Abstract Data Types: No – I have not planed for this but will take advantage of it if an opportunity arises.

Recursion and Data Structures: No – I have not planed for this but will take advantage of it if an opportunity arises.

Exception Handling: Maybe – I might make an exception for illegal moves outlined in the possible additions section.

File Input and Output: Maybe – I might record moves using files outlined in the possible additions section.

Possible Additions

If time permits. I would like to implement multiple view classes to represent a start screen and a game over screen along with popup windows for invalid moves. Invalid moves may also use the creation of an invalidMove exception for example which would trigger the window. I would also like to write the moves of a game to a file and be able to read the file back afterward to replay a game. I would finally like to add a move hint which would show if moving a piece to another location would open an immediate danger to it being captured on the next move by the opponent.