Chess - Project Part 4

1. **What features were implemented?**

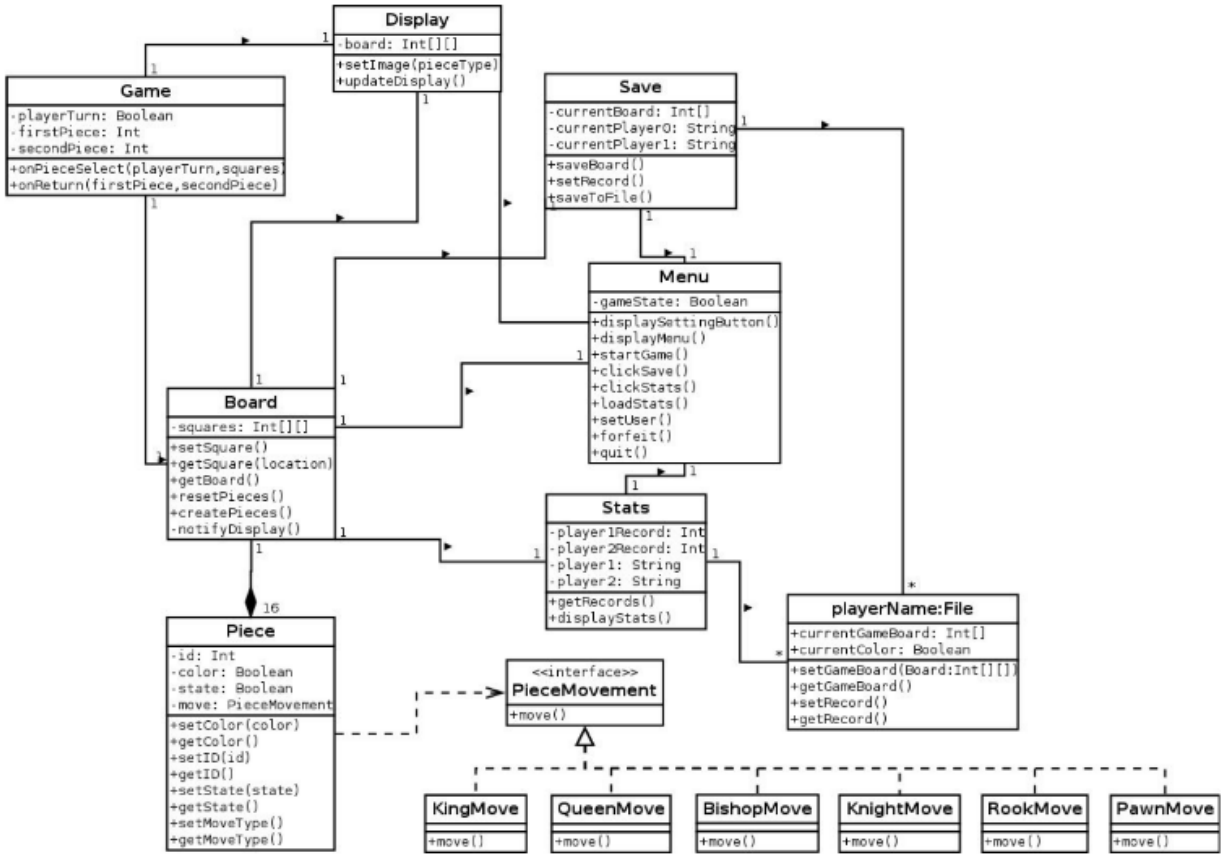
Piece moves & validation:
    King, Queen, Bishop, Knight, Rook, Pawns (up & down)
Commands:
    Move, Quit, Forfeit, Save
Visual:
    Text & Unicode Pieces, Display Board, Display Killed Pieces, Display Prompts
Connection:
    Socket Connection, Exchange Game Info Between Players/Programs
Menu Options:
    Host Game, Connect To Game, Load Saved Game, Display Stats
Persistance:
    Save A Game, Save Player Statistics
Options:
    Select Piece Type (text or unicode), Enter Username

2. **Which features were not implemented from Part 2?**

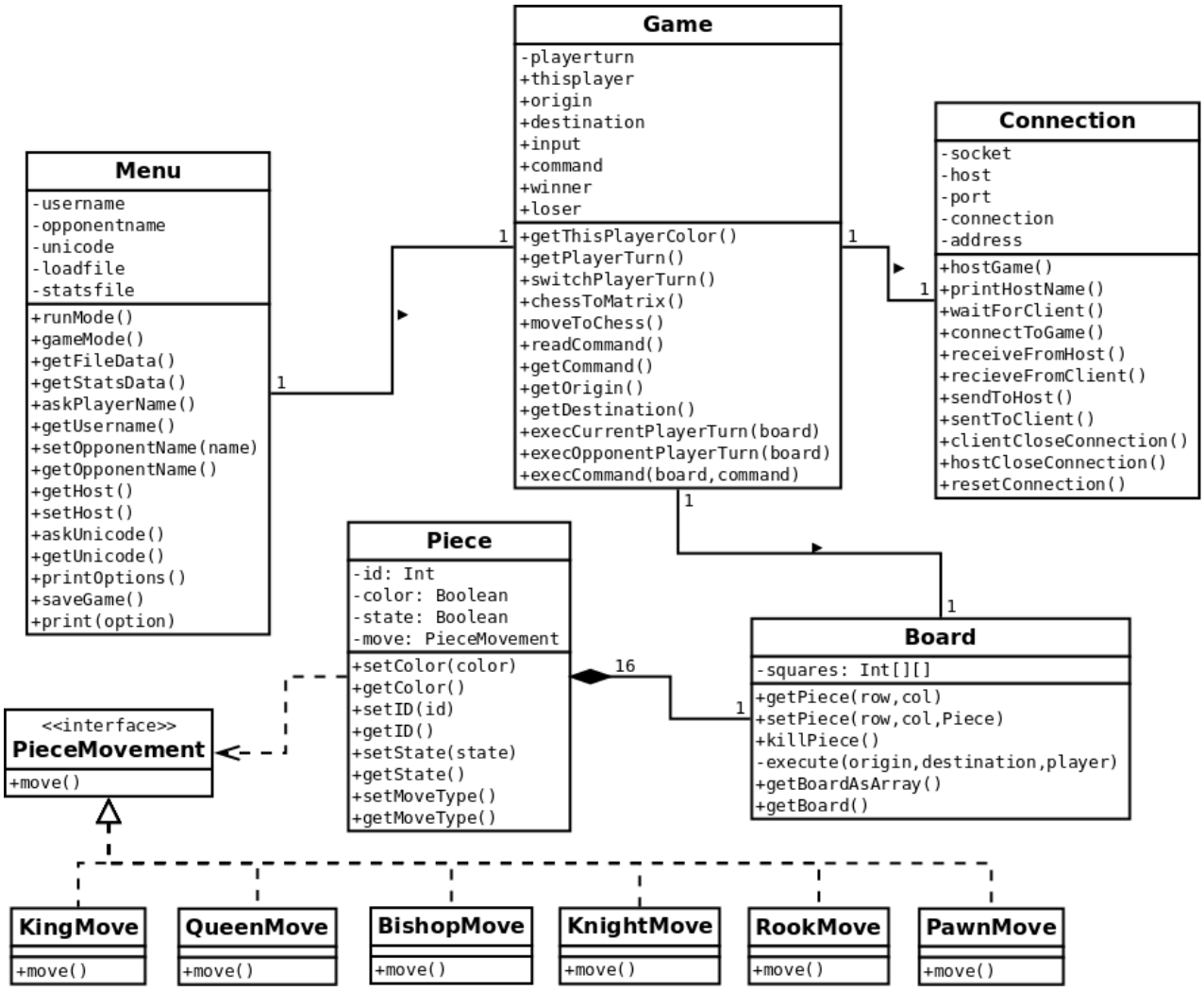
Logging Out & Logging In
Keeping Track Of & Viewing A Leader Board
Playing Against An A.I.

3. **Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.**

Initial Class Diagrams



Final Class Diagrams



The Strategy Pattern remained the same (Board, Piece, & PieceMovement).

What changed somewhat drastically was everything else, mainly because of the decision to switch form hosting the software on a server and building it with Javascript, to having the software run in a shell and building it with Python. It greatly simplified the structure.

The concept change introduced the use of a Front Controller Pattern & a Facade pattern, which is discussed in the next question.

4. **Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns inyour system?**

The Board, Piece, & PieceMovement interface implemented the **Strategy Design Pattern**. When a Piece was constructed, it's moveType() method was defined by a PieceMovement. Python does it a little differently though. The PieceMovement classes were actually just straight-forward functions.

Our original class diagram looked a bit like spaghetti between the following classes: Menu, Board, Display, Save, Stats, & Game. So, the idea of using a **Front Controller Pattern** was developed. We would use the Menu class to accept all user input, which would then send the data to the Game class (the dispatcher), which would then call on other classes (views) depending on the command.

Due to time constraints, we only partially implemented this design pattern. The Save,Stats, & Display classes were all absorbed into the Menu class. Thus our Front Controller Pattern only had one view, the Board class. If we had more time, we would separate the handling of displaying visuals and persistance from the Menu class into their own view classes respectively.

In addition, a Connection class was added to handle the Python socket library. This class is just a **Facade Pattern** which provided two benefits: 1. Several functions that had to always be called together were wrapped into their own methods. 2. The method names made it far easier to understand how to create and use a socket connection.

5. **What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?**

Well, originally we did not want to get too specfic with the requirements because if any major changes occurred then a lot of the design patterns would go out the window, and that's exactly what happened. You have to code within the limitation of the languages and environment your coding in, so the switch to Python and sockets made a lot of the initial class diagrams useless.

Design patterns (and anti-patterns) are helpful to avoid digging yourself into deep trouble, so just having the knowledge of the principles were important during the process. They were used frequently while refactoring the code as we went along. However, it would have been a mistake to try to stick too hard to any of the initial plans.