Kyle Mock
Jake Shay
Matthew Cohen

OOAD Project 6 - Chess Program

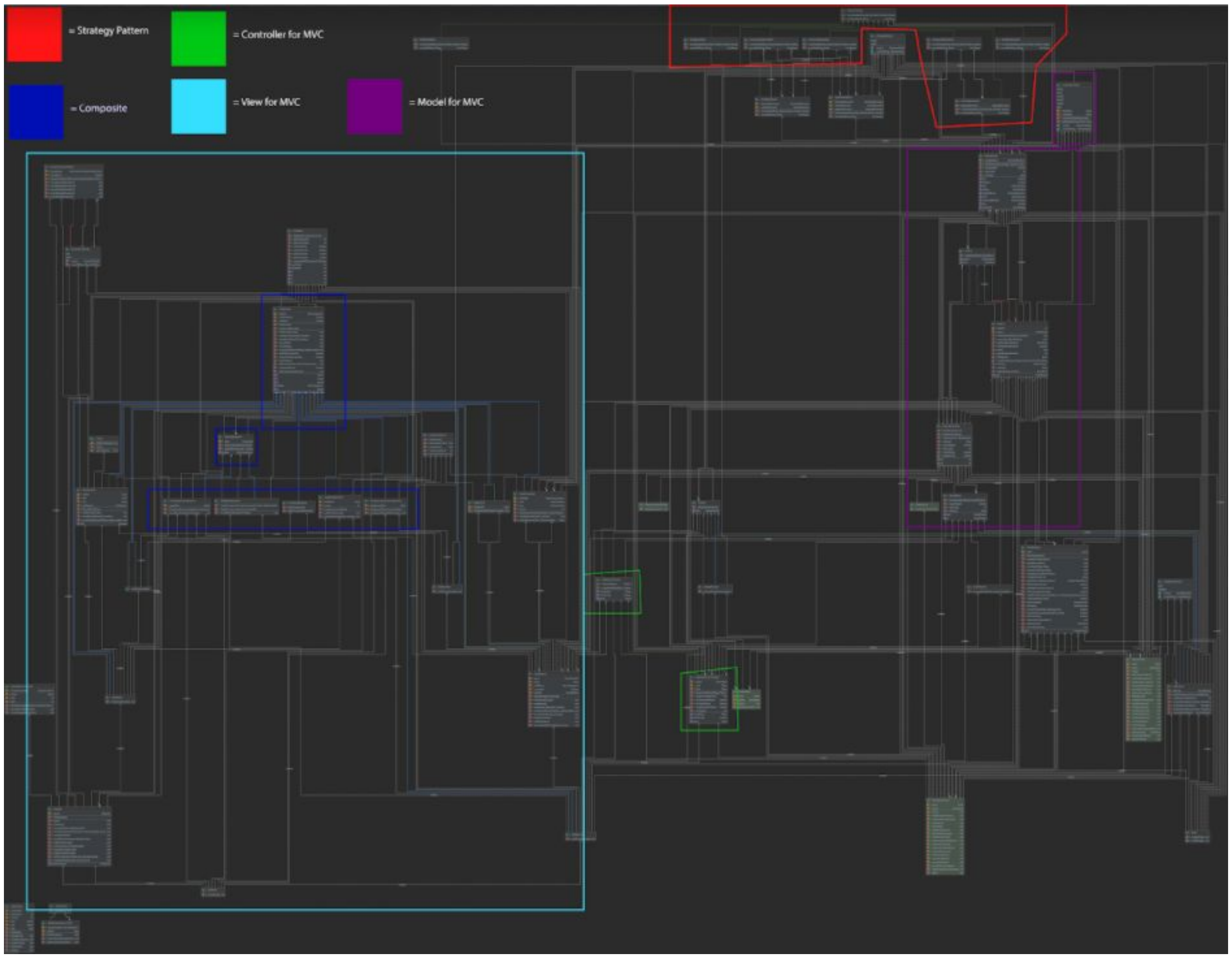
System Statement

Our final design incorporated almost all of the features we had planned. These included playing vs. an AI opponent and playing vs. a human opponent. However, there were some features that we were not able to implement. Initially, we discussed the ability to save a game and resume it at a later time, unfortunately, we did not have time to put this feature into the game. We also wanted to implement a way to track game stats and calculate a high score, as well as a harder AI algorithm. Again, this was something we did not have time to add. We had to cut out a few nice-to-have features because we noticed that our program needed some sort of MVC pattern to facilitate interactions between the UI and game logic. We also wanted to add the ability to restrict pieces from moving if they were blocking an enemy from checking the king, but did not have time to implement.

Kyle Mock
Jake Shay
Matthew Cohen

Class Diagram and Comparison

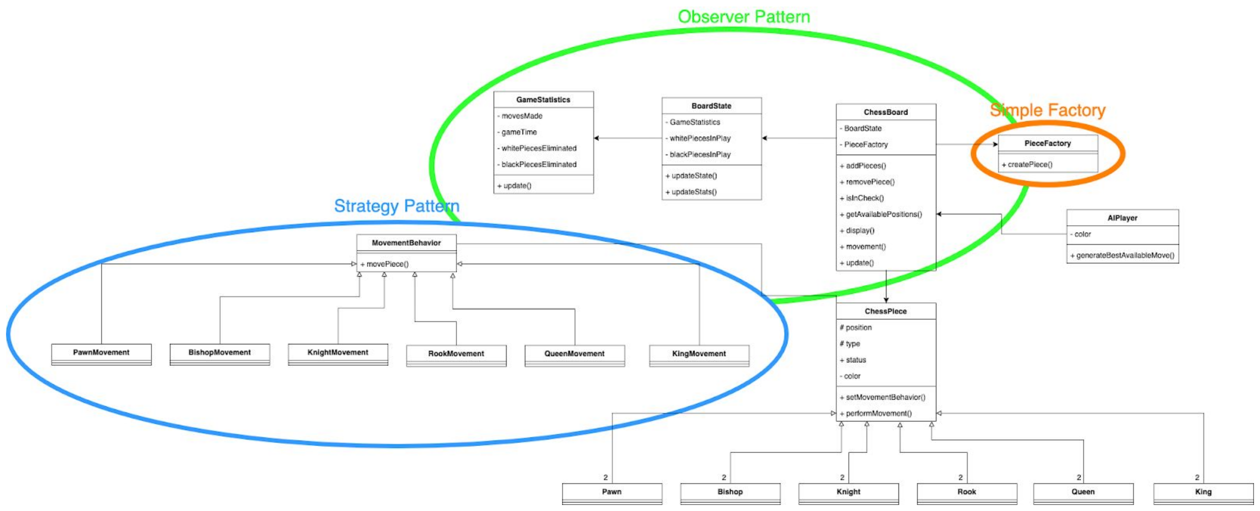
UML



Full-size:
https://drive.google.com/file/d/15aamRv6zCITY7QrFtSXyarFRkhZCnMvl/view?usp=sharing

Kyle Mock
Jake Shay
Matthew Cohen

Project 4 Class Diagram



Changes in UML summary

The original UML that we developed for project 4 was very basic. Most of the classes within remained in our final designs. However, we did choose to omit the observer and simple factory patterns leaving only the strategy pattern in the final iteration. A big difference between project 4 UML and the current UML is that the current UML includes the implementation of a GUI. The GUI implements the composite pattern which was not included in our original plans. The most significant change was the inclusion of an MVC in our final implementation. Originally, we thought we would use an observer pattern but with the addition of the GUI, the MVC pattern became the most logical solution. Finally, there is a significant increase in the number of classes.

Code Attributions

- Matt: I did not include any third-party code in my implementation, however, the cost function written for the medium AI player uses an algorithm found at https://en.wikipedia.org/wiki/Chess_piece_relative_valu
    - I also referenced geeks4geeks for basic java syntax throughout the project.
- Jake: referenced: https://www.geeksforgeeks.org/convert-char-to-int-in-java-with-examples/ for working with board positions as strings.

Kyle Mock
Jake Shay
Matthew Cohen

- Kyle: The GUI uses JOGL as a 3rd-party library: https://jogamp.org/jogl/www/, modified assets: https://joszs.itch.io/chess-pack, and a custom font: *Roboto: Designed by Christian Robertson.*

Process Statement

- UML: though we didn't use UML's as much as we expected, the process of creating one in the initial planning phase was helpful. It forced us to consider how different elements of the game would fit together before we started actually coding it. It also helped us visualize the role design patterns could potentially play in the project.
- Test-Driven Development: we discovered a lot of bugs with the game through the process of actually playing it. However, writing tests was very helpful, in that it allowed us to determine the source of the bug as well as how to fix it.
- Separation of Concerns: By dividing the project into discrete components we were better able to divide up work and keep our code from becoming too interdependent (coupled). By using patterns such as MVC we were able to keep our code well structured and easily testable.