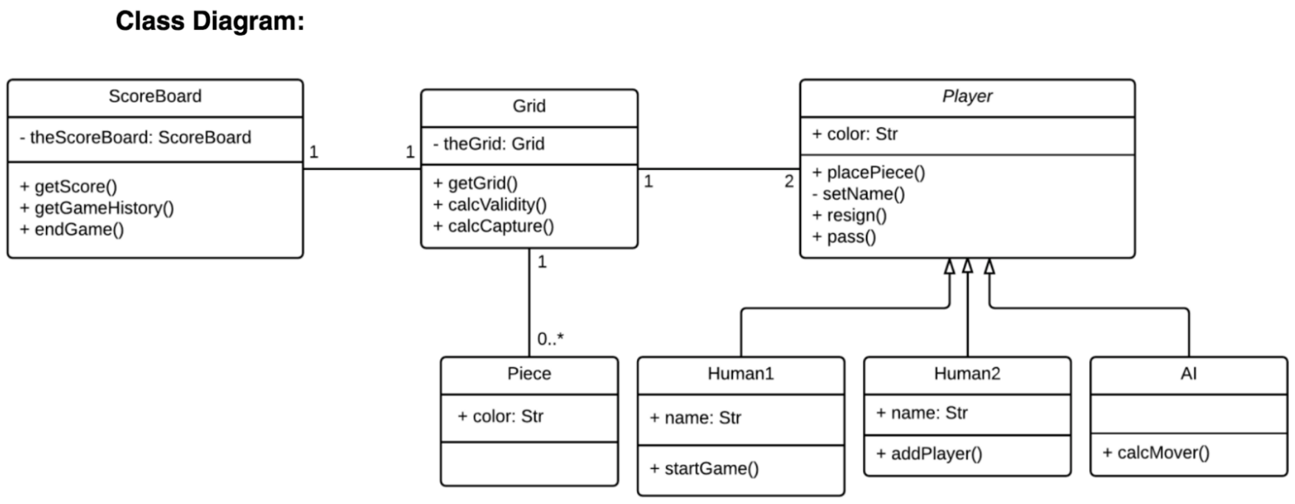
PlayGo! Project Part 4 Write up

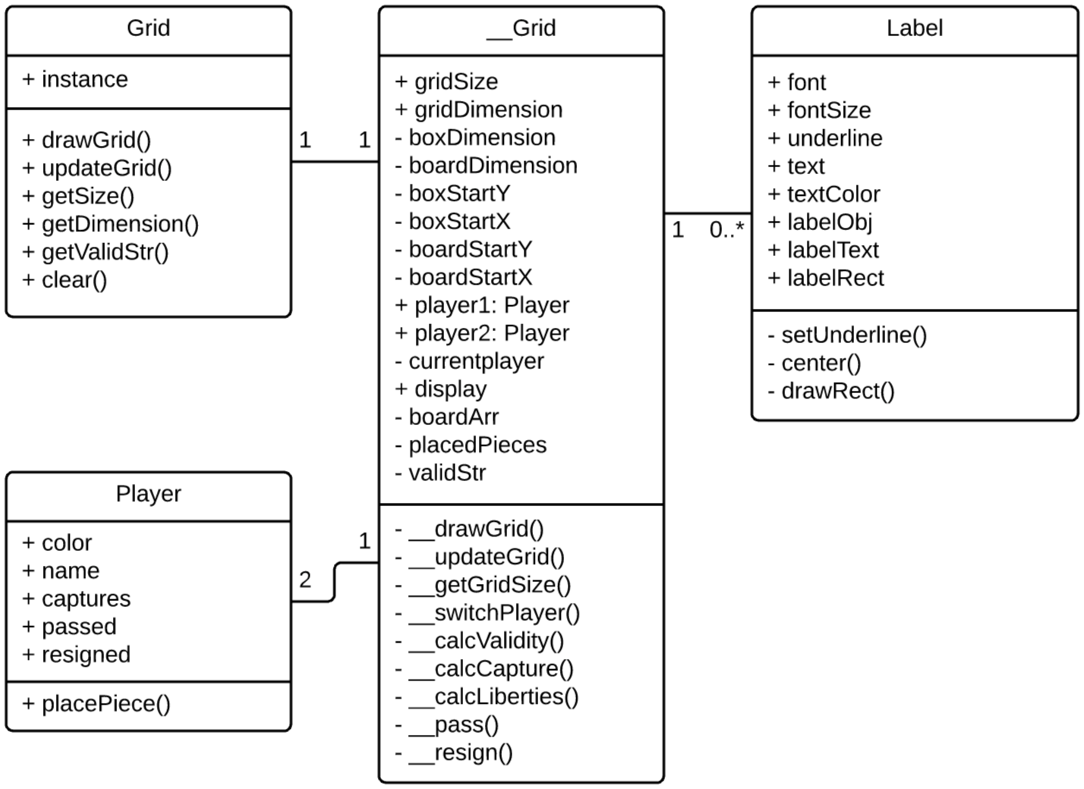
1. Features we implemented:
   - Choose grid size
   - Passing
   - Resigning
   - Set name
   - Only allow legal moves

2. Features we did not implement:
- Playing against an AI (was our stretch functionality)
- Displaying last 5 game results
- Displaying territory score
- Capturing groups of connected stones (can only do one isolated stone captures)

3. Original Class Diagram:



3. Current Class Diagram



Our changes to the class diagram were primarily motivated by our use of the Pygame library. Given our limited experience with OO programming principles, it was hard for us to design an effective class diagram from the outset. We initially had planned on using a CLI but adapted our project to pygame GUI. Using pygame made certain things easier and other things harder. For example, we had to remove our piece class as we couldn't figure out how to use these objects on the board and make it work well with pygame (although we do know now, time was not on our side). Additionally, we removed the scoreboard class as we didn't implement the functionality or it was added to the grid class. Due to time constraints, Grid got bloated and we had a hard time trying to refactor and encapsulate parts of Grid and keep the same working behavior.

4. We primarily relied on the Singleton and Façade design patterns. The singleton pattern was used to ensure we had one and only one instance of the Grid and accessor methods to update its state were limited to protected methods. Time permitting, we would have liked to make use of flyweight for pieces. Having piece objects would have made doing our capturing algorithms for groups a lot easier (using Flood Fill algorithm for example) Our scoreboard functionality could've been implemented using an Observer pattern as well.

5. One of the main things we learned is how much the analysis and design process simplifies necessary acts like refactoring or expanding your code. Initially, we began with a "survival code" mindset, just trying to write code that worked before refactoring it to OO code. We quickly realized that this was causing working together on the project very difficult as it was hard for someone who didn't write the code to understand it. Additionally, the project became very jumbled and confused quickly as more and more code was added. Once we realized this, we stepped back and decided that the ease of "survival coding" was not worth the effort required to refactor or understand and that properly planning, writing, and refactoring code makes it much easier to collaborate. Finally, once the code was broken in to working classes, each of us was able to work on the individual files without causing errors in each other's code or merge conflicts.

Even though our initial analysis and design was not very close to our end product, the thought process helped guide us through this project. We knew what we wanted our product to do and used the template we had set up to format the final class structure. We discovered that certain classes were unnecessary while others were more important than we initially thought. Despite this rough first attempt at a large scale project, we all learned how important the process is. Finally, we have a better picture of a truly designed project should look and the benefits that analysis and design has.