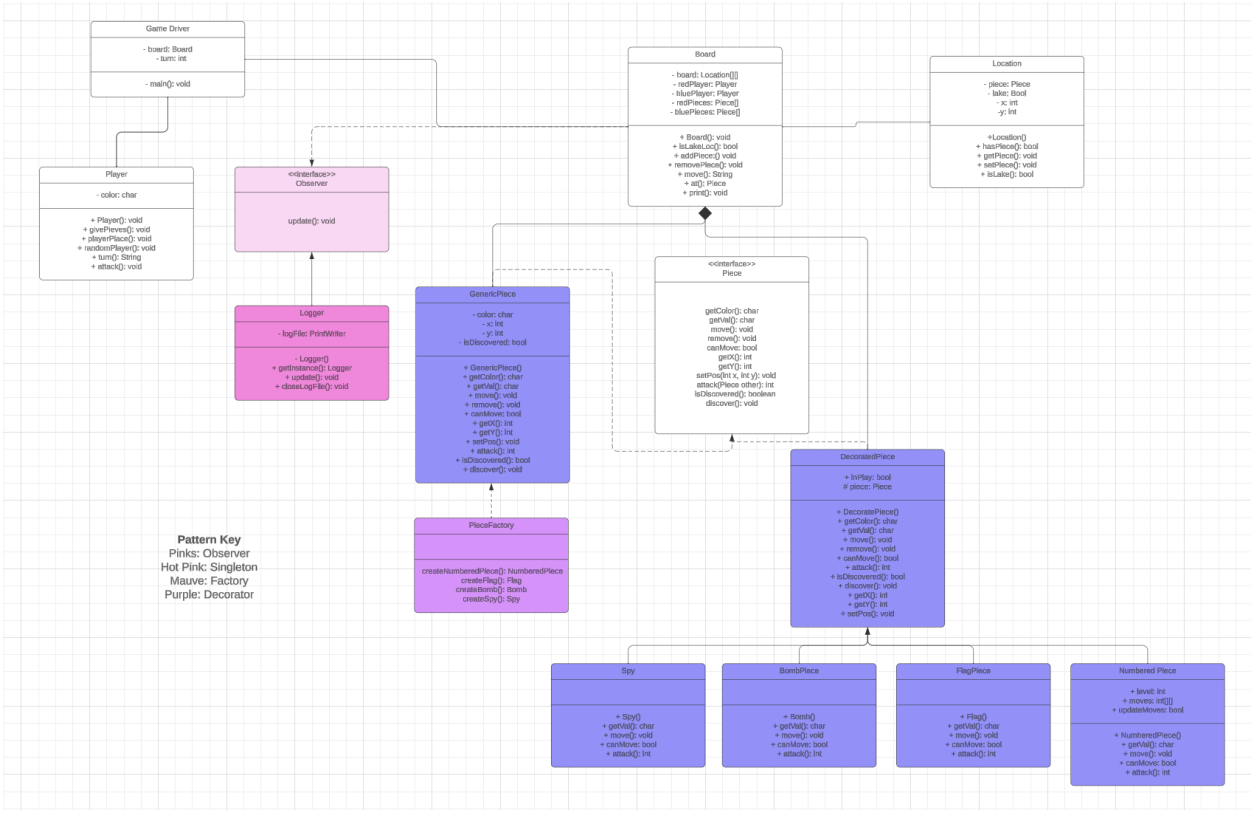
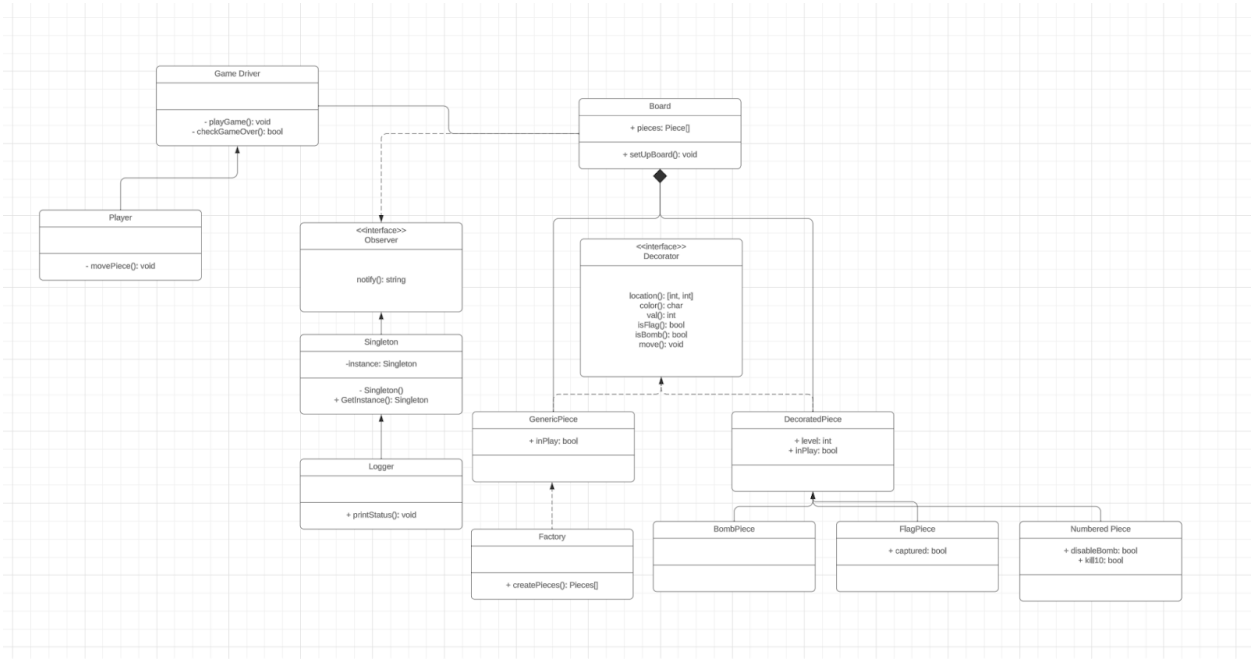
Project name: Strateho
Group members: Emma Worthington and Kyle Paris

Final state of the system:
In the end, we created a fully functioning Stratego game as we initially intended with the exception that it is a command line game rather than a UI. The reason that we made this decision is that we really liked how the command line game looks, it has a cute nerdy feel to it that we think best embodies our techy fun adaption of the game. Ultimately, we really wanted to focus our time on writing good clean code with well-implemented patterns rather than implementing an unnecessary UI. We included the four design patterns we originally intended, the observer, singleton, factory, and decorator patterns. We chose these functions because we liked how they improved our game function in arcane and made sense with the intended architecture of our Stratego game. The main changes that occurred between the past two assignments and this one were changing the game logic of the move function from the player class to utilize the decorator pattern, and expanding to add all of the game features.

Final UML:



Project 5 UML:



What's changed:

Between our original UML and our final UML, the main difference is there are significantly more added functions in the final, there aren't however many different classes. There is an added location class in our final architecture but primarily there are just significantly more classes and variables added in as we built out our game and implemented the different components. As previously stated, one of the most major changes was where the game logic for move occurred, and this was finishing the implementation of our decorated piece pattern, as well as adding the factory pattern in the final project. On top of that the functions of special pieces were added such as the 9 piece being able to move more than one in the direction of the player's choosing.

Third-Party code statement:

We did not use any third-party code in the creation of our Stratego game.

Statement on the OOAD process for the semester:

1. UMLs - UML diagrams helped us organize our ideas between projects very well. A lot changed from our initial UML to our final implementation; however, it gave us a great starting point and forced us to fully flush out our ideas and work together to create our combined vision for what we ended up coding.
2. Writing code asynchronously - Initially it was hard to write code asynchronously because of how dependent different methods are on each other. An example of this is that the decorator class is used to dictate how pieces move and is implemented into the player function so one cannot exist without the other. The dependencies of code on each other make it difficult to code anything without each other's code so it was definitely a process for both the final project and arcane to learn how to code together and work with missing functions.

3. Managing abstraction layers and information - It was hard to plan what should be in which functions as far as interfaces, abstract classes, and concrete classes, in our game. We would plan to put code in certain abstract classes and then realize it should actually be in the concrete class or vice-versa.