Varunjit Srinivas

Isaac Sim

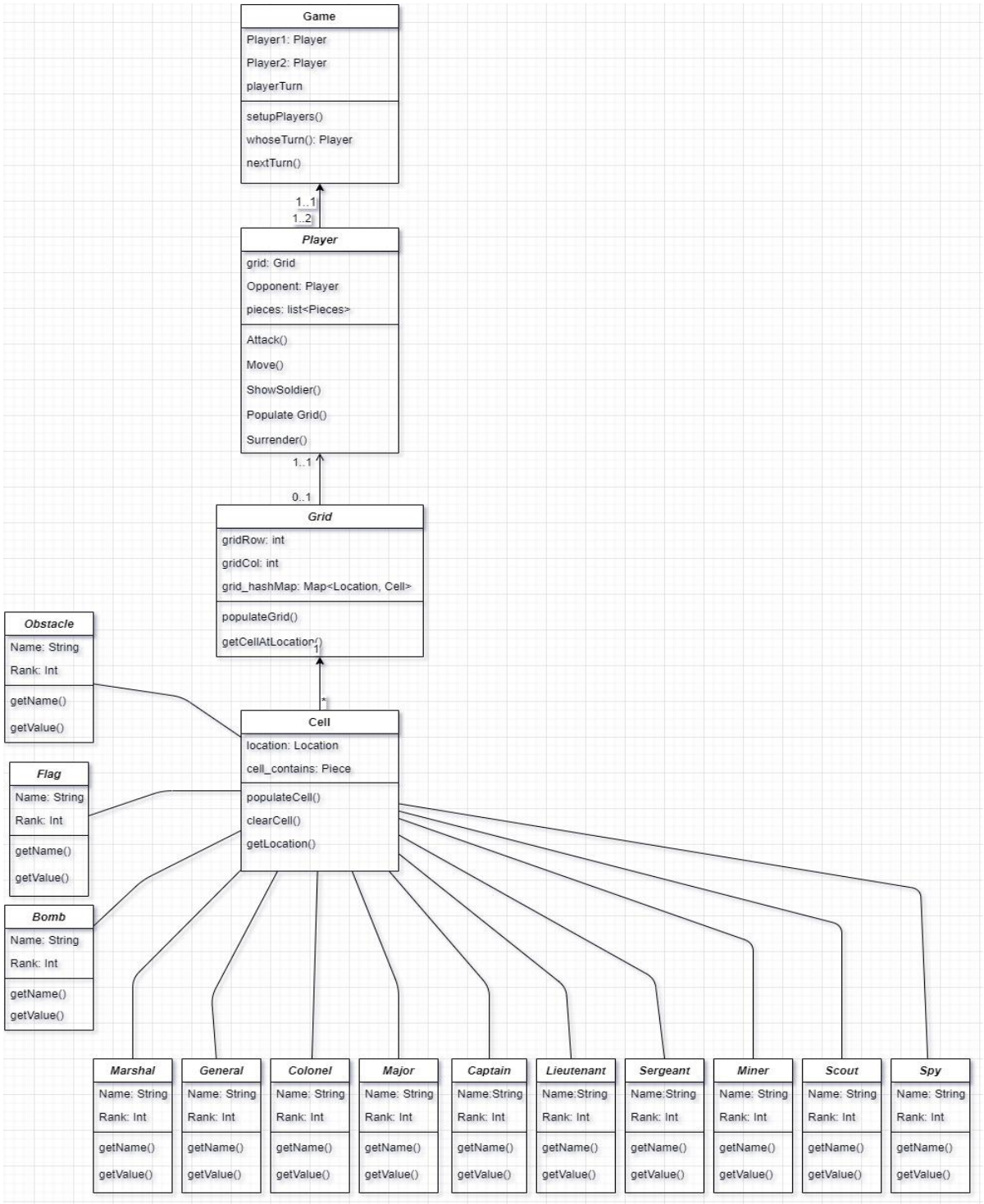
Jason Nguyen

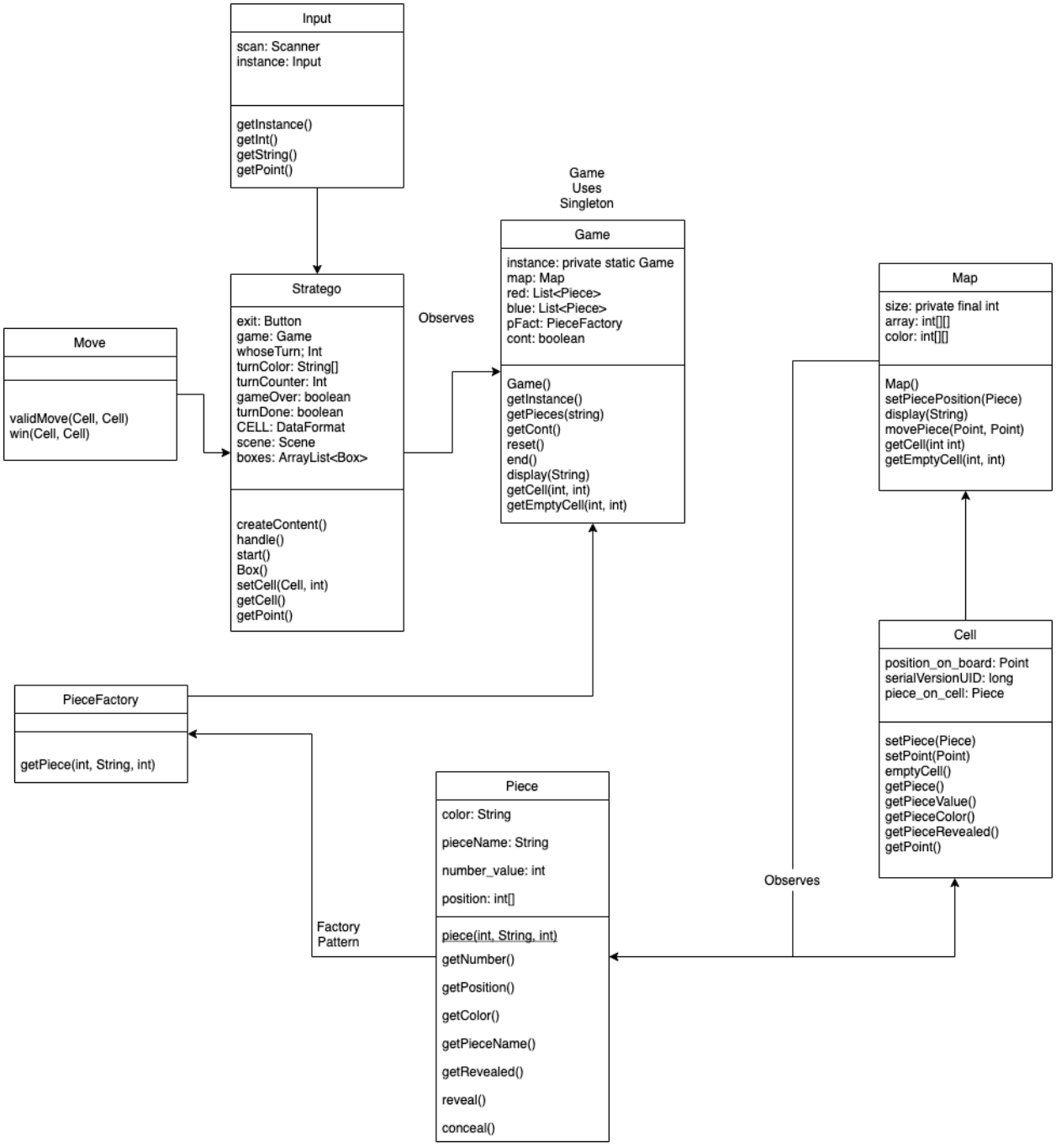
Project 7 Final Report: Stratego Board Game

Final State of System Statement:

When we first started working on implementing the board game *Stratego*, we were very ambitious with our project idea. We originally had planned to do a web application with connections to a back end and a database, but now we have created a Java based computer application. The features we implemented were the base game of Stratego with a player versus player mode and a local front end GUI. Both players are able to move a piece every turn count. All game rules have been implemented, so this is an authentic Stratego experience. A feature we attempted to implement, but dropped due to workload and time constraints, was a player versus bot mode for the board game.

Final Class Diagram and Comparison Statement:





As you can see, there are some very key changes to our UML class diagram from Project 5 to now. Project 5's UML Diagram is what we had planned on implementing, Project 7's is what we actually implemented. The main differences are the delegations of the work for each class. In Project 5, we thought it would be smarter to separate each piece type into their own subclass, when in reality each piece is so similar there was no reason to do so. Our player class became

really similar to our recent game class, and our old game class is more akin to our current Stratego class. Other than our different implementation of the factory pattern, as well as our changed implementation of the drivers of the game, the UML diagrams share a similar structure. It was really interesting on how we built on our UML diagrams over time, and how it retained the same basic concept of what we were trying to do, with major changes.

Third-Party code vs. original code Statement:

We did not use any direct references to any outsourced code. All code that we have written is our own original work. However, we did use resources outside of our knowledge to help develop the game. Jason was in charge of working on our GUI, and since he has had little experience in Java, he researched the best way to implement a GUI and concluded that JavaFX was the way to go. As he was working on the GUI, any questions he had he would search up on the website stackoverflow to use as a guide / reference. Much of the tutorials and documentations used to complete the GUI was from Oracle.

-https://docs.oracle.com/javase/8/javafx/api/index.html

Statement on the OOAD Process:

Some design patterns we used were both the Singleton pattern and the Factory pattern. Using UML to plan out our project before coding or setting a base really helped. Most of us have never really sat down to plan a project in its entirety, we used to just sit down and code until a problem would arise and cause us to re-evaluate the work we are doing. We agreed that the Architecture Diagram was the most useful out of all the diagrams, because it helped us see the perspective and scope of our project. Isaac implemented the Singleton Pattern, he chose it because he wanted to have 2 object pools that are initialized in the beginning, and don't change at all after. Using singleton cleaned up the main function a lot and helped with delegation of work outside of giant classes. We also decided to use the Factory pattern, and Varunjit and

Isaac agreed on using that because they thought it would be a good idea to separate creation from use in the classes.