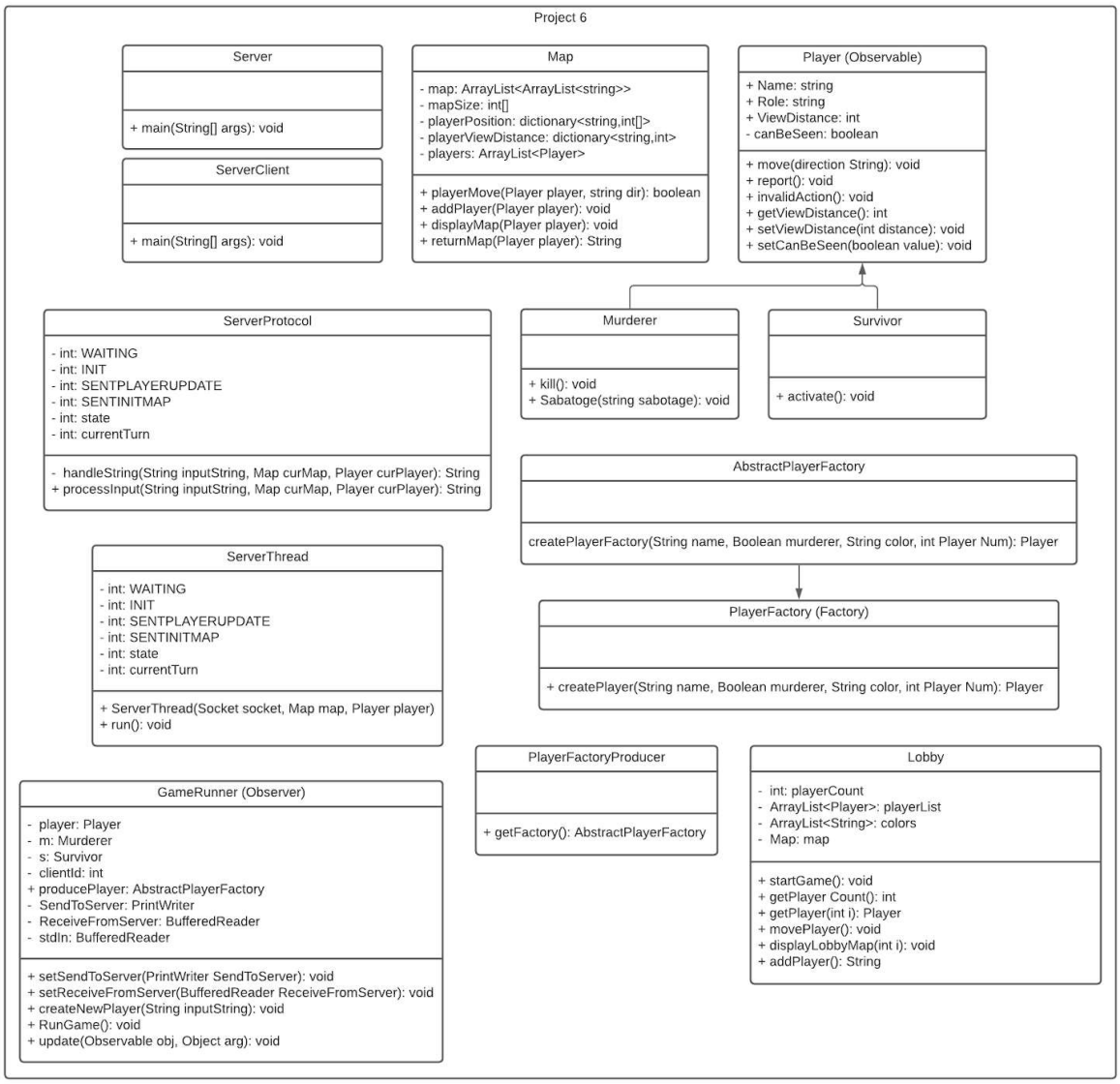
Project Name: Spooky Town
Team Members: Chandler Garthwaite, Tanner Slemmer, Kolin Newby

State of System

In the final state of our project we have 3 separate sections in the project. The player classes, which includes the Top tier Player which is then inherited by the murderer, survivor and ghost. Each having some of their own specific actions due to the game. This also includes a gameRunner program that controls the communication to the client server through an observable. It also handles the creation of the player through the factory created to make creation easier. The next section is the server side implementation. This includes a client server, the actual server, and a protocol handler. The client server being the connection from the gamerunner to the server. It creates a game runner object and starts the game up. The server acts as a socket pipe so that the protocol can take in the strings being sent and handle the correct actions. The protocol will take care of most actions in the game with retrieving players and the map. The last section is the map portion. This section of classes contain the ability to read in a map, Gather view distance based on the players, as well as being able to understand who is next to who and display that on the map.

Class Diagram & Comparison



- In our initial UML diagram we had the player classes with the factories but with nothing observing the actual actions of those players. So with the final implementation we decided to create a GameRunner which communicates with the client Server and observes the player classes and their actions
- The next portion we changed was the sabotage decorator. We decided against doing this because there was a simpler solution that is easier to implement and understand. We decided it could easily just be a variable for the players view and a get/set function. Thus whenever a map function is called and relies on the view it will just grab the new value.
- One thing you will see is that there is no server diagram in the first uml. This is because we were unsure how the server implementation would work.

Third-Party Code vs Original Code

The majority of our code is going to be original code. This incorporates all the map classes and the player classes and gameRunner. The patterns used are very similar to project 4 structure which is taken from the notes. The server side implementation is pulled from a knock knock protocol example pulled from the internet (source here). Now it isn't a direct copy, just a scaffolding so that we can have the client servers to connect and communicate in our own way after solidifying the example.

OOAD Process(Negatives or Positives)

1. When starting the project we had an idea of where all of our patterns would fit into the project. However when starting the project with code we realized that some of those patterns wouldn't work as well as some of them were just slightly off. Sometimes its a good idea to just estimate where they can go because when starting to actually design the project we were able to be flexible and finalize where they would be.
2. One of the most important ooad processes we took was the design first. While this is partly due to us having to design it first for project 4 and 5 it is something that is still valuable to the project and its success. With the design out of the way we were able to design a project with 3 distinct parts that could connect very well and allow each team member to work on it at the same time. Which makes it super easy to get things done.
3. Above we mention that we were able to separate the work into 3 sections to be done. While they may not all seem equal they focused on different aspects of computer science, servers, display maps from reading in files, and a strong player class to provide a good foundation. However with splitting the work and their differences, the connection of these proved to be a little difficult and required some creativity. When connecting the players to the client server, we initially thought we could make the client server observable but in the end we needed another class as a form of adapter so that we could communicate.