**Challenges and Lessons Learned**

Over the course of the semester we learned that we essentially needed better communication, which included:

* Communicating progress with assigned tasks
* Delegating work
* Asking for help when needed
* Offering help when idle

Telling people where you’re at is crucial. Ultimately, we’re on the same team, and there were situations where we could have been better about asking for help with tasks we had previously committed to. Something that could have helped with our communication is having a specific time and place to meet weekly. This would have given us as a group the ability to look into everyone’s progress, make sure we are all in agreement as to the prioritization of tasks, and then delegate work accordingly. We worked best when we were communicating, breaking large tasks into smaller pieces and when everyone had a defined role.

Another challenge for us was the exposure to new technologies and frameworks, such as Gradle, React, and MySQL. Learning them well enough in isolation in order to complete a task was challenging, but then trying to get everything to work together with other components proved much more difficult.

Patterns

**Observer Pattern with Invites**

We used the observer patter between the Invites and the model controllers. When an invite is accepted, it notifies all its ovservers. One of those is the model which will create a game based off of it. We needed this because invites only interact with users, so this is how we get the game controller to be able to create a game.

**Façade Controller between UI and The Server Controllers (RESTful API)**

The RESTful API lets the client control the server remotely. This has to happen because it can’t interact with the Java controller directly. It also has significant security benefits.

**Façade Controller between The Server Controllers and Model**

This façade controller lets the model of the system easy to use. Using the components of the model would be difficult, so using this, the RESTful API can link directly to different parts of the model without knowing how it wokrs internally. This makes it easy to refactor the system as well as redesign components wthout having to change the API.

**Polymorphism**

We used Polymorphism when we designed the pieces. This is so that testing for valid moves is very easy since the implementation to that is put into the classes directly. It is more organized and easier to maintain.