We did our code review on GitHub as part of a pull request. We created a new branch and reverted it back to the very first initial commit. We then created a pull request between that branch and the original master branch, everyone went and look through the changes and commented on things that they thought that should be improved. For the most part, there wasn't anything huge that we found. This is because before we started coding, we all agreed on a class structure to follow for the main parts of our code. This meant that before all of our changed came together, we knew how every different class would interact. One big thing we found is that we were needing to make the same vector utilities. For example: we had three different iterations of some form of "get vector magnitude" function. Due to this, we decided to make a class that will be a supplemental class to the Vector2 class that comes with the SFML library. There were also large parts of the code that were completely undocumented, those were then properly documented. There were also a few other small things with keeping the code base consistent. Everyone was assigned the faults to their own contributions, since that made the most sense. As for the new features, I took the liberty of adding those since at the time I was ahead of schedule. The original idea was to halt development and to the code review, and merge it before we continue with additional features. The issue with this was that the time we had left for the project didn't allow for this. Looking at the code, we also decided that our GameManager class was too cluttered. Up until then, it was handling the base game logic, the collision system, and the UI. We decided the best decision was to split the GameManager classes up. For the UI elements, we originally wanted to have a separate class containing all of the UI elelemts. We ran into issues with this because of the scope of all the objects inside of GameManager. We would end up having to pass around almost everything inside of GameManager, which would have made the rendering of all of our game objects infinitely slower. This downfall lead us to a better option, which was just to move all of the UI elements to another cpp file. This made the clutter of the GameManager file significantly less, although the clutter of the GameManager class remained the same. We didn't see this as too significant though, as we didn't have a better solution. Finally, we moved the collision to the Entity class. This was my job since I originally wrote the Entity class. This was much more difficult to implement, but it was entirely worth doing. The problem was again with scope: in that to do a raycast from one Entity to another, you need to check if the ray collides with every Entity in the game. To do this, you need some kind of reference to all of the Entity objects you have in the game, and having every Entity store a bunch of references to the other Entity objects would very quickly clutter the stack. To solve this, we moved from having everything separated, to having on large vector of Entity objects stored in GameManager. A pointer to this vector was passed into all of the Entity objects, so it would have a small and managable reference to every other Entity object in the game.