We used the object oriented design paradigm for our project. Everything is held in classes, with several singleton classes, and everything operates on an object oriented structure. We wanted to branch out and use a different paradigm, but the language we used, the library we used, and the type of project we were doing all rely heavily on object oriented programming. Our group decided to try and make a gaming using C++. C++ pretty much only functions from object oriented coding, it doesn't handle any of the other paradigms anywhere near as well as object oriented. Additionally, the library we used to create the window and draw the sprites used an object oriented paradigm. Finally, video game logic is entirely designed from object oriented design. There is no legitemate reason to stray from that paradigm because of those. Most of the game logic was handled from one singleton class we dubbed "GameManager." From there we had every in game object as an "Entity." There were specific types of entities that were derived from the base class, but everything inherited the basic logic of an entity from that class. We originally wanted a large array of entities that we would eventually use for collision checking, but we found that would be a lot more complicated in practice. For example, if you press a movement key, you would need to search through the entity array to find the player each time. Additionally, it would get difficult searching through the array to find each type of entity each frame. We ended up having separate arrays for different types of entities. So we had a single player object, an enemy array, and a wall array.