We used state patterns in our design, for our game state. We had various game states within our game manager class that would keep track of if the game is playing, menu, gameOver, exiting. These various game states control the logic of our game in what events are called and ultimately what the client can see and do. We also used singleton design as we only have one true instance of the GameManager that every component reports back to. The GameManager enforces only having one instance. A third example of a design pattern was Composite design. We had Objects that broke into a tree-like structure. We have an overarching Entity class that branches off into child comments including Player, and Enemy. This design offers the potential in the future to expand similar sibling components including something we had already planned which was item entities that can be picked up. This also includes breaking the Player into smaller parts. The biggest pattern we used was prototype. We based everything off of one Entity class, and without needing to copy code from one spot to the other, we allowed our more unique classes to inherit those basic functions while still allowing for the more unique elements of those classes. The player class for example, it still needs health and basic movement, but it should also be able to dodge and counter attacks, as well as receive its movement directly from keyboard input. These things wouldn’t make sense to put in the main Entity class, but it also wouldn’t make sense to copy the basic movement logic for both Player and Entity.