**Public Member Functions Description**

**Actor**

For my Actor class, it was derived from GraphObject. I declared a doSomething function but I made it pure virtual because all actors had different types of doSomething. I had an isDead which determined whether an actor was dead or not. This function is the same for all actors so I put it in my base class. Similarly, I had a setDead function which made an actor dead. Finally, I had a getWorld which returned a pointer to StudentWorld which would be very useful for many of my implementations in this project.

**Star**

My star class only consisted of a virtual doSomething derived from my Actor class. I had to implement this doSomething because the doSomething in my Actor class was pure virtual.

**Projectile**

For this class, I had a virtual doSomething. Again, I had a separate implementation for this function because all actors had different doSomething functions. I also had a playerFired function which tested whether a projectile was fired by a player or alien. This would be very important for determining which actor the bullets were meant for.

**Bullet and Torpedo**

Both my bullet and torpedo classes were derived from my projectile class. I implemented all my functions using my projectile class so I only had constructors for my bullet and torpedos classes.

**Goodie**

My goodie class had a doSomething class because all actors required a separate implementation for doSomething. I also had a doSpecialAction function which I made pure virtual. I did this because all goodies had a different special action and required separate implementations.

**FreeShipGoodie**

For this class, I only had a constructor and a doSpecialAction function. I had to implement the doSpecialAction because I made that function a pure virtual function in my goodie class. This function gave the player an extra life and awarded 5000 points.

**EnergyGoodie**

Similar to FreeShipGoodie, this function only had a constructor and doSpecialAction function. This function restored the energy for the player and awarded 5000 points.

**TorpedoGoodie**

Like the other goodie classes, this class had a doSpecialAction function. This function gave the player 5 torpedos in addition to 5000 points.

**Ship**

This class consisted of a host of trivial functions that would be useful when I had to implement subclasses. The functions I declared here were getEnergy, getEnergyPct, decreaseEnergy, and restoreFullEnergy. I also declared a pure virtual damage function that will be useful when several actors need to inflict damage on themselves for getting hit.

**Player**

For this class, I defined a doSomething for player which implemented tasks such a key input and tracked whether the player collided with an enemy. I also had a damage function which inflicted damage on the player if it was hit with a projectile or enemy. This function decreased life and let the player know that it was hit. I also included 2 trivial functions classed getNumTorpedoes and addTorpedoes that were only for the player class because I needed to keep track of the torpedoes the player had.

**Alien**

In this class, I declared a virtual damage function that inflicted damage for aliens. Unlike players, aliens were killed instantly when hit by a player so I needed a separate damage function for this class. I also had a maybeDropGoodie class which determined what type of goodie an alien would drop if killed. Different aliens drop different types of goodies so I had to make this function virtual.

**NachlingBase**

Nachlings and WealthyNachlings were super similar so I had a base class for them. I defined a doSomething function for this class which would be used for both Nachlings and WealthyNachlings. This function would act as artificial intelligence for the Nachlings.

**Nachling**

All functions needed for nachlings were already implemented in derived classes so I only had a constructor for this class.

**WealthyNachling**

For this class, I had a virtual doSomething because the WealthyNachling behaved slightly differently than a normal nachling. However, I still made use of its base classes’ doSomething. I also had a virtual maybeDropGoodie function that determined if and what type of goodie the WealthyNachling would drop if it was dead.

**Smallbot**

Smallbots weren’t that similar to the Nachling classes so I had virtual functions of doSomething, damage, and maybeDropGoodie for them. The damage function was required because the smallbot needed to know if it was hit so it could dodge another bullet. The maybeDropGoodie determined if and what goodie a smallbot would drop if it was killed. I also had a hit function that determined whether the smallbot was hit with a projectile or not.

**Bugs**

I was able to fully implement all functionality for this game.

**Design Decisions**

One decision I had to make for this project was how I would make some Actors not do something on certain ticks. For actors would could only do things on every other tick, I just had a variable that would incremented as each tick passed. The actor would do something if the variable was even and would return if the variable was odd. For the WealthyNachlings who could malfunction for 30 ticks, I just had a variable continuously increment until it reached 30 which was when the actor could do something again. I also decided to use a list to store all my pointer variables in my StudentWorld because I believed that making a list was the most efficient way add and delete items from the container.

**Testing**

**Star**

For my star class, I just made sure that each star would disappear when it reached the bottom of the page. Originally, my stars would delete too early so I had to fix that.

**Projectile**

For both bullets and torpedoes, I had to make sure that the correct image popped up when I shot each projectile. I also tested the damage each projectile made to the enemy. Torpedoes would kill the enemy much faster than a bullet. Finally, I made sure that bullets would disappear if it hit an enemy or if it left the page.

**FreeShipGoodie**

For this goodie, I made sure that Smallbots would sometimes drop these when they died. I made sure that the goodie would progressively dim as it came down and eventually disappear. Finally, I made sure that I actually got an extra life when I got this goodie.

**EnergyGoodie**

For this goodie, I made sure that WealthyNachlings would sometimes drop these when they died. I made sure that the goodie would progressively dim as it came down and eventually disappear. Finally, I made sure that my energy was actually restored when I got this goodie.

**TorpedoGoodie**

For this goodie, I made sure that WealthyNachlings would sometimes drop these when they died. I made sure that the goodie would progressively dim as it came down and eventually disappear. Finally, I made sure that I received 5 torpedoes when I got this goodie.

**Player**

For the player, I made sure that I could move anywhere I wanted on the field. I tested that I could shoot both bullets and torpedos. I tested the damage when I was hit with both projectiles and enemies and made sure that the round would restart everytime I died and that I also lost a life.

**Nachling**

For Nachlings, I made sure that they would periodically shoot bullets at me and that they would die and disappear if I killed them. I also tested that they would disappear and that the player would lose life if I ran into them.

**WealthyNachling**

I tested the WealthyNachling the same way as I tested a Nachling except that I also made sure that they would periodically drop goodies whenever I killed them.

**SmallBot**

Similar to WealthyNachlings, I had to make sure that they would periodically drop goodies when I kill them. I also tested that they would try to dodge my bullets when I started to hit them.