**The Class Hierarchy (Follow to see what’s derived from what)**

* Actor
  + Ship
    - NachenBlaster
    - Alien
      * Alien Base
        + Smallgon
        + Smoregon
      * Snagglegon
  + Bullet
    - TurnipBullet
    - CabbageBullet
    - FlatulanTorpedo
  + Goodie
    - TorpedoGoodie
    - RepairGoodie
    - LifeGoodie
  + Star
  + Explosion

**High Level Description of Public Member Functions**

:(All Classes have Non-Virtual Constructors and Virtual Destructors)

* ***class Actor : public GraphObject***
  + **bool isStillAlive()**
    - This boolean member function returns true/false to show whether the given actor is alive or not alive. Alive was represented using true, and not alive was represented using false. Not set to virtual because alive/not alive is a very “basic” facet of an Actor; it is universal to all Actors.
  + **bool setAlive()**
    - This member function allows the user to set the given Actor to alive or not alive. It changes the value of the private member variable to true or false respectively. Not set to virtual because alive/not alive is a very “basic” facet of an Actor; it is universal to all Actors.
  + **StudentWorld\* getWorld();**
    - Often times, an actor class need to call functions from studentWorld and this allows them to do that by returning a pointer to the world. It is not set to virtual because using functions from the studentWorld is a very “basic” facet of an Actor; it is universal to all Actors.
* ***class Ship : public Actor***
  + **int getHealth() const;**
    - Returned the value of the private member variable that contained the health value. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **double getHealthPct() const;**
    - Returned the value of the private member variable that contained the health value, in percentage form. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **int getCabbage() const;**
    - Returned the value of the private member variable that contained the number of cabbages. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **int addCabbage();**
    - Allows private member variable that contained the number of cabbages to be incremented to a greater value. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **int setCabbage(int i);**
    - Allows private member variable that contained the number of cabbages to be set to any value. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **double getCabbagePct() const;**
    - Returned the value of the private member variable that contained the number of cabbages, in percentage form. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **void decreaseHealth(int amount);**
    - Allows private member variable that contained the value of health to be decremented. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **void** **restoreFullHealth**();
    - Allows private member variable that contained the value of health to be set to its maximum value. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the Ship class.
  + **bool fire(int type, int firedBy);**
    - This member function is used to mark whether an instance of type Ship fired a shot. The inputs allow for the Type of shot (turnip, cabbage, torpedo) and who the shot was fired by ( 1=NachenBlaster, 2=Alien). This was not made virtual because the derived classes of the Ship type all use this method regardless, there is no specialization with this method.
* ***class NachenBlaster : public Ship***
  + **virtual void doSomething();**
    - This function allows an instance of a NachenBlaster to complete a set task in a single tick. In this function, a NachenBlaster is given the opportunity to (move, shoot, and etc…). I chose to define this as virtual because each type of actor completes doSomething() in a different way.
  + **int getNumTorpedos() const;**
    - Returned the value of the private member variable that contained the number of torpedoes. This was not made virtual because this member function was not a part of the Actor class, and the function is solely used in the NachenBlaster class.
  + **void addTorpedos(int n);**
    - Allows private member variable that contained the number of torpedos to be incremented to a greater value. This was not made virtual because this member function was not a part of the Ship class, and the function is solely used in the NachenBlaster class.
  + **void damage(int amount, bool hitByProjectile);**
    - This function is used to tell how much damage that the NachenBlaster should take, and whether it was hit by a projectile(true) or collided(false). This was not made virtual because this member function was not a part of the Ship class, and the function is solely used in the NachenBlaster class.
  + **void checkForCollisionsWithAliens(NachenBlaster\* p);**
    - This function tests for cases in which there was a collision between an Alien type with the NachenBlaster. This was not made virtual because this member function was not a part of the Ship class, and the function is solely used in the NachenBlaster class.
* ***class Alien : public Ship***
  + **virtual void damage(int points, bool hitByProjectile);**
    - This function is used to tell how much damage that the NachenBlaster should take, and whether it was hit by a projectile(true) or collided(false). This was made virtual because this member function is specialized in the Alien derived classes (Smallgon, Smoregon, Snagglegon).
  + **bool diedByCollision();**
    - This function tests for cases in which there was a collision between an Alien type with the NachenBlaster. This was not made virtual because this member function does not need to be made unique for other derived Alien classes. The function is universal to all Aliens.
  + **virtual void maybeDropGoodie() = 0;**
    - This function was made pure virtual because of the fact that each derived class of alien has its own unique method of dropping a goodie, and determining what type of goodie to drop.
  + **virtual void doSomething() = 0;**
    - This function was made pure virtual because of the fact that each derived class of Alien has its own unique method of moving, shooting, and etc...
  + **int getWorth();**
    - This member function return the value of the private member variable that holds the amount of points that a NachenBlaster will receive if a given Alien is destroyed.
* **class AlienBase : public Alien**
  + **virtual void doSomething();**
    - This function is used by some, but not all of the AlienBase derived classes. It shows basic movements, flight patterns, shooting patterns, and etc… This function was made virtual due to the fact that there is specialization of this function in future, derived, classes.
  + **void setCanAct(bool m);**
    - This function returns whether the Alien is allowed to make a move during a given tick. This was not made virtual because this member function does not need to be made unique for other derived AlienBase classes. The function is universal to all AlienBase types.
  + **void setdirY(int i);**
    - This function allows the private member variable that holds the movement speed in the Y direction to be changed. This was not made virtual because this member function does not need to be made unique for other derived AlienBase classes. The function is universal to all AlienBase types.
  + **void setdirX(int i);**
    - This function allows the private member variable that holds the movement speed in the X direction to be changed. This was not made virtual because this member function does not need to be made unique for other derived AlienBase classes. The function is universal to all AlienBase types.
* ***class Smallgon : public AlienBase***
  + **virtual void maybeDropGoodie();**
    - This function was made virtual because of the fact that each derived class of alien has its own unique method of dropping a goodie, and determining what type of goodie to drop.
* ***class Smoregon : public AlienBase***
  + **virtual void doSomething();**
    - This function is used by some, but not all of the AlienBase derived classes. It shows basic movements, flight patterns, shooting patterns, and etc… This function was made virtual due to the fact that there is specialization of this function in future, derived, classes.
  + **virtual void maybeDropGoodie();**
    - This function was made virtual because of the fact that each derived class of alien has its own unique method of dropping a goodie, and determining what type of goodie to drop.
* ***class Snagglegon : public Alien***
  + **virtual void doSomething();**
    - This function is used by some, but not all of the AlienBase derived classes. It shows basic movements, flight patterns, shooting patterns, and etc… This function was made virtual due to the fact that there is specialization of this function in future, derived, classes.
  + **virtual void maybeDropGoodie();**
    - This function was made virtual because of the fact that each derived class of alien has its own unique method of dropping a goodie, and determining what type of goodie to drop.
* ***class Bullet : public Actor***
  + **void doSomething();**
    - This function is very basic and tells a type of bullet what to do every tick, which is a simple movement in the correct direction. This was not made virtual because this member function does not need to be made unique for other derived Bullet classes. The function is universal to all Bullet types.
  + **int firedBy() const;**
    - This function returns the number, which represents who the respective bullet was fired by. 1=NachenBlaster, 2=Alien. This was not made virtual because this member function does not need to be made unique for other derived Bullet classes. The function is universal to all Bullet types.
  + **void checkForCollision(Bullet\* b);**
    - This function checks to see if the given bullet collides with an Alien or NachenBlaster. This was not made virtual because this member function does not need to be made unique for other derived Bullet classes. The function is universal to all Bullet types.

* ***class TurnipBullet : public Bullet***
  + NO UNIQUE MEMBER FUNCTIONS
* ***class CabbageBullet : public Bullet***
  + NO UNIQUE MEMBER FUNCTIONS
* ***class FlatulenceTorpedo : public Bullet***
  + NO UNIQUE MEMBER FUNCTIONS
* ***class Goodie : public Actor***
  + **virtual void doSomething();**
    - This function was made virtual because of the fact that each derived class of Goodie has its own unique method of dropping a goodie, and determining what type of goodie to drop.
  + **virtual void doSpecialAction(NachenBlaster\* p) = 0;**
    - This function was made pure virtual because of the fact that each derived class of Goodie has its own unique method of dropping a goodie, and determining what type of goodie to drop.
* ***class TorpedoGoodie : public Goodie***
  + **virtual void doSpecialAction(NachenBlaster\* p)**
    - This function has its own unique method of dropping a goodie, and determining what type of goodie to drop.
* ***class RepairGoodie : public Goodie***
  + **virtual void doSpecialAction(NachenBlaster\* p)**
    - This function has its own unique method of dropping a goodie, and determining what type of goodie to drop.
* ***class LifeGoodie : public Goodie***
  + **virtual void doSpecialAction(NachenBlaster\* p)**
    - This function has its own unique method of dropping a goodie, and determining what type of goodie to drop.
* ***class Star : public Actor***
  + **virtual void doSomething();**
    - This function allows the Star object to doSomething() every tick. Since doSomething, is specialized for a star, it is set as virtual since the Actor implementation of doSomething() is completely different.
* ***class Explosion : public Actor***
  + **virtual void doSomething();**
    - This function allows the Explosion object to doSomething() every tick. Since doSomething, is specialized for an explosion, it is set as virtual since the Actor implementation of doSomething() is completely different.

**List of Functionality That I Failed to Finish**

Unless I overlooked something minor in the project specification, I believe that all functionality is correctly implemented.

**Design Decisions and Assumptions You Made**

There is a graphic of sorts that outlines the hierarchy of my Classes, which are further broken down in part one in the member function descriptions. The majority of my design decisions can be seen through my member function descriptions and my implementations of those functions(in the above description). For the most part, I stuck to what the spec had us do.

**A description of how you tested each of your classes (1-2 paragraphs per class).**

* Actor
  + Ship
    - NachenBlaster
    - Alien
      * Alien Base
        + Smallgon
        + Smoregon
      * Snagglegon
  + Bullet
    - TurnipBullet
    - CabbageBullet
    - FlatulanTorpedo
  + Goodie
    - TorpedoGoodie
    - RepairGoodie
    - LifeGoodie
  + Star
  + Explosion

Above I have highlighted the classes that I ran tests for, these classes were chosen due to the fact that they are the “most derived” Class in their respective Class type; the final product in a sense.

**NachenBlaster.** For the NachenBlaster class I tested its functionality mainly by playing the game itself and making sure that the right behavior was being produced respective to given situations. For example, I ran into an issue that caused my cabbage bullets to shoot in the wrong direction(180 degrees backwards). I also ran into the issue of allowing the NachenBlaster infinite amounts of cabbages, which allowed me to plow through aliens with ease; this issue has since been fixed of course. I would have only been able to catch these minute mistakes by actually playing the game itself. Another thing that I made sure to test was the functionality of the keys. Were the keys responding like described in the spec? The only way to thoroughly test this was by playing the game. I would lastly, test for collisions by, of course, playing the game and purposely colliding with alien ships to make sure that damage, explosions, sounds, and alien deaths were according to the spec.

**Smallgon, Smoregon, Snagglegon.** The Smallgon, Smoregon, and Snagglegon classes were all mainly tested by observing their behaviors as they flew around, and as they shot their respective bullets. To test the shooting of the respective Alien types, (one-by-one) I made their positions stationary along the x and y axis, this allowed me to plot the NachenBlaster in the shooting range of the aliens. This way I made sure that the aliens were shooting the correct bullets at the right frequency/speed. I also tested the Smoregon’s rapid speed increase through a similar method, but instead of binding the Smoregon to a specific X and Y positions, I only bound the Y position. I also had a problem with the Aliens going out of bounds, which I observed from test, and I also made the observation that I was not properly destroying aliens that went out of bounds because the amount of aliens roaming kept decreasing after each wave.

**TurnipBullet, CabbageBullet, FlatulanTorpedo.** The TurnipBullet, CabbageBullet, and FlatulanTorpedo classes were mainly tested by me running each type of bullet through the NachenBlaster. By doing this I made sure that each bullet had the right speed, start distance, frequency, etc… In addition to this, I also made sure that bullets were causing the correct amount of damage by allowing the NAchen Blaster to be purposely hit, and observing how many hit points I was losing per collision with each type of bullet.

**TorpedoGoodie, RepairGoodie, LifeGoodie.** The TorpedoGoodie, RepairGoodie, and LifeGoodie were all tested by me spawning random goodies around the map at the start of a level so that I could test to make sure that the NachenBlaster’s attributes were correctly boosted. I did run into problems when it came to colliding with the goodies, because for some reason the game did not recognize that I was colliding with the goodie. I later realized that this was due to the fact that I had not implemented a check to see if the goodie had collided with the NachenBlaster. Lastly, I also spawned many, of one type of, a certain alien so that I could make sure that when I killed an alien the right types of goodies were being dropped, at the correct frequency. i also checked to see if the goodie was moving in the correct direction as well.

**Star.** Star was checked by me spawning many random amounts of stars in random places all over the map. I initially made it so that my move function only implemented stars. I would check to see that stars were correctly moving along the screen, and being destroyed once they reached the far left side of the screen. I also made sure to check that the sizing of the stars was correct; this was a big problem I ran into, because the randInt() function only output integers (not a range of 0.05-0.5) it was until way later into the project that I realized that I could setup my random number generator like such: randInt(5-50)/100.00 in the initializer list to get my random sizing.

**Explosions.** Explosions were checked by me randomly shooting all types of aliens, me colliding the NachenBlaster into other alien ships, and etc… I mainly checked to see that the size of the the explosion was changing over the correct amount of ticks. I originally had trouble implementing this, so I had static sized explosions that looked incorrect. In addition to that error, that I eventually fixed, I had trouble with implementing the explosions because I did this last, it was hard to find every location that explosions were necessary, but I eventually found them.