1. Actor:
   1. I made the doSomething function pure virtual because each of the actors has to doSomething during the tick, and the actor class is made abstract, since we don’t want an Actor object to be made.
   2. The isAlive function is not virtual because it returns a boolean value whether an actor is alive or not and has the same functionality for all the classes.
   3. The setAliveStatus is not virtual because each of the actors is either alive or dead and the actor class holds a variable to keep track of that. This way, you can only change the variable of the alive status of any type of actor through this function.
   4. The getStudentWorld function returns a pointer to the current student world in order to access its functions and returns the variable that the actor construct takes that is a pointer to a student world.
   5. The isHittable function is pure virtual because since the actor should be abstract and is not a hittable object, and every derived class needs to return whether it is hittable by a projectile or not.
   6. The isAlien function returns false because the majority of subclasses are not aliens, unless a subclass overrides this function and returns true, which should only be the aliens.
   7. The setDamaged function is called when a subclass of actor is damaged by an alien ship, nachenBlaster, or a projectile. I put this function in the actor class to access it in the student world class, which can only access actor functions and virtual functions derived from that function.
   8. The sufferDamage function is called when a subclass of actor needs to reduce its hit points due to an alien ship, nachenBlaster, or a projectile. I put this function in the actor class to access it in the student world class, which can only access actor functions and virtual functions derived from that function.
   9. The addPoints function is called when a subclass of actor needs to add its hit points due to rewards from goodies. I put this function in the actor class to access it in the student world class, which can only access actor functions and virtual functions derived from that function.
   10. The getHitPoints returns the number of hit points for the NachenBlaster for the display function in the student world class.
   11. The getCabbagePoints returns the number of cabbage points for the NachenBlaster for the display function in the student world class.
   12. The getTorpedoes returns the number of torpedoes the NachenBlaster has for the display function in the student world class.

NachenBlaster:

1. For the doSomething function:

If the object isn’t alive

Return

Else

Intialize an int value

If the user enters a key

Is its spacebar and cabbage points is greater than 5

Cabbage points is decremented by 5

The student world adds a cabbage

Play the SOUND\_PLAYER\_SHOOT sound

If its down and the nachenblaster is not at the edge of the screen

Move the nachenblaster down six pixels

If its up and the nachenblaster is not at the edge of the screen

Move the nachenblaster up six pixels

If its right and the nachenblaster is not at the edge of the screen

Move the nachenblaster right six pixels

If its left and the nachenblaster is not at the edge of the screen

Move the nachenblaster left six pixels

Is its tab and torpedoes is greater than 0

Torpedoes is decremented by 1

The student world adds a torpedo

Play the SOUND\_TORPEDO sound

Is cabbage points is less than 30

Increment cabbage points by 1

1. The isHittable function overrides the one in the Actor class and returns true because nachenBlaster’s can be hit.
2. The addPoints function overrides the one in the Actor class and adds the number of points in the parameter to nachenBlaster’s hit points.
3. The getHitPoints function overrides the one in the Actor class and returns the number of hit points the NachenBlaster currently has.
4. The addTorpedoes function overrides the one in the Actor class and adds the number of torpedoes in the parameter to nachenBlaster’s torpedoes variables.
5. The sufferDamage function overrides the one in the Actor class and decrements the number of points in the parameter to nachenBlaster’s hit points.
6. The function getCabbagePoints tells the student world object the number of cabbage points the nachenBlaster has.
7. The function getTorpedoes tells the student world object the number of torpedoes the nachenBlaster has.

Star:

* 1. The doSomething function moves a star object to the left by one every tick.
  2. The isAlive function overrides the function in the actor class and it returns false when the x coordinate of the star is less than 0
  3. The isHittable function overrides the actor class function so the star class is not abstract as well and returns false since stars can’t be hit.

Explosion:

1. The doSomething function just increases the explosion object by a factor of 1.5 for four ticks and then the explosion’s alive status is set to false.
2. The isHittable function overrides the actor class function so the explosion class is not abstract as well and returns false since explosions can’t be hit.

Projectile:

1. The isFlown function determines whether a projectile’s x coordinate is greater than the width of the screen or less than 0.
2. The doSomething function is not overrided because a projectile class should be abstract as well.
3. The isHittable function returns false, because all projectiles, cabbages, turnips, and torpedoes are not hittable.

Cabbage:

1. The doSomething function checks if the object is alive and if it is then if the object collides with an alien, then the cabbage is set to dead. Otherwise, the cabbage moves 8 pixels to the right and sees if it collides with an alien.

Turnip:

1. The doSomething function checks if the object is alive and if it is then if the object collides with the nachenBlaster, then the turnip is set to dead. Otherwise, the turnip moves 6 pixels to the left and sees if it collides with the nachenBlaster.

Flatulan Torpedo:

1. The doSomething function checks if the object is alive and if it belongs to the nachenblaster then if the object collides with an alien, then the torpedo is set to dead. If it belongs to the alien, it check for collision with the nachenBlaster. Otherwise, the turnip moves 8 pixels to the left if it belongs to the alien or to the right if it belongs to the nachenBlaster and checks collision again.

Alien:

1. The isDamaged function returns a boolean value if the alien object is damaged in a collision.
2. The setDamaged function changes the wasHit variable to the parameter.
3. The setHitPoints function changes the hit points variable to the parameter based on the type of alien.
4. The getHitPoints function returns the number of hit points the alien has remaining.
5. The setFlightLength function changes the flight length variable to the parameter based on the type of alien.
6. The getFlightLength function returns the flight length the alien has remaining.
7. The setFlightDirection function changes the flight direction variable to the parameter based on the type of alien.
8. The getFlightDirection function returns the direction the alien is currently traveling in.
9. The setTravelSpeed function changes the flight speed variable to the parameter based on the type of alien.
10. The getTravelSpeed function returns the flight speed the alien is currently traveling in.
11. The isFlown function returns whether the alien has flown off the screen.
12. The isAlien function overrides the one in the actor class and returns true because all of the subclasses will be aliens.
13. The isHittable function overrides the one in the actor class and returns true because all of the subclasses will be aliens and aliens are hittable.
14. The sufferDamage function decrements an alien’s hitpoints by the number in the parameter.
15. For the collision function:

If is damaged

If the hit points is less than or equal to 0

Increase the user’s points by the points in the parameter

Call the killed alien function in the student world class

Play sound SOUND\_DEATH

Add an explosion

Set drop goodie to true

Else

Play SOUND\_BLAST

If the alien collides with the nachenBlaster

Set the alien to dead

Call killed alien in the student world class

Increase the user’s points by the points in the parameter

Play sound SOUND\_DEATH

Add an explosion

Set drop goodie to true

Set damage to false

* 1. For the changingFlight function:

If the flight length is 0 or the alien is at the edge of the screen in the y-direction

If the ship is at the top of the screen

Set the flight direction to down and left

If the ship is at the bottom of the screen

Set the flight direction to up and left

If the flight length is 0

There is a 1/3 chance that the direction is changed to up and left

There is a 1/3 chance that the direction is changed to down and left

There is a 1/3 chance that the direction is changed to left

Set the flight length to a random number between 1 and 32

* 1. The moving function, movies the alien up and left by the travel speed if the direction is up, down and left by the travel speed if the direction is down and left, and left by the travel speed if the travel direction is left.

Smallgon

1. For the doSomething function:

If the alien is alive

Call collision with 250 hit points and 5 damage points

Call changing flight

If the nachenBlaster is to the left

Get a random number

If the number is 1

Add a turnip

Play sound ALIEN\_SOUND\_SHOOT

Return

If the alien is alive

Call moving

Call collision with 250 hit points and 5 damage points

Smoregon

1. For the doSomething function

If the alien is alive

Call collision with 250 points and 5 damage

If get drop goodie is true

Get a random number from 1 to 3

If the number is one

Get a random number from 1 to 2

If the number is 1

Add a repair goodie

Else

Add a torpedo goodie

Set drop goodie to false

Call changing flight

If the nachenblaster is to the left

Get a random number

If it is 1

Add a turnip

playSound SOUND\_ALIEN\_SHOOT

return

get another random number

if the random number is 1

set flight direction to left

set flight length to the width of the screen

set travel speed to 5

if the ship is alive

call moving

call collision with 250 and 5 as parameters

If get drop goodie is true

Get a random number from 1 to 3

If the number is one

Get a random number from 1 to 2

If the number is 1

Add a repair goodie

Else

Add a torpedo goodie

Snagglegon:

1. For the doSomething function:

If the ship is alive

Call collision with parameters 1000 and 15

If get drop goodie is true

Get a random number from 1 to 6

If the number is 1

Add a extra life goodie

Set drop goodie to false

If the alien is hitting the bottom or top of the screen

If at the top of the screen

Change flight direction to down and left

If at the bottom of the screen

Change flight direction to top and left

If the nachenBlaster is to the left

Get a random number

If the random number is 1

Add a torpedo

Play sound (SOUND\_TORPEDO)

Return

Move the alien up and left by its travel speed if direction is up/left

Move the alien down and left by its travel speed if direction us down/left

Call collision with parameters 1000 and 15

If get drop goodie is true

Get a random number from 1 to 6

If the number is 1

Add a extra life goodie

Set drop goodie to false

Goodie:

1. The isHittable function returns false because all the subclasses of Goodies are not hittable.
2. The isFlown function is not virtual because all of the goodies have the same implemention for the it, which sets the object to dead if they fly off the screen
3. The collides function sees whether the goodie collides with the nachenblaster, and if it does then the player increases its score, the goodie is set to dead, the sound goodie is played and true is returned.

ExtraLife Goodie:

1. The doSomething function checks to see if the goodie is alive, and if it collides with the nachenBlaster, it increments the player’s life. If it does not collide, the goodie moves and checks again whether it collides.

RepairLifeGoodie:

* + - * 1. The doSomething function checks to see if the goodie is alive, and if it collides with the nachenBlaster, it increases the player’s health. If it does not collide, the goodie moves and checks again whether it collides.

TorpedoGoodie:

1. The doSomething function checks to see if the goodie is alive, and if it collides with the nachenBlaster, the player gets 5 torpedoes. If it does not collide, the goodie moves and checks again whether it collides.

Student World: All of these functions are not virtual because I knew I was not going to make a derived class from this class, and these functions had no reason to be overridden.

1. For the init function:

aliensDestroyed variable is set to 0

the aliens remaining variable is set to a number that increases as the level increases

initialize the maximum number of aliens on the screen at a time variable

repeatedly to 30…

get a random depth

make that depth into a double

create a star with that depth and add it to the vector of actor pointers

create the nachenblaster and put it into a separate data structure

continue the game

1. For the move function:

Reinitialize the current aliens to 0

Repeatedly to the size of the vector…

If the actor is alive

If the actor is an alien

Increment current aliens

Make the actor do something

If the nachenblaster is alive

Make the nachenblaster do something

Else

Decrement the number of lives

Return the player dies

If the number of aliens remaining to kill is 0

Play the sound when you finish a level

Return the level is completed

Repeatedly to the size of the vector…

If the actor is not alive

Delete the actor and its pointer

Initialize a double, smaller

If the number of aliensRemaining is less that maxAliens

Smaller is set to aliens remaining

Else

Smaller is set to maxAliens

If the current aliens is less than smaller

Initialize int s1 to 60

Initialize int s2

Initialize int s3

s equals all the sum of s1, s2, and s3

with the probability s1/s

add a new smallgon to the vector

with the probability s2/s

add a new smoregon to the vector

with the probability s3/s

add a new snagglegon to the vector

with the probability of 1/15

add a new star to the vector

call the display to update game stats

continue the game

1. The cleanup function gets rid of all of the actors in the vector and the nachenblaster at the beginning of each level and the end of the game.
2. The destructor of the student world class just calls the cleanup function because the cleanup function deletes all the remaining dynamically allocated actors.
3. The display function updates the game statistics every tick and allows for the player to see it at the top of the screen.
4. The addCabbage function adds a cabbage to the vector of actor pointers every time the user presses the spacebar key and deletes this object if it hits a hittable thing or it flies off the screen.
5. The addExplosion function adds an explosion to the vector of actor pointers everytime an alien dies, and the explosion is added at the x and y coordinates of the alien that just died. The explosion dies after four ticks.
6. The addTurnip function adds a turnip to the vector of actor pointers every time the user presses the tab key and if the user has torpedoes and deletes this object if it hits a hittable thing or it flies off the screen.
7. The addExtraLife Goodie function adds a extra life goodie if the alien dies and drops a goodie at the place it dies.
8. The addRepair Goodie function adds a repair goodie if the alien dies and drops a goodie at the place it dies.
9. The addTorpedo Goodie function adds a torpedo goodie if the alien dies and drops a goodie at the place it dies.
10. The pushTorpedo function adds a torpedo in terms of the nachenblaster because this torpedo is added to the vector of actors because the user had a least 1 torpedo in its inventory and the user pressed the tab key.
11. The pushTorpedo2 functions adds a torpedo in terms of the alien ship because this torpedo belongs to the snagglegon and is supposed to hit the nachenblaster and not any other alien ship.
12. The isEuclidian function checks to see if a cabbage hit an alien ship, and if it does than it sets the isDamaged function in the alien class to true, so the alien knows its been damaged and so it can do its corresponding behaviors. The cabbage dies after it hits something.
13. The isEuclidian2 function checks to see if something hits the nachenblaster, and if it does than it calls the sufferDamage function with the parameter in the function because each projectile or ship causes different damage to the nachenblaster.
14. The isEuclidian3 function checks to see if the nachenblaster collides with a goodie and if it does, then there is some benefit added based on the parameter, because each type of goodie gives a different kind of benefit to the nachenblaster or user.
15. The totheLeft function checks to see if the nachenblaster is to the left of the alien and either 4 pixels up or down in the y direction, so the alien ship can possibly shoot a turnip or torpedo at it.
16. The killedAlien function is called every time the user kills an alien either using projectiles or running into it. This function decrements the variable that keeps track of how many aliens are remaining for the user to kill to beat the level.

1. There are not any major classes that I did not implement. However, for the display function, instead of using the setw() function, I just added two spaces after each of the labelled values, which made it mostly the same as the sample game, but it might be slightly different.
2. There were not any major assumptions I made since the project spec was really detailed and outlined mostly everything we were supposed to do. However, there were some assumptions I made. One assumption I made was to add all the alien ships in the move function. Also, I assumed that the player does not receive any points for beating the level. Also, I wasn’t sure of whether the torpedos the actor gets in one level carry on if the beat the level, so when I played to game I saw that they reinitialize to 0, so that is what I did in my game.
3. Actor:

In this project, testing things was one of the hardest things to do, since there was a lot of probability and some classes are abstract. I tested the actor class mostly through the other classes. Since mostly all the other classes were subclasses of the actor class, there were a lot of functions that were derived from the actor class, and I tested the actor functions through the testing of the other classes. It is also hard to test since this is an abstract class and you cannot instantiate it.

NachenBlaster:

The NachenBlaster is a really important component of this project, since this is the one that the user controls. Because the user controls it, it made it a lot easier to test this class as compared to the other classes. I tested this class by first seeing whether it is initialized and it shows up on the screen. After that, I tested the ability to move the NachenBlaster whether it can go up, down, left and right with those corresponding keys, and that is does nothing when the other keys are pressed. Then I tested that it doesn’t move if the NachenBlaster is at the edge of the screen. When I added the cabbages and torpedoes, I made sure those work when I press the spacebar and tab keys.

Star:

The Star class was also easier to test, since the stars do not do much. This was one of the very first classes I implemented. The way I tested this class was by checking whether they show up on the screen when I initialized them. Then I tested if they could move and if they all disappear. Also I made sure that they all are different sizes. Then I tested that they could move by seeing if the stars move slowly to the left.

Alien:

The alien class is another base class from which other classes are derived. The way I tested this class and all of its functions is through the testing of its derived classes, the smallgon class, the smoregon class, and the snaggelgon class, since all of this classes functions are inherited by these derived classes.

Smallgon:

The smallgon class is one of the three alien classes and was a little hard to test since there were a lot of random and small details that needed to be checked. First, I checked whether a smallgon would be initialized correctly and did not really look at the probability stuff. After they were being initialized correctly, I added the collision part and the move part for the aliens and the sounds made it easier to test whether the alien was colliding with a cabbage or the NachenBlaster or not. After that, I added the turnip and set the probability to a hundred percent to check whether those would initialize and then added the probability parts to the code.

Smoregon:

The smoregon class was really similar to that of the smallgon class, and since there were a lot of common functions that were written in the common base class of alien, when I tested the smallgon class, I was simultaneously testing the functions for the smoregon class as well. Also, I would check for the initialization and the movement of the smoregon in the similar fashion as I did for the smallgon. However, there were some differenced between the two classes. For example, in the smoregon class, there are goodies dropped sometimes when the smoregon is destroyed. To check whether the goodie was being initialized properly, I would set the probability to 100 percent.

Snagglegon:

Like the smoregon class, the snagglegon class is similar to the smallgon class and even more related to the smoregon class. There are some differences, like how the other two classes have flight lengths, but the snagglegon does not. I checked if the changing flight direction works by seeing whether the alien ship changes its flight direction when the ship hits either the top or botton of the game screen. Also, you can tell if the collision works by seeing whether there is an explosion when a nachenblaster and a ship collide or there is a sound when the ship is hit by a cabbage of torpedo.

Projectiles:

I tested the projectile class through the implementations of its derived classes, the cabbage class, the torpedo class, and the turnip class. Like the actor class and the alien class, the projectile class is also abstract and is not actually instantiated. This class just is a layout for all the other projectiles.

Cabbages:

I tested this class through playing the game and pressing the spacebar. If I press the spacebar and a cabbage comes out then they are being instantiated correctly. If there is a sound that comes when a cabbage collides with an alien ship then the colliding part of the function works correctly as well. Also, I tested continuously pressing the spacebar and saw that there was a continuous string of cabbages coming out with a small break and start again.

Turnips:

This class was a little hard to test since the alien ships threw the turnips, which was based on chance. To test this class easier, I made the probability that the alien would throw a turnip really high. If there was a sound when the nachenBlaster got hit and the health went down, the colliding part of the function worked correctly.

Flatulence Torpedo:

I tested this class in a combination of the way I tested the cabbages and the turnips, since the torpedo is a combination of those two projectiles in a way. It can either belong to the user or be thrown by an alien. If it I press tab and the torpedo collides with an alien and there is a sound it works. If the alien releases the torpedo and the NachenBlaster gets hit and the hitpoints decrease with a sound, the torpedo works both ways.

Explosions:

This class was very minimal and was very easy to test since there was only one component. When a alien ship dies or an alien ship collides with the NachenBlaster, there should be the image of an explosion that lasts for four ticks. This showed that explosions are initialized correctly and they grow for four seconds, after which they go away when they are deleted.

Goodies:

This is another one the base classes that I tested through the implementations of its derived classes, which includes the ExtraLife Goodie class, the Repair Life Goodie class, and the Torpedo Goodie class. Goodies are basically initialized, sometimes when alien ships die and they dispeear if they float off the screen or collide with the NachenBlaster.

ExtraLife Goodie:

I tested this class by making the probability that this goodie is added super high. This goodie is brought into the game with the death of a snagglegon, so when these die, there was a high probability that it drops the extra life goodie. I would check the functionality of this class by seeing that if the nachenBlaster collides with the extra life goodie, then the number of lives are incremented by 1 and nothing happens if it just floats down the screen.

RepairLife Goodie:

Since all the goodies are similar and their functionalities are similar, I tested the repair life goodies class in a similar fashion to the extra life goodie. Every time the smoregon dropped a repair life goodie, I checked when the goodie collided with the nachenblaster that the health of the nachenblaster would increase, and if it just floats off the screen, nothing happens or is affected.

Torpedo Goodie:

For the torpedo goodie, I had to see if the variables in the nachenblaster would change correctly and whether the torpedo goodie would correctly convert to 5 torpedoes for the nachenblaster and he user to utilize. When the torpedo goodie collided with the NachenBlaster, I check if the torpedoes in the nachenblaster was updated by 5 using the display. Then I checked whether five torpedoes would come out if I were to press the tab key five times. I also made the probability of a smoregon dropping a torpedo really high so it would be easier to test.

StudentWorld:

For the StudentWorld class, I mostly tested things through playing the game. At the beginning , I tested whether the initialization function was working if 30 stars showed up on the screen when I pushed thirty star objects onto the STL structure. For the movie function, I tested that I was able to move the nachenblaster at all ticks, and that every object in the vector did something at every tick. Also, for the number of alien ships that had to be added, I checked that the max aliens should be four for the first level. Also it is a little hard to check for memory leaks but its necessary every time dynamic allocation is involved. I tested this through the g32 project code that was provided in the project 3 spec.