The following will be split up into sections that contain the class title (bolded and underlined) followed by its member functions (bolded) and relevant descriptions for each function (their functionality and purpose) and why I chose them to be virtual/pure/not virtual

**Actor Class:**

**Actor():**

The default constructor for the actor class,

**Actor(int imageID, int sX, int sY, int sDir, double size, int depth):**

The non-default constructor for the actor class, creates a new actor with the set parameters, indicating where to place the actor on the screen at the x,y, and depth (z) positions. The actor picture and the size to scale that picture.

**virtual ~Actor()**

The destructor for the actor class, deconstructs the current actor. It is made virtual because it is common practice to make the destructor for you class virtual to allow child classes to define their own destructor if they need to.

**virtual bool isAlive()**

The isAlive function in the actor class, returns the actor’s alive state. If the actor is alive it will not be removed from the screen, and in the case of the natchenBlaster it will not end the game. This function was not created virtually because we need to check the alive states of all actors, not just of specific actors.

**void hasDied()**

The hasDied function in the actor class simply sets the alive state for the current actor to false so that it can be removed properly. It was not created virtually because all actors must die so that they can be properly removed from the screen.

**int getImageID()**

The getImageID function returns the matching ID for the image to determine during collisions and other relevant functions what was collided with.

**virtual void doSomething() = 0**

The doSomething function is very important to the game and gives each actor a change to do something (in the case of a star, move. In the case of the nachenBlaster, move or shoot, etc..) each class that inherits actor MUST do something so therefore it is defined to be pure virtual.

**Explosion Class:**

**Explosion(int sX, int sY)**

Create a new explosion that extends the actor class and create at the x and y location a new explosion.

**virtual void doSomething()**

Explosion’s doSomething function increases a counter for the explosion and then after 4 ticks of increasing the size by a scale of 1.5 it will set itself to false. This class is virtual due to the explanation in the actor class.

**Alien Class:**

**Alien(int starID, int sX, int sY, int sDir, double size, int depth, int hp, StudentWorld\* sWorld)**

The alien class creates a new alien object (nBlaster, all enemies) at the specified location with the specified hp and studentworld pointer.

**int getHP()**

This function returns the HP of the current alien. This function is intilzied because all aliens have an hp but other objects (projectiles, goodies, explosions, etc..) have no health so it makes sense that the HP will be defined in the Alien class. It is also not virtual because all aliens don’t need a special way to define their HP and it can be set easily.

**void setHP(int hp)**

The set HP function allows the user to set the HP to whatever HP they need. This was used for testing and to ensure that when the NachenBlaster (which inherits the alien class) doesn’t exceed 50 hp. This class was not created virtually because it was intended to be used by all subclasses in the same manner.

**void sufferDamage(int damageAmt)**

The sufferDamage functions does exactly what it sounds like, it makes all the actors suffer the specified amount of damage. If the Alien that suffered damage drops its hp to 0 (or below) then it does the correct functions (play dead sounds, increase the score, notify the studentWorld). This function was not created virtually because it determines the actions to take one each type of class inside of the function itself. So it will perform a different action depending on the type of the class that it is getting called on.

**StudentWorld\* getWorld()**

The get world function returns the studentWorld for the aliens. This function was declared here because all aliens they need to access the studentWorld. This was not declared virtually because it will return the same thing regardless of the subclass.

**Enemies Class:**

**Enemies(int starID, int sX, int sY, int sDir, double size, int depth, int hp, double speed, int points, StudentWorld\* sWorld, int tDir, int flightPath):**

The enemies constructor creates an enemy with the specified alien qualities of location and image, as well as the direction, points, direction, and flightpath. These all help give information on what to do for the specified classes.

**void setTravelDir(int dir):**

This function sets the direction for the enemy. This sets which way the alien will be moving. This was declared here because every enemy must move in some direction so there is no reason that an enemy would not have a direction. This was set non-virtually because there is no reason that each enemy should return the direction differently.

**int getTravelDir():**

This returns the flight direction so that it can be determined how to have this alien move, up-left, down-left, left. This function was declared here because every alien has some form of a flight dir. This was not declared virtually because there is no need to make every ship return the flight dir differently.

**void setFlightPath(int dist):**

This sets the flight path which determines how many ticks the enemy will move in the specified direction for. Setting this will change how many ticks the enemy will move in its current direction. This function was declared here because every alien has some form of a flight path. This was not declared virtually because there is no need to make every ship return the flight path differently.

**void flewOneUnit():**

This indicates that the enemy has flown a tick and to lower the flight path distance by one. This function was declared here because every alien must decrease the flight path after one tick. This was not declared virtually because there is no need to make every ship decrease the flight path differently.

**int getFlightPath():**

This function returns the flight path of the enemy. To see if the enemy has run out of flight to determine a new direction. This function was declared here because every alien must return a flight path. This was not declared virtually because there is no need to make every ship return the flight path differently.

**void setSpeed(int speed):**

The setSpeed function sets the speed for the enemy so that they can move a certian number of pixels per tick. This was defined in the enemies class because every enemy needs a certain number of pixels to move. This was also not defined virtually because there is no reason that specific aliens need to change speed differently.

**int getSpeed():**

The getSpeed simply returns the current speed of the enemy. This was defined here because every alien must be able to return their speed for moving during the game. This was also not defined virtually because every enemy must have the same way of returning the speed of itself.

**WorldExtender Class:**

**WorldExtender(int imageID, int sX, int sY, int sDir, double size, int depth, StudentWorld\* sWorld):**

This WorldExtender class creates a way for certain classes to access the student world by creating an actor (basically) and a final parameter for the studentWorld. This takes the usual parameters and creates a new actor using the relevant information asked for by the actor and taking a pointer to studentWorld.

**StudentWorld\* getWorld():**

The getWorld function returns the studentWorld that the current actor points to. This allows for interaction with the studentWorld. This was made here because it is required to have all objects that need to interact with the studentWorld. This was not made virtually as well because they do not need to individually define how the return the studentWorld.

**NachenBlaster Class:**

**NachenBlaster():**

The default constructor created a new nachenBlaster at the intial location with a size of 1 and a nullpointer to the studentWorld.

**NachenBlaster(int starID, int sX, int sY, int sDir, double size, int depth, int hp, int cabPwr, StudentWorld\* sWorld):**

The non-default constructor created a nachenBlaster at the specified location, with the specified power of cabbages and hp with the studentWorld set to its current studentWorld.

**virtual ~NachenBlaster():**

The desctructor is declared virtually and in the nachenBlaster class by convention.

**virtual void doSomething():**

The nachenblaster’s do something allows the user to choose a direction to move with the arrow keys, or they can fire a cabbage or torpedo (if available). This function is declared here because it will be called using a pointer to nachenBlaster to determine what it should do.

**int getCabPower():**

This function in natchenBlaster returns the current power for the cabbage (out of 30). This was declared here because there is no other logical location to have the natchenblaster return the number of cabbages that it has.

**void setCabPower(int pwr):**

This function sets the current cabbage power, (up to 30) if the user needs to add power to the cabbage it can do that here. This was declared here because there is no other logical location to add to the cabbage power of the natchenBlaster. This is not virtual because no class extends the natchenBlaster or would need to redefine this function.

**void incTorpedo():**

This function increases the number of torpedos that the natchenBlaster class has by 5 (indicatin that it hit a torpedo goodie. This is declared here so that you can change the variable in the natchenBlaster class to set the number or torpedos it holds. This is not virtual because no class extends the natchenBlaster or would need to redefine this function.

**int getTorpedo():**

This function returns the current number of torpedos that the class has. This is declared here because the variable that holds the number of torpedos is in this class. This is not virtual because no class extends the natchenBlaster or would need to redefine this function.

**Star Class:**

**Star(int starID, int sX, int sY, int sDir, double size, int depth):**

The star class creates a new star at a random x and y location (which are specified somewhere else) with the starID and a direction of 0 size of whatever is specified (randomly) and the depth which is set at 3. No need to describe why the constructor is made here and why it is not virtual.

**virtual void doSomething():**

The star’s do something function simply checks if it has left the screen if it hasn’t then it moves the star a set amount of distance to the left. This function is defined in the star class so that each star has something that it can do. This function is virtual because it extends the doSomething function in the actor class.

**Note:** The following classes all contain constructors that extend the enemies class so will not be explained here, please consult the “Enemies Class” section of this document to see how the variables are used and more information.

**Smallgon Class:**

**virtual void doSomething():**

The do something function for the Smallgon gives it a chance to move, attack with turnips, check for collision (with nachenBlaster and cabbages), check boundries (moved out of the screen to the left) and make the correct sounds. This is declared here because this function needs to doSomething every time the game ticks so it has been defined here.

**Smoregon Class:**

**virtual void doSomething():**

The do something function for the Smoregon gives it a chance to move, attack with turnips, check for collision (with nachenBlaster and cabbages), check boundries (moved out of the screen to the left) and make the correct sounds. This is declared here because this function needs to doSomething every time the game ticks so it has been defined here.

**Snagglegon Class:**

**virtual void doSomething():**

The do something function for the Snagglegon gives it a chance to move, attack with torpedos, check for collision (with nachenBlaster and cabbages), check boundries (moved out of the screen to the left) and make the correct sounds. This is declared here because this function needs to doSomething every time the game ticks so it has been defined here.

**Note:** The following classes all contain constructors that extend the projectiles class so will not be explained here, please consult the “WorldExtender Class” section of this document to see how the variables are used and more information.

**Cabbage Class:**

**virtual void doSomething():**

In the cabbage’s doSomething function it checks if it has collided with an enemy, if it has then it deals with giving damage appropriately. If not it moves and rotates itself by 20 degrees. It also checks if it has left the screen to mark itself as dead. The functions are defined here because it is where the cabbage will doSomething during a tick. So the function will be called here when any pointer to a cabbage is called. This function is virtual because it extends the doSomething from the actor class.

**Turnip Class:**

**virtual void doSomething():**

In the turnip’s doSomething function it checks if it has collided with the nachenBlaster, if it has then it deals with giving damage appropriately. If not it moves and rotates itself by 20 degrees. It also checks if it has left the screen to mark itself as dead. The functions are defined here because it is where the turnip will doSomething during a tick. So the function will be called here when any pointer to a turnip is called. This function is virtual because it extends the doSomething from the actor class.

**FlatulanceTorpedo Class:**

**virtual void doSomething():**

In the torpedo’s doSomething function it checks if it has collided with the nachenBlaster or an enemy using a boolean determining who it was fired from, it then it deals with giving damage appropriately. It checks if it has left the screen to mark itself as dead. The functions are defined here because it is where the torpedo will doSomething during a tick. So the function will be called here when any pointer to a turnip is called. This function is virtual because it extends the doSomething from the actor class.

**ExtraLifeGoodie Class:**

**virtual void doSomething():**

The extra-life goodie’s doSomething function determines if it has collided with the nachenBlaster to give an extra life, if not it moves and checks for collision again. This function is defined here because it needs to determine for each ExtraLifeGoodie what it does during that tick. This function is virtual because it extends the doSomething from the actor class.

**RepairGoodie Class:**

**virtual void doSomething():**

The repair goodie’s doSomething function determines if it has collided with the nachenBlaster to give a +10hp bumb, if not it moves and checks for collision again. This function is defined here because it needs to determine for each RepairGoodie what it does during that tick. This function is virtual because it extends the doSomething from the actor class.

**TorpedoGoodie Class:**

**virtual void doSomething():**

The torpedo goodie’s doSomething function determines if it has collided with the nachenBlaster to give a +5 torpedo bumb, if not it moves and checks for collision again. This function is defined here because it needs to determine for each TorpedoGoodie what it does during that tick. This function is virtual because it extends the doSomething from the actor class.

**StudentWorld Class:**

**virtual ~StudentWorld():**

destructor for the studentWorld class deletes all the pointers that it creates. Virtual by convention.

**virtual int init():**

The init function creates the intial screen for the level by setting the nachenBlaster (new with health and power, etc..) and the stars, and setting the number of aliens on the screen and the number of aliens that have been killed to 0. This is defined in this class because the studentWorld gets the init function called to create a new level. This is virtual because it extends a pure virtual function in the gameWorld class

**virtual int move():**

This function is executed every time the game ticks. This gives all objects in the game a change to perform their doSomething function. It also checks if the nachenBlaster has died to call a new game or exit. It checks if the level has been completed, sets the text at the top of the game. This also adds another alien and star if the conditions are met to add them. This function is defined here so that it can be called by the rest of the program during a tick. This is also a virtual function because it needs to extend from the gameWorld class.

**virtual void cleanUp():**

The cleanUp function removes all the pointers to actors and resets the relevant information so that the game/level can start. This is defined here so it can be called by the destructor. This is virtual function because it extends the pure virtual function in the studentWorld class.

**Actor\* collidedWith(Actor\* a):**

The collidedWith function determines if the actor passed in collided with any other actor. If it did then it will return the actor that it collided with so that the program can deal with that information appropriately. This function is declared here because it needs to go through the actorList which only this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

**bool withinShootingDistance(Actor\* shooter):**

This withinShootingDistance passes in an actor (an enemy) and determines if the nachenBlaster is withing the range that it needs to be to get shot. This function is declared here because it needs to go through the actorList which only this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

**bool addAnotherAlien():**

This function determines if there should be another alien added to the screen depending on the function that was given in the spec. This function is declared here because it needs to go through the number of aliens on the screen and the number of aliens killed which only this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

**void gotHealth():**

This function gives the nachenBlaster a certain amount of heath for running into the health goodie. This function was declared here so that it could add health to the nachenBlaster pointer which this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

**void alienKilled():**

This function is declared here because it needs to go through the number of aliens on the screen and the number of aliens killed which only this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

**void alienLeftScreen():**

This function changes the variable that determines how many aliens are displayed on the screen. It says that there is now one more alien on the screen. This function is declared here because it needs to go through the number of aliens on the screen which only this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

**void deleteDeadActors():**

This function goes through the actorList and deletes all the actors that have their states set to false. This function is declared here because it needs to go through the actorList which only this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

**void addToActorList(Actor\* a):**

This function adds an actor to the list. So it can add enemies and stars depending on whatever it needs. The actor list is all actors in the game except for the nachenBlaster (which has it own pointer) This function is declared here because it needs to go through the actorList which only this class has access to. This is not virtual because no other class that extends studentWorld needs to change the definition of this function.

Question 2: I finished everything

Question 3: It was not specified if I should finish the level or say it is game over, if I collided with the last alien with <5 health left. I decided that because all the aliens were dead for the level that it would make sense that you finish the level.

Question 4:

**StudentWorld Class:**

The studentWorld class was tested by simply seeing if the functions inside behaved properly wen it they were called upon from other classes. If the class could add things to the list I would print out the list and saw that because things were being added to the actor list I decided it was working.

**Actor Class:**

The actor class was easily tested by deremining if other classes could extend it and call the functions it needed. Since things were dying and getting deleted I decided that it was working properly.

**Alien Class:**

The alien class was determined to be funcitoning properly because I could lower the HP of the aliens including the nachenBlaster and the enemies. Since this property was working it was to be decided that the entire class was fine.

**NachenBlaster Class:**

The nachenBlaster class was decided to be working because I was able to use the buttons to make it move and shoot. Because this was working and it was taking damage and the other functions were also working I decided it was fine.

**Explosion Class:**

The explosion class was tested easily by making an explosion. I saw that the explosion was getting bigger by the correct scaling amount and going away after so I knew it wasn’t a class make by the original source code.

**Enemies Class:**

The enemies class was tested by making sure that the travel direction was getting set properly and that everything was flying correctly and moving in their defined direction. I initially printed out the value of each alien to see if that alien was moving in the direction thatI set.

**Projectiles Class:**

The projectile class was tested by seeing if the studentWorld was getting returned for the classes that I needed to get it.

**Goodies Class:**

The projectile class was tested by seeing if the studentWorld was getting returned for the classes that I needed to get it.

**Smallgon Class:**

The Smallgon class was tested by seeing if it would shoot cabbages and if the correct movement was performed for itself. This was easy to see and I determined if it was shooting correctly by printing the random values and seeing if they were indeed the correct values to see if they should be getting printed that way.

**Smoregon Class:**

The Smoregon class was tested by seeing if it would shoot cabbages and if the correct movement was performed for itself. This was easy to see and I determined if it was shooting correctly by printing the random values and seeing if they were indeed the correct values to see if they should be getting printed that way.

**Snagglegon Class:**

The Snagglegon class was tested by seeing if it would shoot cabbages and if the correct movement was performed for itself. This was easy to see and I determined if it was shooting correctly by printing the random values and seeing if they were indeed the correct values to see if they should be getting printed that way.

**Star Class:**

The star class was determined to be working by printing out a statement whenever a star was deleted. I knew that the stars must have been dying and therefore it must have been working correctly, especially because I could that the stars were moving on the screen.

**Cabbage Class:**

I determined that the cabbage class was working because it was clearly shooting cabbages that were rotating correctly and causing damage to the things that were coming in contact with. This was then decided that the class because it was doing what it was supposed to.

**Turnip Class:**

The turnip class was decided to be working because the turnips were causing damage to the nachenBlaster and it was clear that they were moving and rotating so I decided it was fine.

**FlatulanceTorpedo Class:**

The FlatulanceTorpedo class was decided to be working fine because it was causing the correct amount of damage and it was not hurting the same team (enemy to enemy) noticing these things had me conclude that it was all fine.

**Extra Life Goodie Class:**

The extra life goodie class was working fine because I notice that it was moving in the correct direction and I printed the odds to see if it should drop and it was all correct. The collision was fine and I saw that it was all working correctly because I received an extra life.

**Repair Goodie Class:**

The repair goodie class was working fine because I notice that it was moving in the correct direction and I printed the odds to see if it should drop and it was all correct. The collision was fine and I saw that it was all working correctly because I received 10 extra hp (not greater than 50 total hp).

**Torpedo Goodie Class:**

The torpedo goodie class was working fine because I notice that it was moving in the correct direction and I printed the odds to see if it should drop and it was all correct. The collision was fine and I saw that it was all working correctly because I received 5 more torpedos.