1. **Description of functions**

**STUDENTWORLD:**

1. **void addActor(Actor\* act);**This function adds a new Actor to the vector of existing actors.
2. **int shipsOnScreen();**

This function returns the number of alien ships that are currently on the screen. This function is used to calculate whether or not a new alien ship has to be introduced.

1. **void shipDead();**This function reduces private member variable m\_current, which represents how many ships are on the screen.
2. **int destroyedShips();**This function returns the number of dead ships for the current level. This is used to calculate whether or not a new alien ship has to be introduced.
3. **void killShip();**This function increases the count of dead ships. It is called whenever an alien ship is destroyed from either a projectile fired from the NachenBlaster, or the NachenBlaster itself.
4. **bool isNearNB(double ax, double ay);**This function returns true if the NachenBlaster’s y-coordinate is within [-4, 4] coordinates of the alien’s y-coordinate. This function is used to decide if the alien should fire a turnip at the NachenBlaster.
5. **double distance(double x1, double y1, double x2, double y2);**This function returns the distance between two actors. This is used to see if a collision will happen.
6. **bool collisionNB (double x, double y, double radius);**This function returns true if the distance between the NachenBlaster and the second Actor is less than 0.75 multiplied by (NachenBlaster’s radius + other Actor’s radius). This is the function that detects whether there is a collision or not.
7. **void NBloseHits(int hits);**This function reduces the NachenBlaster’s Hit Points by the integer value passed into the function.
8. **void NBgainHits(int hits);**This function increases the NachenBlaster’s Hit Points by the integer value passed into the function.
9. **void NBgainTorpedoes(int torpedoes);**

This function increases the NachenBlaster’s Torpedo Projectiles by the integer value passed into the function.

1. **Actor\* collideWithAlien(Actor\* act);**This function iterates through the vector of Actors until the iterator is on either a Smallgon, Smoregon, or Snagglegon. Then, the x-coordinate, y-coordinate, and radius of that alien is stored. Then, the distance() function is used to see if there is a collision between the alien and the other actor (either a cabbage or torpedo projectile). If there is a collision, this function returns a pointer to that alien. Otherwise, it returns a nullptr.
2. **bool canActorMoveTo (Actor\* act, double x, double y);**This function returns true if the coordinate the actor wants to move to is within the screen coordinates.
3. **void setDisplayText();**This function gets the information for the status bar at the top of the screen. It gets the lives, score, and level from GameWorld.h, and the health, cabbages, and torpedoes from functions in the NachenBlaster class.
4. **string formatDisplayText (int lives, int health, int score, int level, int cabbages, int torpedoes);**This function formats the text at the top of the screen.
5. **int init();**This function adds stars to the screen at random positions and adds the NachenBlaster.
6. **int move();**This function updates the stats at the top of the screen, adds new stars as old stars disappear, adds a new alien ship if needed, tells each actor to do something, and deletes dead actors and actors that have flown off the screen.
7. **void cleanup();**This function goes through the vector of Actors and deletes each Actor, including the NachenBlaster.

**ACTOR:**

1. **virtual void doSomething() = 0;**This function allows each actor to execute its own actions. It is declared pure virtual because each actor has its own unique actions.
2. **StudentWorld\* getStudentWorld() const;**This function returns a StudentWorld pointer.
3. **void moveToIfCan (double x, double y);**This function uses the given moveTo() function to move the Actor to its specified x- and y-coordinates, as long as the canActorMoveTo() function from StudentWorld.cpp returns true.
4. I also added the functions **virtual void setAlive(), virtual void setDead(), virtual bool isAlive(),** and **virtual int getID().** These are self-explanatory given the names of the functions.

**STAR:**

1. **virtual void doSomething();**This function moves the star one pixel to the left. It is virtual because each Actor has its own unique actions for doSomething().
2. **virtual bool isAlive()** is self-explanatory.

**EXPLOSION:**

1. **virtual void doSomething();**This function increases the size of the explosion by a factor of 1.5 for four ticks, and then sets its state to dead. It is virtual because each Actor has its own unique actions for doSomething().
2. **virtual bool isAlive()** and **virtual bool isDead()** are self-explanatory.

**NACHENBLASTER:**

1. **virtual void doSomething();**This function first checks if the NachenBlaster is alive, or if its hit points have reached 0. If then moves the NachenBlaster, fires a cabbage, or fires a torpedo, depending on the user input. It is virtual because each Actor has its own unique actions for doSomething().
2. **int getHealth();**This function returns the percentage of Hit Points the NachenBlaster has left. This is done by multiplying the amount by 2 since the maximum number of Hit Points the NachenBlaster can have is 50.
3. **int getCabbagePercent();**This function returns the percentage of Cabbages the NachenBlaster has left. This is done by multiplying the amount by (10/3) since the maximum number of Cabbages the NachenBlaster can have is 30.
4. **int getTorpedoInventory();**This function returns the number of torpedo projectiles the NachenBlaster has.
5. I also added **void loseHitPoints(int hits), void addHitPoints(int hits), void addTorpedoes(int torpedoes), void addCabbages(), int getHealth(),** and **virtual bool isAlive(),** which are self-explanatory given the name of the function.

**ALIEN:**

1. **virtual void doSomething() = 0;**This function allows each alien actor to do something. It is pure virtual because each Alien has its own unique actions for doSomething().
2. **void sufferDamage(int hits);**This function reduces the alien’s hit points by the amount passed into the function. If the hit points are 0 or below, whenAlienDies is called. Otherwise, the SOUND\_BLAST sound is played.
3. **virtual void possiblyDropGoodie() = 0;**This function allows the aliens to drop goodies when they are dead. This is pure virtual because each alien drops goodies in its own way.
4. **virtual void whenAlienDies() = 0;**This function is called when the alien dies and does the appropriate methods, such as playing sound or increasing score. This is pure virtual because each alien does something different when it dies.
5. **void moveDownAndLeft(), void moveUpAndLeft(),** and **void moveDueLeft()** set private member variable m\_travelDirection to 1, 2, or 3 respectively. **void travelInDirection()** then moves the alien in the correct direction, depending on m\_travelDirection.
6. **virtual int getID()** is self-explanatory.

**SMALLGON:**

1. **virtual void doSomething();**This function checks if the alien is alive or if it has flown off the screen. It then checks for a collision with the NachenBlaster. If there is a collision, the NachenBlaster loses 5 points, and the whenAlienDies() function is called. Then, this function sees if the flight plan is 0, or if the y-coordinate of the alien is at the edge of the screen. The travel direction is changed and the flightplan is assigned to a random integer. Then, the alien fires a Turnip if the isNearNB() function from StudentWorld.cpp returns true. Then the alien moves in its current travel direction and reduces its flight plan. It is virtual because each Alien has its own unique actions for doSomething().
2. **virtual void whenAlienDies();**

This function increases the score by 250, sets its state to dead, adds an explosion, and plays a sound. This is virtual because the aliens do different things when they die.

1. **virtual void possiblyDropGoodie();**

This function does nothing because Smallgons do not drop goodies. It is declared virtual because each alien does something different in this function.

1. **virtual bool isAlive(), virtual int flightPlan(),** and **virtual int travelSpeed()** are self-explanatory given their names.

**SMOREGON:**

1. **virtual void doSomething();**First, this function checks if the Smoregon is alive or if it has flown off the screen. It then checks for a collision with the NachenBlaster and calls the whenAlienDies() function if there is a collision. If the flight plan is 0 or if the y-coordinate is too low or too high, the travel direction is changed, and then the flight plan is assigned to a random integer. The function then checks if it should shoot a turnip or if it should change its flight plan and speed, using the isNearNB() function from StudentWorld.cpp. The alien then moves in its current travel direction and reduces its flight plan by 1. It is virtual because each Alien has its own unique actions for doSomething().
2. **virtual void possiblyDropGoodie();**There is a 1/3 chance that the Smoregon will drop a goodie. There is a 50% chance that this goodie will be a Repair Goodie and a 50% chance that it will be a Torpedo Goodie. This is virtual because each alien does a different thing in this function.
3. **virtual void whenAlienDies();**This function increases the score by 250, sets the state to dead, adds a new explosion, and calls the possiblyDropGoodie() function. This is virtual because each alien has a different method of dying.
4. **virtual bool isAlive(), virtual int flightPlan(),** and **virtual int travelSpeed()** are self-explanatory given their names.

**SNAGGLEGON:**

1. **virtual void doSomething();**This function first checks if the alien is alive or if it has flown off the screen. It then checks for a collision with the NachenBlaster and calls the whenAlienDies() function if so. Then, it checks if the y-coordinate is too low or too high and changes its travel direction accordingly. If the isNearNB() function returns true, the Snagglegon fires a Torpedo at the NachenBlaster. It then moves in its current travel direction. It is virtual because each Alien has its own unique actions for doSomething().
2. **virtual void possiblyDropGoodie();**

There is a 1/6 chance that this function will add an Extra Life Goodie. This is virtual because the aliens do different things when they drop the goodies.

1. **virtual void whenAlienDies();**This function adds 1000 points to the score, sets its state to dead, adds an explosion, and calls the possiblyDropGoodie() function. This is virtual because each alien does something different when it dies.
2. **virtual bool isAlive(), virtual int flightPlan(),** and **virtual int travelSpeed()** are self-explanatory given their names.

**CABBAGE:**

1. **virtual void doSomething();**This function checks if the cabbage is alive and in the correct coordinate range. It then checks if there is a collision with an alien using the collideWithAlien() function from StudentWorld.cpp. If there is a collision, it tells the alien that it needs to suffer from the damage, and it sets state to dead. Otherwise, it moves 8 pixels to the right and rotates in a counterclockwise direction. It is virtual because each Actor has its own unique actions for doSomething().
2. **virtual bool isAlive()** is self-explanatory.

**TURNIP:**

1. **virtual void doSomething();**The function first checks if the turnip is alive and if it has flown off the screen. It then checks if there is a collision with the NachenBlaster and acts accordingly. Otherwise, it moves 6 pixels to the left and rotates in a counterclockwise direction. It is virtual because each Actor has its own unique actions for doSomething().
2. **virtual bool isAlive()** is self-explanatory.

**TORPEDO:**

1. **virtual void doSomething();**The function checks if the torpedo is alive and if it has flown off the screen. It then determines whether a Snagglegon has fired the Torpedo or if the NachenBlaster has fired the torpedo, based on its direction. It then checks for a collision with its victim (the NachenBlaster or an alien ship) and acts accordingly. If there is no collision, it moves 8 pixels to the left or right, depending on which actor fired it. It is virtual because each Actor has its own unique actions for doSomething().
2. **virtual bool isAlive()** is self-explanatory.

**GOODIE:**

1. **virtual void doSomething();**This function checks if the goodie is alive or if it has flown off the screen. It then checks if there has been a collision with the NachenBlaster. If there is a collision, the score is increased by 100, the goodie’s state is set to dead, a sound is played, and it calls the grantSpecificReward() function. If there is no collision, the goodie moves 0.75 pixels to the left and 0.75 pixels down. It is virtual because each Actor has its own unique actions for doSomething(). This doSomething() function is used by all three goodies.
2. **virtual void grantSpecificReward() = 0;**This function grants the NachenBlaster with its reward. This is pure virtual because each goodie does something unique.
3. **virtual bool isAlive()** is self-explanatory.

**REPAIR GOODIE:**

1. **virtual void grantSpecificReward();**

This function gives the NachenBlaster 5 hit points. It is virtual because each goodie does something different.

**EXTRA LIFE GOODIE:**

1. **virtual void grantSpecificReward();**This function gives the NachenBlaster one extra life. It is virtual because each goodie does something different.

**TORPEDO GOODIE:**

1. **virtual void grantSpecificReward();**This function gives the NachenBlaster 5 torpedo projectiles. It is virtual because each goodie does something different.
2. **Known bugs:** There seemed to be some bugs when checking whether an actor had flown off the screen.

In Actor.cpp, lines 254, 353, and 457, I had to write if (getX() < 2) instead of if (getX() < 0). This is because if I used getX() < 0, the alien would stay on the screen and wouldn’t disappear even though it was on the left edge of the screen. I kept changing this number to see when the aliens disappear and the “magic number” is got was 2.

In lines 507 and 589, I had to write if (getX() >= VIEW\_WIDTH – 8) instead of if (getX() >= VIEW\_WIDTH) to check if a cabbage or torpedo had flown off the right side of the screen. When I used VIEW\_WIDTH, the projectile would stay on the right side of the screen, doing nothing. Once again, I played around with the numbers and found that if I used VIEW\_WIDTH – 8, the projectile would disappear once hitting the right edge.

In lines 540 and 573, I had to write if (getX() < 8) instead of if (getX() < 0) to check if the turnip or torpedo had flown off the left side of the screen. If I had kept getX() < 0, the projectile would stay on the left edge, doing nothing. It would only disappear if I had getX() < 8.

1. **Decisions and Assumptions**
   1. I decided that each actor would have a pointer to the StudentWorld object.
   2. If two sounds should be playing at the same time (i.e. NachenBlaster firing a cabbage and a Smallgon dying), only one sound will be played so that they don’t overlap.
   3. The type of variable for flight plan was not specified so I made it an int.
   4. I decided to store all my Actors in a vector.
   5. The spec did not specify how to handle collisions so I decided to do the following:
      1. The cabbage and torpedo (fired by NachenBlaster) class checks for collision with alien
      2. The alien class checks for collision with NachenBlaster
      3. The turnip class checks for collision with NachenBlaster
      4. The goodie class checks for collision with NachenBlaster
      5. The torpedo (fired by Snagglegon) class checks for collision with NachenBlaster
2. **Testing each class**

**StudentWorld**

Most of the functions in this class are used by other classes, so I tested those by making sure the other classes were correct as well. I tested that init() worked by changing the number of stars that appear on the screen to make sure that the correct number appeared. I checked that the move() function made each actor call their doSomething() function. I did this by writing cout statements to check the original position of each actor and the final position after doSomething() was called.

**Actor**

This class uses polymorphism. Since Actor is an Abstract Base Class, there was no way to

test the class. I made sure that this class worked by testing its derived classes.

**Star**

I tested this class by writing cout statement to see to position before doSomething() was called, and after. The stars moved one pixel to the left so I knew that this class worked.

**Explosion**

I tested this class by using the ‘freeze’ and ‘resume’ commands on my game. When an alien ship died, I froze the screen and went through 4 ticks to make sure that the explosion would increase by a factor of 1.5 for 4 ticks, and then disappear.

**NachenBlaster**

I tested this class by using the arrow keys, the WASD keys, and the number keys to make sure the NachenBlaster would move accordingly. I also tested that it would fire a cabbage or torpedo by pressing the space bar or tab key. I tested the NachenBlaster’s other functions (i.e. addCabbages(), addHitPoints(), etc.) by making sure the health increased when it was supposed to, and that the Cabbage Percentage always returned back to 100%.

**Alien**

Most of the Alien’s functions are pure virtual functions so I did not test those. I tested the travel direction functions by making sure that the aliens moved in the correct direction depending on its random integer.

**Smallgon**

I tested the Smallgon by making sure it began on the right side of the screen and then moved until it disappeared on the left side of the screen. I tested that the random integer works and that it would change its flight plan accordingly. I made sure that the score changed when the Smallgon died and that the NachenBlaster lost health when the Smallgon collided with it. I tested that the Smallgon would fire a turnip depending on a random integer using cout statements. I made sure that the Smallgon would disappear after hitting the left edge.

**Smoregon**

I tested the Smoregon class similar to how I tested the Smallgon. The only difference is that I additionally made sure that there was a 1/3 chance the alien would drop a goodie every time the Smoregon died. Half the time, the goodie would be a Repair Goodie and half the time it would be a Torpedo Goodie. I made sure that the Smoregon would disappear after hitting the left edge.

**Snagglegon**

I tested the Snagglegon class similar to how I tested the Smoregon. The only difference is that there would be a 1/6 change that the Snagglegon drops an Extra Life Goodie. I made sure that the Snagglegon would disappear after hitting the left edge.

**Cabbage**

I tested the cabbage class by continuously pressing the spacebar to see if cabbages would fire from the NachenBlaster. I then used the ‘freeze’ command to go through a tick at a time and make sure that it rotated counterclockwise. To confirm that it was rotating by 20 degrees, I wrote cout statements to check the degrees at each tick. I also used cout statements to make sure that the cabbage moved 8 pixels to the right every tick. I made sure that the cabbage would disappear when it either collided with an alien or when it hit the right edge.

**Turnip**

I tested that the Turnip class works by waiting until an alien fired a turnip. I checked that it would move 6 pixels to the left during each tick. I also checked that it rotated 20 degrees counterclockwise. I also watched the health to make sure that it decreased when it collides with the NachenBlaster. Both of these checks were done with cout statements. I made sure that the turnip would disappear whenever it hit the NachenBlaster or when it reached the left edge of the screen.

**Torpedo**

I tested the Torpedo class by waiting for a Snagglegon to fire a torpedo at the NachenBlaster. I made sure that the health would decrease if they collided and that the Torpedo would disappear. I also checked that the Torpedo would disappear if it reached the left side of the screen. If the NachenBlaster had torpedoes in its inventory, I would press the tab key and make sure the torpedo count decreased at the top of the screen. I checked that it would move towards the right and disappear if it either hit the right edge or collided with an alien.

**Goodie**

I checked the goodie class by making sure they moved down 0.75 pixels and left 0.75 pixels during each tick by using cout statements. I made sure that the correct reward would occur depending on which Goodie the NachenBlaster collided with. I watched that the goodie would disappear if it either collided with the NachenBlaster or if it hit the left/bottom edge of the screen.