**High-Level Description of Public Member Functions:**

**class StudentWorld: public GameWorld**

virtual int init();

Resets all member variables to their default values and adds a new NachenBlaster and 30 stars to a now empty Actors vector. It is virtual because it is a redefinition of the method in GameWorld and it is in StudentWorld because it has more specialized actions that need to be done.

virtual int move();

Runs all the actions that need to be done in a tick by calling all the Actors’ doSomething, checking to see if NachenBlaster is dead, and deleting dead Actors. It is virtual because it is a redefinition of the method in GameWorld and it is in StudentWorld because it has more specialized actions that need to be done.

virtual void cleanUp();

Frees the memory of all Actors and removes them from the Actors vector. It is virtual because it is a redefinition of the method in GameWorld and it is in StudentWorld because it has more specialized actions that need to be done.

void recordAlienDying();

Increments the number of destroyed aliens by 1. It is in StudentWorld because it has to keep track of how many Aliens die and it is not virtual because it is only needed for StudentWorld, not for its parent class.

bool isNachenInLineOfFire(const Actor \* a) const;

Returns true if player it is in the life of fire of a. It is in StudentWorld because it has access to the player and can check to see if it is in the line of fire. It is in StudentWorld because the method is only needed for StudentWorld, not for its parent class.

Alien \* getOneCollidingAlien(const Actor\* a) const;

Returns any alien that collides with a. It is in StudentWorld because it has access to all the Actors and can check to see if they collide with a. It is in StudentWorld because the method is only needed for StudentWorld, not for its parent class.

NachenBlaster \* getCollidingPlayer(const Actor\* a) const;

Returns true if player it collides with a. It is in StudentWorld because it has access to the player and can check to see if it collides with a. It is in StudentWorld because the method is only needed for StudentWorld, not for its parent class.

void addActor(Actor \* actor);

Adds the actor to Actors vector. It is in StudentWorld because only it has access to the Actors vector and it is not virtual because no other class has any use for this function, as they don’t have a vector of Actors.

~StudentWorld();

Calls the cleanup method to free used up memory and remove all Actors from the actors vector.

**class Actor: public GraphObject**

StudentWorld \* getWorld() const;

Returns StudentWorld the Actor is in. This is in Actor because all Actors must be able to get access to the StudentWorld they are in. It isn't virtual because all Actors get their World in the same way.

virtual void doSomething() = 0;

This is in Actor because all Actors need be able to doSomething at every tick. It is pure virtual because all Actors doSomething, but all in a different way.

virtual bool isAlien() const;

Returns false, as most Actors are not Aliens. This is in Actor because all Actors must be able to tell if they are an Alien or not. It is virtual because all Actors can be an alien or not, but it will be redefined for all Alien classes to return true.

void setState(bool state);

Sets alive state of Actor. It is in Actor because all Actors’ state needs to be able to be set. It is not virtual because all Actors’ states are set in the same way.

bool getState() const;

Gets alive state (true -> alive, false ->dead) of Actor. It is in Actor because all Actors’ states must be able to be get. It is not virtual because all Actors’ states are got in the same way.

**class Star: public Actor**

virtual void doSomething();

Does all the actions (moving) to be done in one tick for a Star. It is in Star because Star must be able to doSomething and it is pure virtual in Actor. It is virtual because it is a redefinition of the Actor’s method.

**class Explosion: public Actor**

virtual void doSomething();

Does all the actions (moving, dying after 4 ticks) to be done in one tick for an Explosion. It is in Explosion because Explosion must be able to doSomething and it is pure virtual in Actor. It is virtual because it is a redefinition of the Actor’s method.

**class FloatingObject: public Actor**

virtual void doSomething();

Does all the actions to be done in one tick for all FloatingObjects (Aliens, Goodies, and Projectiles), including calling doSpecialSomething(), which is different for different FloatingObjects. It is in FloatingObject because it is pure virtual in Actor and all FloatingObjects need to be able to do their special actions during every tick. It is virtual because it is a redefinition of the Actor’s method.

**class Character**

void incTorpedoCount(int num);

Increases torpedo count of Character. It is not virtual because all Characters’ torpedo counts are increased in the same way. It is in Character because it has access to the torpedo count and some derived classes of Character need to increment the count.

int getTorpedoCount() const;

Gets torpedo count of Character. It is not virtual because all Characters’ torpedo counts are got in the same way. It is in Character because it has access to the torpedo count and some derived classes of Character need to get the torpedo count.

void setAmmoCount(int count);

Sets ammo count for Character. It is not virtual because all Characters’ ammo counts are set in the same way. It is in Character because it has access to the ammo count and some derived classes of Character need to change the torpedo count.

int getAmmoCount() const;

Gets ammo count of Character. It is not virtual because all Characters’ ammo counts are got in the same way. It is in Character because it has access to the ammo count and some derived classes of Character need to get the torpedo count.

void setHP(int hP);

Sets health points for Character. It is not virtual because all Characters’ health points are set in the same way. It is in Character because it has access to the health points and some derived classes of Character need to change their health points.

int getHP() const;

Gets health points for Character. It is not virtual because all Characters’ health points are got in the same way. It is in Character because it has access to the health points and some derived classes of Character need to get their health points.

void fire(Projectile \* projectile, int type) const;

Fires projectile and plays sound based on the type of projectile. It is not virtual because all Characters’ fire projectiles in the same way and it is in Character because all derived classes of Character will need it.

virtual void damageCharacter(int damage);

Decreased health points by damage for Character. It is virtual because all Characters’ damages are taken care of differently. It is in Character because it has access to the health points and derived classes of Character need to change their health points.

**class NachenBlaster: public Actor, public Character**

virtual void doSomething();

Does all the actions to be done in one tick for NachenBlaster, including checking for key presses and doing their associated actions and incrementing cabbage count. It is in NachenBlaster because it is pure virtual in Actor and NachenBlaster need to be able to do its special actions during every tick. It is virtual because it is a redefinition of the Actor’s method.

int getAmmoPercentage();

Gets ammo percentage for NachenBlaster. It is not virtual and it is in NachenBlaster because only NachenBlaster needs this method to display the percentage on the status text.

int getHPPercentage();

Gets health point percentage for NachenBlaster. It is not virtual and it is in NachenBlaster because only NachenBlaster needs this method to display the percentage on the status text.

virtual void damageCharacter(int damage, int cause);

Damages NachenBlaster and takes care of possible death and sounds that need to be played based on the cause. It is in NachenBlaster because it has a more specialized definition than the one in Character and it is virtual because it is a redefinition.

**class Alien: public Character, public FloatingObject**

virtual bool isAlien() const;

Returns true because all Aliens are Aliens. It is in Alien and it is virtual because it is a redefinition of the method in Actor, as it now returns true instead of false.

virtual void damageCharacter(int damage, int cause);

Damages Alien and takes care of possible death and sounds that need to be played based on the cause. It is in NachenBlaster because it has a more specialized definition than the one in Character and it is virtual because it is a redefinition.

**Other Design Decisions and Assumptions:**

* It was not specified if the collision checks are to be done both ways (e.g. if an Alien checks if it collides with projectiles and if some projectile checks if it collides with an Alien), so I decided to only check the collisions once, as it sufficed to ensure all Actors’ collisions are dealt with.

**Description of Class Testing:**

**class StudentWorld: public GameWorld**

I checked this class by playing the game many times to ensure the stars and aliens are made as they should and that the correct number of kills were registered and the correct number of aliens are on the screen. I checked these numbers by updating them on the status text and checking if they were what the specs said they should be. I used the part 1 Sanity Check to make sure the destructor was working correctly and not leaving any dangling pointers. I also used the status text to see if init() properly reset the member variables by printing size of the actors vector and the various int variables.

**class Actor: public GraphObject**

Actor class was tested mostly using the sanity check from part 1 to see if the movement functions were accessed and working properly. As Actor is an abstract base class with very simple get and set methods, there is not much else to check.

**class Star: public Actor**

I modified StudentWorld to create a lot of Stars to check if they were created with the right coordinates and if they moved at the correct pace. I also checked to see if the random creation of Stars was working as properly by creating a Star with probability of 1 and 1/15 and comparing the difference.

**class Explosion: public Actor**

I killed an Alien in my game and in the sample game and then played it tick by tick after the Alien dies and Explosion is created. I compared the sizes of the ticks and when they died and made sure it was the same in both cases.

**class FloatingObject: public Actor**

I checked to see if the method class in doSomething were done accurately by changing the status text every time something is called. Other than that, there is nothing to check as FloatingObject is an Abstract Base Class.

**class Projectile: public FloatingObject**

I tested the death of projectiles after they collided and after they went off the screen by updating the status text. I also ensured that the collisions were detected properly by seeing if the HP of NachenBlaster was reduced when a Turnip hit it. I also checked the to see if the Projectiles moved and turned as they should by going tick by tick and noticing the differences between each tick.

**class Cabbage: public Projectile**

The Cabbage class is very simple and only sets changes the damage and movement in the Projectile class. Everything else is left to the Projectile class. I checked to see if when 3 Cabbage hit a Smallgon in the first level it dies.

**class Turnip: public Projectile**

The Turnip class is very simple and only sets changes the damage and movement in the Projectile class. Everything else is left to the Projectile class. I checked to see if when 1 Turnip hits the NachenBlaster its HP reduces by 4%.

**class FlatulenceTorpedo: public Projectile**

The FlatulenceTorpedo class is very simple and only sets changes the damage and movement in the Projectile class. Everything else is left to the Projectile class. I checked to see if its HP reduces by 16% when a FlatulenceTorpedo hit the NachenBlaster.

**class Character**

The Character class itself never appears in the game, but its derived classes Aliens and NachenBlaster. I made sure the set and get methods manipulated the right variables, but there is no other check for this. I made sure fire method worked by checking to see if the projectiles were added to the Game when SPACE is pressed and if the correct sound associated with the projectile is played.

**class NachenBlaster: public Actor, public Character**

I tested NachenBlaster by checking to see it can move in all 4 directions properly when the WASD or UP, DOWN, LEFT, RIGHT keys are pressed. Then, I checked to see if a cabbage was fire when SPACE is pressed and cabbage count is not 0 and the cabbage count was reduced appropriately. Then, I checked to see if a flatulence torpedo was fired when TAB was pressed and the count was greater than 0 and if the count was reduced by 1 when it is fired. I made sure that the projectiles started off in the right location and the correct sound was played when cabbage/torpedo was fired. I also checked to see if when the NachenBlaster’s HP goes to 0, the game ends by colliding with multiple Aliens until the player dies. I checked to see if its movement and firing were similar to the sample game by comparing them both.

**class Alien: public Character, public FloatingObject**

Alien handles most of the work needed to be done for the 3 different Aliens. I checked if damageCharacter worked properly by killing different Aliens various times to see if they died, the score was increased, and if the correct sound was played. I checked to see if the Aliens all checked for collisions with the NachenBlaster properly and died if they collided, while increasing the score. I checked to see if the movement of the Aliens were correct, as specified and did not go below or above the game world (y is >= 0 and <= MAX\_HEIGHT - 1).

**class Smallgon: public Alien**

I had nothing to check, as everything that a Smallgon does, the generic Alien does too so by checking the Alien class, I checked to see if Smallgon works as it should.

**class Smoregon: public Alien**

I checked to see if when the NachenBlaster is in front of the Smoregon it changes its flight direction directly left. I also made the probability of the goodie dropping to 100% to see if the Smoregon dropped either a Repair goodie or a Torpedo goodie every time it dies at the position where it dies. Then, I changed the probability to what it says in the specs and counted the number of times Smoregons drops a goodie and divided by the number of times a Smoregon dies and got around 1/3, as it should be. Everything else that the Smoregon does, the generic Alien does too so by checking the Alien class, I checked to see if the Smoregon class works as it should.

**class Snagglegon: public Alien**

I made the probability of the goodie dropping to 100% to see if the Snagglegon dropped a ExtraLife goodie every time it dies at the position where it dies. Then, I changed the probability to what it says in the specs and counted the number of times Snagglegons drops a goodie and divided by the number of times a Snagglegon dies and got around 1/6, as it should be. I also checked to see if the Snagglegon only moves diagonally left and does not go beyond the upper and lower boundary of the screen. I also checked to see the flight plan length that the Alien class has does not affect the flight plan of Snagglegon by checking to direction is only changed when the Snagglegon hits the bottom or top boundary of the screen. Everything else that the Smoregon does, the generic Alien does too so by checking the Alien class, I checked to see if the Smoregon class works as it should.

**class Goodie: public FloatingObject**

Goodie does most of the actions for all derived classes for Goodies other than the effect of the particular Goodie. I checked to see if all Goodies moved down and left with a constant speed during every tick. I also checked to see if the all the collisions of Goodies with a NachenBlaster are checked and the correct effects occur. I made the Aliens drop a Goodie every time they were created and checked to see if the score increased, the Goodie disappeared (as it would die and StudentWorld would remove it), and the sound was played every time the NachenBlaster collided with the Goodie.

**class ExtraLifeGoodie: public Goodie**

Most of the ExtraLifeGoodie’s actions are handled by its parent class, Goodie other than its special effect on the player (increase lives by one). I checked to see if every time the NachenBlaster collided with the ExtraLifeGoodie the lives displayed in the status text would increase by one.

**class RepairGoodie: public Goodie**

Most of the RepairGoodie’s actions are handled by its parent class, Goodie other than its special effect on the player (increase HP by 10). I checked to see if every time the NachenBlaster collided with the RepairGoodie, the HP displayed in the status text would increase by 20% (10 health points) to a maximum total 100% (50 health points).

**class FlatulenceTorpedoGoodie: public Goodie**

Most of the FlatulenceTorpedoGoodie’s actions are handled by its parent class, Goodie other than its special effect on the player (increase HP by 10). I checked to see if every time the NachenBlaster collided with the FlatulenceTorpedoGoodie, the torpedo count displayed in the status text would increase by 5.