**CS 32 Project 3 Report**

**Class Hierarchy:**

StudentWorld

(Not in Hierarchy)

**High Level Description of public member functions:**

Class Actor

1. Actor(int imageID, int startX, int startY, int startDirection, double size, int depth, StudentWorld\* curr\_world);  
   *Constructor for the Actor class. Initializes the private members of the class GraphObject by calling its constructor. Also initializes the values for the actor’s alive status and the StudentWorld it is currently in.*
2. virtual ~Actor();  
   *Virtual destructor for Actor class. It is Virtual because many classes are derived from the Actor class and it is good practice to make destructors virtual when inheritance and polymorphism are used. The absence of the keyword virtual could lead to undefined behavior.*
3. bool getStatus();  
   *Returns Actor’s life status (Dead or alive).*
4. bool setStatus(bool x);  
   *Sets Actor’s life status (Dead or alive).*
5. StudentWorld\* getWorld();  
   *Returns StudentWorld that the Actor exists in.*
6. virtual void doSomething()=0;  
   *Pure, virtual function as all Actors have a doSomething method that has a different implementation. An Actor object will never be created and we cannot say what an abstract Actor would do within the game’s framework.*
7. bool collide(Actor\* x);  
   *Returns whether the current Actor has collided with the Actor passed as an argument to the function. (Determined using the formula* provided in the project spec)
8. virtual bool hasFlightPlan();  
   *Returns whether the current Actor has a flight plan. This function helps in determining if the Actor is an Alien. It is virtual because most classes do not have a flight plan but certain classes do (and must override the method).*
9. virtual bool hasCabbages();  
   *Returns whether the current Actor has cabbages (projectile). This function helps in determining if the Actor is the NachenBlaster. It is virtual because every class except for the NachenBlaster class does not have cabbages. Thus, the NachenBlaster class must override this function.*
10. virtual bool spins();  
    *Returns whether the Actor spins. This function helps in determining if the Actor is a specific projectile (Cabbage or Turnip). It is virtual because most classes cannot spin but certain classes do (and must override the method)*
11. virtual bool playerProjectile();  
    *Returns whether the Actor is a projectile fired by the player. It is virtual because most classes are not player projectiles while certain classes are (and must override the method)*
12. virtual bool hasPathLength();  
    *Returns whether the Actor has a path length. This helps in determining the kind of Alien an actor is as Snagglegons do not have path lengths. It is virtual as most classes do not have path lengths while certain classes do (and must override the method)*
13. virtual void sufferDamage(Actor\* a);  
    *Causes the Actor to suffer damage. It is virtual as only certain kinds of Actors can suffer damage and do so in different ways (they must override this method)*

Class Star

1. Star(int startX, int startY, StudentWorld\* curr\_world);  
   *Calls the Actor Class’ constructor to initialize Actor’s data members to known values. Also initializes the Star’s starting coordinates and the Student World that it belongs to as per arguments.*
2. virtual ~Star();  
   *Destructor for the Star class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. void doSomething();  
   *Moves the Star 1 pixel to the left. Then if the Star has moved past the left end of the screen its status is set to dead.*

Class NachenBlaster

1. NachenBlaster(StudentWorld\* curr\_world);  
   *Calls the Actor class’ constructor to initialize the Actor’s data members to known values. Also initializes the Student World the NachenBlaster currently belongs to as per arguments.*
2. virtual ~NachenBlaster();  
   *Destructor for the NachenBlaster class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. void doSomething();  
   *If the NachenBlaster’s health is less than or equal to 0 set it’s life status to dead and decrease lives by 1. Don’t do anything else in the tick.  
   If the player hits a key,   
   If a directional key is hit, move the player 6 pixels in the corresponding direction   
   If the spacebar is hit, fire a cabbage starting 12 pixels to the right of the NachenBlaster and decrease cabbage energy by 5. Play appropriate sound.  
   If the tab key is hit and the player has torpedoes, fire a flatulence torpedo starting 12 pixels to the right of the NachenBlaster and decrease torpedoes by 1. Play appropriate noise.  
     
   Damage any alien the Nachenblaster may have collided with.  
   Increase cabbage energy by 1 if energy is less than 30.*
4. int getHealth();  
   *Return the NachenBlaster’s health.*
5. void setHealth(int x);  
   *Set the NachenBlaster’s health as per argument passed.*
6. int cabbageEnergy();  
   *Return the NachenBlaster’s cabbage energy.*
7. int torpedoes();  
   *Return the NachenBlaster’s number of torpedoes.*
8. void setTorpedoes(int x);  
   *Set the NachenBlaster’s number of torpedoes.*
9. void sufferDamage(Actor\* a);  
   *If the NachenBlaster has hit an alien with a path length (Smallgon or Smoregon) decrease its health by 5. Otherwise (if it hits a Snagglegon) decrease its health by 15.  
   If the NachenBlaster has hit a spinning projectile (Turnip) decrease its health by 2. Otherwise (if it hits a Flatulence Torpedo) decrease its health by 8.  
   Play appropriate sound.*
10. bool hasCabbages();  
    Return true.

Class Alien

1. Alien(int imageID, int startX, int startY, StudentWorld\* curr\_world, double health, int path, double speed);  
   *Calls the Actor class’ constructor to initialize the Actor’s data members to known values.* *Also initializes Alien’s starting coordinates, Student World it belongs to, health, path and speed as per arguments.*
2. virtual ~Alien();  
   *Destructor for the Alien class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. int getPath();  
   *Returns alien’s path length.*
4. void setPath(int x);  
   *Sets alien’s path length to x.*
5. std::string getDirection();  
   *Returns alien’s flight direction (Down and left, Up and left, or left).*
6. void setDirection(std::string);  
   *Sets alien’s flight direction as per argument.*
7. int getSpeed();  
   *Returns alien’s speed.*
8. void setSpeed(int x);  
   *Sets alien’s speed to x.*
9. void doSomething();  
   *Does stuff common to all aliens (Returns if alien is dead. Sets alien’s life status to dead if it goes past the left end of the screen. Also decreases aliens in the student world by 1 in this case and returns.)  
   Does stuff specific to each kind of alien. (Fire a projectile or dash)  
   Does more stuff common to all aliens (If path length becomes zero, change flight direction and set new path length. If alien reaches the top, change direction to down and left, if it reaches the bottom change direction to up and left. Move the alien in its specific direction as per its speed and decrease path length by 1.*
10. void sufferDamage(Actor\* a);  
    *The alien undergoes damage common to all aliens (Loses 2 hit points if hit by a cabbage and loses 8 hit points if hit by a player fired flatulence torpedo. The alien’s life status is set to dead if it hits the Nachenblaster)  
    If the alien’s health is dead do alien specific actions. (Update score, number of aliens remaining, add an explosion, drop goodies)  
    Play appropriate sound.*
11. bool hasFlightPlan();  
    *Return true.*
12. virtual bool hasPathLength();  
    *Return true. The function is virtual as not all aliens have a path length. Snagglegon’s override this method.*

Class Smallgon

1. Smallgon(int startX, int startY, StudentWorld\* curr\_world);  
   *Calls Alien’s constructor to initialize data members to known values. Also initializes the Smallgon’s starting coordinates and the Student World it belongs to.*
2. virtual ~Smallgon();  
   *Destructor for the Smallgon class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*

Class Smoregon

1. Smoregon(int startX, int StartY, StudentWorld\* curr\_world);  
   *Calls Alien’s constructor to initialize data members to known values. Also initializes the Smoregon’s starting coordinates and the Student World it belongs to.*
2. virtual ~Smoregon();  
   *Destructor for the Smoregon class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*

Class Snagglegon

1. Snagglegon(int startX, int StartY, StudentWorld\* curr\_world);  
   *Calls Alien’s constructor to initialize data members to known values. Also initializes the Snaggelgon’s starting coordinates and the Student World it belongs to.*
2. virtual ~Snagglegon();  
   *Destructor for the Snagglegon class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. bool hasPathLength();  
   *Return false.*

Class Projectile

1. Projectile(int imageID, int startX, int startY, int startDirection, StudentWorld\* curr\_world);  
   *Calls Actor’s constructor to initialize data members to known values. Also initializes the projectile’s image ID, starting coordinates, direction and the Student World it belongs to.*
2. virtual ~Projectile();  
   *Destructor for the Projectile class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. void doSomething();  
   *Do stuff common to all projectiles (If the projectile is dead, return without doing anything else. If the projectile goes past the left or right ends of the screen, set its life status to dead.)  
   Then do stuff specific to each type of projectile. (Damage other actors on collision, and move the projectile)*
4. virtual bool spins();  
   *Returns true. It is virtual as not all kinds of projectiles spin. The flatulence torpedo class must override this method.*
5. virtual bool playerProjectile();  
   *Returns true. It is virtual as not all kinds of projectiles belong to the player. The turnip class must override this method.*

Class Cabbage

1. Cabbage(int startX, int startY, StudentWorld\* curr\_world);  
   *Calls Projectile’s constructor to initialize data members to known values. Also initializes the cabbage’s starting coordinates and the Student World it belongs to.*
2. virtual ~Cabbage();  
   *Destructor for the Cabbage class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*

Class Turnip

1. Turnip(int startX, int startY, StudentWorld\* curr\_world);  
   *Calls Projectile’s constructor to initialize data members to known values. Also initializes the turnip’s starting coordinates and the Student World it belongs to.*
2. virtual ~Turnip();  
   *Destructor for the Turnip class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. bool playerProjectile();  
   *Returns false.*

Class FlatulenceTorpedo

1. FlatulenceTorpedo(int startX, int startY, int startDirection, StudentWorld\* curr\_world);  
   *Calls Projectile’s constructor to initialize data members to known values. Also initializes the torpedo’s starting coordinates, direction and the Student World it belongs to.*
2. virtual ~FlatulenceTorpedo();  
   *Destructor for the FlatulenceTorpedo class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. bool spins();  
   *Returns false.*

Class Explosion

1. Explosion(int startX,int startY,StudentWorld\* curr\_world);  
   *Calls Actor’s constructor to initialize data members to known values. Also initializes the explosion’s starting coordinates and the Student World it belongs to.*
2. virtual ~Explosion();  
   *Destructor for the Explosion class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. void doSomething();  
   *If the explosion has been alive for 4 ticks, set its life status to dead.   
   Increase the explosion’s size by 1.5 times its current size.  
   Decrease the number of ticks remaining (Explosion’s life span)*

Class Goodie

1. Goodie(int imageID, int startX, int startY, StudentWorld\* curr\_world);  
   *Calls Actor’s constructor to initialize data members to known values. Also initializes the goodie’s image ID, starting coordinates and the Student World it belongs to.*
2. virtual ~Goodie();  
   *Destructor for the Goodie class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*
3. virtual void doSomething();  
   *If the goodie’s status is dead return without doing anything else.  
   If the goodie has gone past the edges of the screen, set its status to dead.  
   If the player gets the goodie, increase the score, set the goodie’s status to dead, play a sound and give the player a bonus specific to the kind of goodie. Return without doing anything else.  
   Move the goodie.  
   If the player gets the goodie, increase the score, set the goodie’s status to dead, play a sound and give the player a bonus specific to the kind of goodie. Return without doing anything else.*

Class extraLifeGoodie

1. extraLifeGoodie(int startX, int startY, StudentWorld\* curr\_world);  
   *Calls Goodie’s constructor to initialize data members to known values. Also initializes the extra life goodie’s starting coordinates and the Student World it belongs to.*
2. virtual ~extraLifeGoodie();  
   *Destructor for the extraLifeGoodie class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*

Class repairGoodie

1. repairGoodie(int startX, int startY, StudentWorld\* curr\_world);  
   *Calls Goodie’s constructor to initialize data members to known values. Also initializes the repair goodie’s starting coordinates and the Student World it belongs to.*
2. virtual ~repairGoodie();  
   *Destructor for the repairGoodie class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*

Class flatulenceTorpedoGoodie

1. flatulenceTorpedoGoodie(int startX, int startY, StudentWorld\* curr\_world);  
   *Calls Goodie’s constructor to initialize data members to known values. Also initializes the flatulence Torpedo* *goodie’s starting coordinates and the Student World it belongs to.*
2. virtual ~flatulenceTorpedoGoodie();  
   *Destructor for the flatulenceTorpedoGoodie class. Virtual as it belongs in an inheritance hierarchy. Refer to explanation in Actor’s destructor.*

Class StudentWorld

1. StudentWorld(std::string assetDir);  
   *Calls the GameWorld constructor.*
2. ~StudentWorld();  
   *StudentWorld Destructor. Calls the cleanUp method.*
3. virtual int init();  
   *Initializes the values for number of aliens destroyed, number of aliens remaining in current level, max aliens on screen in current level and the number of aliens currently on the screen.   
   A new Nachenblaster object is created.   
   30 new stars are created and randomly placed on the screen.  
   It is virtual as it overrides a pure virtual method in the GameWorld class.*
4. virtual int move();  
   *Allows each actor (including the NachenBlaster) currently alive to do something during the tick.  
   Remove all the dead game objects.  
   Add a new star as per the probability given in the project spec.  
   Add a new alien according to the conditions and probability given in the project spec.  
   Create a StringStream with the values and formatting specified in the project spec.  
   Set the game text to this StringStream.  
   If the NachenBlaster is dead return the appropriate constant.  
   If there are no aliens left to kill in the current level return the appropriate constant.  
   Otherwise continue the game.  
   It is virtual as it overrides a pure virtual method in the GameWorld class.*
5. virtual void cleanUp();  
   *Delete the NachenBlaster object and set the pointer pointing to it to nullptr.  
   Delete all the actors in the game and set the pointers pointing to them to nullptr.  
   It is virtual as it overrides a pure virtual method in the GameWorld class.*
6. void addActor(Actor\* a);  
   *Add an actor to the game as per argument.*
7. void setAliens(int x);  
   *Set number of aliens to x.*
8. int getAliens();  
   *Return the number of aliens on the screen.*
9. int getPlayerX();  
   *Return the player’s x coordinate.*
10. int getPlayerY();  
    *Return the player’s y coordinate.*
11. void increasePlayerHealth(int x);  
    *Increase the player’s health by x with an upper bound of 50.*
12. void increaseTorpedoes(int x);  
    *Increase the player’s torpedoes by x.*
13. bool alienDamaged(Actor\* a);  
    *Check if the actor a has collided with an alien.   
    If it has, deal damage to that alien.  
    If the actor a is the NachenBlaster, cause it to take damage.  
    If the alien collided with dies, increase the number of destroyed aliens and decrease the number of remaining aliens.   
    Return true.  
    If the actor a has not collided with an alien return false.*
14. bool playerDamaged(Actor\* a);  
    *If the actor a has collided with the NachenBlaster cause the NachenBlaster to take damage and return true.  
    Otherwise return false.*
15. bool playerGetsGoodie(Actor\* a);  
    *Return whether the actor a collided with the NachenBlaster.*

**Incomplete functionality and known bugs:**None.

**Design decisions and assumptions made:**

1. The StudentWorld class has a separate pointer to the NachenBlaster and a vector of Actor pointers to all the other game objects.
2. Collisions are only checked for in the Projectile, NachenBlaster and Goodie classes by calling methods in the StudentWorld class.
3. The NachenBlaster and Alien classes have sufferDamage methods that deal damage to them based on its parameters.

**Description of how I tested each class:**

Class NachenBlaster

**Before implementation of Projectiles and Aliens**

I started the game and first checked to see if the NachenBlaster was in the right position. I then hit the directional keys and made sure the NachenBlaster could move in all 4 directions. I also checked to ensure the NachenBlaster did not go past the edges of the screen.

**After implementing Projectiles and Aliens**

I hit the spacebar to check if the NachenBlaster fired cabbages. I also checked the StringStream to ensure that the number of cabbages decreased with every shot. I allowed to NachenBlaster to be hit by turnips and flatulence torpedoes and checked the StringStream to ensure that the NachenBlaster’s health decreased with every hit.  
  
**After implementing Goodies**

I killed the appropriate alien and picked up the different kinds of goodies. I checked the StringStream to see that the appropriate effect had taken place.

After picking up a flatulence torpedo goodie I hit the tab key to check if the NachenBlaster fired torpedoes.

For minute details in every tick I pressed the ‘f’ key to progress tick by tick.

Class Star

I started the game and first checked if there were 30 stars randomly placed across the screen, each with a random size. I then checked if they all moved to the left and disappeared after passing the edge. I also ensured that new stars were being created at the right end of the screen.

Class Explosion

I killed the different aliens and checked if an explosion was being created at the right location. I ensured that the explosion’s size increased with each tick and that it disappeared after 4 ticks.

Class Cabbage

I fired cabbages and made sure that they rotated, moved to the right and went past the edge of the screen. I fired at aliens and verified that they died after the appropriate number of hits. I also ensured that they did not collide with other projectiles.

I also ensured that the projectile disappeared after any collisions.

Class Turnip

I ensured that Turnips were being fired by aliens and that they rotated, moved to the left and went past the edge of the screen. I allowed the NachenBlaster to get hit by the turnips and verified that they were doing the right amount of damage. I ensured that they did not collide with other projectiles or aliens.

I also ensured that the projectile disappeared after any collisions.

Class FlatulenceTorpedo

**For NachenBlaster**

I fired torpedoes and made sure that they moved to the right and went past the edge of the screen. I fired at aliens and verified that they died after the appropriate number of hits. I also ensured that they did not collide with other projectiles.

**For aliens**

I ensured that torpedoes were being fired by aliens and that they moved to the left and went past the edge of the screen. I allowed the NachenBlaster to get hit by the torpedoes and verified that they were doing the right amount of damage. I ensured that they did not collide with other projectiles or aliens.

I also ensured that the projectile disappeared after any collisions.

Class extraLifeGoodie

I picked up the goodie and checked if the number of lives had incremented by 1 in the StringStream. I also ensured that the goodie moved down and left and disappeared after being picked up or after going past the edge of the screen.

Class repairGoodie

I picked up the goodie and checked if the health had increased by the right amount in the StringStream. I also ensured that the goodie moved down and left and disappeared after being picked up or after going past the edge of the screen.

Class flatulenceTorpedoGoodie

I picked up the goodie and checked if the number of torpedoes had increased by 5 in the StringStream. I also ensured that the goodie moved down and left and disappeared after being picked up or after going past the edge of the screen.

Class Smallgon

I ensured that the Smallgon moved in the right pattern and changed directions when it reached the top or bottom of the screen or when its path length reached zero. I checked if the Smallgon was taking damage when being hit by projectiles by making it output its health in its doSomething function (only for testing). I checked if the Smallgon died on colliding with the NachenBlaster and created an explosion. I checked if the Smallgon fired Turnips when almost in line with the NachenBlaster. I also ensured that the Smallgon disappeared when it died or went past the edge of the screen.

Class Smoregon

I ensured that the Smoregon moved in the right pattern and changed directions when it reached the top or bottom of the screen or when its path length reached zero. I checked if the Smoregon was taking damage when being hit by projectiles by making it output its health in its doSomething function (only for testing). I checked if the Smoregon died on colliding with the NachenBlaster and created an explosion. I checked if the Smallgon fired Turnips or dashed to the left when almost in line with the NachenBlaster. I also ensured that the Smoregon disappeared when it died or went past the edge of the screen.

I also checked if the Smoregon was occasionally dropping goodies when it died.

Class Snagglegon

I ensured that the Snagglegon moved in the right pattern and changed directions when it reached the top or bottom of the. I checked if the Snagglegon was taking damage when being hit by projectiles by making it output its health in its doSomething function (only for testing). I checked if the Snagglegon died on colliding with the NachenBlaster and created an explosion. I checked if the Snagglegon fired torpedoes when almost in line with the NachenBlaster. I also ensured that the Snagglegon disappeared when it died or went past the edge of the screen.

I also checked if the Snagglegon was occasionally dropping goodies when it died.

Class StudentWorld

Every part of the above test cases was proof that the StudentWorld class worked properly. The game moved smoothly showing that the ticks worked properly. All the Actors were created and appeared on the screen proving that the init function worked as well. On finishing a level or dying, the right message popped up. The StringStream displayed the right values with the right formatting. On ending the game, there were no issues with the cleanUp method or the destructor.