Question 1

1. Actor class

virtual void doSomething() = 0: This is called for each actor in each tick of the game, in the move() function of StudentWorld. This function is pure virtual because all actors can do something, and each of its derived classes has a different implementation of doSomething(). The actor itself is an abstract class and can not do something, therefore it is pure virtual.

virtual void DeathSequence(): This is the sequence of commands an alien performs when it is killed by a Nachenblaster fired projectile. It has to be called on the vector of Actors, therefore it is an actor function, however only aliens have an implementation of DeathSequence. It is not pure virtual as it is not required to be implemented by all the derived classes(only the various types of aliens), but it is virtual since it has to be overridden by the Smallgon, Smoregon and Snagglegon versions of the function.

StudentWorld\* getWorld() const: returns a pointer to the StudentWorld that the actor is located in.

virtual bool isAlive() const = 0: This returns whether an actor is alive. It is pure virtual as the derived classes have various implementations of isAlive(), and it is required to be implemented by all derived classes.

virtual void sufferDamage(double amt): reduces an actor’s hitpoints by amt. It is virtual because not all actors(only DamageableObjects) can sufferDamage and do not need to implement it. It however has to be overridden by the sufferDamage function in DamageableObject.

virtual bool collidableWithPlayerFiredProjectile() const: returns whether an actor is collidable with a player projectile(is an alien). It is virtual because it is overridden in the alien class where it is true, but false for all other actors.

1. Star class

void doSomething(): moves the star 1 pixel to the left.

bool isAlive() const: returns false if the star has flown off the left of the screen.

1. Goodie class

void move(): moves the goodies 0.75 pixels down and 0.75 pixels left.

bool isAlive() const: returns false if the goodie is dead(flown off the screen or picked up).

1. ExtraLifeGoodie class

void doSomething(): checks if collided with a player, and if so executes the appropriate sequence of events. Calls Goodie move() function, then checks for collision again.

1. RepairGoodie class

void doSomething(): checks if collided with a player, and if so executes the appropriate sequence of events. Calls Goodie move() function, then checks for collision again.

1. TorpedoGoodie class

void doSomething(): checks if collided with a player, and if so executes the appropriate sequence of events. Calls Goodie move() function, then checks for collision again.

1. DamageableObject class

double hitPoints() const: returns the DamageableObject’s hitpoints.

void increaseHitPoints(double amt): increases the DamageableObject’s hitpoints by amt.

void sufferDamage(double amt): reduces the DamageableObject’s hitpoints by amt.

1. Alien class

bool collidableWithPlayerFiredProjectile() const: always returns true, overriding the actor class’s default return false.

void move(): moves the alien as per its y direction and its speed.

void setYdirection(int dy): sets y direction to dy. 2 is up, 1 is no y direction(only moves in x), 0 is down.

void setSpeed(double speed): sets the alien’s speed to the speed parameter.

int getflightpath() const: returns the alien’s flight path.

void setFlightPath(int a): sets the alien’s flight path.

1. Smallgon class

bool isAlive() const: returns false if hitpoints <=0.

void doSomething(): contains the Smallgon’s actions for each tick. The Smallgon checks if it has collided with the Nachenblaster, and takes appropriate action if it has. It calls the move() function from Alien, and then checks for collision again. It changes direction and flight path as the spec directs, and also has a chance to shoot at the player.

void DeathSequence(): This function is run when an alien is killed by a projectile or collides with the player. The game score is increased appropriately and the alien kill recorded, game sounds played and an explosion is created.

1. Smoregon class

bool isAlive() const: returns false if hitpoints <=0.

void doSomething(): contains the Smoregon’s actions for each tick. The Smoregon checks if it has collided with the Nachenblaster, and takes appropriate action if it has. It calls the move() function from Alien, and then checks for collision again. It changes direction and flight path as the spec directs, and also has a chance to shoot at the player. It can also charge at the player.

void DeathSequence(): When Smoregon dies, increases score, plays game sounds, adds explosion and may drop RepairGoodie or TorpedoGoodie.

1. Snagglegon class

bool isAlive() const: returns false if hitpoints <=0.

void doSomething(): contains the Snagglegon’s actions for each tick. The Snagglegon checks if it has collided with the Nachenblaster, and takes appropriate action if it has. It calls the move() function from Alien, and then checks for collision again. It changes direction and flight path as the spec directs, and also has a chance to shoot torpedoes at the player.

void DeathSequence(): When Snagglegon dies, increases score, plays game sounds, adds explosion and may drop ExtraLifeGoodie.

1. Projectile class

bool isAlive() const: Returns true if a projectile is alive

void setAliveStatus(bool a): Sets the projectile to alive or dead

1. Turnip class

void doSomething(): checks if collided with player, moves, then checks again. If it has collided, damages the player appropriately.

1. Cabbage class

void doSomething(): checks if collided with an alien. If it has, damages it. If it kills it, calls DeathSequence() on that alien. Moves, then checks for collision again.

1. Torpedo class

No public member functions

1. PlayerLaunchedTorpedo class

void doSomething(): checks if collided with alien, and damages it if it has. If it kills the alien, it calls DeathSequence() on that alien. Moves, then checks again.

1. AlienLaunchedTorpedo class

void doSomething(): checks if collided with player, moves, then checks again. If it has collided, damages the player appropriately.

1. NachenBlaster class

void doSomething(): increases cabbage energy points by 1, then checks if a key has been pressed. If a key has been pressed, the NachenBlaster either moves or fires a projectile depending on which key was pressed.

bool isAlive() const: returns false if hitpoints<=0, true otherwise.

double getCEP() const: returns the number of cabbage energy points the player has.

int getTorpedoes() const: returns the number of torpedoes the player has.

void increaseTorpedoes(int a): increases the number of torpedoes by a.

1. Explosion class

void doSomething(): increases size to 1.5 times previous and reduces lifespan by 1.

bool isAlive() const: returns false if lifespan reaches 0 or less, true otherwise.

1. StudentWorld class

virtual int init(): called when a level begins, initializes the game screen.

virtual int move(): called on each tick of the game – calls doSomething() on all actors, deletes dead actors, spawns new actors if they should be and sets the scoreboard.

virtual void cleanUp(): deletes all actors from the game.

Actor\* getOneCollidingAlien(const Actor\* a) const: returns a pointer to an alien that is colliding with the actor parameter by the collision detection system provided.

NachenBlaster\* getCollidingPlayer(const Actor\* a) const: returns a pointer to the player if it is colliding with the actor parameter by the collision detection system provided.

void recordAlienDestroyed(): records if an alien has flown off the screen.

void recordAlienKilled(): records if an alien has been killed by the player.

void addActor(Actor\* a): adds the actor to the game.

NachenBlaster\* getPlayer() const: returns a pointer to the player.

Question 2

All components of the specification were implemented, and there are no known bugs.

Question 3

I handled the collision of aliens with projectiles from the projectile class only – aliens do not detect whether they have collided with a projectile. The projectile checks if it has hit an alien and then calls DeathSequence() on that alien.

Apart from this, I followed the specification thoroughly and did not make any other assumptions/decisions to divert from the specification.

Question 4

I only tested the non-abstract classes(which have an implementation of doSomething()), since the abstract classes can not be instantiated and tested in the game.

NachenBlaster class: I tested the motion of the player by removing all the aliens and moving the player around the empty screen, making sure it stopped at the right boundaries. I tested the shooting of cabbages by comparing whether the energy points allowed the cabbages to fire at the same rate as the sample game. I then added each alien individually to see if they dropped the right goodies that could be picked up by the player. I checked that each goodie carried out its function when picked up by the player, and that torpedoes could be fired correctly.

I also checked that the NachenBlaster was damaged appropriately by collisions with aliens, turnips and torpedoes by spawning only Smallgons, only Smoregons and only Snagglegons so that I could create each damage taking situation more frequently than they would occur in a game.

Star class: I made sure stars spawned at random sizes, and paused the game to check that the correct number of stars were created on initialization. I also tested that the stars were moving as they were supposed to, 1 pixel per tick, and then being deleted properly using the sanity checker provided for part 1.

Explosion class: I shot at ships and paused the game, proceeding 1 tick at a time. I could then check whether the explosion was growing over each tick by 1.5 times it size, and then getting deleted after 4 game ticks.

Cabbage class: I checked that the highest possible fire rate(by holding space bar down) was identical to the sample game. I tested that cabbages were getting deleted if they flew off the screen, as well as getting deleted if they collided with an alien. I also tested whether the collision detection worked 100% of the time by sending aliens in straight lines and shooting at them, making sure the aliens took the right amount of damage from an individual cabbage.

Turnip class: I set the game to only spawn Smoregons and Smallgons, so that I could only have turnips fired at me. I counted the total turnips fired at me over a fairly large period of time and compared it to the sample game, making sure that the probability of turnips being fired was approximately the same. I also made sure that the turnips damaged the player appropriately if collided, and tested collisions by pausing and proceeding frame by frame when a turnip was fired: making sure the turnip was deleted and the player damaged correctly.

Flatulence Torpedo: I tested player launched torpedoes and alien launched torpedoes individually. For the player launched torpedoes, I spawned only Smoregons and Snagglegons. I shot at only Smoregons, making sure Flatulence Torpedo goodies were spawned occasionally. When they did, I picked them up, making sure I got the right number of torpedoes. I fired torpedoes at both Snagglegons and Smoregons, making sure they did the right amount of damage and were deleted after flying off screen or hitting an alien.

For the alien launched torpedoes, I only spawned Snagglegons, allowing them to fire torpedoes at the player. I made sure they did the correct amount of damage and were deleted appropriately.

Repair Goodie and Torpedo Goodie: I spawned only Smoregons, attempting to get as many goodies as possible. I checked that the repair goodie worked at both full health and less than full health, and did not cause the player’s hitpoints to go over maximum. I also made sure that both goodies moved correctly, down and left at the correct speed. When I picked up Torpedo Goodies, I made sure that the number of torpedoes increased by the right amount.

Extra Life Goodie: I spawned only Snagglegons, attempting to get as many extra life goodies as possible. I made sure they were picked up/flew across the screen correctly, and deleted when required. I checked that they incremented my lives correctly and played the extra lives to make sure they worked correctly in the game.

Smallgon: I set the game to only spawn Smallgons, so that I could test whether they could be shot down correctly as well as did the correct thing when colliding with the player. I made sure they spawned explosions on death and added the correct number of points when killed, and not adding points when just flying off screen. I made sure they had the correct number of hitpoints and were damaged correctly by cabbages and torpedoes, and made sure they didn’t drop any goodies. I also tested the turnip shooting function, making sure they didn’t shoot turnips when not in the correct range, and checking if the player was damaged correctly.

Smoregon: I set the game to only spawn Smoregons, and made sure the hitpoints, player- fired projectile interactions and collision detection was correct. I made sure that they dropped goodies at the approximate probability they were meant to be dropped at, and that the goodies were spawned at the correct position by pausing the game. I also made sure the turnip shooting worked in a similar manner to Smallgons, and checked that the charge function worked in the same way as the sample game.

Snagglegon: I set the game to only spawn Snagglegons, and made sure the hitpoints, player-fired projectile interactions and collision detection was correct. I made sure they moved in the correct manner, and allowed them to fire torpedoes, checking if the player was damaged correctly. I also made sure they damaged the player correctly on collision.