Aseem Sane

1.

class StudentWorld : public GameWorld

StudentWorld(std::string assetDir);

* Given constructor for creating the world

virtual int init();

*Actor a new Nachenblaster*

*Allocate 30 new Stars and place them in the Actor vector*

*Set number of Aliens dead and alive to 0*

*Set the string to be displayed at the top*

* For initializing the world

virtual int move();

*Make the blaster and each actor doSomething*

*If blaster dies, return value specifying a life lost*

*If the number of aliens needed to be killed are killed, advance to the next level*

*Deallocate actors that have died this tick*

*If the number of aliens on screen is below what is needed*

*Allocate a new alien of type determined by chance to the right of screen*

*Possibly allocate a star based on chance to the right of the screen*

*Update top display*

* For each tick, to move and remove actors as needed

virtual void cleanUp();

* To deallocate every actor that has been allocated during the current life

void addActor(Actor\* newActor);

* To add a new actor to the vector, called by individual actors to add another actor themselves

bool hasCollided(Actor\* toCheck, Actor\*& toChange);

*Iterate through the vector to check whether toChange is one of the actors in the vector,*

*and if so*

*Start collision search from the next actor in the vector*

*Else*

*Start collision search from the beginning of the vector*

*If the positions and the radii between toCheck and an object in the rest of the vector are close enough for collision*

*Change toChange to the collided actor in the vector*

*Return true*

*Return false*

* Checker function to go through each actor in the list and check whether their position attributes are the same

bool hasCollidedwithBlaster(Actor\* toCheck, NachenBlaster\*& blaster);

* To check whether the toCheck actor has collided with the blaster

NachenBlaster\* getBlaster();

* Returns a pointer to the nachenBlaster

void killAlien();

* Increments number of aliens killed and decrements number alive

void dieAlien();

* Decrements number of aliens alive

class Actor : public GraphObject

Actor(int imageID, double startX, double startY, Direction dir, double size, int depth, StudentWorld\* theWorld);

* Each Actor would have these attributes, as well as a pointer to the StudentWoruld it lives in

virtual void doSomething()

* This function was pure virtual since no instance of an Actor class was to be created – since every object to be created would have its own version of doSomething

void setDead(bool status);

* I chose to define this in Actor since every Actor would die at some point

bool getDead();

* The setter and getter functions for death status were due to the death status Boolean member variable being private in Actor

StudentWorld\* getWorld();

* The Actor class had a studentWorld private member variable that would be passed into its constructor

virtual bool isAlien();

* By default, this function would be false, unless for the inherited Alien class, which overrode this virtual function to be true

virtual bool isProjectile();

* Set to false unless overridden by the inherited Cabbage class

virtual void takeDamage(int damage);

* Every Actor had a health member variable, so that polymorphism could be used to call this function and give damage only to those objects in the world’s container that returned true for the “isAlien” function

double gethealth();

* The getter function for the health member variable

void sethealth(double health);

* Setter function for the health

class NachenBlaster : public Actor

NachenBlaster(StudentWorld\* world);

* The only attribute that needed to be passed in was the world pointer – everything else was preset and passed automatically to the Actor contructor

virtual void doSomething();

*Detect if dead, and if so, return*

*If user pressed a key*

*Move or allocate and shoot a projectile based on which key is pressed*

*Increase cabbage energy*

* The NachenBlaster would doSomething each tick in response to the user and automatically

void incTorpedoes(int incr);

* Only the NachenBlaster class could have a counter on the number of Flatulan Torpedoes it has

int getPercentCabbages();

* Only the NachenBlaster class would have a counter on the number of cabbages it has

int getNumTorpedoes();

* Getter function for number of torpedoes for top display

class Star : public Actor

Star(double startX, double startY, StudentWorld\* world);

* There were 2 custom attributes to each star to mark the starting location, plus the world pointer. Everything else was preset

virtual void doSomething();

* Stars would move left

class Explosion : public Actor

Explosion(double startX, double startY, StudentWorld\* world);

* 2 custom attributes for starting pos, plus world pointer

virtual void doSomething();

* Would grow in size, then die

class Cabbage : public Actor

Cabbage(double startX, double startY, StudentWorld\* world);

* 2 position attributes

virtual void doSomething();

* Would move right and possibly hurt aliens if collided

virtual bool isProjectile();

* Overriden virtual function to assist in collision detection

class Torpedo : public Actor

Torpedo(double startX, double startY, StudentWorld\* world, bool isEnemy);

* 3 custom attributes – 2 for position, and the isEnemy to tell whether the torpedo was shot by an enemy or the blaster

virtual void doSomething();

*If dead or out of bounds, return*

*If enemy*

*Check for collision with blaster, and if so, damage the blaster*

*Else*

*Check for collision with alien, and if so, damage or kill the alien based on alien’s remaining health*

*If enemy, move left, else, move right*

*Check for collision just like the previous steps*

* Torpedo would move left/right and possibly hurt the blaster/aliens according to whether it was enemy or not

class Turnip : public Actor

Turnip(double startX, double startY, StudentWorld\* world);

* Starting position attributes

virtual void doSomething();

* Would move left and possibly hurt the blaster if collided

class Alien : public Actor

Alien(int imageID, double startX, double startY, Direction dir, double size, int depth, StudentWorld\* theWorld);

* Similar attributes to the Actor class

virtual bool isAlien();

* Overriden virtual function to assist in collisions

void setPathLength(int lgt);

* Majority of the aliens had a path length attribute that could be changed

int getPathLength();

* Getter function for path length

void setPathDir(int direc);

* All the aliens had a path direction attribute

int getPathDir();

* Getter for path direction

void setTravelSpeed(double spd);

* All the aliens had a travel speed attribute

double getTravelSpeed();

* Getter for speed

void collideWithBlaster(NachenBlaster\* blaster, int damage, int score);

*Make the blaster take damage*

*Set the alien as dead*

*Increase score*

*Play death sound*

*Make explosion*

* Function for what to do when aliens collide with blaster

void getKilled(int score);

*Set the alien as dead*

*Increase score*

*Play death sound*

*Make explosion*

* Function for what to do when aliens were killed

void movePath();

* Function to make the aliens move on their designated path.

class Smallgon : public Alien

Smallgon(double startX, double startY, StudentWorld\* world);

* 2 custom attributes for pos

virtual void doSomething();

*Check if dead or out of bounds, if so return*

*Check if collided with blaster or cabbages, and if so, perform death procedures*

*If at maximum or minimum y position or no path length remaining*

*Change path direction or length as needed*

*If blaster is within vertical shooting range of alien*

*Possibly shoot a turnip*

*Check again for collisions as before*

* Smallgons had their own movement and attack patterns

class Smoregon : public Alien

Smoregon(double startX, double startY, StudentWorld\* world);

* 2 pos attributes

virtual void doSomething();

*Check if dead or out of bounds, if so return*

*Check if collided with blaster or cabbages, and if so, perform death procedures and also possibly drop goodies*

*If at maximum or minimum y position or no path length remaining*

*Change path direction or length as needed*

*If blaster is within vertical shooting range of alien*

*Possibly shoot a turnip, or activate ramming speed*

*Check again for collisions as before*

* Smoregons had their own movement/attack/dropping patterns

class Snagglegon : public Alien

Snagglegon(double startX, double startY, StudentWorld\* world);

* 2 pos attributes

virtual void doSomething();

*Check if dead or out of bounds, if so return*

*Check if collided with blaster or cabbages, and if so, perform death procedures and also possibly drop goodie*

*If at maximum or minimum y position*

*Change path direction as needed*

*If blaster is within vertical shooting range of alien*

*Possibly shoot a torpedo*

*Check again for collisions as before*

* Snagglegons had their own movement/attack/dropping patterns

class Goodie : public Actor

Goodie(int imageID, double startX, double startY, StudentWorld\* theWorld);

* Custom attributes for each type of goodie

void getGoodie();

* Common function for when player got each goodie

class ExtraLifeGoodie : public Goodie

ExtraLifeGoodie(double startX, double startY, StudentWorld\* world);

* 2 pos attributes

virtual void doSomething();

* For the special positive effects of an extra life when colliding with player

class RepairGoodie : public Goodie

RepairGoodie(double startX, double startY, StudentWorld\* world);

* 2 pos attributes

virtual void doSomething();

* For the special positive effects of repairing health when colliding with player

class FlatulenceTorpedoGoodie : public Goodie

FlatulenceTorpedoGoodie(double startX, double startY, StudentWorld\* world);

* 2 pos attributes

virtual void doSomething();

* For the special positive effects of getting torpedoes when colliding with player

2.

No more detected functionality I have missed or bugs. Some oddities are that the above display seems to twitch more in my version than the given executable, and my game seems to run a but worse on Visual Studio.

3.

I assumed that polymorphism had to be used in detecting each collision, so I gave a health attribute to my Actor class itself, even if derived classes from Actor didn’t have a health attribute to care about. I also chose not to create a projectile base class, since I had my torpedo class work for both friendly and enemy torpedoes.

4.

I tested my NachenBlaster by first moving it in all directions and shooting every type of projectile. Then I shot cabbages and spammed them to see if their depletion worked properly. I also took damage from each ship to see if the damage system worked properly. I checked whether the display at the top corresponded to the ship’s state.

To test the stars, I checked whether the correct amount of stars were spawned and whether they moved correctly. To test the explosions, I killed ships and saw if explosions were spawned and destroyed correctly.

To test cabbages, I shot them out of the ship and checked whether they moved forward at the right speed and rotation by comparing it to the sample executable. I checked whether cabbages were properly destroyed after they hit an enemy ship, and whether the ships are destroyed in the right number of bullets compared to the sample executable.

To test the turnips, I placed my NachenBlaster ahead of Smallgons and Smoregons, and checked whether the aliens shot them at the right speed and rotation, and positioned correctly. I checked whether they were destroyed correctly by checking how much damage they did to the NachenBlaster.

To test the torpedoes, I checked whether they shoot the right way if they are enemy torpedoes or not, and at the right speed. Then I checked whether they do the right amount of damage to the NachenBlaster, and to the ships (easier to test in higher levels).

To test the Smallgon, I checked whether the Smallgon moved in the correct erratic patterns compared to the sample executable, and at the right speed. I also checked that they shot turnips as indicated, and played the correct sounds. I checked that they were killed correctly when colliding with the player or its projectiles, by checking the damage that was done and should have been done.

To test the Smoregon, I checked whether the Smoregon acted like the Smallgon in most situations, testing it in a similar manner. In addition, I positioned the NachenBlaster ahead of it and checked whether it activated ramming speed randomly. Also, I checked that it dropped the correct drops when killed.

To test the Snagglegon, I checked that its flight pattern was completely diagonal. I checked whether it would shoot torpedoes when ahead of the ship rather than turnips. I also checked that it was harder to kill due to more health, and checked whether it would drop the correct drops when killed.

To test the extra life goodie, I made the player collide with Snagglegons to pick it up randomly. I checked whether its movement pattern was correct. Then I checked whether the upper display incremented the lives correctly, and whether it would take one more life for the player to reach game over.

To check the repair goodie, I made the player kill Smoregons, and pick up the repair goodies when they appeared. I checked the upper display before and after pickup to see whether health was restored correctly. I also checked whether health would never increase past 100%.

To test the flatulence torpedo goodie, I made the player kill Smoregons again, and pick up the torpedo goodies when appeared. I checked whether the correct number of torpedoes was added to the ship by checking the upper display. I also made the player shoot the 5 torpedoes to make sure the right number of torpedoes was added.