Hermmy Wang

Professor Smallberg

CS 32 Lecture 3

1 March 2018

Report

1. **Description**

A description of only the complex member functions in Actor, NachenBlaster, Star, Explosion, Projectile, Goodie, and Alien. I provide pseudocodes for the functions that I believe are nontrivial. I omit some classes such as the derived classes of Goodie, Projectile, and Alien, which do not have nontrivial algorithms.

**class Actor:**

* doSomething(): I chose to define it as a nonvirtual function in Actor class because every actor needs to check if it is alive. Inside this function, I will call a pure virtual function doSomethingDiff() if the actor is alive.
* doSomethingDiff(): This is a pure virtual function in Actor class. Each type of actor is able to do something , and they do it in different ways.
* sufferDamage(double d): This is a virtual function in Actor class. Aliens and Nachenblaster are able to suffer damage, and others simply die. Therefore, Nachenblaster and Aliens will deal with damage in their own ways, and others have a default version of sufferDamage in Actor class, which is set them dead.
* collideNB(): This is a nonvirtual function because every actor in Actor class collides with the NachenBlaster in the same way (except for the Nachenblaster can’t collide with itself, but the NachenBlaster will never call this function).
* getDamagePoints(): This is a virtual function because each type of actors have different damage points (e.g. a cabbage has 2 and a torpedo has 8). For stars and explosion which do not cause damage, the base version in Actor class is to return 0.
* alienShip(): This is a virtual function in Actor class. The base version returns false, which can be applied to every type of actors except for alien ships. Alien’s version of alienShip() returns true, so StudentWorld can identity it is an alien ship when it deletes one.
* Other accessor functions, mutator functions, and const functions are nonvirtual because every actor behaves in the same way.
* **class NachenBlaster:**
  + doSomethingDiff(): This is a virtual function in NachenBlaster class. It is inherited from its base class.
    - Pseudocode:

Get key from Student World

If user presses Space and enough cabbages

create cabbages

activate them

play sound

decrement the number of cabbage energy points

If user presses Tab and enough torpedoes

create a torpedo

activate it

play sound

decrement the number of torpedoes

If user presses direction

move if it is not offscreen.

Increase the number of cabbage if it is not full.

* + sufferDamage(double d): This is a virtual function in NachenBlaster class, inherited from Actor class.
    - Pseudocode

Decrease hit points by d

Set it dead if hit points are below 0

* **class Star:**
  + doSomethingDiff(): This is a virtual function in Star class. It is inherited from its base Actor class.
    - Pseudocode:

Move 1 px to the left

If it is offscreen

Set it dead

* **class Explosion:**
  + doSomethingDiff(): This is a virtual function in Explosion class. It is inherited from its base Actor class.
    - Pseudocode:

Increase size by 1.5 every tick.

Increment the number of ticks.

If the number of ticks is greater than 4

Set it dead

* **class Projectile:**
  + doSomethingDiff(): This is a virtual function because each projectile needs to check if it is offscreen, if it collides with anything, if it continues to move, and if it collides with anything again.
    - Pseudocode:

If offscreen

Set it dead

Else

Check collision

If not dead

Move the projectile

Check collision again

* + checkCollision(): It is a pure virtual function because each type of projectile is able to collide with different actors, and causes different amount of damage to the victim.
    - Pseudocode:

If collide with an alien

Let alien suffer damage

Set itself dead

If collide with the NachenBlaster

Let the NachenBlaster suffer damage

Set itself dead

* + moveProjectile(): It is a pure virtual function because each type projectile moves in different speed and direction.
* **class Goodie:**
  + doSomethingDiff(): It is a virtual function because it is inherited from its base Actor class. Every goodie needs to check if it is offscreen, if it is collected by the player, if it can continue to move, and if it collides with NachenBlaster again after moving. (Similar to Projectile’s doSomethingDiff).
  + collisionReaction(): It is a nonvirtual function because Goodie’s derived classes do not call it, and every goodie reacts to the collision in the same pattern.
  + bonus(): It is a pure virtual function in Goodie because each type of goodie gives the player different bonus.
* **class Alien:**
  + doSomethingDiff(): It is a virtual function because it is inherited from its base Actor class. Every alien ship does something in the same pattern.
    - Pseudocode

If offscreen

Set it dead

Else

Check collision

If not dead

Check if it needs new fight plan

Change flight plan

Check if the player is in the sight

Possibly Fire a projectile

Continue moving

Check collision again

* + checkCollision(int damage, int score): It is a nonvirtual function in Alien class that checks if an alien collides with the NachenBlaster. Every alien ship checks collision with the player in the same way.
    - Pseudocode

If it collides with the NachenBlaster

NachenBlaster suffer some damage

Fatal collision for the alien ship

Possibly drop a goodie

* + fatalCollision(): It is a nonvirtual in alien class because all aliens ship deals with fatal collision in the same fashion.
    - Pseudocode:

Increase the player’s score

Set it dead

Increment the player’s number of destroyed aliens

Play sound

Create and activate an explosion

* + sufferDamage(): It is a virtual function because it inherits from its base Actor class. It decreases a certain amount of hit points from the alien ship and makes it do something.
    - Pseudocode:

Suffer some damage

If the damage causes it to die

Fatal collision for the alien ship

Possibly drop a goodie

Else

Play sound

* + changeDirection(): It is a nonvirtual function because none of the derived classes need to call it, and every alien changes direction partially in this fashion.
    - Pseudocode:

If reaches top

Set direction to downleft

If reaches bottom

Set direction to upleft

Call differentiated version of change direction

* + changeDirDiff(): It is a virtual function in Alien class because Smallgons and Smoregons deal with changing direction in the default way, where Snagglegon has a different way and needs to override this function.
    - Pseudocode:

If flight length reaches 0

Randomly change to a different direction

Randomly choose a new flight length

* + move(): It is a nonvirtual function in Alien class. Every alien moves in the same way according to its different travel direction.
  + moveDiff():It is a virtual function in Alien class because Smallgons and Smoregons deal with changing flight length in the default way, where Snagglegon does not have a flight length.
  + dropGoodie(): It is a pure virtual function in alien class. Every alien needs to drop a goodie, but they drop it in different patterns. Smallgons are not able to drop; Smoregons and Snagglegons drop different goodies.
  + fire(): This is pure virtual function in alien class because every type of alien is able to fire. They fire different projectiles with different possibilities.

**class StudentWorld:**

* init(): It is a virtual function inherited from GameWorld.
  + Pseudocode:

Repeatedly 30 times:

Create a new star at a random location

Push the star into the actor vector

Create a new NachenBlaster

Initialize data members

* move(): It is a virtual function inherited from GameWorld.
  + Pseudocode:

For every actor in the actor vector

If the actor is not dead

Let it do something

If the player died

Decrement a life

Return player died

Else

If the level is finished

Play sound

Return finished level

Else

Erase the dead actor from the actor vector.

If it is an alien

Decrement the counter from alien ships

Delete the dead actor

Let the player do something

Update game text

Possibly insert stars

Possibly insert Aliens

Return continue game

* cleanUp(): It is a virtual function inherited from GameWorld.
  + Pseudocode

If the actor collection is not empty

Push the player into the collection

Repeatedly:

Erase every actor in the collection

* animate(Actor\* obj): It is a nonvirtual function because Student World does not have a derived class. It allows the actor being passed in to do something.
* collide(Actor\* obj): It is a nonvirtual function because Student World does not have a derived class. It checks if the actor being passed in, which should be a player projectile, collides with an alien ship. If it does, return a pointer to that actor, so the player projectile can let the alien ship suffer damage.

1. **Failure**

As far as I know, I did not fail to implement any functionality. There are no notable bugs in my program. The only thing I did not do exactly according to the specification is the size of explosion signs. I think the explosion sign appears too large by starting at size 1.0, so I changed it to 0.5.

1. **Assumptions**
   1. The instruction did not specify how to distinguish an alien projectile (which cannot collide with a player projectile), and an alien ship (which is collidable with a player projectile); therefore, I set up a data member called “m\_label”, which labels all player projectiles with PLAYER and all alien ships with ENEMY. The function collide(Actor\* obj) in StudentWorld class becomes easy to implement. To find out a valid collision, I just need to check if two collided actors have different labels.
   2. Every type of actor keeps a pointer to the StudentWorld that it exists. Any actor could easily call an accessor function getWorld() to use the functions in the StudentWorld class.
   3. I include a hit-point value for every actor, including stars and explosions. I initialize the actors with no specified hit points with a constant value 50. setDead() and dead() function are therefore easy to implement. setDead() function will simply set any actor’s hit points to 0, and dead() function will check if an actor’s hitpoints is equal or below 0.
   4. I also include a flight length for every alien, though Snagglegons do not have a flight length. I did this so that all alien ships can share the same algorithm to change direction, except that Snagglegons will never encounter the situation where their flight length reaches 0.
   5. The instruction does not specify if a player’s lives have an upper limit, so I decide to give it no upper limit.
2. **Test**

* StudentWorld
  + I test StudentWorld by making sure it can complie correctly with no compilation errors and runtime errors. The initialized scenario is correctly set with certain actors. I monitor the program by using F and R to make sure that each actor has to be able to do something during each tick. The dead objects have to be removed by C++ delete statement. Since the delete statement will call the specific actor class’s and the base actor’s destructors, I add cerr statement to those destructors to make sure that they are called at the right time. I add cerr statement within multiple functions within StudentWorld class to make sure that they are called appropriately for the right number of times.
* Actor
  + I test Actor by making sure it contains all the common functions share by its derived classes. Some functions first do common things for all actors, and then call a pure virtual that performs differentiated tasks for each class. I make sure that for the virtual functions that are only overridden in some derived classes, the base Actor class has a default version for other classes to use. Therefore my program can use polymorphism adequately.
* NachenBlaster
  + I test that the player can use space key to fire a cabbage and the energy points will decrement and recover properly, use tab key to fire a torpedo and the torpe will decrement properly, use directional keys to move around and not be off screen. Other keys except for F and R should have no effects. When alien ships shoot at NachenBlaster, I make sure it suffers damage properly by monitoring its health percentage. Also, I add a temporary data on the game status text called “Remained” to make sure the level advances when the number of ships remained to destroy reaches 0. The destructor is called before the last Actor destructor when the game ends.
* Star
  + I test that the star is properly destructed when it is off screen. New stars are inserted randomly from the right edge and move towards the left. The stars move in a constant and slower speed than other actors.
* Explosion
  + I test that the explosion sign will disappear after exact 4 ticks by using F to moniter each tick. The explosion sign appears exactly when the alien ship is destroyed by player projectiles or by the player itself. The destructor of explosion is called at the right time.
* Projectile
  + I find the common behaviors of all the projectiles, such as checking if it is off screen, moving in a certain speed and direction, and checking collision with other actors. Projectile allows polymorphism, as different projectiles do the same task in different ways. Therefore, within the Projectile base class, I make sure I include pure virtual functions for the differentiated versions of these tasks for each type of projectile.
* Cabbage
  + I test that a cabbage will only collide with an alien ship and cause exactly 2 points of damage to the ship. I slow down the speed of alien ships so that a cabbage is easier to hit them. At level 1, exact three cabbages can kill a smallgon/smoregon, exact 5 cabbages can kill a snagglegon. I add a cerr statement to inform me that a cabbage is killed right after colliding with an alien ship. I freeze the screen to check if a cabbage rotates properly duringe each tick.
* Turnip
  + I test that a turnip will only collide with the NachenBlaster and cause exactly 2 points of damage to the player. Since alien ships do not always shoot turnips, I temporarily remove the possibility check and let alien ships always shoot turnips at the player within a range of locations, and avoid hitting by torpedoes. The player dies after being attacked by exact 25 turnips. I slow down the speed of alien ships so that the NachenBlaster is easier to be hit. I add a cerr statement to inform me that a turnip is killed right after colliding with the NachenBlaster. I freeze the screen to check if a turnip rotates properly duringe each tick. I also check if the sound is played properly.
* Torpedo
  + I test that a torpedo can collide with the NachenBlaster or the alien ships, and nothing else. It causes exactly 8 points of damage to the player or the alien ship. Since snagglegons do not always shoot torpedoes, I temporarily remove the possibility check and let them always shoot torpedoes at the player within a range of locations. Every time the player is hit, its health percenatge decreases 16%. The player dies after being attacked by exact 7 torpedoes if it avoids being hit by turnips. I slow down the speed of alien ships so that the NachenBlaster is easier to be hit.
  + Since the player starts off with no torpedoes, I temporarily change the initialized m\_nTorpedo to 10. At level 1, a smallgon or a smoregon is killed by exact 1 torpedo and a snagglegon 2. I add a cerr statement to inform me that a torpedo is killed right after a collision. I also check if the sound is played properly.
* Goodie
  + I test that the goodie is dead if it is off screen. Since all kinds of goodies share a lot of functionality, I make sure I have those functions in the Goodie base class. The goodie should increase the player’s score, be set to death, play sound, and reward the player right after colliding with the player. If it is picked up during the current tick, it will no longer move or check collision again.
* ExtraLife
  + I let the snagglegon always drop an extra life goodie after being killed. The player gains one life after colliding with the goodie with no upper limits.
* Repair
  + I let the smoregon always drop a repair goodie and no TorpedoGoodie after being killed. The player’s health percentage should increase 20% if it is less than 80%, otherwise just increase to 100% after colliding with the goodie.
* TorpedoGoodie
  + I let the smoregon always drop a TorpedoGoodie after being killed. The player’s inventory of torpedoes should increase 5 after colliding with the goodie.
* Alien
  + I test the number of aliens on the screen should always follow the given formula. Smallgons and Smoregons should have random flight lengths in different directions. Every alien ship should change direction when reaching the top or the bottom of the screen. Every alien ship should be set to death if it moves off the left edge. If the player is in the shoot range, let the alien ship always fire.
* Smallgon
  + Colliding with the player should always cause the ship to die, increase the player’s score by 250, and hurt the player properly. The player should die after colliding with 5 smallgons. The smallgon should fly in three directions randomly. The smallgon never drops goodie. At level 1, 3 cabbages or 1 torpedo kill a smallgon.
* Smoregon
  + Colliding with the player should always cause the ship to die, increase the player’s score by 250, and hurt the player properly. The player should die after colliding with 5 smallgons. The smoregon should fly in three directions randomly. The smoregon should randomly accelerate to the left edge. At level 1, 3 cabbages or 1 torpedo kill a smoregon.
* Snagglegon
  + Colliding with the player should always cause the ship to die, increase the player’s score by 1000, and hurt the player properly. The player should die after colliding with 4 snagglegons. The snagglegon will fly either down left or up left. It changes direction only when reaching the top or the bottom. At level 1, 5 cabbages or 2 torpedoes kill a snagglegon.