**Project 3**

Actors organization

Actor (public GraphObject)

NachenBlaster

Star

Alien

Smallgon

Smoregon

Snagglegon

Projectile

Cabbage

Turnip

FlatulenceTorpedo

Explosion

Goodie

RepairG

ExtraLifeG

FlatulenceTorpedoG

Member functions descriptions

StudentWorld (public GameWorld)

1. There are the constructor, virtual destructor that call cleanUp(), and virtual init(), move() and cleanUp() that are required to write.
2. I added a addActor function that takes in an actor pointer and adds the actor that’s pointed to to the actor vector. Since the actor vector is inside the StudentWorld class, all the actor classes cannot add actors to the vector. Therefore, this function allows actors to be added no matter which class it’s in.
3. I have a getPlayer() that returns a pointer to the NachenBlaster. By having this function, all the aliens, projectiles and goodies can increase or decrease the objects the player has (hit points, Flatulence Torpedoes).
4. I have a updateDisplayText() that collects useful player’s data and sends them to the given setGameStatText() to display the current user info. Since any one actor shouldn’t be able to control the display text (otherwise an alien might let the player die immediately), this function is placed inside this world class.
5. I have a destroyedAnAlien() that is called every time an alien is destroyed. It decrements the number of aliens need to be destroyed by one and is used to keep track of when to proceed to the next level. Since this function is used in the world and shouldn’t be controlled by any actors, I put it inside this class.
6. I have a setAlienOffScreen() that is called every time an alien is set to dead. It decrements the number of aliens on the screen by one and is used to keep track of when to add new aliens onto the screen. Since this function is used in the world and shouldn’t be controlled by any actors, I put it inside this class.
7. I have a removeDeadGameObjects() that is called in move() during each tick. It erases dead actors from the actor vector.
8. I have a willCollide function that takes in to actor pointers and compares the two pointed-objects’ *Euclidian distance* and return if they will collide or not. This function can be used in all the nachenblaster, alien, projectile and goodie classes, but not in star or explosion classes, so I didn’t put it inside the actor class. Also, only the god, the world, knows whether tow actors collide or not. By logic, each actor will only know if they collide with another actor but can’t know if any two actors will collide. So this function shouldn’t be placed inside the actor class.
9. I have a getOneCollidingAlien function that takes in an actor pointer and checks if the actor pointed to will collide with an alien in the actor vector. If it will collide, the function returns a pointer to the first actor it collides; otherwise it returns a nullptr. This function is called inside the projectile classes. Again, since any actor only knows its own state, this function should be inside this world class which contains all the information about the actors.
10. I have a getCollidingPlayer function that takes in an actor pointer and checks if the actor pointed to will collide with the player. If it will collide, the function returns a pointer to the player; otherwise it returns a nullptr. This function prevents having long code to check whether an actor collides with the player or not, since it is inside the StudentWorld class and doesn’t need to call getWorld().
11. Since this class doesn’t have any more derived classes, all the additional functions I wrote are all non-virtual.

Actor

1. The constructor passes corresponding arguments to GraphObject base class. It also initializes the state of the actor (not dead), stores the actor’s image ID (to later determine if an actor is an alien, and stores the pointer to the world it’s in.
2. The destructor is virtual because of inheritance. It does nothing. (since the StudentWorld pointed to by the pointer object will be destroyed in the GameController, I don’t need to delete it here.)
3. Each actor has its own different things to do, so the doSomething() is declared here and is pure virtual.
4. I have a getWorld() that returns the StudentWorld the actor is in. There is no modification to any variables, so the function is const.
5. I have a isDead() that returns whether the actor is dead and should be removed from the actor vector or not. There is no modification to any variables, so the function is const.
6. I have a killIt() that set the state of the actor to dead. This function is called when an actor flies offscreen, through an explosion, being picked up by the player (goodies), or got hit.
7. I have a onScreen() that checks and returns whether the actor is within the range of the screen. There is no modification to any variables, so the function is const.
8. I have a isAlien() that checks the ID of the actor and returns if it is an alien or not. There is no modification to any variables, so the function is const.

NachenBlaster (public Actor)

1. The constructor takes in a pointer to the StudentWorld the player is in and passes the identification (ID, position, direction, etc.) of a NachenBlaster to the Actor base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. The doSomething() checks whether the player is dead or not. It determines if the user press any key and do corresponding movements accordingly.
4. I have a getCabbage() that returns the number of cabbages the player currently has. It is used in the StudentWorld to display the state of the game. There is no modification to any variables, so the function is const.
5. I have a getTorpedo() that returns the number of FlatulenceTorpedoes the player currently has. It is used to determine whether the player can fire a Flatulence Torpedo and also used in the StudentWorld to display the state of the game. There is no modification to any variables, so the function is const.
6. I have a getHit() that returns the number of hits the player currently has. This function is called in the StudentWorld to display the state of the game. There is no modification to any variables, so the function is const.
7. I have a setHit() that takes in the number of hits the player receives and subtracts it from the current hits. This function sets the player to the state of dead when the number of hits gets to zero and also prevents the number of hits to exceed 50 (maximum). This function is mainly called when the player collides with an alien.
8. I have a increaseTorpedo() that takes in the amount of Flatulence Torpedoes to add and adds them to the current number of Flatulence Torpedoes. This function is called when the player picks up a Flatulence Torpedoes goodie.

Star (public Actor)

1. The constructor takes in a pointer to the StudentWorld the player is in and the location of the star (which, in some cases, is a random number), and passes the identification (ID, position, direction, etc.) to the Actor base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. The doSomething() sets the star to dead if it moves off screen, otherwise continue moving the star leftward.

Alien (public Actor)

1. The constructor takes in a pointer to the StudentWorld the alien is in and the location of the alien, its ID, speed and hit points, and then passes the identification (ID, position, direction, etc.) to the Actor base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. Each alien has its own different things to do, so the doSomething() declared here and is pure virtual.
4. I have a getHit() that returns the number of hits the alien currently has. This function is called when deciding if the alien collides with any projectile. There is no modification to any variables, so the function is const.
5. I have a setHit() that takes in the number of hits the alien receives and subtracts it from the current hits. This function is when the alien gets hit by a projectile.
6. I have a getSpeed() that returns the speed of the alien. This function is used to move the alien during each tick. There is no modification to any variables, so the function is const.
7. I have a setSpeed() that changes the speed of the alien according the argument passed in. It is used to change the speed of Smoregons when they suddenly start to rush forward. Since the Alien base class holds all the data for speed, this function has to be put inside the Alien base class instead of the Smoregon derived class.
8. I have a getPlan() that returns the current flight plan the alien has. It is used to determine if the alien needs to change the direction it currently has. There is no modification to any variables, so the function is const.
9. I have a setPlan() that changes the current flight plan the alien has. This is used when the alien needs to change direction and needs a new flight plan. Within the plan range, the alien maintains its current direction.
10. I have a getDir() that returns the current flying direction the alien has. It is used to determine which direction the alien should be moving to. There is no modification to any variables, so the function is const.
11. I have a setDir() that change the current flying direction the alien has. This function is used to store the new flight direction generated when the alien needs to change its flying direction.
12. I have a collideWProjectile function that receives the number of point an alien loses when it gets hit by any projectile. The number of points depends on the type of aliens. If the hit points the alien has is zero or below, set its state to dead and do other actions (drop goodies if applicable (by random), create an explosion, etc.).
13. I have a collideWPlayer function that takes in an actor pointer and hit points. If collide with the player, the alien will die and the player will lose hit points depending on the type of the alien, and then do afterCollide() actions of that alien.
14. I have a pure virtual afterCollide() because each type of alien has different actions to do after collision (drop goodies if applicable (by random), create an explosion, etc.).
15. I have a withinRange() that returns if the alien is on the same horizontal line and within 4 pixels vertically with the player. This function is used in each alien’s doSomething() to determine if the alien can shoot a projectile.

Smallgon (public Alien)

1. The constructor takes in a pointer to the StudentWorld the Smallgon is in and the location. It then passes the identification (ID, position, direction, etc.) to the Alien base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a doSomething() that checks if the Smallgon is dead, goes off screen, collide with the player, need to change a flight plan, can fire a turnip, and if need to move. It does the above actions accordingly.
4. I have a afterCollide() that increases the player’s score. This function exists as a distinguisher because different aliens behave differently in collideWPlayer function.

Smoregon (public Alien)

1. The constructor takes in a pointer to the StudentWorld the Smoregon is in and the location. It then passes the identification (ID, position, direction, etc.) to the Alien base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a doSomething() that checks if the Smoregon is dead, goes off screen, collide with the player, need to change a flight plan, can fire a turnip, and if need to move. It does the above actions accordingly.
4. I have a afterCollide() that increases the player’s score and drops a goodie if applicable (by random). This function exists as a distinguisher because different aliens behave differently in collideWPlayer function.

Snagglegon (public Alien)

1. The constructor takes in a pointer to the StudentWorld the Snagglegon is in and the location. It then passes the identification (ID, position, direction, etc.) to the Alien base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a doSomething() that checks if the Snagglegon is dead, goes off screen, collide with the player, need to change a flight plan, can fire a turnip, and if need to move. It does the above actions accordingly.
4. I have a afterCollide() that increases the player’s score and drops a goodie if applicable (by random). This function exists as a distinguisher because different aliens behave differently in collideWPlayer function.

Projectile (public Actor)

1. The constructor takes in the identification (ID, position, direction, etc.) of a projectile and passes them to the Actor base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. Each projectile has its own different things to do, so the doSomething() declared here and is pure virtual.

Cabbage (public Projectile)

1. The constructor takes in a pointer to the StudentWorld the cabbage is in and the location. It then passes the identification (ID, position, direction, etc.) to the Alien base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a doSomething() that checks if the cabbage is dead, goes off screen, or collide with an alien. It does the above actions accordingly. Otherwise, it moves the cabbage forward.

Turnip (public Projectile)

1. The constructor takes in a pointer to the StudentWorld the turnip is in and the location. It then passes the identification (ID, position, direction, etc.) to the Alien base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a doSomething() that checks if the turnip is dead, goes off screen, or collide with the player. It does the above actions accordingly. Otherwise, it moves the turnip forward.

FlatulenceTorpedo (public Projectile)

1. The constructor takes in a pointer to the StudentWorld the Flatulence Torpedo is in and the location. It then passes the identification (ID, position, direction, etc.) to the Alien base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a doSomething() that checks if the Flatulence Torpedo is dead, goes off screen, or collide with an alien or with the player. It does the above actions accordingly. It then moves the Flatulence Torpedo forward.

Explosion (public Actor)

1. The constructor takes in a pointer to the StudentWorld the explosion is in and the location. It then passes the identification (ID, position, direction, etc.) to the Actor base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a doSomething() that sets the explosion to dead after enlarging four times.

Goodie (public Actor)

1. The constructor takes in a pointer to the StudentWorld the goodie is in, its ID, and the location. It then passes the identification (ID, position, direction, etc.) to the Actor base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a collideWPlayer() that checks if the goodie collides with the player. The player will increase score and the goodie will die, then the afterCollide()of that goodie is called. Otherwise, the goodie will continually moving in its direction.
4. I have a pure virtual afterCollide() because each type of goodie has different effects on the player.
5. I have a doSomething() that checks if the goodie is dead, on screen, or collide with the player.

RepairG (public Goodie)

1. The constructor takes in a pointer to the StudentWorld the goodie is in and the location. It then passes the identification (ID, position, direction, etc.) to the Goodie base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a afterCollide() that increase the player’s hit points.

ExtraLifeG (public Goodie)

1. The constructor takes in a pointer to the StudentWorld the goodie is in and the location. It then passes the identification (ID, position, direction, etc.) to the Goodie base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a afterCollide() that increase the player’s lives.

FlatulenceTorpedoG (public Goodie)

1. The constructor takes in a pointer to the StudentWorld the goodie is in and the location. It then passes the identification (ID, position, direction, etc.) to the Goodie base class.
2. The destructor is virtual because of inheritance. It does nothing.
3. I have a afterCollide() that increase the player’s number of Flatulence Torpedoes.

StudentWorld

1. The constructor passes its parameters to its GameWorld base class.
2. The destructor calls cleanUp().
3. I have a addActor() that push new actor onto the vector.
4. Other non-trivial functions see the pseudocodes below.

Important pseudocodes

int StudentWorld::init()

create the player and 30 stars

initialize to track the number of aliens to know when to add new ones

int StudentWorld::move()

let the player and each actor do something

introduce new stars by randomly generating them

remove dead game objects

update display text

void StudentWorld::cleanUp()

if the actor vector is not empty (prevent calling cleanup twice)

set the player each actor pointer to nullptr

clear the vector

delete the player pointer

Actor\* StudentWorld::getOneCollidingAlien(const Actor\* a)

Go through the actor vector and check if it is at the same position as the actor passed in

if yes

return that pointer

otherwise

return nullptr

NachenBlaster\* StudentWorld::getCollidingPlayer(const Actor\* a) const

check if the actor passed in is at the same location as the player

if yes

return the player pointer

otherwise

return nullptr

void NachenBlaster::doSomething()

check if the player is dead or should be dead

determine if the user is moving the player or firing

do actions accordingly

add cabbages

void Star::doSomething()

if the star goes off the screen

remove it

move

void Alien::collideWProjectile(const int pt)

decrement the hit points of the alien

if the alien has no hit points left

the alien dies

create an explosion

do the things each type of alien behave differentlu

otherwise, the alien suffers a blast

void Alien::collideWPlayer(Actor\* a, int hitPt)

if collide with the player

decrement the player’s hit points

the alien dies

create an explosion

do the things different aliens do differently

void Smallgon::afterCollide()

if this function gets called, this alien is already dead. Thus increment the player’s score

void Smallgon::doSomething()

determine if the alien is dead or should be dead

check if collide with the player

if need to change flight plan, change accordingly

determine if need to fire a turnip at the player

move

void Smoregon::afterCollide()

if this function gets called, this alien is already dead. Thus increment the player’s score

decide if to drop a goodie

void Smoregon::doSomething()

determine if the alien is dead or should be dead

check if collide with the player

if need to change flight plan, change accordingly

determine if need to fire a turnip at the player

move

void Snagglegon::afterCollide()

if this function gets called, this alien is already dead. Thus increment the player’s score

decide if to drop a goodie

void Snagglegon::doSomething()

determine if the alien is dead or should be dead

check if collides with the player

if need to change flight plan, change accordingly

determine if need to fire a flatulence torpedo at the player

move

void Cabbage::doSomething()

determine if the projectile is dead or should be dead

if collides with an alien

die

otherwise

move and rotate

void Turnip::doSomething()

determine if the projectile is dead or should be dead

if collides with the player

the player loses hit points

die

otherwise

move and rotate

void FlatulenceTorpedo::doSomething()

determine if the projectile is dead or should be dead

if is fired by the player and collides with an alien

die

if is fired by an alien and collide with the player

the player loses hit points

die

move towards its enemy

void Explosion::doSomething()

increase its size by 1.5 times each time

die after four times

void Goodie::collideWPlayer()

if collide with the player

decrement the player’s hit points

the alien dies

create an explosion

do the things different goodies do differently

otherwise

move

void Goodie::doSomething()

determine if the goodie is dead or should be dead

check if it collide with the player

Bugs

I haven’t spot any bugs now. All have fixed.

~~Explosions takes place behind the player.~~

~~The program might crash at some time during the game. I’ve experienced it crashing during level 5, 6 and 11. The problem might be in the iterators while tracing through the actor vector. But mostly the program behaves well and there are no spotted bugs.~~

Design decisions and assumptions

* I used a vector instead of a list, because I think there is a lot of trace-through. Vectors are easier to do that than a list since they are in an order.
* For collisions, the spec says check for collision again after moving. But since the next tick will check for collision immediately, there is no need to spend time checking at the end of this tick again.
* Computing the maximum number of aliens that should be on the screen involves a multiplication with 0.5, so the answer can be a double. But since the number of aliens can’t be a fraction, I use an int to store this maximum number.

Testing descriptions

Actor (public GraphObject)

Since this is an abstract base class, many functions are used in other classes as well. Thus, I only tested if an actor can successfully pass its parameters to its base class’s GraphObject constructor, and the StudentWorld pointer is correctly pointing to the student world by running the program.

NachenBlaster (public Actor)

* I tested if the player can be correctly created. If it appears and can do actions, it is created.
* I tested if the player can move as will by playing the game and moving the player.
* I tested if the player can fire a cabbage or a flatulence torpedo by playing the game, firing cabbages, and picking up flatulence torpedo goodies to fire flatulence torpedoes.
* I tested if the number of flatulence torpedoes will be cleared when proceeding to the next level.
* I tested if the player can fire a flatulence torpedo when it doesn’t have one.
* I also tested if the player’s data (hit points, flatulence torpedoes, etc.) is correctly added and used by playing the game and counting by myself to compare with the displayed messages.
* will produce sound accordingly

Star (public Actor)

* I tested if stars are created when the game begins.
* I tested if every star created later is appearing from the left side of the screen.

Alien (public Actor)

Since this is an abstract base class, I only tested if an alien can be successfully created by passing its parameters to its Actor base class. The rest of the functions are tested in its derived classes.

Smallgon (public Alien)

By playing the game and check if the intended actions occur, I tested if a smallgon

* + can fire turnips
  + can collide with the player or player fired projectiles
  + can change directions correctly
  + will not go off screen except when hitting the left side of the screen
  + will be removed when it gets hit
  + will increase the player’s score by a correct amount when colliding or when it dies
  + will produce sound accordingly

Smoregon (public Alien)

By playing the game and check if the intended actions occur, I tested if a smoregon

* + can drop goodies
  + can fire turnips
  + can collide with the player or player fired projectiles
  + can change directions correctly
  + can increase its speed when on the same line as the player
  + will not go off screen except when hitting the left side of the screen
  + will be removed when it gets hit
  + will increase the player’s score by a correct amount when colliding or when it dies
  + will produce sound accordingly

Snagglegon (public Alien)

By playing the game and check if the intended actions occur, I tested if a snagglegon

* + can drop goodies
  + can fire flatulence torpedoes with the correct direction
  + can collide with the player or player fired projectiles
  + can change directions correctly
  + will not go off screen except when hitting the left side of the screen
  + will be removed when it gets hit
  + will increase the player’s score by a correct amount when colliding or when it dies
  + will produce sound accordingly

Projectile (public Actor)

Since this is an abstract base class, I only needed to test if it can pass the parameters to its base class’s Actor constructor correctly.

Cabbage (public Projectile)

By playing the game and check if the intended actions occur, I tested if a cabbage

* + can move forward
  + can collide with an alien
  + will produce sound accordingly
  + is rotating
  + will be removed after collision

Turnip (public Projectile)

By playing the game and check if the intended actions occur, I tested if a turnip

* + can move forward
  + can collide with the player
  + will produce sound accordingly
  + is rotating
  + will be removed after collision

Flatulence Torpedo (public Projectile)

By playing the game and check if the intended actions occur, I tested if a flatulence torpedo

* + can move forward
  + has the correct direction depending on from where it is fired
  + can correctly collide with the player or an alien according to from where it is fired
  + will produce sound accordingly
  + will be removed after collision

Explosion (public Actor)

By playing the game and check if the intended actions occur, I tested if an explosion

* + can take place at the right location
  + takes place in front of the player
  + will increase in size
  + will produce sound accordingly

Goodie (public Actor)

Since this is an abstract base class, I only tested if an goodie can be successfully created by passing its parameters to its Actor base class. The rest of the functions are tested in its derived classes.

Repair goodie (public Goodie)

By playing the game and check if the intended actions occur, I tested if this goodie

* + is dropped at the right location
  + can move correctly
  + can be picked up and will disappear
  + will add scores to the player
  + can add hit points to the player

Extra life goodie (public Goodie)

By playing the game and check if the intended actions occur, I tested if this goodie

* + is dropped at the right location
  + can move correctly
  + can be picked up and will disappear
  + will add scores to the player
  + can add another life to the player.
  + The added life will not be reset at the next level.

Flatulence torpedo goodie (public Goodie)

By playing the game and check if the intended actions occur, I tested if this goodie

* + is dropped at the right location
  + can move correctly
  + can be picked up and will disappear
  + will add scores to the player
  + can add 5 more flatulence torpedoes to the player.
  + The added flatulence torpedoes will be reset at the next level.

Student World

By playing the game and check if the intended actions occur, I tested if

* + the displayed messages:
    - are of correct amount
    - the number of cabbages will not be negative
    - extra lives will not be rest when proceeding to the next level
    - flatulence torpedoes will be reset when proceeding to the next level
    - the spaces between each item is 2
  + colliding with aliens and the player will take place correctly
  + the game is properly initialized
    - with 30 stars and the player at first
    - with the amount of each item is correct
  + each actor has something to do during every moment (tick)
  + the world is properly cleaned up after the game ends by checking if the game crashes when I exist using different methods (quit, close the window, etc.)