Omar Ahmad

CS project 3 report

Prof. Smallberg lecture 2



* class Actor : public GraphObject
  + Actor(int imageID, int x, int y, int dir, double size, int depth, StudentWorld\* world = nullptr):
    - calls GraphObject’s constructor, sets m\_alive to true, and m\_world to world
  + virtual ~Actor();
    - empty
  + void setAlive(bool living);
    - sets m\_alive to living
  + virtual bool isAlive() const;
    - returns m\_alive
    - virtual because the healthObject class redefines it to include if object’s health is greater than 0
    - all Actors need to know if they’re alive so that the game can destroy the ones that are no longer considered alive
  + bool isAliveAndInBounds() const;
    - returns true if alive and in bounds
  + virtual void doSomething() = 0;
    - pure virtual because don’t have any objects that are strictly Actors
  + bool checkCollision(Actor\* a, Actor\* b);
    - checks if actor a and b are colliding with each other
  + StudentWorld\* getWorld() const;
    - returns the world the object is in
    - used to call StudentWorld functions

* class healthObject : public Actor
  + healthObject(int imageID, int x, int y, int dir, double size, int depth, int health, StudentWorld\* world);
    - calls Actor’s constructor and sets m\_health (hit points) to 0
  + virtual ~healthObject();
    - empty
  + int Health() const;
    - returns m\_health
  + void setHealth(int health);
    - sets m\_health to health
  + void sufferDamage(int damage);
    - decrements m\_health by damage
    - use this instead of sethealth when attacking a healthObject
  + virtual bool isAlive() const;
    - redefined to return true only if m\_health is greater than 0 and m\_alive is true
  + virtual void doSomething() = 0;
    - pure virtual because don’t have any objects that are strictly healthObjects
* class nachenBlaster : public healthObject
  + nachenBlaster(StudentWorld\* world = nullptr);
    - calls healthObject’s constructor and sets m\_cabbageEnergy to 30 and m\_nTorpedoes to 0
  + virtual ~nachenBlaster();
    - empty
  + virtual void doSomething();
    - Pseudocode:

if ( not alive )

return

if (user pressed key ch)

{

Switch (ch)

{

case right:

if can move right, move right

break

case left:

if can move left, move left

break

case up:

if can move up, move up

break

case down:

if can move down, move down

break

case space:

if m\_cabbageEnergy is greater than 5

decrement 5 from m\_cabbageEnergy

fire a cabbage 12 units to the right of the blaster

break

case tab:

if m\_nTorpedoes is greater than 0

decrement m\_nTorpedoes

fire a torpedo 12 units to the right of the blaster

break

}

}

* + int cabEnergy() const;
    - returns m\_cabbageEnergy
  + int nTorps() const;
    - returns m\_nTorpedoes
  + void incTorps(int amt);
    - increases m\_nTorpedoes by amt
* class alien : public healthObject
  + alien(int imageID, int x, int y, int health, StudentWorld\* world, double speed, int type = 0, int initDir = 0);
    - calls healthObject’s constructor, sets m\_travelspeed to speed, m\_travelDirection to initDir, and m\_type to type
    - type refers to the type of alien ( 0 = smallgon, 1 = smoregon, and 2 = snagglegon)
    - calls the addActor, addAlien, and addShip functions from StudentWorld
    - these add the alien to the list of actors, the list of aliens and increases the counter for aliens that are alive
  + virtual ~alien();
    - calls decShip from StudentWorld to decrease the ship counter
    - if the health of the alien is below 0 (meaning the alien was killed and did not drift off the screen), the destroyShip function from StudentWorld is called to increment the counter for number of ships the user has destroyed
  + virtual void doSomething();
    - sets m\_alive to false if out of bounds and returns if not alive
    - returns if checkCollideWithBlaster() which checks if the alien has collided with the nachenBlaster (and if so damages the blaster and kills the alien) returns true
    - calls newFPL() which sets a new flight path and/or length
    - calls attack() which attacks according to the type of alien
    - calls move() which moves according to the flight plan and travel speed
    - calls checkCollideWithBlaster() again
  + void dropGoodie();
    - causes alien to drop goodie based off what type of alien it is
  + int type() const;
    - returns m\_type
* class Smallgon : public alien
  + Smallgon(int x, int y, StudentWorld\* world);
    - calls Alien’s constructor
  + virtual ~Smallgon();
    - empty
* class Smoregon : public alien
  + Smoregon(int x, int y, StudentWorld\* world);
    - calls Alien’s constructor
  + virtual ~Smoregon();
    - empty
* class Snagglegon : public alien
  + Snagglegon(int x, int y, StudentWorld\* world);
    - calls Alien’s constructor
  + virtual ~Snagglegon();
    - empty
* class projectile : public Actor
  + projectile(int imageID, int x, int y, int dir, double size, int depth, StudentWorld\* world, int type);
    - calls Actor’s constructor and sets m\_type to type
    - type refers to the type of projectile (0 = cabbage, 1 = turnip, and 2 = torpedo)
    - calls addActor to add the projectile to the actor list
  + virtual ~projectile();
    - empty
  + virtual void doSomething();
    - Pseudocode:

If (not alive or not in bounds)

set m\_alive to false and return

if (cabbage)

call checkCollideWithAlien which calls StudentWorld’s checkCollisionAlienProjectile

return

if (turnip)

call checkCollideWithBlaster which calls StudentWorld’s checkCollisionBlasterActor

returns

if (torpedo)

if direction = 0

call checkCollideWithAlien

else call checkCollideWithBlaster

move according to type

check collisions again

* class cabbage : public projectile
  + cabbage(int x, int y, StudentWorld\* world);
    - calls projectile’s constructor
    - plays sound SOUND\_PLAYER\_SHOOT
  + virtual ~cabbage();
    - empty
* class torpedo : public projectile
  + torpedo(int x, int y, int dir, StudentWorld\* world);
    - calls projectile’s constructor
    - plays sound SOUND\_TORPEDO
  + virtual ~torpedo();
    - empty
* class turnip : public projectile
  + turnip(int x, int y, StudentWorld\* world);
    - calls projectile’s constructor
    - plays sound SOUND\_ALIEN\_SHOOT
  + virtual ~turnip();
    - empty
* class explosion : public Actor
  + explosion(int x, int y, StudentWorld\* world);
    - calls Actor’s constructor and set’s lifeLength to 0
    - calls StudentWorld’s addActor and plays sound SOUND\_DEATH
  + virtual ~explosion();
    - empty
  + virtual void doSomething();
    - increments lifeLength and increases its size by a factor of 1.5
    - calls setAlive(false) if lifeLength is greater than 3
* class Star : public Actor
  + Star(int x, int y, StudentWorld\* world);
    - calls Actor’s constructor
    - calls studentWorld’s addActor
  + virtual ~Star();
    - empty
  + virtual void doSomething();
    - moves one pixel left
    - if out of bounds sets m\_alive to false
* class goodie : public Actor
  + goodie(int imageID, int x, int y, StudentWorld\* world, int type);
    - calls Actor’s constructor and sets m\_type to type
    - type refers to the type of goodie (0 = life, 1 = repair, and 2 = torpedo)
    - calls StudentWorld’s addActor and plays sound SOUND\_GOODIE
  + virtual ~goodie();
    - empty
  + virtual void doSomething();
    - sets alive to false and returns if not alive or not in bounds
    - returns if collided with blaster
    - moves to bottom left 0.75 pixels
    - checks if collided with blaster again
* class lifeGoodie : public goodie
  + lifeGoodie(int x, int y, StudentWorld\* world);
    - calls goodie’s constuctor
  + virtual ~lifeGoodie();
    - empty
* class repairGoodie : public goodie
  + repairGoodie(int x, int y, StudentWorld\* world);
    - calls goodie’s constructor
  + virtual ~repairGoodie();
    - empty
* class torpedoGoodie : public goodie
  + torpedoGoodie(int x, int y, StudentWorld\* world);
    - calls goodie’s constructor
  + virtual ~torpedoGoodie();
    - empty
* class StudentWorld : public GameWorld
  + StudentWorld(std::string assetDir);
    - sets user to nullptr, nShips (# of aliens alive) to 0, and nDestroyed (# of aliens killed by the user) to 0
  + ~StudentWorld();
    - calls cleanUp
  + virtual int init();
    - sets user to new nachenBlaster
    - creates field of thirty stars
    - sets nShips and nDestroyed to 0
  + virtual int move();
    - called every 1/20th of a second by game framework
    - Pseudocode:

if (user is alive)

call user’s doSomething

else

call decLives to decrement the number of lives

return GWSTATUS\_PLAYER\_DIED

repeatedly, for (all members of the list of all actors)

if ( alive)

doSomething

repeatedly, for (all members of the list of all aliens)

if (not alive)

erase from aliens list

repeatedly, for (all members of the list of all actors)

if(not alive)

delete actor

erase from actors list

have 15% chance of creating a new star at a random height

if (not enough alien on screen)

add new alien according to probabilities in spec

return GWSTATUS\_FINISHED\_LEVEL if killed enough aliens

sets a stringstream to stats data

sets the game stat text to string of stringstream

* + virtual void cleanUp();
    - deletes user and sets to nullptr if not already set to nullptr
    - deletes all actors in actors list and erases their pointers
    - erases all pointers in aliens list
  + template<typename T>

void addActor(T\* actor);

* + - pushes actor onto actors list
  + template<typename A>

void addAlien(A\* alien);

* + - pushes alien onto aliens list
  + void addShip();
    - increments nShips
  + void decShip();
    - decrements nShips
  + void destroyShip();
    - increments nDestroyed
  + void userPos(int& x, int& y);
    - sets x to the nachenBlaster’s x and y to its y
  + bool checkCollisionAlienProjectile(int damage, projectile\* projectile);
    - returns whether the projectile collided with any alien in the aliens list
    - if so, damages the aliens
    - if they died, increases the score, creates a new explosion, and asks the alien to drop a goodie
  + bool checkCollisionBlasterActor(Actor\* threat, int damage = 0, int amtInc = 0, int goodieType = -1);
    - returns whether the actor has collided with the nachenBlaster
    - if so, sets the actor to dead, causes damage to the nachenBlaster, increases the score by amtInc, and applies effects of the corresponding goodie (goodtype of -1 means no goodie and no goodie effects)

1. I am unaware of any functionality I didn’t implement or any bugs

* Decided to have goodies handle blaster – goodie interactions
* Decided to have aliens handle blaster – alien interactions
* Decided to have turnips handle blaster – turnip interactions
* Decided to have torpedoes handle blaster – torpedo interactions
* Decided to have cabbages handle alien – cabbage interactions
* Decided to have turnips handle alien – turnip interactions
* Decided to have goodie’s, explosion’s, and all projectiles’ constructors handle playing their respective sounds
* Decided to have alien’s constructor and destructor handle the current aliens and killed aliens counters



Actor: I tested the actor class by creating an object out of bounds to test the isAliveAndInBounds function. Then I created an actor and set it to dead to test the alive functions. I created sets of two objects at various distances from each other to test the checkCollision function.

healthObject: I tested the healthObject class by temporarily defining its doSomething function to create an actual heathObject. Then I set the health to specific values to test the isAlive function and the other health related functions,

nachenBlaster: I tested the nachenBlaster class by building the code and seeing if the blaster would react how the spec said it should. I made sure the cabbages and torpedoes would only fire when they had enough of their respective ammo.

alien: I tested the alien class by building the program and making sure the aliens behaved as described in the specification and in the sample game. I made sure the aliens fired their respective weapons at the correct frequencies and that they performed the correct actions. I made sure they moved in the correct manner and at the correct speed as well. I also tested to make sure the aliens would drop the correct goodies and at the correct frequencies.

Smallgon: I tested the Smallgon class by making sure its movement, speed, and attack method/frequency matched the specification. I created a non-moving Smallgon to make it easier to test the attack mechanism.

Smoregon: I tested the Smoregon class by making sure its movement, speed, and attack method/frequency matched the specification, I created a non-moving Smoregon to make it easier to test the attack and charging mechanism.

Snagglegon: I tested the Snagglegon class by making sure its movement, speed, and attack method/frequency matched the specification. I created a non-moving Snagglegon make it easier to test the attack mechanism.

projectile: I tested the projectile class by testing if its doSomething did the correct things for the correct types of projectiles.

cabbage: I tested the cabbage class by making sure the cabbages moved the correct amounts at the correct speed and rotated the correct amount in the correct direction. I also made sure cabbage’s constructor correctly played the correct sound.

turnip: I tested the turnip class by making sure turnips moved the correct amounts at the correct speed and rotated the correct amount in the correct direction. I also made sure turnip’s constructor correctly played the correct sound.

torpedo: I tested the torpedo class by making sure they moved the correct amounts at the correct speed and didn’t rotate. I made sure they worked correctly if they were fired by the nachenBlaster and a snagglegon. I also made sure turnip’s constructor correctly played the correct sound.

explosion: I tested the explosion class by creating an explosion in the middle of the screen and watching frame by frame, making sure the explosion increased in size for four frames and then disappeared.

Star: I tested the star class by having thirty stars create randomly on the screen and move towards the left, not interacting with any other objects.

goodie: I tested the goodie class by creating a random goodie in the middle of the field and making sure it had the correct effects when a blaster collided with it

lifeGoodie: I tested the lifeGoodie class by creating lifeGoodies in the game and making sure they had their desired effects.

repairGoodie: I tested the repairGoodie class by creating repairGoodies in the game and making sure they had their desired effects.

torpedoGoodie: I tested the torpedoGoodie class by creating torpedoGoodies in the game and making sure they had their desired effects (+5 torpedos).

StudentWorld: I tested the StundentWorld class using the sanity checker for part 1 and 2 to make sure it destructed objects correctly. I tested the init, and move functions by playing the game and stepping through one tick at a time to make sure objects were created at the right times, and collisions were handled correctly. I tested the addShip, decShip, and destroyShip functions by making sure new aliens were created with the correct frequencies. I tested the collision functions by making sure collisions were handled correctly when objects were in collision range.