Jaden Booher

405-333-407

Project 3 Writeup

1. A high-level description of each of your public member functions in each of your classes, and why you chose to define each member function in its host class; also explain why (or why not) you decided to make each function virtual or pure virtual.

//StudentWorld Class

virtual int init();

-Initializes all dirt, food, pits, and Socrates

virtual int move();

-Moves all actors using an iterator and a vector for all actors, and Socrates, deletes dead actors, adds new actors, sets game stats, checks if Socrates is alive, and checks if all bacteria are dead

virtual void cleanUp();

-Deletes all actors upon winning a level, loosing a level, or loosing all 3 lives and the game ending

void scoreBoard();

-Uses the “setGameStatsText()” command and string streams to create the scoreboard that record she number of lives, ammo, score, and health

void addToVec(Actor\* temp);

-Used to add a new actor, such as a new bacteria spawn, to the vector of actors so that it can be called in the move function with the other actors

void randPoint(int& x, int& y, int radius);

-Used to create a random point given the desired radius; if given 128, it creates a random x and y value with a 128 radius; if given any other value, it creates a random point within the radius

bool overlaps(int x, int y);

-Passes in an x and y value, creates by the randPoint function, and compares all actors to its location to see if any overlap; passes false if an item overlaps it

void addNewActors();

-Called by the move() function with a percent chance to add new actors, such as goodies or fungus

void addDirt();

-Called by the init() function to add the initial amount of dirt based on the level of the game

void addFood();

-Called by the init() function to add the initial amount of food based on the level of the game

void addPits();

-Called by the init() function to add the initial amount of pits based on the level of the game

void deleteDead();

-Called by the move() function to delete the actors that are declared “dead” and removes them from the vector and from the game as a whole

bool dirtOverlap(const Actor\* temp);

-Called by the Bacteria class to check if dirt gets in the way of movement for an individual bacteria and returns true if it overlaps

void addBacteria(int num);

-Adjusts the int of the number of bacteria supposed to be spawning per round; if it reaches 0, the round ends

Socrates\* getPlayer();

-Returns the pointer to Socrates

Socrates\* overlapSocrates(const Actor\* temp) const;

-Compares the actor passed into the function to the position of Socrates and checks if it overlaps; if the variable overlaps, it returns a pointer to Socrates; if not, returns a nullptr

Actor\* findMove(const Actor\* temp, std::string type) const;

-Function passes a pointer to an actor and a string, which is used to determine what variation of the find move function is called; the function is used to find the location of the nearest food for bacteria, used to see if an enemy overlaps food, and used to see if “flame” or “spray” overlap an enemy

//Actor base class

virtual void doSomething();

-The function called by most of the base classes, but not all, like bacteria and food, so that is the reason it is not a pure virtual function

bool isAlive();

-Returns the private member variable bool function about whether an actor is alive; this is used by every actor to determine if it is alive or not and if it needs to be removed from the game

void kill();

-Sets the private member variable determining if an actor is alive to false

void setType(std::string type);

-Sets the private member variable of the actor class “m\_type” to the string that is set for each individual type of actor; used to find only certain actors for some functions, such as excluding type “food” from being damaged by “flame” or “spray”

std::string getType();

-Returns the private member variable string “m\_tpye” of what type of actor each individual actor is

virtual void takeDamage(int damage) {};

-Used to adjust the private member variable of an actor’s health “m\_health” by whatever integer is passed; is virtual because some actors have different specifications on what happens when you die; not pure virtual because some actors do not have health and don’t call “takeDamage”

int getHealth();

-Returns the private member variable “m\_health”

void setHealth(int health);

-Sets the private member variable “m\_health” to the passed integer

int getAngle();

-Returns the “m\_angle” private member variable that all actors have;

void setAngle(int angle);

-Sets the “m\_angle” function to the passed integer

void setDamage(int damage);

-Sets the “m\_damage” private member variable to the passed integer; “m\_damage” is assigned per bacteria, weapons, and fungus and tells the game how much to damage Socrates

int getDamage();

-Returns the “m\_damage” integer when called

StudentWorld\* myWorld();

-Returns the pointer to the “StudentWorld” game

//Dirt class : Actor

virtual void takeDamage(int damage);

-Specific to dirt: since dirt is set to either alive or death, whenever it takes any damage it dies, so the kill function is called within

//Socrates class :Actor

virtual void doSomething();

-Specific to Socrates: first checks if Socrates is alive, then takes a keyboard press using the “getKey” function; Upon arrow key press, Socrates either moves by increments of 5 degrees by moving Socrates to the middle, changing angle, moving to the new point, then changing the angle again; if space is pressed, it creates a new “spray” actor, adds it to the actor vector, and plays the spray sound; if enter is pressed, it creates 16 new “flame” actors in increments of 22 degrees, adds them to the actor vector, and plays the flame sound; if no key is pressed, the count of “m\_spray” (the ammo left for spray) is increased by one

void addFlameAmmo(int ammo);

-Adds the passed integer to the “m\_flaame” to indicate how much flame ammo is left

virtual void takeDamage(int damage);

-Gets Socrates health and damages him the passed amount; plays Sorates’ hurt sound; if Socrates has 0 or less health, the kill() function is called

//Food class : Actor

No public member functions declared (is killed by bacteria if eaten)

//Goodie base class : Actor

virtual void doSomething();

-Checks if the goodie is alive; calls the “ifOverlap()” function to check if Socrates overlaps, returns if it does; checks the life span of the goodie by comparing it to the max ticks alive per goodie (initialized in each goodies’ constructor) and adding a tick every time “doSOmething()” is called

virtual void takeDamage(int num);

-(only applies to Fungus) if shot, the kill() command is called

bool ifOverlap();

-Calls the “overlapSocrates()” function declared in “StudnetWorld” to check if Socrates overlaps with any of the goodies; if Socrates does, the function calls “getType()” and checks the type of each goodie, each goodie/ fungus doing different things and awarding different points; damages player if it is a fungus

//restoreHealthGoodie class : Goodie

Bacteria class handles everything

//extraLifeGoodie class: Goodie

Bacteria class handles everything

//flameThrowerGoodie class

Bacteria class handles everything

//fungus class: goodie

Bacteria class handles everything

//Weapon class : Actor

bool ifOverlap(int damage);

-Calls the “findMove()” function in “StudentWorld” to see if any valid actor overlaps the weapon; if it overlaps, it makes the actor take damage and kills the weapon

void setMaxPixels(int max);

-Sets the “m\_maxPixels” to the pixels passed; this function is called by weapon subclass constructors to initialize how far they are supposed to move

virtual void doSomething();

-Checks if weapon is alive; checks if overlaps with any enemies; checks if max range has been achieved, if not, move the weapon; if max pixels are reached, kill the weapon

//Flame class : Weapon

Weapon class handles everything

//Spray class : weapon

Weapon class handles everything

void takeDamage(int damage);

-Subtract the passed integer from “m\_health” and play damage sound; if health is 0, call the “killBacteria()” command

void setHurtSound(int sound);

-Called by contructor to set the hurt sound per bacteria

void setDeathSound(int sound);

-Called by contructor to set the death sound per bacteria

int getSound(std::string type);

-Returns the type of sound per bacteria and type of sound base of what string is passed

void setFood(int food);

-Set “m\_food” for how many food objects an individual bacterium has eaten

int getFood();

-Returns “m\_food” how many food objects an individual bacterium has eaten

void setMovePlan(int plan);

-Sets the number of “m\_movement” to the integer passed; “m\_movement” is the number int for move plan as per the spec

bool moveAroundDirt(int angle, std::string type);

-Moves the bacteria forward alls the “dirtOverlap()” function created in “StudentWorld” to check overlap; after, moves the bacteria back to its original position and returns true or false on whether bacteria is in the way; if bacteria is in the way it creates a new random direction and sets the move plan to 10

void moveToFood();

-Calls the “findMove” function to find the nearest food; if none are found a random direction and set move plan to 10; else get the coordinates of the nearest food and test the radius in 360 degrees, testing every 1 degree to find the nearest point, then setting its angle to the point closest to the food (which overlaps the food)

virtual bool regularMovement();

-Movement performed by both the salmonella and then overwritten by “Ecoli”; increments movement plan and moves in direction already moving in and calls “moveAroundDirt()” function to check if dirt is in the way; otherwise, move to the nearest food using the “moveToFood()” command

void killBacteria();

-Increase score by 100 and play death sound set by bacteria using the “getSound()” function and kill the bacteria; use “randInt()” to create a 50% chance to spawn food upon death at the x and y coordinates of the bacteria killed

bool ifOverlapSocrates();

-Calls “StudentWorld” “overlapSocrates” function and damages Socrates by set damage which is given by “getDamage()”

void ifOverlapFood();

-calls the “StudentWorld” findMove()” function; if overlaps; the food is killed and it increases the food count for the bacteria using the “setFood()” command

virtual void doSomething();

-First chekcs if it is alive; next it calls the “agroSalMove()” function, which passes false unless passed when “agroSal” moves towards Socrates; next checks if it overlaps Socrates using the :ifOverlapSocrates()” function; then checks how much food has been eaten and spawns the correct bacteria that has eaten 3; then checks if overlaps food using the “ifOverlapFood()” command; finally, calles the “regularMovement” command for everything unless the “agroSalMove()” command returns true;

virtual bool agroSalMove();

-Redefined by AgroSalmonella; returns false by default

//RegSalmonella class : Bacteria

virtual bool agroSalMove();

-Gets position of the player and checks if radius is under 72 pixels; if is test every degree for 360 degreees given the radius and find the nearest point and have it move towards it; call the :moveAroundDirt()” command and if it fails, the agroSalmonella stays still; returns false unless Socrates is found

//AgroSalmonellaclass : Bacteria

virtual bool agroSalMove();

-If Socrates is within 256 pixels, get Socrates position and angle by checking every degree up to 360 degrees and returning the nearest point and angle; if it is blocked by dirt, it increments itself by 10 degrees 10 times to move around and if it cannot it will stay still

1. A list of all functionality that you failed to finish as well as known bugs in your classes

* Whenever I create “flames” my game tends to lag and/or stutter
* Everything works as specified by the spec

1. A list of other design decisions and assumptions you made

* Assumed that other key presses specified in the spec, such as q, were already implemented; implemented the arrow and shoot keys on my own per the spec
* When spawning an item with a 128 pixel radius, it is practically impossible without using trig and doubles to get the nearest “random” spot with a radius of 128; I would use doubles and trig then make the radius an integer to round to 128, otherwise other “random” spots would be impossible to find
* Assumed that there was no max ammo for the flame thrower, on account that he picks up the “flamethrowerGoodie”, adding 5 ammo regardless what ammo Socrates has before

1. A description of how you tested each of your classes
   1. class StudentWorld : public GameWorld
      1. Tested the number of dirt, foods, and pits per level without spawning any enemies
      2. Only spawned dirt and tested both weapons on how they interact with it
      3. Spawned all the items unable to be damaged by both weapons and tested their interactions
      4. Tested “addNewFactor()” function by upping the spawn rate and seeing if everything spawned at the 128 radius and that Socrates was able to interact with them
      5. Spawned only one bacteria and killed it to see if the end level command and deconstruction would work correctly
      6. Spawned one bacteria and no other actors to see how it would interact with Socrates using the “overlapSocrates()” command
      7. Spawned an increased number of dirt and no other actors to test the “randPoint()” command and the “overlaps” command
   2. class Actor : public GraphObject
      1. Would spawn just one dirt and test the “isAlive()” command and the “kill()” command on it
      2. Spawned one bacteria and testes the “getHealth(),” “setHealth(),” “getAngle(),” “setAngle(),” “getDamage(),” and “setDamage()” functions on it individually
      3. Spawned a dirt and bacteria to test that each virtual implementation of “doSomething()” and “takeDamage()” had their own implementations and did not call “Actros” version of it
   3. class Dirt : public Actor
      1. Called “init()” “addDirt()” command and then tested how it was killed by having Socrates shoot it with both flame and spray
   4. class Socrates : public Actor
      1. Spawned Socrates as the only actor and had him move around using the “doSomething()” command in addition to shooting both flames and spray
      2. Spawned Socrates with just 1 bacteria and had him take damage to test the “takedamge()” function redefined in this subclass
      3. Spawned dirt and then all other non-damageable objects to test the weapons on
   5. class Food : public Actor
      1. Spawned a large number of food and dirt to check that no overlapping was happening and then tested weapons on the food to make sure they did not interact with each other
      2. Spawned 1 bacteria and a large sum of food to see if the bacteria would pick it up
   6. class Goodie : public Actor
      1. Spawned just Socrates and used the “addNewActors()” function in “StudentWorld” to see how Socrates interacted with any of the goodies using the “ifOverlap()” command
      2. Spawned only Socrates and Fungus to test the “takeDamage()” function when Socrates shoots it
   7. class extraLifehGoodie : public Goodie
      1. Spawns only Socrates and the extra life goodie to test their interaction and make sure that it added a life to Socrates while also playing the “gotGoodie” sound and killing the goodie
   8. class flameThrowerGoodie : public Goodie
      1. Spawns only Socrates and the flame thrower goodie to test their interaction and make sure that it adds 5 flame to Socrates while also playing the “gotGoodie” sound and killing the goodie
   9. class restoreHealthGoodie : public Goodie
      1. Spawns only Socrates and the restore health goodie to test their interaction and make sure that makes Socrates health 100 while also playing the “gotGoodie” sound and killing the goodie
   10. class Fungus : public Goodie
       1. Spawns only Socrates and the restore health goodie to test their interaction and make sure that makes Socrates takes damage and Socrates 50 points from his while also playing the “hurtSocrates” sound and killing the goodie; also tested that Socrates could damage and kill the Fungus
   11. class Weapon : public Actor
       1. Spawned dirt and then all other non-damageable objects to test the weapons on
       2. Spawned each individual type of bacteria and fungus to test that the weapons could damage and kill each of them
       3. Spawned Socrates alone to test the distance of each of the projectiles and test how they interact when not hitting any objects
   12. class Flame : public Weapon
       1. All testing done by “Weapon” and “Socrates” class
   13. class Spray : public Weapon
       1. All testing done by “Weapon” and “Socrates” class
   14. class Pit : public Actor
       1. Only Spawned pits and no oter actors to make sure that it was randomly being spawned
       2. Spawned pits and all other actors besides bacteria to test random location and no overlap
       3. Just spawned the pit and ran its “doSoemthing()” function to test the random spawn times of bacteria, test spawn location of bacteria (on top of the pit), and test that the pit dies once the certain number of each type of bacteria is spawned
   15. class Bacteria : public Actor
       1. Spawned 1 of one singular bacteria at a time to test the individual movement and logic of each
       2. Spawned in 1 of each bacteria to test the hurt and death sounds of each
       3. Spawned an excess of food and one singular bacteria to test that the “overlapFood()” command and test that duplication upon eating 3 food worked; it also tested the “moveToFood” for the bacteria that call it
       4. Commented certain sections of “doSomething()” to test each individual part; commented out everything but the “regularMovement()” function to test move paths and the move plan of each bacteria that called it; commented out everything but the “reguarMovement()” and “moveToFood()” commands to test how it moved with food; set the bacteria alone in the map and implemented only the “agroMove()” function used by aggressive Salmonella; removed all dirt and called the “regularMovement()” and “overlapSocrates()” functions to test interactions with Socrates; finally tested all the functions with “Dirt” implemented
   16. class RegSalmonella : public Bacteria
       1. Everything tested by the “Bacteria” class
   17. class AgroSalmonella : public Bacteria
       1. Tested the “agroSalMove()” function only and commented out the rest of “doSomething()” in the “Bacteria” class; “agroSalMove()” returns false by default
   18. class Ecoli : public Bacteria
       1. The “virtual regularMovement()” function of Ecoli overrides the regular movements of the bacteria base class; spawned 1 Ecoli with no other actors and tested that the logic of the function worked correctly; then testes with dirt actors to test the angle move increments that are implemented withing the function when it hits a dirt pile