In this report, some constructors and destructors are left out because there’s really not much to say.

1. Student World class: public:

StudentWorld(std::string assetPath);constructor.

~StudentWorld();destructor;

virtual int init();

virtual int move();

virtual void cleanUp();

bool getshot(Projectile\* p);actions that a projectile will take when hitting an actor

void addFlame();add flames to the actor list when player pressed enter key

void addSpray();add sprays to the actor list when player pressed space key

void addBacteria(int type,double x, double y);add bacteria to the actor list when a new bacteria is born either from a pit or a duplication of a full bacteria

void reducepits();inform the student world class that a pit has already released all bacteria by decreasing the number of pits that is present by one

bool gethealth(RHealth\* r);things going on when player gets restore health goodies(return true if got, false if not)

bool getflame(FThrower\* f); things going on when player gets Flame thrower goodies(return true if got, false if not)

bool getgoodie(Goodie\* g);return true if player gets a goodie, otherwise false.

bool getlife(ELife\* e); things going on when player gets extra life goodies(return true if got, false if not)

bool getfungus(Fungus\* fungus); things going on when player runs into a fungus(return true if did, false if not)

bool hurtSocrates(Bacteria\* b);used to determine if a bacteria has hurt a Socrates and also implement the effect of damage(decrease player’s hp, play sound)

bool inrange(Bacteria\* b);used to determine if the player is in 72pixels around a bacteria(for aggressive salmonella use)

bool inrange2(Bacteria\* b);used to determine if the player is in 256 pixels around a bacteria(for ecoli use)

bool blockbydirt(double x, double y);determine if the result of the next movement’s position is going to cause an overlap with a dirt

bool getFood(Bacteria\* b);determine if a bacteria got food and implement the effect of getting a food(increase the number of food eaten by the bacteria by one and set the food to dead)

void dirtoplayer(Bacteria\* b);get the direction from the bacterium to the player and set it to the direction of the bacterium

bool dirtofood(Bacteria\* b); get the direction from the bacterium to the closest food and set it to the direction of the bacterium

I chose to make the above functions public in this class because the spec says that the StudentWorld class is the only place that should know where the actors are. So it is ought to be in charge of all kinds of interactive actions.

double countDistance(double x1, double y1, double x2, double y2);count the distance between two x,y pairs(two coordinates) I chose to make this one public because it could also be used by other functions in other classes so I need not rewrite another one for that class.

2. Actor:

Actor(int id, double startX, double startY,StudentWorld\* s,int dir = 0, int depth = 0);

void setDead();set the status of the actor to dead

StudentWorld\* getstudent();return a pointer of the studentworld object

virtual ~Actor(); virtual destructor for a base class

virtual bool damageable()const; to determine if the actor is damageable

virtual bool isProjectile()const; to determine if the actor is a Projectile

virtual bool isFood()const; to determine if the actor is a Food

virtual bool isDirt()const; to determine if the actor is a Dirt Pile

virtual bool isFungus()const; to determine if the actor is a Fungai

virtual bool isBacteria()const; to determine if the actor is a bacterium

virtual bool isEcoli()const; to determine if the actor is an ecoli

virtual bool cantoverlap()const; to determine if the actor could be overlapped while contructed

virtual bool couldblock()const; to determine if the actor could block the movement of bacteria

virtual bool hashp()const; to determine if the actor has hitpoints

I decided to make all the above virtual functions non pure virtual, instead, I made them return false by default because if so I would not have to let the majority base classes return false repeatedly. I only need to modify the ones that should return true in the base classes

int gethp()const; return the hitpoints of the actor

void changehp(int a); change the hitpoints of the actor

bool alive() const; return true if the actor is alive false otherwise

virtual void doSomething()=0; let doSomething be pure virtual because we will never implement is for only an “actor” we would only use it for base objects where we override the function.

3.Socrates:

Socrates(StudentWorld\* student);

~Socrates();No need to be virtual since no derived classes

virtual bool hashp() const; since Socrates hashp, override actors hashp() to return true

void getPositionInThisDirection1(Direction angle, int units, double& dx, double& dy); override this function for the purpose of the Socrates move

virtual void doSomething(); virtual for the purpose of clarity, get keys and implement corresponding actions, if no keys get, then replenish sprays to 20

int getflamenum()const;to get the number of flame left for Socrates (for studentworld use)

int getspray()const; to get the number of spray left for Socrates (for studentworld use)

void increaseflame(int i); increase the number of flames(for student world use when Socrates got flame thrower goodies)

4.Dirt Pile:

virtual bool damageable()const;is damageable

virtual bool isDirt()const;is a dirt pile

virtual bool couldblock()const; could block movements of bacteria

virtual void doSomething();do nothing

all virtual keywords for clarity

5.Food:

virtual bool damageable()const;is damageable

virtual bool isFood()const; is a Food

virtual bool couldblock()const; could block movements of bacteria

virtual void doSomething();do nothing

virtual bool cantoverlap()const; cannot be overlapped with pits and food and dirt

6.Projectile:

virtual~Projectile(); virtual destructor because it serves as a base class

virtual bool isProjectile()const;is a projectile

virtual int getDamage() const=0;pure virtual because base class could have different damages

virtual void doSomething();though need not be virtual, but just in case we need to modify it later in base classes. Here’s what it does: if the projectile is dissipated, return immediately, if it hits something, do the damage and dissipate, and return immediately, if none of the above happened, move in its current direction in the specified distance, if it moved a total distance more than it should, it dissipates.

7.Flame:

virtual int getDamage()const; return flame’s damage, virtual for clarity

8.Spray:

virtual int getDamage()const; return spray’s damage, virtual for clarity

9.Goodie:

virtual ~Goodie(); base class so virtual destructor

virtual bool getSomething()=0; pure virtual because depends on different types of base class

virtual void doSomething(); if dissipated, return immediately, if not, then see if player gets the goodie, tell studentworld to play sound and do corresponding action to the player and dissipate the goodie. If nothing happened, reduce the tick left for the goodie before it dissipates, if time left not greater than 0, dissipate itself.

virtual bool damageable()const; all goodies are damageable, virtual for clarity

10.Restore health goodie:

virtual bool getSomething();tell studentworld to check if the player got a restore health goodie if so then do corresponding actions in student world and return true; if not then return false

11.Flame thrower goodie:

virtual bool getSomething();tell studentworld to check if the player got a Flame thrower goodie if so then do corresponding actions in student world and return true; if not then return false

12.Extra life goodie:

virtual bool getSomething(); tell studentworld to check if the player got an extra life goodie if so then do corresponding actions in student world and return true; if not then return false

13.Fungus:

virtual bool getSomething(); tell studentworld to check if the player ran into a fungus if so then do corresponding actions in student world and return true; if not then return false

virtual bool isFungus()const; return true, virtual for clarity. This function is for studentworld to determine which sound to play when the player ran into a goodie

14. Pit:

virtual void doSomething(); if it contains no more bacteria, then dissipate and inform the student world class that a pit dissipated. Otherwise add bacteria into the studentworld actor list according to the spec and decrease the number of corresponding bacteria held by one.

virtual bool cantoverlap()const; a pit cannot overlap with other pits or food which are also cantoverlap. And dirt piles also cannot overlap with them while constructed.

15.Bacteria:

virtual~Bacteria(); base class, so destructor has to be virtual

virtual void doSomething();

virtual bool damageable()const; return true because could be damaged by projectile

virtual bool hashp()const; return true since all bacteria have hitpoints

virtual bool isBacteria()const; return true

virtual int getdamage()const=0; pure virtual because different bacteria have different damages

virtual bool aggressive()const; determine whether the bacteria is an aggressive salmonella

int getnumoffood()const; return the number of food that the bacteria had ate

void eatfood(); increase the number of food the bacteria had ate by one

void resetfood(); set the number of food the bacteria had ate to 0

int getmoveplan()const; get the move plan distance of bacteria

void decmoveplan(); decrease the move plan distance of the bacteria

void resetplan(); set the move plan distance to 10

void trymove(); try to move getpixel() amount of pixels in current direction, if failed, then set direction to a new direction

virtual int getpixel()const; return the pixels the bacteria should move during each tick, if it could. It returns 3 by default which is the number of pixels salmonellas move.

16.Regular Salmonella:

virtual int getdamage()const; return regular salmonella’s damage

17.Aggressive Salmonella:

virtual int getdamage()const; return aggressive salmonella’s damage

virtual bool aggressive()const; return true since this is an aggressive salmonella

18.Ecoli:

virtual int getdamage()const;return ecoli’s damage

virtual bool isEcoli()const; return true since this is an ecoli

virtual int getpixel()const; return 2 since it moves 2 pixels everytime

Following the spec, any bacteria that ran into the player tends to oscillate around the player while in the demo game, it is not like that. So I submitted a version following the spec and has another version that tells bacteria to stop when they have a distance less than 2 from the player, so that the game looks a bit like the demo version.

How I tested my Socrates class: press enter to see if there are flames generated around, press direction key to move around clockwise and counterclockwise for a couple of rounds. Press enter to see if sprays are generate in front of the player. Use cerr to print out the flames left for the player. See if the flame would be depleted.

How I tested my flames and spray class: test the normal cases along with the player class test. Then change the distance they travel before they dissipate to see if things work as expected. See if flames and spray could destroy dirt, food, all kinds of goodies and fungus and bacteria.

How I tested my Restore Health goodie class: See if it dissipates normally, see if it dissipates after player runs into it, change the rate that it appears to see if things goes as expected. See if the player’s health is filled after getting it. Hit it with flame to see if it could be destroyed.

How I tested my Flame thrower goodie class: See if it dissipates normally, see if it dissipates after player runs into it, change the rate that it appears to see if things goes as expected. See if the player’s flames left is increased by 5 after getting it. Hit it with flame to see if it could be destroyed.

How I tested my Extra Life goodie class: See if it dissipates normally, see if it dissipates after player runs into it, change the rate that it appears to see if things goes as expected. See if the player’s life is increased by 1 after getting it. Hit it with flame to see if it could be destroyed.

How I tested my Fungus class: See if it dissipates normally, see if it dissipates after player runs into it, change the rate that it appears to see if things goes as expected. See if the player’s hp is decreased by 20 and score decreased by 50 after running into it. Hit it with flame to see if it could be destroyed and if it is destroyed, did the players score increase by 100.

How I tested my Pit class: change the rate that it produces bacteria and observe if the rate actually changed as expected. Count the number of different types of bacteria it emits and see if they are right. Change the storage of its different type of bacteria and see if the new numbers are correct. See if the pit disappears after producing all bacteria. Let the move function in student world class always return finished level and see how the pit number goes with increasing levels. And if pits would overlap with each other.

How I tested my Food class: See if it disappears after a bacteria contacts with it. See if any of it overlaps with dirt piles, pits, or other food. See if it could be destroyed by projectiles. Move on to different levels to observe the number of food.

How I tested my dirt class: See if they could overlap with each other when allocated and that they should not overlap with food or pits.

How I tested my bacteria classes: See if ecoli is always directed towards the player, see if aggressive salmonella is directed towards the player if the player is in range. See if the above two actors stop their movements when blocked by dirt. See if all bacteria have the intelligence of changing direction when cannot move in the current direction, see if salmonella changes direction after moving in the same direction for some time and will find the closest food object within range. Hit bacteria with spray and see if they will be damaged at first and killed when being hit too much(damage exceeds bacteria’s hp) See if bacteria reproduce after eating three food(tracing one bacterium).

How I tested my StudentWorld class: For the init() function, implement one at a time while commenting others out, just to see if things go right. Then add them together to see if they are initialized properly together (eg. Objects that cannot overlap will not overlap.) I also let the move function always return level completed to see if the init function works properly to initialize random and correct number of different objects at the beginning of each leveled game. For the move() function, I would let the player run into a fungus while having a 0 score to see what is displayed at top of the screen if the score is negative. Aside from all methods I used to test other classes that can also be used to test this function, I would let the player die in different circumstances and see if the program runs correctly. For other circumstances that I cannot reach easily by playing the game and pass the levels, I chose to set the player to a status undamageable or add eternal flames around the player and see how the game goes in high levels( aka hacking.) For cleanup(), I only need to let player die and till game over, see if there are any bad access or read access violation( deleting a pointer twice) or any memory leaks.