Darren Wu

CS 32 Project 3

1.

Class Hierarchy

GraphObject

Actor

BorderLine

Agent

GhostRacer

Pedestrian

HumanPedestrian

ZombiePedestrian

ZombieCab

Spray

nCAWActor

OilSlick

SprayableGoodie

HealingGoodie

HolyWaterGoodie

SoulGoodie

Uses the same hierarchy as the design provided with the exception of sprayable goodie.

Class Actor

doSomething() is a pure virtual function as the subclasses each implement their own doSomething function in a different way.

gotSprayed() is a virtual function. This is virtual as sprays only affect certain actors, to the ones that it did not affect, they could inherit the gotSprayed function. Others that do be affected by the spray get their own implementations for what happens when sprayed.

double getSpeed() const;

void setSpeed(double s);

bool isAlive() const;

void setAlive(bool b);

bool isCAWActor() const;

void setCAWActor(bool b);

int getHSpeed() const;

void setHSpeed(int n);

The getter and setter functions for each of the Actors member variables. Every Actor needs to have these so it is defined in the Actor class.

GetSpeed GetHspeed and setSpeed and setHSpeed are used to access and change the Actor’s vertical and horizontal speeds.The isAlive and setAlive are used as a marker for when an Actor dies. In the case of isCAWActor and setCAWActor, those functions are used to determine if an Actor is “Collision Avoidance Worthy”.

StudentWorld\* getWorld(); returns a pointer to the world passed in by each actor’s constructor. Every Actor needs to have access to the world to potentially tell the world to do any actions.

int determineCurrentLane() const;

This function takes the position of the actor that called the function and returns which lane it is in from 0 ,1, or 2 which represent the left, middle and right lanes respectively.

bool determineOverlap(Actor\* a, Actor\* b);

Returns whether or not two Actors are overlapping as defined by the spec.

void moveActor();

Every actor moves in the same way based off their own speed and the ghost racer’s speed so this should be in the actor class.

Class Agent:public Actor

int getHealth() const;

void setHealth(int h);

void damageHealth(int n);

These were defined in the Agent Class as agents are the only actors with health. GetHealth returns the member Health variable, setHealth changes the member variable to the parameter, and damage subtracts n health from its original

int getPlan() const;

void setPlan(int n);

Each of the agents have some sort of movement plan, not every actor has this so it is defined in the agent class. getPlan returns the movement plan, setPlan changes the movement plan to its parameter n.

class GhostRacer: public Agent

virtual void doSomething();

Ghost Racer’s doSomething implements going left, right, speeding up, slowing down, spraying water, and getting damaged by the left and right borders. The virtual function is implemented here as no other actor shares any functionality with the GhostRacer

int getSprays();

void addSprays(int s);

Only Ghost racer has sprays so they are defined in the ghost racer class. getSprays returns the amount of sprays left, setSprays changes the amount of sprays to n sprays

void slick();

This changes the direction of the ghost Racer whenever it overlaps with an oil slick. I decided to have it defined in the ghostRacer class as opposed to the oil class as it made more sense for me to have the oil tell the ghostRacer to be spun rather than actually setting the direction of the ghostRacer in the oil class.

void heal(int n);

Instead of using the setHealth, I defined a heal function for readability. Adds n health to the ghostracer. Defined in ghostracer as no other Actor can be healed.

Class nCAWActor:public Actor

This class represents the non “collision avoidance worthy” objects excluding borderlines and the spray. When I reference nCAW actors, exclude borderlines and the spray in description of the class.

virtual void doDistinctNCAWActivity() = 0;

This is defined as pure virtual in the nCAW actor class as every subclass underneath this has a different functionality as well as sharing the same movement.

virtual void doSomething();

Calls both the moveActor functions and the distinctNCAW actor Activity functions. Defined here as each NCAW actor won’t need to implement their own doSomething since the distinct actions for each different actor are defined in the pure virtual function described earlier.

class BorderLine: public Actor

I decided to have this not inherit nCAW actors as I did not want to have a doDistinct activity which did nothing.

virtual void doSomething();

Calls the moveActor function. Defined here as borderlines are the only Actor that solely move as their functionality.

class Pedestrian : public Agent

virtual void doSomething();

Calls both the moveActor and doPedActivity functions. Defined here as the lower level classes share the same functionality with the exception of doPedActivity which is just called by doSomething.

virtual void doPedActivity() = 0;

Defined as pure virtual as both zombie and human subclasses do a pedestrian activity but it is not necessarily the same. Should be implemented by the subclasses.

Class ZombiePedestrian : public Pedestrian

virtual bool gotSprayed();

Damages the Zombie and either makes the world play the zombie hurt noise or kills the zombie and potentially drops a healing goodie based on whether the zombie loses enough hit points. Increases the student world’s score if kills successfully. This has to be redefined and cannot inherit the Actor class gotSprayed function as the two functionalities are different

virtual void doPedActivity();

If a ghostRacer runs over a zombie, then damage the zombie, the ghost racer and play its respective sound for dying. Zombies also are drawn towards the ghostRacer, which set its direction to walk towards the ghostracer if it is within range and they have had the same x coordinate in the zombie’s lifetime. This is unique to the Zombie Pedestrians so it must be defined in this class.

class HumanPedestrian : public Pedestrian

virtual bool gotSprayed();

This just changes the direction and horizontal Speed of the Human. This has to be redefined and cannot inherit the Actor class gotSprayed function as the two functionalities are different

virtual void doPedActivity();

This tells the world that the player died if the ghostRacer and the human ped overlap because you are not allowed to run over humans. This is unique to the Human Pedestrians so it must be defined in this class.

class ZombieCab : public Agent

virtual void doSomething();

If the zombieCab is alive, check if it and the ghostRacer overlap, and damage the GhostRacer, and change the direction and horizontal speed of the cab. Moves the cab. Then adjusts speed if it is too close to another “Collision Avoidance Worthy” actor. Also decreases its movement plan which intermittently changes the zombie cab’s speed. Defined here as most of the functionality is specific to the zombie cab (moveActors is called, defined from Actor class).

bool didDamageGhost() const;

void setDidDamage(bool b);

Getter and Setter functions so the Zombie cab can know if it damaged the GhostRacer already. If so set its member variable to true. Defined in Zombie cab class as only Zombie Cabs care if they damaged the ghost already.

virtual bool gotSprayed();

Damage the cab, if not dead play its respective sound. If health is less than or equal to zero, set its alive status to die. Increase the student world score and add oil actor if necessary. This has to be redefined and cannot inherit the Actor class gotSprayed function as the two functionalities are different, and return different values

class OilSlick: public nCAWActor

virtual void doDistinctNCAWActivity();

Tells the ghostRacer to change direction as specified in slick function for every tick they overlap. This is unique to oil slicks so it needs to be defined in this class.

Class SprayableGoodie: public nCAWActor

Virtual bool gotSprayed();

Sets sprayable goodies to false so they can be removed when sprayed. Defined here to prevent repeating code for both healing goodies and holy water goodies

Class HealingGoodie: public SprayableGoodie

virtual void doDistinctNCAWActivity();

If overlapping with ghost racer, Increases score by 250, heals ten health unless it will go over 100, then it will only heal to 100. Only healing goodies heal so this is its unique activity

Class HolyWaterGoodie: public SprayableGoodie

virtual void doDistinctNCAWActivity();

If overlapping with ghost racer, Increases score by 50, adds ten more sprays to the ghost racer object through the student world pointer. Only holy water goodies can add water so it is defined in this distinct activity function.

Class SoulGoodie: public nCAWActor

virtual void doDistinctNCAWActivity();

If overlapping with ghost racer, Increase score by 100. Add to the souls collected. Also rotate by ten degrees. This is from the pure virtual doActivity from the nCAW class, this functionality is unique only to soul goodies

Class HolyWaterProjectile: public Actor

virtual void doSomething();

If its alive, tell the world to see if its overlapping with any other Actors, if they are splashable, set the projectile alive status to false so it dies. Moves forward with each tick, and if it has gone 160 ticks, set status to false so it can be removed. This virtual do something function has to be redefined for the projectile as its functionality is unique only to Holy water projectiles

int getTravelDistance()const;

void setTravelDistance(int n);

These are getter and setter functions for the member variable travel distance which either return or change the the travel distance member value respectively

Student World Class Additional Public Functions

GhostRacer\* getGhost();

Returns a pointer to the ghostracer which is held in a separate pointer as opposed to the vector of pointers. Used so other actors can tell the ghostracer to do different things.

void addHolyWaterProj(double x, double y, int dir,StudentWorld\* s);

void addHealingGoodie(double x, double y, StudentWorld\*s);

void addOil(double x, double y, StudentWorld\* s);

Used so different actors can communicate to the world to add new Actors at specified positions by the parameters

void setHumanContact(bool b);

Used by the human ped when ran over to communicate to the world that it has been ran over and to end the game and return a dead status.

void addSoulsCollected();

bool checkWaterSplash(Actor\* a);

bool checkBehindActors(Actor\* a);

bool checkinFrontActors(Actor\* a);

These three functions take in Actor pointer parameters and compare it to the m\_actors vector of Actor Pointers.

checkWaterSplash takes in the projectile as an Actor parameter and sees if it hit any splash worthy objects. The first one it hits, it returns true and does its respective gotSprayed function

Both checkbehind and checkinfront actor functions are used by the zombie cab to see whether or not there is a “collision avoidance worthy” actor in the lane that is close enough to where it needs to slow down or speed up respectively.

Pseudocode For More complex functions

bool checkBehindActors(Actor\* a);

Loop through every single Actor

If the current actor is in the same lane as the actor which was passed in, and is “CAW”

If the actor’s pointer from looping through each actor isn’t the same as Actor a

Save the difference if it is the smallest difference

Return true if an actor is too close from behind.

bool checkinFrontActors(Actor\* a);

Works in the same way as checkBehindActors, but swaps the subtraction to keep a positive value between the Actor a and the distance between the actors in front

void StudentWorld::tryAddZombieCab()

Choose a random lane and loop maximum 3 times, checking each lane

If lane is clear from either top or bottom, break out of loop and mark whether to add to top or bottom from the current lane

Set Y position, x position and speed based off where you are adding a new zombie cab

Add new cab to the vector of actor pointers

2. I wasn’t able to replicate it after multiple attempts(maybe it is fixed), but when adding a zombie cab at every applicable tick going at the default speed, then speeding up to the max speed as quick as possible, one of the zombie cabs overlapped with each other.

My actors and borderlines move noticeably faster than the sample executable, but as mentioned in the spec it is rare but possible, so I believe it is worth noting.

When playing the game, I would sometimes get errors with playing the audio. I noticed that it couldn’t rapidly play the sounds. When I go frame by frame however, it works as expected.

I believe the rest of my classes work as described in the spec.

3.

In regard to the zombie pedestrian following the ghost racer, it said to turn the direction when in front and the x coordinate was 30 px away either left or right. After playing the game I interpreted it to mean that if the ghost racer passes directly in front of the zombie, then it must follow the ghost racer as long as it is within 30 px of the racer.

I assumed the holy water projectile direction to be the same as the Ghost Racer’s direction.

You don’t get points if you run into a zombie cab.

When adding a Zombie Cab it says to choose a random lane to start evaluating, then later it states (you may check the three lanes in any order, so long as you check each lane just once). I assumed that the first lane you choose must be random, then evaluation of the other two lanes don’t need to be random.

I interpreted “random x position on the surface of the road” to mean “random x position between 0 and VIEW\_WIDTH”