**Chelsea Biala**

**CS32 Smallberg Lecture 2**

1. Actor:

* doSomething(): I chose to make this pure virtual because each actor has to perform an action each tick, and each actor does something different.
* isAlive(): returns whether an actor should be deleted at the end of each tick. This is part of the Actor class because each actor has a status of dead or alive.
* getSpeed(): returns the speed of each actor at the current moment. This is a part of the Actor class because each actor a speed in a particular direction.
* die(): changes the status of an actor from alive to dead. This is part of the Actor class because each actor must be removed if its status is dead.
* setSpeed(): changes the speed of the actor. This is part of the Actor class because each actor has a speed that can change based on the speed of the GhostRacer or some other factor each tick.
* getSW(): returns the StudentWorld the actor is a part of. This is part of the Actor class because each actor is a part of a StudentWorld class and must sometimes get information about other actors on the screen.
* isOffScreen(): determines whether an actor is off the screen and therefore should be removed. This is part of the Actor class because all actors must not be alive if they are not displayed on the screen.
* areOverlapping(): determines whether two actors are overlapping so they can possibly interact with each other. This is part of the Actor class because all actors are affected in some way when they come into contact with one another.
* isCollisionAvoidanceWorthy(): returns whether each actor is collision avoidance worthy. This is part of the Actor class because I gave Actor a private data member bool of whether or not it is collision avoidance worthy; every actor either is or is not.
* getSprayed(): I chose to make this function virtual because actors perform different actions when sprayed with a holy water projectile. This is part of the Actor class because actors of different base classes are able to be sprayed.
* moveRelativeToGhostRacer(): moves the actor based on the current speed of the ghost racer. This is part of the Actor class because each actor moves during each tick based on the speed of the ghost racer.
* isSprayable(): returns whether an actor is affected by holy water projectiles. I chose to make this function pure virtual because some actors are sprayable and some are not.

LiveActor:

* getHitPoints(): returns the remaining number of hit points for a LiveActor. This is part of the LiveActor class because each LiveActor has a certain number of hit points, and there must be a way to access them.
* changeHP(): changes hit points to a different value. This is part of the LiveActor class because each actor needs a way to decrease or increase hit points.
* soundWhenHurt(): returns the sound a LiveActor makes when it loses hp. This is part of the LiveActor class because each LiveActor makes a sound when damaged. I made this function pure virtual because each actor makes a different sound.
* soundWhenDie() returns the sound a LiveActor makes when it dies. This is part of the LiveActor class because each LiveActor makes a sound when it dies. I made this function pure virtual because each actor makes a different sound.

GhostRacer:

* doSomething(): as described in the spec, the ghost racer takes damage when it hits border lines, takes user input through keys, and accordingly shoots holy water and changes direction. It then moves forward in this direction.
* soundWhenDie(): returns the sound a ghost racer makes when it dies, inherited from LiveActor.
* soundWhenHurt(): returns the sound a ghost racer makes when hurt, inherited from LiveActor.
* numSprays(): returns the number of holy water projectiles the ghost racer has left in order to determine whether it can still shoot.
* addSprays(): adds a specified number of holy water projectiles to the ghost racer object.
* spin(): as described in the spec, changes the position of the ghost racer when it activates an oil slick.
* isSprayable(): inherited from actor, shows that the ghost racer cannot be affected by holy water projectiles.

Pedestrian:

* soundWhenHurt(): returns the sound a pedestrian makes when hurt. Both derived classes make the same sound when hurt, so this function is just inherited from the Actor class.
* soundWhenDie(): returns the sound a pedestrian makes when it dies. Both derived classes make the same sound when they die, so this function is just inherited from the Actor class.
* getHorizSpeed(): returns the pedestrian’s horizontal speed. This is a part of the pedestrian class because pedestrians have horizontal speeds, unlike other actors.
* setHorizSpeed(): changes the pedestrian’s horizontal speed. This is a part of the pedestrian class because pedestrians have horizontal speeds, unlike other actors.
* moveAndPossiblyPickPlan(): as described in the spec, moves the pedestrian along its path and randomizes its movement, sometimes changing its direction. This is a part of the pedestrian class because they follow similar plans of movement.
* takeDamageAndPossiblyDie(): inflicts damage on the pedestrian and dies based on number of hit points. This is a part of the pedestrian class because they have the same number of hit points and can die in similar ways.

Human Pedestrian:

* doSomething(): dies and ends the level if the ghost racer hits it, otherwise moves based on inherited function from pedestrian class.
* getSprayed(): inherited from actor class, loses the appropriate number of hit points when hit with a holy water projectile.
* isSprayable(): returns true because human pedestrians are affected by projectiles.

Zombie Pedestrian:

* doSomething(): sustains damage if hits ghost racer, changes direction if is in close enough proximity to ghost racer, and otherwise moves in accordance to its movement plan inherited from the pedestrian class.
* getSprayed(): inherited from actor class, loses the appropriate number of hit points when hit with a holy water projectile.
* isSprayable(): returns true because zombie pedestrians are affected by projectiles.
* isCloseToGhostRacer(): determines whether the zombie pedestrian should change direction based on its proximity to the ghost racer.

Zombie Cab:

* doSomething(): if overlapping with the ghost racer, a zombie cab will veer off track. Otherwise, it changes its speed based on the position of the ghost racer and collision avoidance worthy actors and moves.
* getSprayed(): as inherited from the actor class, the zombie cab sustains damage and possibly dies when overlapping with a projectile. It possibly drops a healing goodie, which is why this function must be overridden for this class.
* getHorizSpeed(): returns the horizontal speed of the zombie cab. This is redefined in zombie cab and pedestrian because a ghost racer, while an agent, does not have horizontal speed.
* setHorizSpeed(): changes the horizontal speed of the zombie cab. This is redefined in zombie cab and pedestrian because a ghost racer, while an agent, does not have horizontal speed.
* soundWhenHurt(): returns the sound a zombie cab makes when it’s hurt, as inherited from the live actor class.
* soundWhenDie(): returns the sound a zombie cab makes when it dies, as inherited from the live actor class.
* isSprayable(): returns true because a zombie cab is affected by projectiles.

Borderline:

* doSomething(): as described in the spec, moves relative to the ghost racer’s speed.
* hitRacer(): decreases the ghost racer’s health if the two overlap.
* isSprayable(): returns false because borderlines are not affected by projectiles.

Ghost Racer Activated Object:

* doSomething(): moves according to the ghost racer’s speed and calls the doActivity function, to be explained later. If overlapping with the ghost racer, increases its score by the specified number of points and dies.
* doActivity(): this is what each object should do when activated by the ghost racer. This function is pure virtual because each ghost racer activated object does something different when the ghost racer hits it.
* getScoreIncrease(): returns the number of points each ghost racer activated object will increase the ghost racer’s score by when they overlap. This function is pure virtual because each object increases its score by a different number.
* getSound(): returns the sound made when the ghost racer overlaps with the object. This function is pure virtual because each object makes a different sound.
* selfDestructs(): returns whether the object disappears when it hits the ghost racer. This function is pure virtual because some disappear and some don’t.

Soul Goodie:

* doActivity(): decreases the number of souls left for the ghost racer to save if they’re overlapping. Changes the soul goodie’s direction.
* getScoreIncrease(): as inherited from ghost racer activated object, returns the number of points collecting this goodie gives to the player.
* getSound(): as inherited from ghost racer activated object, returns the sound made when the ghost racer collects this goodie.
* selfDestructs(): as inherited from ghost racer activated object, returns true because this object self destructs when it hits the ghost racer.
* isSprayable(): as inherited from actor, returns false because this goodie is not affected by projectiles.

Healing Goodie:

* doActivity(): increases the ghost racer’s health by 10 if they are overlapping
* getScoreIncrease(): as inherited from ghost racer activated object, returns the number of points collecting this goodie gives to the player.
* getSound(): as inherited from ghost racer activated object, returns the sound made when the ghost racer collects this goodie.
* selfDestructs(): as inherited from ghost racer activated object, returns true because this object self destructs when it hits the ghost racer.
* isSprayable(): as inherited from actor, returns true because this goodie is affected by projectiles.
* getSprayed(): sets the object’s status to not alive.

Holy Water Goodie:

* doActivity(): adds 10 sprays to the ghost racer’s spray count
* getScoreIncrease(): as inherited from ghost racer activated object, returns the number of points collecting this goodie gives to the player.
* getSound(): as inherited from ghost racer activated object, returns the sound made when the ghost racer collects this goodie.
* selfDestructs(): as inherited from ghost racer activated object, returns true because this object self destructs when it hits the ghost racer.
* isSprayable(): as inherited from actor, returns true because this goodie is affected by projectiles.
* getSprayed(): sets the object’s status to not alive.

Oil Slick:

* doActivity(): spins the ghost racer if they overlap
* getScoreIncrease(): returns zero because this object does not increase the player’s score when they overlap
* getSound(): as inherited from ghost racer activated object, returns the sound made when the ghost racer overlaps with the oil slick
* selfDestructs(): as inherited from ghost racer activated object, returns false because this object does not self destruct when it hits the ghost racer.
* isSprayable(): as inherited from actor, returns false because this goodie is not affected by projectiles.

Spray:

* doSomething(): if there is an actor that overlaps with the projectile, it will get sprayed. If not, it continues traveling forward.
* isSprayable(): as inherited from actor, returns false because this goodie is not affected by projectiles.

Student World:

* setBorderY(): changes the Y coordinate of the last white border line added.
* getBorderY(): returns the Y coordinate of the last white border line added.
* getGR(): returns a pointer to the ghost racer in the student world to access its functions.
* decSouls(): decreases the number of souls left to save in a level.
* calcStartX(): gives a random x coordinate on the road where various actors can be initialized.
* willInsert(): determines whether to insert an actor in each tick based off of the probability of each given in the spec
* getSoulCount(): returns number of souls left to save in the level.
* getBonus(): returns the current bonus amount in that particular tick
* addHealingGoodie(): pushes a new healing goodie onto the actor container at the specified x and y coordinates
* getLane(): returns which lane (1, 2, or 3) the given actor is in

if x coordinate is between left edge and left lane border

return 1

else if x coordinate is between left lane border and right lane border

return 2

else if x coordinate is between right lane border and left edge

return 3

else return 0

* collisionAvoidanceWorthyActorInLane(): determines whether there is a collision avoidance worthy actor in the same lane as a given actor

for each actor in container

if the actor is collision avoidance worthy and in the same lane

return true and assign parameters of that actor

if the ghost racer is in the same lane

return true and assign parameters of that actor

* addOilSlick(): push a new oil slick with the specified position and a random size onto the container of actors
* addZombieCab(): if there is an appropriate spot to introduce a zombie cab in that tick, push it onto the container of actors at the specified position

create 3 lists for actors in each lane during the current tick

while no spot is found

iterate through actor list and find closest collision worthy actor to the bottom, then the top if none

if none found, go to next lane

if a lane is chosen, introduce a zombie cab in that lane at an x and y coordinate based on the ghost racer and chosen actor’s position

* holyWaterActivated(): determines if a holy water projectile will have an effect

for all actors in container list

if is sprayable and overlapping with projectile

return this actor

return null pointer

* addHolyWater(): adds a holy water projectile at the specified spot with the specified direction
* init(): initializes all objects and data members for each level

initializes actor list and ghost racer

calculates number of souls to save based on level

adds yellow and white border lines at specified intervals all up page

* move(): all actors in list and ghost racer perform their actions for each tick, dead actors removed

for all actors

do something

if ghost racer is not alive

decrement lives

return player died

if all souls are saved

add bonus point

return finish level

change m\_border\_y based on ghost racer speed

ghost racer does something

for all actors

if not alive

erase from list and delete

insert new border lines onto list

insert all new actors based on chance onto list

decrease bonus by one

display text at top of screen

* cleanUp(): deletes all remaining actors and frees pointers

for all actors in container

remove from list and delete

2. My zombie cabs can sometimes overlap and don’t often spawn at the top of the screen. I am not sure if their movement pattern is completely correct; I can’t tell from watching them on the screen. They spawn often right behind the ghost racer and hit it immediately.

My inheritance sometimes duplicates code is small functions. I followed the recommended structure, but kept some things from my own that I didn’t have time to clean up.

3. I assumed that each individual actor does not need a specifically implemented destructor, even though one of its data members is a pointer to student world, because in deleting the pointer, you would delete the entire interface of the game.

I assumed that when a healing goodie will put a ghost racer’s health to above 100, you shouldn’t increment its health at all. This is probably an incorrect assumption, but I didn’t have time to correct it.