The inheritance hierarchy of my actor classes is:

**GraphObject**

**Actor**

**BorderLine**

**Agent**

**GhostRacer**

**Pedestrian**

**HumanPedestrian**

**ZombiePedestrian**

**ZombieCab**

**Spray**

**GhostRacerActivatedObject**

**OilSlick**

**HealingGoodie**

**HolyWaterGoodie**

**SoulGoodie**

1.

* In my base **Actor** class, I define the following public member functions:

**pure virtual** **doSomething()**: I make this function pure virtual because each type of actor should have a chance to do something, and what they do may differ.

**isDead()**:Returns a Boolean value that indicates whether the actor is dead. I did not make it virtual because it works in the same way for each type of actor.

**setDead()**: Mark the actor as dead. I did not make it virtual because it works in the same way for each type of actor .

**world()**: Returns a pointer to the student worldall actors are in. I did not make it virtual because it works in the same way for each type of actor.

**getVerticalSpeed()**: Returns the actor’s vertical speed. I did not make it virtual because it works in the same way for each type of actor.

**setVerticalSpeed()**: Set the actor’s vertical speed. I did not make it virtual because it works in the same way for each type of actor.

**moveRelativeToGhostRacerVerticalSpeed()**: Move to a position determined by the actor’s horizontal speed (dx) and the actor’s vertical speed relative to Ghost Racer’s vertical speed. I did not make it virtual because it works in the same way for each type of actor.

**virtual beSprayedIfAppropriate()**: If the actor is affected by holy water projectiles (sprays), then inflict that effect on it and return true; otherwise, return false. I make it return false (which indicates that the actor is not sprayable) and make it virtual so that each sprayable actor can override it.

**virtual isCollisionAvoidanceWorth()**: Returns a Boolean value that indicates whether the actor is “collision avoidance-worthy”. I make it return false and make it virtual so that each “collision avoidance-worthy” actor can override it.

* In my **BorderLine** class, I define the following public member functions:

**virtual doSomething()**: Implements what a border line must do during a tick. It is virtual because it overrides the base **Actor** class’s **pure virtual doSomething()**.

* In my **Agent** class, I define the following public member functions:

**virtual isCollisionAvoidanceWorth()**: Since all agents (**GhostRacer**, **HumanPedestrian**, **ZombiePedestrian**, and **ZombieCab**) are “collision avoidance-worthy”, this function overrides the base **Actor** class’s **virtual isCollisionAvoidanceWorth()** and returns true.

**getHP()**: Returns the actor’s hit points. I did not make it virtual because it works in the same way for each type of agent.

**increaseHP()**: Increases the actor’s hit points. I did not make it virtual because it works in the same way for each type of agent.

**getHorizSpeed()**: Returns the actor’s horizontal speed. I did not make it virtual because it works in the same way for each type of agent.

**setHorizSpeed()**: Set the actor’s horizontal speed. I did not make it virtual because it works in the same way for each type of agent.

**virtual takeDamageAndPossiblyDie()**: Decreases the actor’s hit points and check if it is dead. I make it virtual so that the subclasses **ZombiePedestrian** and **ZombieCab** can add their specific behaviors to this function.

**virtual soundWhenHurt()**: Returns the sound that this agent should play when it is damaged but does not die. I make it return SOUND\_NONE and make it virtual so that each agent with a playable sound can override it.

**virtual soundWhenDie()**: Returns the sound that this agent should play when it is damaged and dies. I make it return SOUND\_NONE and make it virtual so that each agent with a playable sound can override it.

* In my **GhostRacer** class, I define the following public member functions:

**virtual doSomething()**: Implements what the Ghost Racer must do during a tick. It is virtual because it overrides the base **Actor** class’s **pure virtual doSomething()**.

**virtual soundWhenDie()**: Returns SOUND\_PLAYER\_DIE. It is virtual because it overrides the **Agent** class’s **virtual soundWhenDie()**.

**getNumSprays()**: Returns the number of holy water projectiles (sprays) the Ghost Racer has.

**increaseSprays()**: Increases the number of holy water projectiles (sprays) the Ghost Racer has.

**spin()**: Spins as a result of hitting an oil slick.

* In my **Pedestrian** class, I define the following public member functions:

**virtual soundWhenHurt()**: Returns SOUND\_PED\_HURT. It is virtual because it overrides the **Agent** class’s **virtual soundWhenHurt()**.

**moveAndPossiblyPickPlan()**: Moves the pedestrian and possibly picks a new movement plan. I did not make it virtual because it works in the same way for each type of pedestrian (human and zombie).

* In my **HumanPedestrian** class, I define the following public member functions:

**virtual doSomething()**: Implements what a human pedestrian must do during a tick. It is virtual because it overrides the base **Actor** class’s **pure virtual doSomething()**.

**virtual beSprayedIfAppropriate()**: Inflicts the effects of the spray on the human pedestrian and return true. It is virtual because it overrides the base **Actor** class’s **virtual beSprayedIfAppropriate()**.

* In my **ZombiePedestrian** class, I define the following public member functions:

**virtual doSomething()**: Implements what a zombie pedestrian must do during a tick. It is virtual because it overrides the base **Actor** class’s **pure virtual doSomething()**.

**virtual takeDamageAndPossiblyDie()**: Calls **Agent::takeDamageAndPossiblyDie()** first, then does its specific behaviors (possibly create new healing goodie and increase the player’s score). It is virtual because it overrides the **Agent** class’s **virtual takeDamageAndPossiblyDie()**.

**virtual beSprayedIfAppropriate()**: Inflicts the effects of the spray on the zombie pedestrian and returns true. It is virtual because it overrides the base **Actor** class’s **virtual beSprayedIfAppropriate()**.

**virtual soundWhenDie()**: Returns SOUND\_PED\_DIE. It is virtual because it overrides the **Agent** class’s **virtual soundWhenDie()**.

* In my **ZombieCab** class, I define the following public member functions:

**virtual doSomething()**: Implements what a zombie cab must do during a tick. It is virtual because it overrides the base **Actor** class’s **pure virtual doSomething()**.

**virtual takeDamageAndPossiblyDie()**: Calls **Agent::takeDamageAndPossiblyDie()** first, then does its specific behaviors (possibly create new oil slick and increase the player’s score). It is virtual because it overrides the **Agent** class’s **virtual takeDamageAndPossiblyDie()**.

**virtual beSprayedIfAppropriate()**: Inflicts the effects of the spray on the zombie cab and returns true. It is virtual because it overrides the base **Actor** class’s **virtual beSprayedIfAppropriate()**.

**virtual soundWhenHurt()**: Returns SOUND\_VEHICLE\_HURT. It is virtual because it overrides the **Agent** class’s **virtual soundWhenDie()**.

**virtual soundWhenDie()**: Returns SOUND\_VEHICLE\_DIE. It is virtual because it overrides the **Agent** class’s **virtual soundWhenDie()**.

* In my **Spray** class, I define the following public member functions:

**virtual doSomething()**: Implements what a holy water projectile (spray) must do during a tick. It is virtual because it overrides the base **Actor** class’s **pure virtual doSomething()**.

* In my **GhostRacerActivatedObject** class, I define the following public member functions:

**virtual doSomething()**: Implements what a Ghost Racer-activated object must do during a tick. I define this function here because all of the subclasses have a similar doSomething() structure:

move relative to the Ghost Racer

if overlaps with the Ghost Racer:

do specific activity

play sound

set itself to dead if appropriate

increase the player’s score

It is virtual because it overrides the base **Actor** class’s **pure virtual doSomething()**.

**virtual beSprayedIfAppropriate()**: Inflicts the effects of the spray on the Ghost Racer-activated object and returns true. It is virtual because it overrides the base **Actor** class’s **virtual beSprayedIfAppropriate()**.

**pure virtual doActivity()**: I make this function pure virtual because each type of Ghost Racer-activated object has different special activity to do.

**pure virtual getScoreIncrease()**: I make this function pure virtual because each type of Ghost Racer-activated object increases the player’s score by different amount.

**virtual getSound()**: Returns the sound to be played when the Ghost Racer-activated object is activated. I make it return SOUND\_GOT\_GOODIE and make it virtual so that each Ghost Racer-activated object with a different sound can override it.

**virtual selfDestructs()**: Returns a Boolean value that indicates whether the Ghost Racer-activated object should destruct itself after activation. I make it return true and make it virtual so that each Ghost Racer-activated object that should not destruct itself can override it.

**virtual isSprayable()**: Returns a Boolean value that indicates whether the Ghost Racer-activated object is sprayable. I make it return true and make it virtual so that each Ghost Racer-activated object that is not sprayable can override it.

* In my **OilSlick** class, I define the following public member functions:

**virtual doActivity()**: Tells the Ghost Racer to spin. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual doActivity()**.

**virtual getScoreIncrease()**: Returns 0. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual getScoreIncrease()**.

**virtual getSound()**: Returns SOUND\_OIL\_SLICK. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **virtual getSound()**.

**virtual selfDestructs()**: Returns false. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **virtual selfDestructs()**.

**virtual isSprayable()**: Returns false. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **virtual isSprayable()**.

* In my **HealingGoodie** class, I define the following public member functions:

**virtual doActivity()**: Increases the Ghost Racer’s hit points. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual doActivity()**.

**virtual getScoreIncrease()**: Returns 250. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual getScoreIncrease()**.

* In my **HolyWaterGoodie** class, I define the following public member functions:

**virtual doActivity()**: Increases the Ghost Racer’s number of sprays. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual doActivity()**.

**virtual getScoreIncrease()**: Returns 50. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual getScoreIncrease()**.

* In my **SoulGoodie** class, I define the following public member functions:

**virtual doSomething()**: Calls **GhostRacerActivatedObject::doSomething()** first, then rotates itself. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **virtual doSomething()**.

**virtual doActivity()**: Tells the student world that a soul is saved. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual doActivity()**.

**virtual getScoreIncrease()**: Returns 100. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **pure virtual getScoreIncrease()**.

**virtual getSound()**: Returns SOUND\_GOT\_SOUL. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **virtual getSound()**.

**virtual isSprayable()**: Returns false. It is virtual because it overrides the **GhostRacerActivatedObject** class’s **virtual isSprayable()**.

2.

The sound effect is strange. Two sounds cannot play at once.

3.

I assumed that the bonus points cannot be negative.