CS 32 Project 3 Report

By Tomas Kaljevic

Actor:



The one constructor defined for Actor invokes the constructor of GraphObject to initialize the image ID, starting X, starting Y, direction, size, and depth of an Actor. Additionally, it initializes the pointer to StudentWorld, which controls all the Actors in the game. An Actor can never be initialized on its own because it contains a pure virtual function. In other words, while all entities in the game are Actors, none of them are only an Actor.



The destructor defined for Actor is virtual because all the other classes that inherit Actor must also be destroyed, which having a virtual destructor ensures. The body of the implementation of Actor’s destructor is empty because the destructor does not need to actually do anything.



The method doSomething() is a pure virtual function because while each Actor does something, they do something in a different way. For example, a BorderLine doesn’t do the same thing in a tick as a ZombieCab. This method is called every tick by every Actor currently active.



Every Actor moves, and some Actors follow the exact same movement algorithm. So the default implementation of move() implements this common movement algorithm as defined in the spec, returning false only if the Actor is off the screen after moving and returning true otherwise. For Actors that follow unique movement algorithms, like GhostRacer, they can override this function and implement their own movement algorithm, hence why it is virtual.



Every Actor is considered either “collision avoidance-worthy” or not “collision-avoidance worthy.” Most Actors are not “collision avoidance-worthy,” so the default implementation of isCollisionAvoidanceWorthy() returns false. For Actors that are “collision avoidance-worthy,” however, they can override this method and return true, hence why it is virtual.



Some Actors can be damaged, and some cannot. Most Actors cannot be damaged, so the default implementation of canBeDamaged() returns false. For Actors that can be damaged, they can override this method and return true, hence why it is virtual.



Some Actors can be affected by overlapping with holy water projectiles, and some cannot be affected. Most Actors cannot be affected by holy water, so the default implementation of affectedByHolyWater() returns false. For Actors that can be affected, they can override this method and return true, hence why it is virtual.



For Actors that are affected by overlapping with holy water projectiles, there needs to be behavior defined for when they are hit by holy water. By default, Actors that are affected by holy water are damaged by 1 hit point when they are hit by holy water. For Actors that are affected differently by holy water, they can override this method and implement their own behavior, hence why it is virtual.



There needs to be a way for an Actor to use the methods in StudentWorld (e.g. to get the container of actors, or get GhostRacer, or add a new Actor), so this method returns a pointer to StudentWorld that can be utilized by all derived classes of Actor.



We need to know if an Actor is active or not because if they are active, they should doSomething() every tick. So this method returns whether an Actor, or any derived class of Actor, is active.



If any Actor needs to be deleted (e.g. GhostRacer has overlapped with that Actor and destroyed it), we need to say that it is no longer active so it gets deleted the next tick.



A lot of the times we need to know an Actor’s vertical speed to adjust how fast it (or another Actor) moves down the screen. This method returns an Actor’s vertical speed.



A lot of the times we to adjust how fast an Actor moves down the screen by changing its vertical speed. This method sets an Actor’s vertical speed to a new value.



A lot of the times we need to know an Actor’s horizontal speed to adjust how fast it (or another Actor) moves across the screen. This method returns an Actor’s horizontal speed.



A lot of the times we to adjust how fast an Actor moves across the screen by changing its horizontal speed. This method sets an Actor’s horizontal speed to a new value.



For Actors that can be damaged, we need to be able to know their health to see if it is 0, and therefore needs to be removed from the game. This method returns an Actor’s health.



For Actors that can be damaged, we need to be able to modify their health (e.g. when they are initialized with a given amount of health). This method sets an Actor’s health to a new value.



For Actors that can be damaged, we need to be able to decrease their health (e.g. when they are damaged by holy water). This method decreases an Actor’s health by a given amount.



For Actors that can be damaged, we need to be able to increase their health (e.g. when they pick up a healing goodie). This method increases an Actor’s health by a given amount.

Damageable:



The one constructor defined for Damageable invokes the constructor of Actor to initialize the image ID, starting X, starting Y, direction, size, and depth of a Damageable Actor. It also initializes the pointer to StudentWorld, which controls all the Actors in the game. Finally, it initializes the Damageable’s health to whatever value is provided in the constructor.



As its name suggests, all Damageables can be damaged, so this method is overridden from Actor’s virtual implementation of it and returns true.

Pedestrian:



The one constructor defined for Pedestrian invokes the constructor of Damageable to initialize the image ID, starting X, starting Y, direction, size, and depth of a Pedestrian. It also initializes the pointer to StudentWorld, which controls all the Actors in the game. Finally, it initializes the Damageable’s health to whatever value is provided in the constructor.



All Pedestrians are considered “collision avoidance-worthy” actors, so this method is overridden from Actor’s virtual implementation and returns true.



All Pedestrians are affected by overlapping with holy water projectiles, so this method is overridden from Actor’s virtual implementation and returns true.



Pedestrians follow a movement pattern dictated by their movement plan distance, an integer value. This method returns a Pedestrian’s movement plan distance.



A Pedestrian’s movement plan distance needs to be modified sometimes during a tick. This method sets a Pedestrian’s movement plan distance to a new value.



A Pedestrian’s movement plan distance needs to be decreased sometimes during a tick. This method decreases Pedestrian’s movement plan distance by 1.

BorderLine:



The one constructor defined for BorderLine invokes the constructor of Actor to initialize the image ID, starting X, starting Y, direction, size, and depth of a BorderLine. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



During a tick, a BorderLine, no matter its color, moves down the screen at a rate based on GhostRacer’s vertical speed. This method is overridden from Actor’s pure virtual function and conducts the movement behavior typical of both colors of BorderLines.

Goodie:



The one constructor defined for Goodie invokes the constructor of Actor to initialize the image ID, starting X, starting Y, direction, size, and depth of a Goodie. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



All Goodies do roughly the same thing during a tick. They are given the chance to move down the screen. Then, if they overlap with GhostRacer, they are picked up, then rendered inactive, then the score is increased. Otherwise, they might do some additional behavior if it is defined by the doOtherBehavior() method below.



When a Goodie is picked up, the total score of the game is increased by some amount. However, each Goodie increases the total score by a different amount. So this is a pure virtual function that returns the amount of points GhostRacer attains by picking up a Goodie.



When a Goodie is picked up, something must happen (i.e. a unique sound is played and GhostRacer is affected in some way). Since each Goodie affects GhostRacer in a different way, this is a pure virtual function that dictates what happens when a Goodie is picked up.



Some Goodies have behavior outside of what happens when GhostRacer overlaps with it. An implementation of this method concerns what happens during a tick that GhostRacer doesn’t overlap with a Goodie. The default implementation is empty, but this method can be overridden by some Goodie that requires additional behavior.

GhostRacer:



The one constructor defined for GhostRacer invokes the constructor of Damageable to initialize the image ID, starting X, starting Y, direction, size, and depth of a GhostRacer to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



During a tick, if GhostRacer’s health is at or below 0, then it plays its death sound and is rendered inactive. Otherwise, if GhostRacer bumps into the left or right yellow border lines, then it needs to take damage and its direction needs to be adjusted. Otherwise, if the user presses a key that is recognized by GhostRacer, then it needs to respond correctly (e.g. shoot a holy water projectile, move left or right, or speed up or slow down).



GhostRacer follows a unique movement algorithm that none of the other Actors have in common. Thus, it must override Actor’s virtual implementation of move() and conduct its own movement. This method will always return true because GhostRacer can never go off the screen.



GhostRacer is considered a “collision avoidance-worthy” actor, so this method is overridden from Actor’s virtual implementation and returns true.



Returns the number of holy water projectile sprays GhostRacer has.



Increase the number of holy water projectile sprays GhostRacer has by some amount.



Returns the number of souls GhostRacer has saved.



Increase the number of souls GhostRacer has saved by 1.

YellowBorderLine:



The one constructor defined for YellowBorderLine invokes the constructor of BorderLine to initialize the image ID, starting X, starting Y, direction, size, and depth of a YellowBorderLine to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.

WhiteBorderLine:



The one constructor defined for WhiteBorderLine invokes the constructor of BorderLine to initialize the image ID, starting X, starting Y, direction, size, and depth of a WhiteBorderLine to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.

HumanPedestrian:



The one constructor defined for HumanPedestrian invokes the constructor of Pedestrian to initialize the image ID, starting X, starting Y, direction, size, and depth of a HumanPedestrian to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



During a tick, if a HumanPedestrian overlaps with GhostRacer, then GhostRacer’s number of lives must be decremented and the level must end. Otherwise, a HumanPedestrian must be given a chance to move, and then its movement plan must be adjusted as defined by the spec as it moves horizontally across the screen.



Since Pedestrians are affected by holy water, HumanPedestrian needs to override this function from Actor’s virtual implementation of it and define its own behavior. When it’s hit by holy water, its horizontal speed must be multiplied by -1, its direction must flip, and it must play its hurt sound.

ZombiePedestrian:



The one constructor defined for ZombiePedestrian invokes the constructor of Pedestrian to initialize the image ID, starting X, starting Y, direction, size, and depth of a ZombiePedestrian to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



During a tick, if a ZombiePedestrian’s health is at or below 0, then it plays its death sound and is rendered inactive. Otherwise, if it overlaps with GhostRacer, then is damaged and the ZombiePedestrian is killed. Otherwise, a ZombiePedestrian’s horizontal speed must be adjusted as it moves across the screen, and it must be given a chance to move down the screen. Then, its movement plan must be adjusted as defined by the spec.



Since Pedestrians are affected by holy water, ZombiePedestrian needs to override this function from Actor’s virtual implementation of it and define its own behavior. When it’s hit by holy water, it must invoke Actor’s call of onHitByHolyWater(). Then, if its health is at or below 0, it has a 0.20 chance of spawning a HealingGoodie at its location. Otherwise, it must play its hurt sound.

ZombieCab:



The one constructor defined for ZombieCab invokes the constructor of Pedestrian to initialize the image ID, starting X, starting Y, direction, size, and depth of a ZombieCab to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game. Finally, it initializes the ZombieCab’s vertical speed and lane to whatever values are provided in the constructor.



During a tick, if a ZombieCab’s health is at or below 0, then it plays its death sound and is rendered inactive. Otherwise, if it overlaps with GhostRacer and it hasn’t already damaged GhostRacer, then GhostRacer must be damaged by 20 hit points and the cab must go flying off the screen. Otherwise, ZombieCab must be given a chance to move and speed up or slow down if the closest Actor in front of or behind the ZombieCab is a reasonably safe distance for it to do so.



ZombieCabs are considered “collision avoidance-worthy” actors, so this method is overridden from Actor’s virtual implementation and returns true.



ZombieCabs are affected by overlapping with holy water projectiles, so this method is overridden from Actor’s virtual implementation and returns true.



Since ZombieCabs are affected by holy water, ZombieCab needs to override this function from Actor’s virtual implementation of it and define its own behavior. When it’s hit by holy water, it must invoke Actor’s call of onHitByHolyWater(). Then, if its health is at or below 0, it has a 0.20 chance of spawning an OilSlick at its location. Otherwise, it must play its hurt sound.

OilSlick:



The one constructor defined for OilSlick invokes the constructor of Actor to initialize the image ID, starting X, starting Y, direction, size, and depth of a OilSlick to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



During a tick, an OilSlick is given a chance to move. Then, if it overlaps with GhostRacer, GhostRacer must be spun around randomly.

HolyWaterProjectile:



The one constructor defined for HolyWaterProjectile invokes the constructor of Actor to initialize the image ID, starting X, starting Y, direction, size, and depth of a HolyWaterProjectile to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



During a tick, an OilSlick must loop through all active Actors that are affected by holy water and see if it has overlapped with any of them. If it has, then that Actor must call its onHitByHolyWater() method and the holy water must be rendered inactive. Then, it must be given a chance to move and if it has exceeded its max travel distance, it must be removed.



HolyWaterProjectile follows a unique movement algorithm, so it must override Actor’s virtual implementation of move() and conduct its own movement. This method will return true if Actor moved successfully and false if it moved off the screen.

HealingGoodie:



The one constructor defined for HealingGoodie invokes the constructor of Pedestrian to initialize the image ID, starting X, starting Y, direction, size, and depth of a HealingGoodie to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



Returns the number of points (250) GhostRacer attains by picking up a HealingGoodie.



When a HealingGoodie is picked up, GhostRacer’s health must be increased by 10 hit points and the sound for picking up a Goodie must be played.



Returns true because HealingGoodies are affected by holy water.



When a HealingGoodie is hit by holy water, it must be destroyed.

HolyWaterGoodie:



The one constructor defined for HolyWaterGoodie invokes the constructor of Pedestrian to initialize the image ID, starting X, starting Y, direction, size, and depth of a HolyWaterGoodie to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



Returns the number of points (50) GhostRacer attains by picking up a HolyWaterGoodie.



When a HolyWaterGoodie is picked up, GhostRacer must be given 10 holy water projectile sprays and the sound for picking up a Goodie must be played.



Returns true because HolyWaterGoodies are affected by holy water.



When a HolyWaterGoodie is hit by holy water, it must be destroyed.

LostSoulGoodie:



The one constructor defined for LostSoulGoodie invokes the constructor of Pedestrian to initialize the image ID, starting X, starting Y, direction, size, and depth of a LostSoulGoodie to given values defined in the spec. It also initializes the pointer to StudentWorld, which controls all the Actors in the game.



Returns the number of points (100) GhostRacer attains by picking up a LostSoulGoodie.



When a LostSoulGoodie is picked up, GhostRacer must save 1 soul and the sound for picking up a Goodie must be played.



LostSoulGoodies have additionally defined behavior in that they rotate by 10 degrees every tick that they are active, this method is overridden from Actor’s virtual implementation of it.

StudentWorld:



The one constructor for StudentWorld initializes the pointer to GhostRacer to be a nullptr, the bonus score to be 0, and the Y of the last white BorderLine added to be -1.



The destructor for StudentWorld calls the cleanUp() method to delete all pointers so as to prevent any possible memory leaks.



The init() function creates GhostRacer and adds the initial BorderLines to the screen. It is a virtual function to denote that it is inherited from GameWorld.



In the move() function, every Actor that is currently active does something as dictated by their doSomething() methods. If GhostRacer dies during any tick, the level ends. If GhostRacer saves the required number of souls on a level, the level ends and the next level begins. All inactive Actors are removed, and new Actors are added. The game stat text is updated and the bonus score decreases at a rate of 1 point/tick. It is a virtual function to denote that it is inherited from GameWorld.



In the cleanup() function, every Actor – including GhostRacer – is deleted so as to prevent any possible memory leaks.



Returns a vector that stores a pointer to all Actors currently active in the game.



Returns a pointer to GhostRacer.



Returns values corresponding to the left and right edges of the road respectively.



Determine whether new BorderLines should be added during a tick as dictated by the requirements in the spec and add them if they need to be added.



Determine whether new ZombieCabs should be added during a tick as dictated by the requirements in the spec and add them if they need to be added.



Given a starting X and a starting Y, add a new OilSlick at that location.



Determine whether new OilSlicks should be added during a tick as dictated by the requirements in the spec and add them if they need to be added.



Determine whether new ZombiePeds should be added during a tick as dictated by the requirements in the spec and add them if they need to be added.



Determine whether new HumanPeds should be added during a tick as dictated by the requirements in the spec and add them if they need to be added.



Determine whether new HolyWaterGoodies should be added during a tick as dictated by the requirements in the spec and add them if they need to be added.



Determine whether new LostSoulGoodies should be added during a tick as dictated by the requirements in the spec and add them if they need to be added.



Given a starting X, a starting Y, and a direction, add a new HolyWaterProjectile at that location.



Given a starting X and a starting Y, add a new HealingGoodie at that location.



Return whether an Actor is in a lane or not.

1. To my knowledge, I implemented everything that was required in the spec.
2. When adding new BorderLines, I was repeatedly having a problem where new WhiteBorderLines weren’t being added. To fix this, I had to update the value of the Y of the last WhiteBorderLine added each tick, which wasn’t directly specified in the spec. Furthermore, when adding a new ZombieCab, I was having a problem where the ZombieCab would spawn directly under GhostRacer and therefore crash directly into it. To fix this, I added my GhostRacer pointer to the same vector that stores pointers to all other Actors currently active, and that fixed that problem I was having.