Eric Zhang

CS 32

Project 3 Report

This report describes the class structure and public member function implementations in my Ghost Racer video game.

StudentWorld class

This class contains the functions for the primary states of the game, init(), move(), and cleanUp(). It also contains any functions that may be used by Actors but require access to the vector that stores all the actor pointers in a game, or other private member variables like the Ghost Racer.

* int init() : Dynamically allocate a new GhostRacer to m\_player. Create new yellow and white borderlines from the bottom to the top of the screen, and point lastYellow and lastWhite pointers to the yellow and white borders closest to the top of the screen to help with the algorithm for adding new borderlines later. set the value of soulsToSave depending on the level, and set the bonus score to 5000.
* int move() : Tell the GhostRacer to do something. Then, loop through m\_actors. If the actor is alive, tell it do something. Then check if the Ghost Racer is still alive, as well as if it has saved enough souls, and perform the according actions if so. If the actor is not alive, it is deleted and erased from the vector. Then, add new actors to the screen:
  + Zombie cabs are added with the following pseudocode:
  + if the chance of a vehicle spawning is true:
    - create array of 3 false bool values
    - cur\_lane = randint between 1 and 3 inclusive
    - int addZombieCab = -1;
    - loop up to 3 times (increments cur\_lane with a mod of 3):
      * if no collision avoidance-worthy actor exists in cur\_lane or the closest one to the bottom is at least VIEW\_HEIGHT / 3 pixels away:
        + set addZombieCab to cur\_lane
        + break out of loop
      * if no collision avoidance-worthy actor exists in cur\_lane or the closest one to the top is at least VIEW\_HEIGHT \* 2/3 pixels from the bottom:
        + set addZombieCab to cur\_lane
        + break out of loop
    - if we should add zombie cab (addZombieCab != -1):
      * add zombie cab in the lane determined by value of addZombieCab
      * append the zombie cab to the vector of actors
  + after zombie cabs are added, add oil slicks, zombie pedestrians, human pedestrians, holy water refills, and lost souls using their spawn chances, and append to the actors vector if added.
  + Finally, update the game text on the screen.
* void cleanUp() : delete m\_player (Ghost Racer), and delete and erase the first element of the actors vector repeatedly until the vector is empty.
* GhostRacer\* getPlayer() : simply returns m\_player, the GhostRacer object in the game. Since all of my other actor objects will be passed a pointer to StudentWorld, this lets them also access all public member functions of a GhostRacer without adding another pointer.
* void addActor(Actor\* a) : this function simply pushes back a new dynamically allocated actor to the vector m\_actors that stores all actors on screen. I added this function to StudentWorld after realizing that the Ghost Racer, zombie pedestrians and zombie cabs may create new actors in their actions (holy water projectiles, healing goodies and oil slicks, respectively) and could use access the world to add them to the vector of actors.
* void recordSoulSaved() : increases number of souls saved.
* int findLane(Actor\* a) : used to find the lane in which a collison avoidance-worthy actor is located, returning 1, 2, or 3 for the 3 lanes in the game screen. This function is useful for the following functions that locate certain actors within a desired lane.
* Actor\* closestAbove(Actor\* a), Actor\* closestBelow(Actor\* a) : determine the closest actor above or below and in the same lane as the actor. Both keep track of a shortestDist double variable initialized to VIEW\_HEIGHT and an actor pointer initialized to nullptr. After finding the lane of the actor, both functions traverse the m\_actors vector. If an actor is both collision avoidance-worthy and in the same lane as the target actor, its vertical distance from the target actor is checked.
* Actor\* closestToTop(int lane), Actor\* closestToBottom(int lane) : implemented similarly to the previous two functions, but instead check for the largest or smallest Y coordinate of eligible actors instead of distance from a given actor. All four of these functions that return an actor pointer are used in the creation and movement of zombie cabs.
* bool sprayFirstAppropriateActor(Actor\* a) : gets passed in a pointer to a holy water projectile actor, and checks each actor in the m\_actors vector for an overlap and if it is affected by holy water sprays.

Actor class

* virtual void doSomething() = 0 : this function must have a redefined implementation for every child class of Actor, since each actor performs a different series of actions per tick when its doSomething() function is called.
* virtual bool isAlive() const : returns the private variable m\_alive to determine whether an actor is still alive. This function is virtual because it is later redefined for Actors whose alive status is better characterized by their remaining hit points (namely, whether they have more than 0) than a boolean value.
* void kill() : sets the private variable m\_alive to false, so that isAlive() returns false when next checked for. All actors who only possess a true or false alive status can easily use this function to remove their alive statuses.
* StudentWorld\* getWorld() : returns a pointer to the world the actor is created in, allowing access to playing various sounds, adding new Actors, Ghost Racer functions, and much more functionality.
* StudentWorld\* getPlayer() : returns a pointer to the Ghost Racer in getWorld()’s private members. Saves me from writing getWorld()->getPlayer() every time I want to access the Ghost Racer in an Actor function.
* int getVertSpeed() const, void setVertSpeed(int speed), int getHorizSpeed() const, void setHorizSpeed(int speed) : getters and setters for vertical and horizontal speed. Since almost all actors use both horizontal and vertical speed in some way for their movement calculations, I placed them both together in the main Actor class for easy access. (for the few actors that don’t use one of the speeds, that member variable is disregarded so a new subclass doesn’t have to be created to accommodate.)
* bool offScreen() const : checks if an actor’s x and y coordinate values have exceeded the range of the screen, which almost every actor beside the Ghost Racer needs in order to be removed when it has passed the screen.
* bool overlap(Actor\* other) : uses the spec-provided overlap algorithm to determine if this Actor and other are overlapping.
* void moveOnScreen() : uses GraphObject’s moveTo() function to translate actors along the screen. I placed this in the overarching Actor class because almost all Actors automatically move along the screen through the same algorithm.
* virtual bool isCollisionAvoidanceWorthy() const : returns false, but is virtual so that a subclass of all collision avoidance-worthy actors can later redefine this function.

SentientActor class

This class contains all the actors who are collison avoidance-worthy, and also have a given health which may be subtracted (and gained, in the case of the Ghost Racer).

* void decHP(int hp) : decrements the private member variable m\_hp by the given value. All SentientActors have a set number of starting hit points and may be killed by those hit points decrementing.
* int getHP() const : gives the amount of hp remaining. \*Though imperfect, I decided to replace any instance of kill() with the decHP() function that passes getHP() as its parameter, since m\_alive from the base Actor class is no longer accessible.
* virtual bool isAlive() const : returns true if m\_hp is greater than 0. Again, all actors in this class have an alive status that cannot simply be toggled.
* virtual bool isCollisionAvoidanceWorthy() const : returns true, setting an opposite definition from the other actors, all of which are not collision avoidance-worthy.

GhostRacer class

* virtual void doSomething() : checks for alive status, wall collisions, user key inputs and performs the next tick movement, all of which are described in the spec.
* void heal() : increments m\_hp by calling decHP with -10 as its parameter. Since Ghost Racer is the only entity that can restore hit points and can only do so through one event, I decided not to create a new private variable just to accommodate it.
* int getHolyWater() : returns the amount of holy water units, which is used in the game status text.
* void increaseHolyWater() : increases the private member variable holyWaterUnits by 10.
* void spin() : called when oil slick is run over. Chooses a random integer from -20 to 20 until its absolute value is greater than or equal to 5, then performs a direction change by that amount if it does not turn the Ghost Racer by too much.

MovingAgent class

This class contains all the actors in SentientActor except for the Ghost Racer. Since these actors must move intelligently on their own, new functions related to the movement plan distance are incorporated.

* int getMovementPlanDistance() : returns the movement plan distance. All moving agents have a movement plan distance that must reset when it reaches 0 and must call this getter function
* void decMovementPlanDistance() : decreases movement plan distance by 1.
* void resetMovementPlanDistance() : sets movement plan distance to a random integer between 4 and 32. All MovingAgent objects reset their movement plan distance in the same way.
* virtual void pickNewMovementPlan() : sets the horizontal speed to a value between -3 and 3 while it is not 0, calls resetMovementPlanDistance, and changes the direction based on the horizontal speed. I made this function virtual because zombie cabs change their vertical speed instead of horizontal speed when picking a new movement plan.

HumanPedestrian class

* virtual void doSomething() : return immediately if not alive. If overlap with Ghost Racer occurs, kill the player and immediately return. then, move on screen, decrement movement plan distance, and pick new movement plan if needed.
* virtual bool beSprayedIfAppropriate() : sets horizontal speed to its negative counterpart, adds 180 degrees to direction (automatically mods with 360), and play a sound. Return true.

ZombiePedestrian class

* virtual void doSomething() : return immediately if not alive. If overlap with Ghost Racer occurs, decrease the Ghost Racer’s hp by 5 and decrease own hp by 2. If no longer alive, play the according sound. If the absolute value of the difference between x coordinate and x coordinate of ghost racer is less than 30 and y coordinate is less than ghost racer’s y coordinate, set direction to 270 and move towards ghost racer depending on relative position. Decrease ticks to next grunt, and play sound and set ticks to next grunt to 20 if it is 0. Then, move on screen, decrement movement plan distance and return if greater than 0, and pick new movement plan otherwise.
* virtual bool beSprayedIfAppropriate() : decrease hp by 1. If no longer alive, play a sound and create a healing goodie with a 1 in 5 chance if not overlapping with ghost racer, and also increase score by 150. If still alive, play a different sound. Return true.

ZombieCab class

* virtual void pickNewMovementPlan() : set vertical speed to the vertical speed plus a random integer between -2 and 2, inclusive. Then reset the movement plan distance. This function redefines the one from MovingAgent, since only zombie cabs change vertical speed when resetting movement.
* virtual void doSomething() : return immediately if not alive. If overlap with Ghost Racer occurs and has not already damaged ghost racer: play a sound, decrement Ghost Racer’s hp by 20, and swerve to a side depending on relative position by changing horizontal speed and direction. Indicate that the ghost racer has been damaged by this cab. Then, move on screen. Adjust vertical speed if necessary using following pseudocode:
  + if vertical speed > ghost racer’s vertical speed:
    - if an actor above self in this lane exists and difference in Y < 96:
      * decrease vert speed by 0.5
      * return;
  + else
    - if an actor below self in this lane exists and difference in Y < 96:
      * increase vert speed by 0.5
      * return;

lastly, decrease movement plan distance and pick a new one if it reaches 0.

* virtual bool beSprayedIfAppropriate() : decrease hp by 1. If no longer alive, play a sound and create an oil slick with a 1 in 5 chance. Increase score by 200 points. Otherwise, play a different sound. Return true.

NoncollidingActor class

This class contains objects that generally move down the screen uneventfully, but have some interaction with the Ghost Racer if they overlap.

* virtual bool beSprayedIfAppropriate() : kills itself and returns true if it is a sprayable actor, and returns false otherwise. Since all noncolliding actors that respond to holy water are affected in the same way, this function is dependent only on the isSprayable return value of the actor.
* virtual bool isSprayable() const = 0 : returns whether a noncolliding actor is affected by holy water, is true for some actors and false for others.

Borderline class

* virtual void doSomething() : return immediately if no longer alive, move on screen, and kill if moved off screen.
* virtual bool isSprayable() const : returns false.

OilSlick class

* virtual void doSomething() : move on screen, and tell the world to play a sound and the player to spin if they overlap.
* virtual bool isSprayable() const : returns false.

HealingGoodie class

* virtual void doSomething() : move on screen, and tell the player to heal and the world to play a sound and increase score by 250, then set own status to not alive, if they overlap.
* virtual bool isSprayable() const : returns true

HolyWaterGoodie class

* virtual void doSomething() : move on screen, and increase player’s holy water supply, play a sound, and increase score by 50 if they overlap.
* virtual bool isSprayable() const : returns true

LostSoul class

* virtual void doSomething() : move on screen, tell the world to record a saved soul, play a sound, and increase score by 100, and set own status to not alive, if overlap with player occurs.
* virtual bool isSprayable() const : returns false

List of missing/incomplete/buggy functionality that remains in my video game:

* when an oil slick is run over, the user can hear overlapping/echoing oil slick sounds being played, and sometimes (for bigger oil slicks) the ghost racer will rapidly shift direction multiple times.
* Zombie cabs, when another zombie cab comes close to them in the same lane, often match their speeds instead of increasing their distance from each other. This did not happen in the demo game but their speeds under those circumstances were being increased accordingly, so I could not find the error.

Other design assumptions:

* the bonus score can continue decreasing to negative numbers if a round lasts to that point. I made this assumption having never reached that distance while playing, and deciding that the player deserves to be penalized if they choose to drag the game that long.