Report

The base class for all actors is called **actorBase,** which has the following public function

* 1. doSomething()
  2. Accessor functions (getXspeed, getYspeed, getHealth, getAvoid, getAliveStatus, getSWptr (which returns a pointer to Student World)
  3. isOffScreen(), which returns true if the actor is offscreen, which it set itself to notalive
  4. isOverLap(), which returns true if the actor overlaps with Ghost Racer
  5. Mutator Functions, which changes the actor’s base attribute (changeHealth, changeAliveStatus, changeXspeed, changeYspeed)
  6. changeInYalg(), which changes Y based on Ghost Racer’s Speed
  7. Common Method Functions, which splits all actors based on its characteristics: isZombie(), isBeing(), isWaterInteractive()

-doSomething() is a pure virtual functions since it is getting redefined in each actor, and actorBase shouldn’t be able to doSomething because it isn’t an actor itself

-Accessor and Mutator Functions are both non-virtual functions, since every actor uses the same function and they really don’t alter much since all they do is returning a value of their properties. So every actor uses the same function.

-isOffScreen, changeInYalg , and isOverLap() are also non-virtual functions since every actor uses the same method to determine if it’s offscreen or if it’s overlapping Ghost Racer

-all the common methods and changeInYalg are virtual functions since they can be changed by each actor.

**Pseudo for actorBase**

void changeInYalg(double& new\_x, double& new\_y, actorBase\* ptr)

{

Vert\_speed = ptr’s Y speed - Ghost Racer’s Speed

Hor\_speed = ptr’s X speed

New\_y = ptr’s Y + VertSpeed

New\_x = ptr’s X + HorSpeed

}

Bool isOffScreen( actorBase\* ptr)

{

If (ptr’s X < 0 or Y < 0 or X < VIEW\_WIDTH or Y > ViewHeight)

Ptr’s alive status = false

Return true

Return false;

}

Bool isOverLap( StudentWorld\* ptr)

{

deltaX = abs( actor’s X - GR’s X)

deltaY = abs( actor’s Y - GR’s Y)

sumOfRadius = actor’s radius + GR’s Radius

If ( deltaX < SOR/4 && deltaY < SOR \* 0.6)

Return true

Return false

}

**CLASS MOVABLE**

1. getPlan() , which returns m\_plan (getter)
2. changePlan(), which changes plan (Mutator)
3. Virtual determineNextPlan()

-Both getters and mutators are non-virtual since they are simple functions that won’t be needed to be override

-determineNextPlan() is virtual since it is the same for all class BUT zombieCab.

**PSEUDO for MOVABLE**

Void determineNextPlan( actorBase\* ptr)

{

Int tempxspeed = 0

For loop

...keeps looping under tempXspeed is not 0 but is still between -3 & 3…

Ptr’s speed = tempxspeed;

changePlan (random int between 4 and 32)

If (ptr’s xspeed < 0)

Set ptr’s direction to 180

Else

Set ptr’s direction to 0

return

}

**GHOST RACER CLASS , PUBLIC ACTORBASE**

1. doSomething(): virtual since it inherited from actorBase
2. getHurt() : non virtual since it’s a helper class
3. changeNumOfSprays() : nonvirtual since it’s a mutator within the class
4. ger\_numofSprays() :non-virtual since it’s a getter within the class

**PSEUDO FOR GHOST**

doSomething()

{

If dead

Return

If (x < Left edge)

If direction > 90

...get hurt by 10..

Set direction = 82

Play sound

Else if x >= Right Edge

If Direction < 90

..getHurt(10)..

Set Direction = 98

Play Sound

Else

Int key = getKey

If key == space

...let SW know we need to add water…

...play sound of water

...change number of spray -1..

Else if key == left and direction < 114

… set direction to current direction + 8..

Else if key == right and direction > 66

...set direction to current direction - 8…

Else if key == up and y speed < 5

...increase y speed by 1…

Else if key == down and yspeed > -1

...decrease y speed by 1

max tick = 4

Direction = current direction (IN RADIAN)

Delta\_x = cos Direction \* maxTick

Move to (current’s x + delta X, current’s Y)

}

Void getHurt( int damage)

{

newHealth = health - damage

Change health to new health

If health < 0

...change status to dead..

..play sound..

}

**CLASS BORDERLINE : PUBLIC ACTORBASE AND MOVABLE**

1. isBeing() and isZombie(), both return false. Both are inherited and redefined to distinguish different class
2. doSomething(), virtual since inherited from actorBase

**PSEUDO for BORDERLINE**

doSomething()

{

Double new x, double newY;

changeInYalg( newX, newY, this) ----> from movable

Move to (newX, newY)

If offscreen

return

}

**CLASS HUMAN PEDESTRIAN: PUBLIC ACTORBASE AND MOVABLE**

1. isZombe() and isWaterInteractive() , zombie returns true, Water returns false. Both virtual since inherited
2. doSomething(), virtual since inherited from actorBase

**PSEUDO for HUMAN PED**

doSomething()

{

If dead

Return

If overlaps with GR

Change GR to dead

Return

Double newY, newX;

changeInYalg(nexX, newY)

moveTo (newX, newY)

If offscreen

Return

changePlan (decrease by 1)

If (plan > 0)

Return

Else

determineNextPlan(this) ------> from movable class

}

**CLASS ZOMBIE PED : PUBLIC ACTORBASE AND MOVABLE**

1. isWaterInteractive() , which returns true. Virtual since inherited
2. doSomething(), virtual since inherited from actorBase
3. getTicks(), nonvirtual since a getter for zombie ped’s private
4. changeTicks(), nonvirtual since it changes a ped’s ticks

**Pseudo for Zombie Ped**

Void doSomething()

{

If dead

Return

If overlaps with GR{

...Hurt GR with 5 damage…

...change status to dead…

...play Sound…

...increase Score by 150…

return

}

If ( abs ( x - GR’s X) <= 330 and Y > GR’s Y){

...setDirection to 270…

If (x - GR’s x < 0)

changeXspeed to 1

If x - GR’s x > 0

changeXspeed to -1

Else

change X speed to 0

changeTicks(decrease by 1)

If ticks <= 0

Play Sound

changeTicks to 20

}

Double newY, newX;

changeInYalg(nexX, newY)

moveTo (newX, newY)

If offscreen

Return

If (plan > 0)

Return

Else

determineNextPlan(this) ------> from movable class

}

**CLASS ZOMBIE CAB: PUBLIC ACTORBASE, MOVABLE**

1. getIsDamageGR(), getLeftLane(), and getRightLane() : all non virtuals since they are getters for members within the class
2. changeisDamageGR(), non virtual since since its a mutator for members within its class
3. determineNextPlan(), virtual since it’s inherited from movable but redefined
4. doSomething(), virtual since it’s inherited from actorBase
5. isBeing() and isWaterInteractive(), virtual since they are different method inherited from actorBase

**PSEUDO for ZOMBIE CAB**

determineNextPlan(actorBase \*ptr)

{

changePlan (randint 4, 32)

Int tempyspeed = randint (-2,2)

Change ptr’s Y speed = ptr’s Y speed + tempyspeed

}

doSomething()

{

If dead

Return false

If overlaps with GR

If it hasn’t damaged GR

Play sound

Hurt GR by 20

If (X <= GR’s X

...changeXspeed to -5…

...change direction to 120 + randint (0, 19)

If ( X > GR’s X)

...change X speed to 5

...change direction to 660 - randint (0, 19)

...record that it has done damage to GR already...

Double newY, newX;

changeInYalg(nexX, newY)

moveTo (newX, newY)

If offscreen

Change alive status to false

Return

If (ySpeed > GR’s Y speed and if there’s an actor infront of it within 96 pixels)

Decrease yspeed by 0.5

Return

If (yspeed <= GR’s Y speed and there’s an actor behind if within 96 pixels)

Increase yspeed + 0.5

Return

If (plan > 0)

Return

Else

determineNextPlan(this) ------> from movable class

}

**CLASS OILSLICK: PUBLIC ACTORBASE**

1. doSomething(), virtual since inherited from actorbase
2. Iszombie and isBeing(), both return false and virtual since inherited from actorBase
3. spinGR(), nonvirtual because it’s a helper function for doSomething()

**PSEUDO for OILSLICK**

doSomething()

{

Double newY, newX;

changeInYalg(nexX, newY)

moveTo (newX, newY)

If offscreen

Return

If overlaps with GR

playSound

spinGR()

}

spingGR()

{

Set randClock = randint(0, 1)

If randclock == 0

Randclock == 1

Else

Randclock == -1

newDirection = randint(5, 20) \* 20

If (GR’s Direction + newDirection < 60 OR GR’s Direction + newDirection > 120

Return

Else

Increase GR’s Direction by newDirection

return

}

**CLASS HEALINGGOOD PUBLIC ACTORBASE**

1. isZombie() and isbeing() return false, isWaterInteractive() return true, all of which are all virtual since inherited
2. doSomething() is virtual since doSomething is inherited

**PSEUDO for HEALINGGOOD**

doSomething()

{

Double newY, newX;

changeInYalg(nexX, newY)

moveTo (newX, newY)

If offscreen

Return

If overlaps with GR

Increase GR’s Health by 10

Change alive status to dead

Play sound

Increase score by 250

}

**CLASS HOLYWATERGOODIE PUBLIC ACTORBASE**

1. isZombie() and isbeing() return false, isWaterInteractive() return true, all of which are all virtual since inherited
2. doSomething() is virtual since doSomething is inherited

**PSEUDO for HOLYWATERGOODIE**

doSomething()

{

Double newY, newX;

changeInYalg(nexX, newY)

moveTo (newX, newY)

If offscreen

Return

If Overlaps with GR

Increase numOfSpray in GR by 10

Change alive status to dead

Play sound

Increase score by 50

}

**CLASS SOULGOODIE PUBLIC ACTORBASE**

1. isZombie() and isBeing() both return false and both are virtual
2. doSomething() is virtual sense inherited

**PSEUDO for SOULGOODIE**

doSomething()

{

Double newY, newX;

changeInYalg(nexX, newY)

moveTo (newX, newY)

If isOffscreen

Return

If overlaps with GR

Change GR’s soul by -1

Change status to dead

Play sound

Increase score by 100

Increase Direction by 10

}

**CLASS WATERPROJECTILE : PUBLIC ACTORBASE**

1. doSomething() is virtual because it’s inherited
2. isZombie() is virtual because inherited as a different method
3. getPixels() is accessor and nonvirtual since it access members only for WP
4. changePixels() is a mutator and nonvirtual since it changes members only for WP

**PSEUDO FOR WATERPROJECTILE**

doSomething()

{

If dead

Return

Bool proceed = true

If (other actor overlap with water)

If other is waterineractive

Proceed = false

If iszombie()

If isbeing()

Change the zombie ped’s health by -1

Play sound

If ( zombie ped isnt overlapping with GR and randint(0,4) == 1

...let SW know to add heal

...let SW remember zombie’s ped X and Y

increaseScore by 150

Else

Decrease Zombie Cab’s health by 1

If Cab is not alive

playSound

If (randInt (0,4 ) == 1

Let SW know to add Oil Slick

Let SW remember cab’s X and Y

increaseScore by 200

Else

playSound

Else

If isBeing()

Flip X’s speed

If (human’s direction == 0)

Set Human’s direction to 180

Else

Set Human’s direction to 0

playSound

Else

Change Heal/Water alive to death

If proceed == false

Return

moveForward by SPRITE HEIGHT

Decrease pixels by SPRITEHEIGHT

If isOffScreen

Return

If getPixels() <=0

Change status to dead

}

2. I think I implemented all of them. Might have direction wrong because sometimes it works in Radian but other times it works in Degree. However, I play tested many times and it works as it should when direction needed to be changed.

3. Some assumption I made is that I could allocate work of actors to StudentWorld so that I do not have to remember a pointer and pass it in for actors like cab or projectile to determine what’s close to them. I also did not pass in any iterators as that is a bad practice, and I mainly let iterating through the loop methods to be assigned within the StudentWorld since it can access the container easier.