Aaron Minkov

Computer Science 32

CS32 Project 3 Report

1. Class functions explained!
   1. Actor Class
      * doSomething() is pure virtual because each actor has its own unique implementation of what to do.
      * setAsDead() is a void function that is NOT virtual because it will only ever be defined in the Actor class. It simply sets the boolean private data member m\_living to false.
      * alive() is a boolean function in Actor that is NOT virtual because it simply returns the m\_living data member that is private in Actor,
      * takeDamage is a virtual boolean function that has a base definition in Actor which returns false. Derived classes of the Actor class may have a different implementation.
      * The virtual boolean const function blocksBacteriumMovement() simply returns if the Actor blocks a bacteria from moving. All but Dirt, which is derived from Actor, return false.
      * The virtual boolean function isEdible(), which is const, returns if the object can be eaten. Food, a derived class of Actor, is the only one that returns true.
      * preventsLevelCompleting() is a virtual boolean const function that returns whether the object is significant in keeping that level from concluding. This is changed in classes like Bacterium and Pit.
      * getWorld() returns a pointer to the StudentWorld class that governs the game.
   2. Agent Class : Derived from Actor
      * doSomething() is defined as pure virtual because it’ll never be directly called by an Agent, only something derived from it (and it’d be specific to that class)
      * Int getHealth() returns m\_hitpoints, a private member of Agent; it does not need to be virtual because it’ll never be redefined.
      * Void restoreHealth() sets m\_hitpoints to a specified value ‘amount’. It’s also not virtual because it directly alters m\_hitpoints which is private.
      * The virtual function bool takeDamage(int damage) is redefined to subtract a specified amount of health, and set the object to dead if m\_hitpoints then becomes equal to or less than zero.
      * soundWhenHurt() & soundWhenDie() are pure virtual because each derived object has its own sounds.
   3. Socrates Class : Derived from Agent
      * doSomething() is defined to follow the instructed steps in the spec.
      * The non-virtual functions int getFlames() & getSprays() return the amount of m\_flame & m\_spray respectively, private members of Socrates.
      * Void addFlames(int amount) adds the specified amount of flames to m\_flame.
      * soundWhenHurt() & soundWhenDie() are specified to return the constant value of the sounds it should play when hurt or injured.
   4. Bacterium : Derived from Agent
      * doSomething() is defined to follow the spec for general instructions of Bacterium.
      * specificDoSomething() does a specific Bacterium’s task. It’s pure virtual in Bacterium.
      * The function int getFoodCount() is non virtual and returns the amount of food (m\_foodCount) eaten by a Bacteria.
      * Void incFoodCount() increases m\_foodCount by one.
      * resetFoodCount() sets m\_foodCount to zero.
      * Int getMovePlanDistance() returns the private data member m\_movePlanDistance.
      * Void setMovePlanDistance(int distance) sets the move plan distance data member to the specified distance.
      * setRandomDistance selects a random integer from 0 to 359 and sets the object’s direction to that angle. It’s no different in any derived Bacterium class so it’s non virtual.
      * move(int x, int y) sets the new position of a Bacterium. It’s generic and never changed (non-virtual)
      * Virtual bool takeDamage(int damage) is redefined to check if it’s alive, call Agent::takeDamage(), then see if it’s still alive & act accordingly.
      * preventsLevelCompleting() returns true because all alive Bacterium prevent the level from ending.
   5. Salmonella : derived from Bacterium
      * specificDoSomething() completes the specific task necessary for regular salmonella. It’s called by Bacterium::doSomething().
      * soundWhenHurt() & soundWhenDie() are defined to return the appropriate sound constants for regular salmonella.
   6. Aggressive Salmonella : derived from Bacterium
      * specificDoSomething() completes the specific task necessary for aggressive salmonella. It’s called by Bacterium::doSomething().
      * soundWhenHurt() & soundWhenDie() are defined to return the appropriate sound constants for aggressive salmonella.
   7. E. coli : derived from Bacterium
      * specificDoSomething() completes the specific task necessary for E. coli. It’s called by Bacterium::doSomething().
      * soundWhenHurt() & soundWhenDie() are defined to return the appropriate sound constants for E. coli.
   8. Pit : derived from Actor
      * doSomething() is defined to follow the spec for the Pit class.
      * preventsLevelCompleting() returns true because Pits that are visible & alive prevent the level from finishing.
   9. Dirt : derived from Actor
      * doSomething() does nothing, but it’s defined because the function is called by StudentWorld’s move().
      * Virtual bool takeDamage(int damage) sets the Dirt object to dead.
      * blocksBacteriumMovement() returns true because Bacterium can’t run over Dirt objects.
   10. Activating Object : derived from Actor
       * The only public function has a pure virtual void doSomething class.
   11. Food : derived from Activating Object
       * doSomething() does nothing. It’s food! It does get defined though & returns nothing.
       * Virtual bool isEdible() const returns true because food is edible.
   12. Projectile : derived from Activating Object
       * doSomething() is pure virtual void function.
       * Int getFlightDistance() returns m\_flightDistance.
       * Void decFlightDistance() decreases m\_flightDistance by one.
   13. Flame : derived from Projectile
       * doSomething() simply follows the spec for Flame
   14. Spray : derived from Projectile
       * doSomething() simply follows the spec for Spray
   15. Goodie : derived from Actor
       * takeDamage() sets the Goodie to dead.
       * decLifetime() decreases m\_lifetime by 1.
       * getLifetime() returns m\_lifetime.
   16. RestoreHealthGoodie, FlamethrowerGoodie, ExtraLifeGoodie, and Fungus : derived from Goodie
       * doSomething() is defined for each specific goodie type to do the proper actions specified in the spec.
2. As far as I know, I’ve implemented all of the requirements listed in the spec! The game seems to function properly in all circumstance.
3. There weren’t any things that I truly needed to guess on, aside from the formatting of certain functions. I changed up the way move() operates because I wasn’t able to complete it all in one “while” loop.
4. Testing Classes (I will only specify the most derived classes, as those are the only ones that appear in the actual GameWorld)
   * Socrates
     + I tested socrates by checking if each key press responded properly within doSomething(). I made sure that a full rotation of movements would bring Socrates to the exact same starting position. I also confirmed that the deployment of Spray & Flame were identical to the full version of Kontagion provided to us.
   * Regular Salmonella
     + I first coded Regular Salmonella without the other Bacterium. I created a circle around each food object in the game with a random goodie to see if, when Regular Salmonella was within that radius, it would attempt to move toward the food. All the other features of Regular Salmonella were easy to observe, so I just compared their actions to those expected from the spec & the full version of Kontagion.
   * Aggressive Salmonella
     + I did the exact same thing for Aggressive Salmonella as I did for Regular Salmonella, but I commented out the specificDoSomething() code of Regular Salmonella to confirm that I was only observing the aggressive ones. I ensured that while Aggressive Salmonella was near Socrates, it followed him properly without moving out of the boundaries of the petri dish.
   * E. Coli
     + Again, I did the same first steps as the previous Bacterium to test and ensure E. Coli was moving properly. I simply moved Socrates around and ensured that the E. coli would get stuck at the proper times (if it hits dirt and can’t turn any other way) and that in other cases, E. Coli moved toward Socrates without exiting the petri dish.
   * Pit
     + Pit was not a hard class to test. I ensured that the bacterium spawned appropriately out of the Pit, and that once all of the m\_salmonella, m\_aggSalmonella, and m\_eColi were spawned, the Pit disappeared. I logged the value of these private data member variables to see when they reached zero.
   * Dirt
     + To test that dirt was functioning properly, I made sure that it did nothing… When hit by flame or spray, I observed that takeDamage deletes the dirt object.
   * Food
     + Food also did nothing. I checked to make sure that it was picked up/deleted when overlapped by a Bacterium.
   * Flame
     + To test that Socrates’ flame action worked properly, I created a circle of dirt exactly 32 pixels away from Socrates’ starting position (Socrates at the center). Then, I fired a Flame and ensured that the flame objects got far enough to destroy the dirt
   * Spray
     + To test that Socrates’ spray action worked properly, I repeatedly fired around the map. Seeing that I was able to clear the petri dish of dirt, I knew that the spray object worked properly.
   * Goodie
     + All I did to test that Goodies worked was observe them appear & disappear throughout each level. It’s hard to test exactly how often the Goodies are being placed because the chance of placing one is random.