# Design Scheme

Below is the schematic of my StudentWorld class

StudentWorld

init()

move()

cleanUp()

(Note: function arguments omitted)

bool isBlocked() // Return true if there’s a blockage at location

bool hasActorType() // Return true if there’s an Actor of a specific type at location

void addActor() // Add a pointer of an Actor to location

void moveActor() // Move an Actor’s pointer to another location

int reduceFood() // Reduce the amount of food at location by a certain amount, returns the actual amount reduced or -1 if there’s no food

void stackFood() // Add an amount of food to location, or add a new pointer to a Food object if there’s no food present

void stunAOE() // Calls the getStunned method for all Actors at location

void damageAOE() // Deals an amount of damage to all Actors at location

bool damageRand() // Deals damage to a random actor at location. Each actor rolls an integer, the actor with largest integer takes the damage. A pointer to an Actor is passed in so that the source of damage wouldn’t take damage itself.

vector<Actor\*> m\_actors[VIEW\_WIDTH][VIEW\_HEIGHT] // A two-dimensional array of vectors containing pointers to Actors, corresponding to coordinate (x, y) on the cartesian plane

Also, I organized my Actor class inheritance as shown below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Actor // Every method defined here are called either by StudentWorld or by its subclasses  int getType() // Returns an int specifying the type of the Actor, to let the StudentWorld know what it is dealing with  int getFaction() // Returns an int specifying the faction the Actor belongs to: 0-4 for the ant colonies and 5 for neutral  bool isBlockage() // Returns true if the Actor blocks the terrain  bool isImmobile() // Returns true if the Actor is immobile and can’t be attacked  bool isActive() // Returns true if the Actor hasn’t acted during current tick  StudentWorld\* getWorld() // Returns a pointer to the StudentWorld the Actor belongs to  virtual int getHP() = 0 // Pure virtual because the two subclasses, Mortal and Terrain, exhibit completely different behavior  virtual bool isDead() = 0 // (Same as above)  void activate() // Activate the Actor so that it can perform action in current tick  void deactivate() // Deactivate Actor so that it can’t perform action in current tick  The two methods above are correlated to the move() method in StudentWorld Class. When an Actor moves to another coordinate, since the move() method traverses the Actor container structure in a certain order, a same Actor can be visited twice during each tick. By deactivating an Actor after it performs action, StudentWorld is not allowed to call doSomething for a same actor twice in a row in the same tick.  virtual void getStunned() // Does nothing. Mortal /Insect may exhibit other behavior.  virtual void getBitten() // (Same as above)  virtual void gainHP() // (Same as above)  virtual void loseHP() // (Same as above)  virtual void doSomething() = 0 | | | | | | | | |
| Terrain // Does not have HP, immutable  virtual int getHP() // Returns 0 because Terrain don’t have HP  virtual bool isDead() = 0 // Returns false because Terrain can’t die  virtual void doSomething() = 0 // Pure virtual because different Terrain do completely different things | | | Mortal // Has HP, can die, can “eat”  virtual int getHP() // Returns the actual HP value  virtual bool isDead() = 0 // Returns false if HP <= 0  virtual void gainHP() // Actually gains HP  virtual void loseHP() // Actually loses HP  bool consumeFood() // Calls StudentWorld’s reduceFood method to eat food at current location  virtual void doSomething() = 0 // Still pure virtual because its subclasses exhibit different behavior | | | | | |
|
| Pebble  virtual void doSomething() // Does nothing | PoolofWater  virtual void doSomething() // Calls StudentWorld’s stunAOE method | Poison  virtual void doSomething() /// Calls StudentWorld’s damageAOE method | Pheromone  virtual void doSomething() // loseHP(1) | AntHill  virtual void doSomething() // Performs the AntHill’s actions described in the spec | Food  virtual void doSomething() // Does nothing | Insect | | |
|
| Ant | BabyGrasshopper | AdultGrasshopper |

# Bugs & Unfinished Parts

|  |  |  |
| --- | --- | --- |
| Insect // Can move, bite,  (A number of trivial functions are omitted here)  bool bite() // Calls StudentWorld’s damageRand method  bool eat() // Calls Insect’s consumeFood method, with a fixed value as input  bool move() // Moves the Insect elsewhere  bool attemptAct() // Returns true if not dead or sleeping  void dropfood() // Calls StudentWorld’s stackFood method  void getUnstunned() // Unstuns self  virtual void getStunned() // Stuns self  virtual void getBitten() // Calls Mortal’s loseHP method  virtual void doSomething() = 0 // Still pure virtual because its subclasses exhibit different behavior | | |
|
| Ant  virtual void getBitten() // Calls Insect’s getBitten method and in addition set self status to bitten  virtual void doSomething() // Does the ant things as required by the spec | BabyGrasshopper  virtual void doSomething() // Does the baby grasshopper things | AdultGrasshopper  virtual void getStunned() // Do nothing  virtual void getBitten() // Roll a number between 0 and 1. Calls bite() if it rolls a one.  virtual void doSomething() // Does the adult grasshopper things |

# Assumptions

# Tests

ANTHILL

If the anthill produces 5 ants at the start of the game, and by the time the game ends any one of the four anthill produces more than 6 ants, the class behaves properly.

POISON

Set the poison damage to 9999. Observe the game, if any insect instantly turns into food when it steps onto the poison, the class behaves properly.

POOL OF WATER

Observe the game. If ants are stunned by 2 (and no more than 2) ticks , baby grasshoppers are stunned by 4 (and no more than 4) ticks, and adult grasshoppers are never stunned, the class behaves properly.

PEBBLE

Observe the game. If no insect ever passes through the pebble, the class behaves properly.

PHEROMONE

Observe the game. If pheromone’s color corresponds to the ant colony color and it disappears after ~256 rounds, the class behaves properly.

BABY GRASSHOPPER

Observe the game. If the baby grasshopper walks 1 grid every 2 ticks, waits 2 tick when it is blocked, drops food when stepping onto poison (whose damage is set to 9999 as above), is stunned for 4 ticks on pool of water, and can eventually become adult grasshopper, then the class behaves properly.

ADULT GRASSHOPPER

1. Observe the game. The adult grasshopper should occasionally jump.
2. Set a break point when its bite() method returns true. If the break point is triggered, check the location of the adult grasshopper and see if another insect is present at current location.
3. Set a break point when eat() method returns true. If the break point is triggered, check the location of the adult grasshopper and see if there’s food at current location
4. Set a break point when bite() is called. If the breakpoint is triggered, look at the call stack and check if the bite() function is called on top of a getBitten() function.
5. f the adult grasshopper walks 1 grid every 2 ticks, waits 2 tick when it is blocked, drops food when stepping onto poison (damage 9999), is never stunned by pool of water, and satisfies 1-4 above, then the class behaves properly.

ANT

1. Set a break point when its getBitten() is called. If the break point is triggered, check the call stack and see if the getBitten() function is called on top of the doSomething() method of an adult grasshopper or an enemy ant at the same location, and the ant’s state is set to bitten.
2. Remove all commands from uscant.bug, check if all ants die within the same tick they are born because they are unable to fetch commands
3. Using the original uscant.bug, observe the game and confirm the following behavior:
   1. The ant gets a random direction, and in the next tick it moves forward 1 grid
   2. New ants are produced, which means that the ants successfully fetch food to the anthill
   3. Number of ants on screen decreases because they die
   4. Set a breakpoint when its gainHP() (resulting from eating food) is called. If the breakpoint is triggered, check to see if the ant’s current HP is below 25, and the method is called by a command evaluating to eatFood.
4. Add the following lines below the on\_food line of uscant.bug, and check if the ants keep spinning clockwise on food:

rotateClockwise

goto on\_food

1. Add the following lines below the on\_food line of uscant.bug, and check if the ants keep spinning counterclockwise on food:

rotateCounterClockwise

goto on\_food

1. Add the following lines at the bottom of uscant.bug:

emit:

emitPheromone

goto start

* 1. Add the line “if i\_am\_carrying\_food then goto emit” under the start line. Check if the pheromones are distributed around the food piles on the map.
  2. Add the line “if i\_smell\_danger\_in\_front\_of\_me then goto emit” under the start line. Check if the pheromones are distributed around the poisons on the map.
  3. Add the line “if i\_was\_blocked\_from\_moving then goto emit” under the start line. Check if the pheromones are distributed around the pebbles on the map.
  4. Add the following lines under the start line. Observe the game and check if the ant produces some spaced pheromones as it moves

generateRandomNumber 2

if last\_random\_number\_was\_zero then goto emit

* 1. Add the line “if i\_am\_standing\_with\_an\_enemy then goto emit” under the start line. Observe the game and check if the pheromones are produced when the ant meets with another insect of different faction
  2. Add/edit the following lines under the start line, observe the game and check if food is dropped around the anthills.

if i\_am\_standing\_on\_my\_anthill then goto emit

if i\_smell\_pheromone\_in\_front\_of\_me then goto on\_hill

* 1. Add the line ”if i\_was\_bit then goto emit” under the start line. Observe the game and check if pheromone is dropped when an ant meets with enemy.