Class Actor

1. Constructor
2. Destructor: virtual ~Actor() I decide to define it virtual because some derived classes need to have specific destructors.
3. Virtual int eatOneTime() const In order to know how many food an insect can eat at one time, I add this function to the base class. Since other classes don’t need this function and I need polymorphism, I make it virtual.
4. Virtual void doSomething() = 0 Since each class must be have a doSomething() function, I add it to the base class. Each class has a different doSomething() function, I make it pure virtual.
5. Bool checkAlive() const Most classes might be dead and then be removed from the simulation. This function can find which one is to be removed.
6. Void setDead() When those energy holders get out of hit point, they need to be set dead and then get removed.
7. Actor\* getAnActorAtTheProposedLocation(int X, int Y) const Those insects need to know if there is food or pebble in the square to make sure whether they can eat or whether they are blocked. This function returns the first element in the list and then use blockAnt() and findFood() function to make sure if it belongs to one of these two kinds.
8. StudentWorld \* myWorld() const Each class need to have a pointer to the StudentWorld class and can call functions belonging to StudentWorld class, so each of them need to have a myWorld() function to return this pointer.
9. void setMoved() During one tick, after one Actor has been called to doSomething, they must be marked in case it might be called again.
10. void setUnmoved() At the end of each tick, after every actor has been called to do Something, they must be set unmoved and then they’ll be able to do Something in the next tick.
11. virtual bool blockAnt() const To know whether an Actor is a pebble or not, I add this function to the base class. Everything will return false except a pebble, so it is virtual.
12. virtual bool isEdible() const To know whether an actor is food or not, I add this function to the base class. Everything will return false except a food class, so it is virtual.
13. virtual bool isPheromone() const To know whether an actor is pheromone or not, I add this function to the base class. Everything will return false except a pheromone class, so it is virtual.
14. virtual bool amAHill() const To know whether an actor is pheromone or not, I add this function to the base class. Everything will return false except an Anthill class, so it is virtual.
15. virtual bool isDangerous() const Poison and Water are dangerous to Ant, so only those two classes will return true.
16. void setMyHit(int m) Most classes are energy holders, they need to be set hit points when constructed.
17. Void decrease(int m = 1) During each tick, each energy holders must decrease it hit by one. An insect might decrease more hit points during one tick, so the parameter need to be a variable as well.
18. Void increase (int m) Most classes should be able to increase their hit points.
19. int getMyHit() const Most classes should be able to check their hit points and have a responded action.
20. virtual void setPoisoned() Sone classes are able to get poisoned, but others may not. So it is virtual.
21. virtual void setWatered() Some classes can be stunned by water but others can not. So it is virtual as well.
22. virtual bool isEnemy() const Some classes are the enemies of Adult\_Grasshopper class, but others are not.
23. virtual int myHarm() const { return 0; } Adult\_Grasshopper and Ant can bite others and cause others to decrease a certain number of points. Other classes can’t do this, so this is virtual as well.
24. virtual int getMyColony() const { return -1; } Ant and Anthill have a meaningful function getMyColony(), but others don’t. I just make it return -1 which is different from Ant and Anthill to make sure they won’t cause some trouble.
25. virtual void setBitten(Actor\* p) Some classes are able to be bitten but others don’t so I make it virtual. Adult\_Grasshopper has different behavior when gets bitten as well.

Class Poison: public Actor

1. Constructor
2. virtual void doSomething() It need to poison the Actors that can be poisoned in the list when it’s called.
3. virtual bool isDangerous() const It returns true, which is different from the base class.

Class Water: public Actor

1. Constructor
2. virtual void doSomething() It need to change the status of Actors that can be stunned by water in the list when it’s called.
3. virtual bool isDangerous() const It returns true, which is different from the base class.

Class Pebble: public Actor

1. Constructor
2. virtual void doSomething() Although it can’t do anything, when this is called, it’s status must be set to moved.
3. virtual bool blockAnt() const It returns true, which is different from the base class.

Class Food: public Actor

1. Constructor A food object is created when loading the field.
2. Constructor A food object is created because an insect is dead.
3. virtual void doSomething() It sets its status to moved and check it’s hit points.
4. virtual bool isEdible() const It returns true, which is different from the base class.

Class Pheromone: public Actor

1. Constructor
2. virtual void doSomething() It sets its status to moved, decrease its hit points by one and check if its hit point reaches zero.
3. virtual bool isPheromone() const It returns true, which is different from the base class.
4. virtual int getMyColony() const It returns its colony type, which is different from the base class.

Class Insect: public Actor

1. Constructor
2. Virtual ~Insect() The destructor of insects is different from the base class. It adds 100 units of food to the square.
3. virtual void doSomething() = 0 It’s pure virtual because each derived class has a different doSomething() function.
4. bool checkStunned() const Some classes can be stunned, so the Insect class should be able to check whether it is stunned or not.
5. virtual void setStunned() Some classes are able to be stunned, others aren’t. So this is virtual.
6. virtual int eatOneTime() const All derived classes of Insect can eat, and has a data member(int m\_eatOneTime), which is different from the base class.
7. virtual void setPoisoned() Some classes can be poisoned while others can’t. So this is virtual.
8. void setUnbitten() set m\_bitten to false. This is the same for each class. So it doesn’t need to be virtual.
9. void decreaseStunnedTime() Decrease the stunned time by one. This is the same for each class.
10. void setUnStunned() Set m\_stunned to false. This is the same for all classes.
11. bool goodStatus() Each derived classes need to check if it’s now in good status before they really do something. This is the same for all derived classes.
12. virtual bool move() Each class’s move function differs. But some of them have some similarities. So I make it virtual and has some functionality.
13. void setMemory(int X, int Y) Some classes need to remember they have been bite, stunned or poisoned at the current square.
14. int getMemoryX() const
15. int getMemoryY() const Return the memory
16. void pickRandomDirection() most derived classes of Insect can pick a random direction.
17. bool isBlocked(int X, int Y) const most derived classes of Insect can detect whether the proposed location is blocked or not.
18. Actor\* findFood(int X, int Y) const It returns an pointer to an actor, which is actually a food object or a nullptr. I push\_front each food object. If there is a food object, it must be in the first position.
19. void eat(Actor\* a) The parameter is actually a pointer to a food object.
20. virtual void setWatered() Some classes can be stunned by water but others cann’t. So it is virtual.
21. virtual bool isEnemy() const This returns true which is different from the base class.
22. virtual void setBitten(Actor\* p) This is virtual because some classes can be bitten, Adult\_Grasshopper behaves differently when it’s bitten.

Class Anthill: public Insect

1. Constructor
2. virtual void doSomething()
3. virtual bool move() Of all the derived classes of Insect, only Anthill can’t move. It does nothing and returns true.
4. virtual void setStunned() An Anthill can’t be stunned. So this is virtual and does nothing.
5. virtual void setPoisoned() An Anthill can’t be poisoned. So this is virtual and does nothing.
6. virtual void setWatered() An Anthill can’t be stunned by water. So this is virtual and does nothing.
7. virtual bool isEnemy() const This returns false, different from its base class, which is Insect.
8. void produce() This produces an ant. Only the Anthill Object can produce, so it is not virtual.
9. virtual void setBitten(Actor\* p) An Anthill can’t be bitten. So this is virtual and does nothing.
10. virtual bool amAHill() const This returns true, different from Actor class. So this is virtual.
11. virtual int getMyColony() const This is different from Actor class, which returns -1. So this is virtual.

Class Ant: public Insect

1. Constructor
2. Destructor virtual ~Ant() An Ant Object needs a virtual destructor because it needs to call the StudentWorld to decrease the number count.
3. virtual void doSomething()
4. virtual int myHarm() It’s virtual because it’s different from the Actor class.
5. bool simpleInterpreter() Since only Ant has this functionality, it is not virtual.
6. void setCounterValue(int m) Since only Ant has this functionality, it is not virtual.
7. virtual int getMyColony() const It’s virtual because it’s different from the Actor class.

Class Grasshopper: public Insect

1. Constructor
2. Destructor virtual ~Grasshopper() It has a destructor but does nothing because it’s the base class for Baby\_Grasshopper and Adult\_Grasshopper.
3. virtual void doSomething() = 0 It’s pure virtual because each of its derived class has a different doSomething() function.
4. virtual bool move() It’s virtual because it’s different from Insect class.
5. bool haveARest() This function decides whether the grasshopper wants to have a rest or not after eating. This is the same for its derived classes. So it’s not virtual.
6. void decreaseRemainingDis() This is the same for its derived classes. So it’s not virtual.
7. void setRandomDis() This is the same for its derived classes. So it’s not virtual.
8. int getRemainingDistance() const This is the same for its derived classes. So it’s not virtual.
9. void setRemainingDis(int m) This is the same for its derived classes. So it’s not virtual.

Class Baby\_Grasshopper: public Grasshopper

1. Constructor
2. virtual void doSomething()

Class Adult\_Grasshopper: public Grasshopper

1. Constructor
2. virtual void doSomething()
3. virtual void setPoisoned() It’s virtual because it can’t be poisoned, which is different from Insect class.
4. virtual void setWatered() It’s virtual because it can’t be stunned by water, which is different from Insect class.
5. bool decideToBite() const It’s not virtual because only Adult\_Grasshopper has this functionality.
6. bool decideToJump() It’s not virtual because only Adult\_Grasshopper has this functionality.
7. virtual int myHarm() It’ virtual because it’s different from Actor class.
8. virtual void setBitten(Actor\* p) It’s virtual because an Adult\_Grasshopper may bite back when it’s bitten, which is different from the Insect class.

I think I finish all the functionalities.

My Assumption is the Anthill may be set dead after it’s hit points run out, and then removed from the simulation.

Another assumption is a square can hold as many as four different pheromone object, and each object has a maximum hit point of 768.

My Tests

To test whether pebble can block an Ant and Grasshopper, I modified the field.txt to have pebbles around a grasshopper and the anthill. As a result, the grasshopper and ant produced by the ant hill cannot move at all.

To test whether water can stun an Ant and Baby\_Grasshopper, but not an Adult\_Grasshopper, I surround an Anthill and A Baby\_Grasshopper by water Object and watch tick by tick.

To test whether a poison can decrease the hit point of an insect and stun it, I add a “cerr” in the setPoisoned function and watch their stunning time.

To test whether a food works, I set it initial hit point to 200, and surround a grasshopper with four food objects. After one tick, one food object just disappears from the simulation, and it’s no longer exists in myActors[X][Y] list as well.

To test whether an anthill can produce whenever its hit points reaches 2000. I initiate its hit points to 20000, and watch how many ants it can produce.

To test whether a Baby\_Grasshopper can be stunned by water, I surround a Baby\_Grasshopper by water and see if it’s going to be stunned for exactly 4 ticks. To test if Baby\_Grasshopper can be poisoned by poison, a I also initiate Baby\_Grasshopper with 154 hit points, surround one by four poison, and see if it is removed after 4 ticks. Then I initialize it to 500 hit points again, and see if it can move out from the poison it run into at the sixth tick. Similarly with water.

To test whether an Adult\_Grasshopper can be stunned by water or poison, I initialize a Baby\_Grasshopper with 2000 hit points and see if the Adult\_Grasshopper is stunned. I also surround four Baby\_Grasshoppers with 2000 points with pebbles to see if the Adult\_Grasshopper are going to bite back and forth.

To test whether an ant can emit pheromone, and whether a pheromone can decrease its hit point and get removed from the simulation, I edit the USC.bug file, add emitPheromone to its fifth command, and initialize pheromone with 5 hit points.