Amir Ali Yazdi-Nejad

UID: 204846462

CS 32

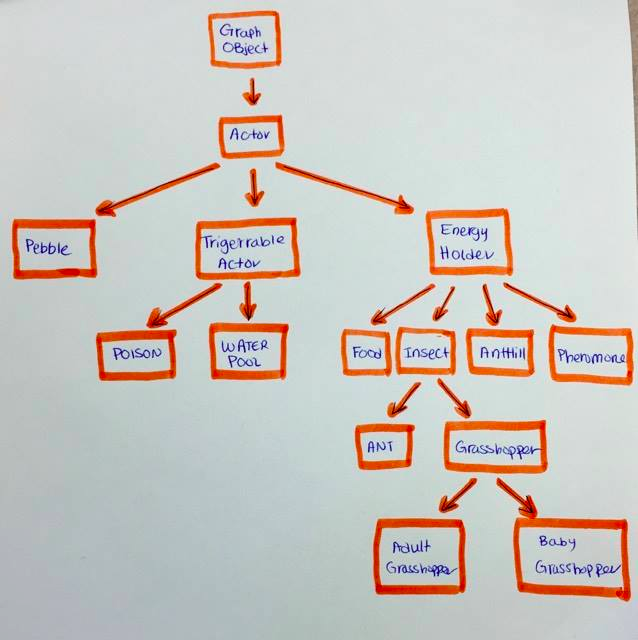
Winter 2017

Professor Nachenberg

**Project 3 Report**

1. **Introduction:**

This year, we made a graphical Ant simulation to gain more experience on Object Oriented Programming, inheritance, and polymorphism. Our hierarchy was split to three subgroups (Pebble, Danger, and Energy Holder). Danger, itself, was split into two other subgroups (Poison and Water Pools), and EnergyHolder was split to Food, Insect, AntHilll, Pheremone. Insect was split into Ant and Grasshopper, and Grasshopper was split into Baby Grasshopper and Adult Grasshopper. The design for our Actor inheritance Hierarchy looks like this:

****

**Description of the different classes and each of the public member functions:**

**--------------------------------------------------------------------------------------------------------------------------------**

**ACTOR CLASS:**

Actor class is the first superclass inside our Actor.h, and everything else inherits from this class. This class is an abstract class, since it has a pure virtual function (doSomething()).

**Non-Virtual functions:**

**Actor(int imageID, int startX, int startY, Direction dir, int depth, StudentWorld \*student);**

**// The constructor initializes all the member functions of the class. It makes sure nothing is left uninitialized.**

**void setMoved(bool change);**

**// This function sets the moving condition to true. Therefore, this actor will not be able to move during that tick again.**

**bool getMoved() const;**

**// It gets a Boolean to see if the actor has already moved or not. If it has moved, it will not move again.**

**StudentWorld\* getWorld() const;**

**// Get this actor's world. It will let the user have access to StudentWorld pointers throughout Actor.cpp**

**Pure Virtual Functions:**

**virtual void doSomething() = 0;**

**Virtual functions:**

**Virtual ~Actor() {}**

**// Virtual destructor is always needed for the base class of a hierarchy when you are using polymorphism.**

**virtual bool blocksMovement() const;**

**// This function checks to see if the given actor is a pebble or anything else. If it is a pebble, it will return true, else it will return false.**

**virtual bool isDead() const;**

**// This function checks to see if the given actor is dead or alive. If it is dead, it will return true, else it will return false.**

**virtual void getBitten(int amt);**

**// Cause this actor to be be bitten, suffering an amount of damage.**

**virtual void getPoisoned();**

**// Cause this actor to be be poisoned, suffering an amount of damage.**

**virtual void getStunned();**

**// Cause this actor to be be stunned for 2 extra ticks.**

**virtual bool isEdible() const;**

**// This function checks to see if the given actor is a food or not. If it is a food, it will return true, else it will return false.**

**virtual bool isPheromone(int colony) const;**

**// This function checks to see if the given actor is a pheromone of the same colony as an ant.**

**virtual bool isEnemy(int colony) const;**

**// Is this actor an enemy of an ant of the indicated colony?**

**virtual bool isDangerous(int colony) const;**

**// Is this actor detected as dangerous by an ant of the indicated colony?**

**virtual bool isAntHill(int colony) const;**

**// Is this actor the anthill of the indicated colony?**

**PEBBLE CLASS: public Actor**

Non-Virtual functions:

**Pebble(int startX, int startY, StudentWorld \*student);**

**A Pebble object must have an image ID of IID\_ROCK.**

**A Pebble must always start at the proper location as specified by the field's data file.**

**A Pebble must start out facing right.**

**A Pebble object must start out with a depth of 1.**

Virtual functions:

**virtual bool blocksMovement() const;**

**// This function returns true for pebbles which means that there exists a pebble in that square.**

**virtual void doSomething();**

**This function does not do anything. It only returns.**

**DANGER CLASS: public Actor**

Non-Virtual functions:

**Danger(int imageID, int startX, int startY, StudentWorld \*student);**

**// This class is a derived class of the Actor class which also is a superclass for the water and poison classes too.**

Virtual functions:

**virtual bool isDangerous(int colony) const;**

**// This functions let the user know that water and poison are dangers to the ants.**

**WATER CLASS: public Danger**

Non-Virtual functions:

**Water(int startX, int startY, StudentWorld \*student);**

**A Pool of Water object must have an image ID of IID\_WATER\_POOL.**

**A Pool of Water must always start at the proper location as specified by the field's data**

**file.**

**A Pool of Water always starts out facing right.**

**A Pool of Water always has a depth of 2.**

Virtual functions:

**virtual void doSomething();**

**The Pool of Water must attempt to stun all Insects on the same square as it. Stunning an**

**insect (if the insect can be stunned), causes it to sleep for 2 additional ticks (above and**

**beyond any existing sleeping the insect is doing).**

**POISON CLASS: public Danger**

Non-Virtual functions:

**Poison(int startX, int startY, StudentWorld \*student);**

**1. A Poison object must have an image ID of IID\_POISON.**

**2. A Poison object must always start at the proper location as specified by the field's data**

**file.**

**3. A Poison object always starts out facing right.**

**4. A Poison object always has a depth of 2.**

Virtual functions:

**virtual void doSomething();**

**The Poison object must attempt to poison all Insects on the same square as it**

**ENERGYHOLDER CLASS: public Actor**

Non-Virtual functions:

**EnergyHolder(int imageID, int startX, int startY, Direction dir, int depth, StudentWorld \*student, int hitPoints);**

**// This class is another derived class from the actor class which also is a superclass for ants, anthills, grasshoppers, pheromones and anthills.**

**int getHitPoints() const;**

**// This function returns the amount of energy that is left in each energy holder.**

**void updateHitPoints(int hitPoints);**

**// This function adds or reduces the amount of energy that is left in an energy holder.**

**void addFood(int amt);**

**// Add an amount of food to this actor's location.**

**int pickupFood(int amt);**

**// Have this actor pick up an amount of food.**

**int pickupAndEatFood(int amt);**

**// Have this actor pick up an amount of food and eat it.**

Virtual functions:

**virtual bool isDead() const;**

**virtual bool becomesFoodUponDeath() const;**

**// Does this actor become food when it dies?**

**ANTHILL CLASS: public Eergyholder**

Non-Virtual functions:

**AntHill(int startX, int startY, StudentWorld \*student, int colonyNum, Compiler\* program);**

**An AntHill object must have an image ID of IID\_ANTHILL (all ant colonies share the same anthill graphic).**

**An AntHill object must have its x,y location specified for it.**

**All new AntHill objects must have a starting hit points of 8,999 units.**

**48**

**An AntHill object must always start out facing right.**

**An AntHill object must start out with a depth of 2.**

**Each AntHill object must have its colony number passed in, so it knows which colony of**

**ants it represents.**

**Each AntHill object must have a Compiler object passed in (by the StudentWorld::init()**

**method), where the Compiler object holds the compiled Bugs Commands that will**

**govern the behavior of ants born from that anthill**

Virtual functions:

**virtual void doSomething();**

**It decreases its (queen’s) hit points by 1 unit.**

**If the anthill’s (queen’s) hit points reaches zero units, then:**

**a. The anthill must set its status to dead so it can be removed from the simulation**

**after the current tick.**

**b. The anthill must immediately return.**

**The anthill checks to see if there is any food on its square. If so:**

**a. It will eat up to 10,000 units of food from the square, and this amount will directly**

**increase its hit points.**

**b. The anthill must immediately return.**

**The anthill checks to see if it has enough energy - at least 2,000 hit points - to produce a**

**new ant. If so:**

**a. It adds a new ant of the same colony number to its square in the simulation.**

**b. It reduces its own hit points by 1,500.**

**c. It asks StudentWorld to increase the count of the total number of ants that this**

**colony has produced. This needs to be tracked in order to determine the winner**

**ant colony.**

**virtual bool isAntHill(int colony) const;**

**// This function checks to see if the given anthill belongs to the given ant. If it does, it will return true, else it will return false.**

**FOOD CLASS: public EnergyHolder**

Non-Virtual functions:

**Food(int startX, int startY, StudentWorld \*student, int hitPoints);**

**A Food object must have an image ID of IID\_FOOD.**

**A Food object must have its x,y location specified for it.**

**All Food objects that are created at the start of the simulation must hold 6000 units of**

**food. All Food objects that are created due to an insect dying must hold 100 units of**

**food.**

**A Food object must always start out facing right.**

**A Food object must start out with a depth of 2.**

Virtual functions:

**virtual void doSomething();**

**// This function does not do anything. It only returns.**

**virtual bool isEdible() const;**

**// This function only returns true for squares which contain food**

**--------------------------------------------------------------------------------------------------------------------------------**

**PHEROMONE CLASS: public EnergyHolder**

Non-Virtual functions:

**Pheromone(int startX, int startY, StudentWorld \*student, int colony);**

**A Pheromone object must have an image ID of IID\_PHEROMONE\_TYPE0 (if generated**

**by ants of colony 0), IID\_PHEROMONE\_TYPE1 (if generated by ants of colony 1), IID\_PHEROMONE\_TYPE2 (if generated by ants of colony 2),**

**IID\_PHEROMONE\_TYPE3 (if generated by ants of colony 3)**

**A Pheromone object must have its x,y location specified for it.**

**All new Pheromone objects must have a strength of 256 units.**

**A Pheromone object must always start out facing right.**

**A Pheromone object must start out with a depth of 2.**

**void increaseStrength();**

**// This function increases the amount of pheromone at a point by 256 points max. Pheromone points cannot exceed 768 points.**

Virtual functions:

**virtual void doSomething();**

**They decrease their strength by 1 unit.**

**If their strength reaches zero units, then they must set their status to dead so they can**

**be removed from the simulation after the current tick.**

**virtual bool isPheromone(int colony) const;**

**// This function checks to see if there exists pheromone inside the square of the given ant. If it does, increase its strength, else create new pheromone.**

**INSECT CLASS: public EnergyHolder**

Non-Virtual functions:

**Insect(int imageID, int startX, int startY, StudentWorld \*student, int hitPoints);**

**// Insect is a derived class of EnergyHolder which is also a superclass for ants, baby grasshoppers, and adult grasshoppers.**

**int getSleepTicks() const;**

**// This function returns the number of ticks that each insect sleeps.**

Virtual functions:

**virtual void getBitten(int amt);**

**// This functions decreases the hit points of the given actor by the given amount**

**virtual void getPoisoned();**

**// This function poisons the insect at that position, causing it to lose hit points.**

**virtual void getStunned();**

**// This function stuns the insect, causing it to sleep two extra ticks.**

**virtual bool isEnemy(int colony);**

**// This function lets the ants know if there exists an enemy inside the square that they are in.**

**virtual bool becomesFoodUponDeath() const;**

**virtual void setPool(bool change);**

**// This function lets the insect know that it is already stunned by the pool, and it should leave after it sleep ticks reaches zero.**

**virtual bool moveForwardIfPossible();**

**// Move this insect one step forward if possible, and return true, otherwise, return false without moving.**

**void increaseSleepTicks(int amt);**

**// Increase the number of ticks this insect will sleep by the indicated amount.**

**ANT CLASS: public Insect**

Non-Virtual functions:

**Ant(int imageID, int startX, int startY, int colony, StudentWorld\* student, Compiler\* program);**

**An Ant must have an image ID of IID\_ANT\_TYPE0 (if it’s an ant of colony 0),**

**IID\_ANT\_TYPE1 (if it’s an ant of colony 1), IID\_ANT\_TYPE2 (if it’s an ant of colony 2),**

**IID\_ANT\_TYPE3 (if it’s an ant of colony 3).**

**The Ant’s graphical depth must be 1 (this is eventually passed into GraphObject’s depth**

**field during construction, and indicates that the ant graphic should be in the foreground,**

**covering other objects with a greater depth, like an anthill or food).**

**The Ant must always start at the same location as the anthill that produced it.**

**The Ant must know its colony number, which is the same as the anthill that produced it.**

**The Ant must be provided a pointer to its Compiler object so it can get the instructions**

**that govern its behavior.**

**The Ant, in its initial state:**

**Has 1,500 hit points.**

**Faces in a random direction.**

**Is not in a stunned state.**

**Holds 0 food.**

**Was not previously bitten.**

**Was not previously blocked from moving.**

**Has a last random number value of 0.**

**Has a starting instruction counter value of 0, indicating that the first command a**

**new ant must execute during its first tick is the first command in its Compiler**

**object’s vector of commands**

**bool Interpret();**

**// This function takes the commands from the compiler, interprets it and sends it to the ants to let them know what to do:**

**i. moveForward:**

**1. If an ant moves successfully, then it needs to:**

**a. Remember that it was not blocked from moving.**

**b. Remember that it was not bit while on its current square**

**(since it’s now on a new square, it hasn’t been bitten there**

**yet, by definition).**

**2. If an ant tries to move but is blocked by a pebble, then it needs to**

**remember that it was blocked from movement.**

**ii. eatFood: An ant will try to eat 100 units of food at a time, assuming it is**

**holding food (an ant can only eat food that it previously picked up). If it is**

**holding less than 100 units of food, it will eat as much as it has left.**

**iii. dropFood: An ant will drop all of its food at once onto its current square. If**

**a square already holds a food object, the ant should simply increase the**

**units of that food in that object rather than creating another food object in**

**the same square. (Better: The ant asks the StudentWorld to figure this**

**out, and StudentWorld either creates a new food object or adds units to**

**an existing food object)**

**iv. bite: An ant will bite an enemy on the current square, if an enemy is**

**present. If there is more than one enemy on the square, then the ant will**

**choose a random enemy to bite. An ant considers all grasshoppers as**

**well as ants from all other colonies its enemy. An ant bite does 15 points**

**of damage to the enemy.**

**v. pickupFood: An ant will try to pick up 400 units of food at a time from its**

**current location. Ants may only hold a total of 1800 units of food. So if the**

**ant tries to pick up more than this, it will be limited to picking up an**

**amount of food that causes it to hold no more than 1800 total units. If**

**less than 400 units of food are left on the square, then the ant will pick up**

**the remaining food on the square, up to its 1800 total unit capacity.**

**vi. emitPheromone: Increases the pheromone scent of the current square of**

**the grid by 256 points, up to a maximum of 768. If the pheromone**

**strength already in the square were 700, and the ant emitted a new**

**pheromone, the square’s pheromone strength would jump from 700 to**

**768. Note: Each ant colony has its own pheromones, so there may be up**

**to 4 different pheromone objects (each with its own strength) in each**

**square of the field. If a square already holds a pheromone object for a**

**given colony of ants, the ant should simply increase the units of that**

**pheromone object rather than create another pheromone object in the**

**same square. (Better: The ant asks the StudentWorld to figure this out,**

**and StudentWorld either creates a new pheromone object or adds units to**

**an existing pheromone object)**

**vii. faceRandomDirection: Causes the ant to face a random direction (which**

**could be the same as the current direction faced by the ant).**

**viii. rotateClockwise: Causes the ant to turn 90 degrees clockwise.**

**ix. rotateCounterClockwise: Causes the ant to turn 90 degrees**

**counterclockwise.**

**x. generateRandomNumber: Generates a random number between 0 and**

**N-1, where the value of N is specified in operand1 of the command. If**

**operand1 is 0, then set the random number to 0.**

**xi. goto\_command: The command must set the instruction counter to the**

**position specified in operand1.**

**xii. if\_command: The if command must check the specified condition**

**(specified in operand1)**

**bool conditionTriggered(int cmd);**

**// This function lets the ants know what to do in different situations:**

**1. last\_random\_number\_was\_zero: Checks to see if the last random**

**number generated was equal to zero. If so, sets the instruction**

**counter to the location specified in operand2.**

**2. I\_am\_carrying\_food: If the ant is currently carrying any food, sets**

**the instruction counter to the location specified in operand2.**

**3. I\_am\_hungry: If the ant currently has 25 or fewer hit points (which**

**means it’s hungry), then set the instruction counter to the location**

**specified in operand2.**

**4. I\_am\_standing\_with\_an\_enemy: If the ant is currently on the**

**same square as either an ant from a different colony or a**

**grasshopper, then set the instruction counter to the location**

**specified in operand2.**

**5. I\_am\_standing\_on\_food: If the ant is currently on the same square**

**as a food object with at least 1 unit of food left, then set the**

**instruction counter to the location specified in operand2.**

**6. I\_am\_standing\_on\_my\_anthill: If the ant is currently on the same**

**square as its own anthill (and the anthill is still alive), then set the**

**instruction counter to the location specified in operand2. Note:**

**This must not trigger if an ant is standing on a competitor ant’s**

**anthill.**

**7. I\_smell\_pheromone\_in\_front\_of\_me: If there is a pheromone with**

**a strength of at least 1 in the square in front of where the ant is**

**facing (not the square the ant is on, but the square in front of the**

**ant), then set the instruction counter to the location specified in**

**operand2.**

**8. I\_smell\_danger\_in\_front\_of\_me: If there is danger (e.g., an enemy**

**ant, poison, water, or a grasshopper of any type) in the square in**

**front of where the ant is facing (not the square the ant is on, but**

**the square in front of the ant), then set the instruction counter to**

**the location specified in operand2.**

**43**

**9. I\_was\_bit: If the ant was bitten by another insect while on its**

**current square (since it last moved onto this square), then set the**

**instruction counter to the location specified in operand2. (Note:**

**The moment the ant moves successfully to a new square where it**

**has not yet been bitten, this state is cleared and it would no longer**

**considered itself bitten)**

**10. I\_was\_blocked\_from\_moving: If, during its last attempt to move,**

**the ant was blocked from moving forward by a pebble, then set**

**the instruction counter to the location specified in operand2. (Note:**

**The moment the ant moves successfully to a new square, this**

**state is cleared and it would not longer be considered blocked)**

Virtual functions:

**virtual void doSomething();**

**// This function gets the order from the interpret function, and it will make the ant do something that is appropriate to the given order.**

**virtual void getBitten(int amt);**

**// This function makes the ant lose hit points. It also give it a chance to retaliate immediately by biting another insect which does not belong to its own colony.**

**virtual bool isEnemy(int colony) const;**

**// This function lets the ant know if there exists enemies in the square that it is standing. If there exists an enemy, it will return true, else it will return false.**

**virtual bool moveForwardIfPossible();**

**// This function will move the insect forward if there is nothing blocking it. Then it will set the sleeping ticks back to 2.**

**GRASSHOPPER CLASS: public Insect**

Non-Virtual functions:

**Grasshopper(int imageID, int startX, int startY, StudentWorld \*student, int hitPoints);**

**// This class is a derived class from the insect class and it is a superclass for baby grasshopper and adult grasshopper.**

**BABYGRASSHOPPER CLASS: public Grasshopper**

Non-Virtual functions:

**BabyGrasshopper(int startX, int startY, StudentWorld\* student);**

**The BabyGrasshopper object must have an image ID of IID\_BABY\_GRASSHOPPER.**

**The BabyGrasshopper’s graphical depth must be 1.**

**The BabyGrasshopper object must always start at a starting location as specified by the**

**field's data file.**

**The BabyGrasshopper must start out facing a random direction.**

**The BabyGrasshopper must pick a random distance to walk in this random direction.**

**The distance must be between [2,10], inclusive.**

**The BabyGrasshopper must start out with 500 hit points.**

**The BabyGrasshopper must start out in a non-sleeping/stunned state, so it gets a**

**chance to move during the first tick of the simulation**

Virtual functions:

**virtual void doSomething();**

**The baby grasshopper loses one hit point (as it gets hungrier).**

**The baby grasshopper must check to see if its hit points have reached zero. If so, it**

**must:**

**a. Add 100 units of food to the simulation world in its current x,y location (the baby**

**grasshopper can ask the StudentWorld object to do this on its behalf)**

**b. Set its state to dead.**

**c. Immediately return.**

**Otherwise, the baby grasshopper must check to see if it is currently sleeping/stunned**

**(e.g., because it sleeps 2 out of 3 ticks, or because it might have been additionally**

**stunned by stepping onto a Pool of Water). If the baby grasshopper is sleeping/stunned,**

**then the baby grasshopper must:**

**a. Decrement the count of ticks left for it to be sleeping.**

**b. Immediately return.**

**Otherwise, the baby grasshopper is going to do something this round.**

**The baby grasshopper checks its hit points. If its hit points are greater than or equal to**

**1,600, then it will moult and turn into an adult grasshopper. It must:**

**a. Create and add a new adult grasshopper object to the simulation in the same**

**square as the baby.**

**b. Set the baby’s status status to dead, resulting in a pile of 100 units of food being**

**dropped in the current square.**

**c. Return immediately**

**The baby grasshopper then attempts to eat any food on the current square. It will**

**attempt to eat 200 units of food at a time (or however much food is available, if the**

**amount of food on the current square is less than 200 units). If the baby grasshopper**

**does find food, the eaten number of units must be deducted from the food object on the**

**square (possibly causing it to disappear) and this energy is given to the baby**

**grasshopper as hit points.**

**If the baby grasshopper did eat, then there is a 50% chance it will want to immediately**

**rest, in which case it proceeds to step 12.**

**Otherwise, check if the baby grasshopper has finished walking the desired distance in**

**the current direction. If so:**

**The baby grasshopper must pick a new random direction (which could be the**

**same as the current direction)**

**The baby grasshopper must pick a random distance to walk in this new random**

**direction. The distance must be between [2,10], inclusive.**

**The baby grasshopper attempts to move one square in its currently-facing direction.**

**If the baby grasshopper was blocked from moving (e.g., by a pebble) then it must:**

**a. Set the desired distance to continue walking in the current direction to zero**

**(which will cause it to pick a new direction to move during the next tick)**

**b. Proceed to step 12.**

**Otherwise, the baby grasshopper decreases its desired distance to walk in the current**

**direction by one. (Note: If the desired distance reaches zero, this will cause it to pick a**

**new direction to move during the next awake tick)**

**Set the number of ticks to sleep to 2, so that the baby grasshopper sleeps two ticks**

**before doing something meaningful again.**

**ADULTGRASSHOPPER CLASS: public Grasshopper**

Non-Virtual functions:

**AdultGrasshopper(int startX, int startY, StudentWorld\* student);**

**The AdultGrasshopper object must have an image ID of IID\_ADULT\_GRASSHOPPER.**

**The AdultGrasshopper’s graphical depth must be 1.**

**The AdultGrasshopper object’s starting location must be passed into its constructor.**

**The AdultGrasshopper must start out facing a random direction.**

**The AdultGrasshopper must pick a random distance to walk in this random direction.**

**The distance must be between [2,10], inclusive.**

**The AdultGrasshopper must start out with 1,600 hit points.**

**The AdultGrasshopper must start out in a non-sleeping/stunned state, so it gets a**

**chance to move during the first tick of the simulation.**

Virtual functions:

**virtual void getBitten(int amt);**

**// This function will cause damage to the adult grasshopper, and it will also give it a chance to retaliate and bite another insect in the same square.**

**virtual void doSomething();**

**1. The adult grasshopper loses one hit point (as it gets hungrier).**

**2. The adult grasshopper must check to see if its hit points have reached zero. If so, it**

**must:**

**a. Add 100 units of food to the simulation world in its current x,y location (the adult**

**grasshopper can ask the StudentWorld object to do this on its behalf)**

**b. Set its state to dead.**

**c. Immediately return.**

**3. Otherwise, the adult grasshopper must check to see if it is currently sleeping (e.g.,**

**because it sleeps 2 out of 3 ticks). If the adult grasshopper is sleeping, then the adult**

**grasshopper must:**

**a. Decrement the count of ticks left for it to be sleeping.**

**b. Immediately return.**

**4. Otherwise, the adult grasshopper is going to do something this round.**

**5. There is a one in three chance that the adult grasshopper will try to bite another insect**

**on the same square during the current tick. If the adult grasshopper does decide to bite**

**an enemy during the tick (⅓ chance), and there are one or more enemies (all other baby**

**and adult grasshoppers and all ants) on the current square that can be bit, then:**

**a. The adult grasshopper will randomly choose one of the enemies and bite them,**

**doing 50 hit points of damage.**

**b. Proceed to step 13.**

**6. Otherwise, there is a one in ten chance that the adult grasshopper will decide to jump to**

**another square. If it decides to jump AND there is an open square for it to jump to (one**

**without a pebble), then it will:**

**a. Select a random open square within a circular radius of 10 squares of itself, and**

**moveTo() to that square (Hint: use cos() and sin() from <cmath> for this)**

**b. Proceed to step 13.**

**7. The adult grasshopper then attempts to eat any food on the current square. It will**

**attempt to eat 200 units of food at a time (or however much food is available, if the**

**amount of food on the current square is less than 200 units). If the adult grasshopper**

**does find food, the eaten number of units must be deducted from the food object on the**

**current square (possibly causing it to disappear) and this energy is given to the adult**

**grasshopper as hit points.**

**8. If the adult grasshopper did eat, then there is a 50% chance it will want to immediately**

**rest, in which case it goes to step 13.**

**9. Otherwise, see if the adult grasshopper has finished walking its desired distance in the**

**current direction. If so:**

**a. The adult grasshopper must pick a new random direction (which could be the**

**same as the current direction)**

**b. The adult grasshopper must pick a random distance to walk in this new random**

**direction. The distance must be between [2,10], inclusive.**

**10. The adult grasshopper attempts to move one square in its currently facing direction.**

**11. If the adult grasshopper was blocked from moving (e.g., by a pebble) then it must:**

**a. Set the desired distance to continue walking in the current direction to zero**

**(which will cause it to pick a new direction to move during the next awake tick)**

**b. Proceed to step 13.**

**12. Otherwise, the adult grasshopper decreases its desired distance to walk in the current**

**direction by one. (Note: If the desired distance reaches zero, this will cause it to pick a**

**new direction to move during the next tick)**

**13. Set the number of ticks to sleep to 2, so that the adult grasshopper sleeps two ticks**

**before doing something meaningful again.**

1. I was able to finish all my classes, and I think they work probably. However, I am concerned about the actions of my ants and anthills a little. The anthills do not always produce new ants. As a result, I will only get 5 ants most of the time. It seems like the ants never bring any food back to their anthills. However, the problem may be related to the USCAnt.bug file. The program is also not able to tell who wins if there is a tie.
2. When adult grasshoppers attempt to jump from one square to another square in a radius of 10 squares around them, they may try to jump on the top of a pebble. In order to prevent that from happening, I made sure that if the Adult grasshopper’s destination is a pebble, it will set the distance equal to 0. As a result, the adult grasshopper will always stay inside the field, and it will never jump outside.
3. For my StudentWorld class, I called cout in different positions to make sure that my program would go through each and every step. I also put breakpoints inside my functions to make sure that my program was going through all my functions and everything was working properly.

My actor class, everything was just returning either a Boolean or it was not returning anything.

My pebble class did not need that much to do either. The doSomething() does not return anything, and there is no other function that we should worry about.

Inside my poison and water class, I put cout to make sure it would go through my if else statements and everything would work properly and in order. I would also freeze my program to make sure that my insects were getting poisoned and stunned correctly and everything works well.

Inside my Adult GrassHopper and Baby GrassHopper, I put breakpoints to make sure my program was working and get rid of my programing bugs. For my ant class, I added commands to my USCAnt.bug to see how it works under different instructions and circumstances. It was also a good way to debug my code and solve my program’s problems.