1. StudentWorld:

The StudentWorld class takes care of any interactions between Actors, or between Actors and the simulation.

StudentWorld Helper Functions:

* EatFood takes a position and returns an integer. It decreases the amount of food at the location by a specified amount.
* dealWithNewAnt takes a position and colony number and creates a new ant of that colony.
* dealWithDropFood takes a position and amount and either adds it to the current food item or creates a new one.
* dealWithPickupFood takes a position and an amount and returns the amount of food picked up.
* dealWithAntAttack takes a position and a colony number and reduces the appropriate bug’s health points and if the bug is an Ant, it is bit
* dealWithPheromone takes a position and colony number and creates a new pheromone of that colony or adds to the current pheromone, up to the max value of 768
* isEnemyHere takes a position and a colonyNum and returns true if there is an enemy ant or grasshopper on the position and false if otherwise
* isFoodHere takes a position and returns true if there is a food item at the location and false if otherwise
* isAnthillHere takes a position and takes a position and a colony number and returns true if the anthill of that colony is on that position and false if otherwise
* isPheromoneHere takes a position and takes a position and a colony number and returns true if the pheromone of that colony is on that position and false if otherwise
* isDangerHere takes a position and a colony number and returns true if an ant of another colony, a grasshopper, a poison item, or a pool of water is at that location and false if otherwise
* canMoveTo takes a position and returns true if there is not a pebble there and false if otherwise
* addFood takes a position and an amount and either adds it to the current food item or creates a new one
* dealWithHazard takes a position and either stuns or poisons the ants or baby grasshoppers at the position, if it is the first time they encountered it
* addActor takes an actor pointer and a position and creates an actor at that location
* dealWithGAttack

get current position

find all enemies on that location

if there are enemies on that square

choose a random one to bit

if an AdultGrasshopper was bit, give it a chance to

biteback

else

return false

* dealWithGJump

get random location within 10 square radius

keep generating random locations until one is with the arena

if there is a pebble at the random location

try again, up to 10 times

else

jump to square

* setDisplayText updates the text at the top of the screen using string streams
* getLeader returns whichever ant colony has the most colonies

StudentWorld:

* calls each actor’s doSomething
* if actor is dead, deallocate it, and either add to the current food item on the square or create a new food item

Actor:

* doSomething is a pure virtual function to be overridden by the derived classes
* getPoisoned is overridden by a derived class
* getStunned is overridden by a derived class
* getWorld returns a pointer to the StudentWorld object
* There are getters and setters for each Actor that is used in StudentWorld(Pebble, Water, Poison, Food, Pheromone, Anthill, Ant, BabyGrasshopper, and AdultGrasshopper)

Pebble:

* Pebble inherits Actor
* Actor’s isPebble is set to true
* Pebble’s doSomething does nothing

Water:

* Water inherits Actor
* Actor’s isWater is set to true
* Water’s doSomething does nothing

Poison:

* Poison inherits Actor
* Actor’s isPoison is set to true
* Poison’s doSomething does nothing

EnergyHolder:

* EnergyHolder inherits Actor
* EnergyHolder’s doSomething does nothing
* getHP returns the hp of the item
* addHP add to the hp of the item
* reduceHP reduces the hp of the item

Food:

* Food inherits EnergyHolder
* Actor’s isFood is set to true
* Food’s doSomething checks if it is dead, and if it is it sets isDead to true and returns

Pheromone:

* Pheromone inherits EnergyHolder
* Pheromone’s isPheromone is set to true
* Pheromone’s doSomething checks if it is dead, and if it is it sets isDead to true and returns

Anthill:

* Anthill inherits EnergyHolder
* Actor’s isAnthill is set to true
* Anthill’s doSomething checks if it is dead, and if it is it sets isDead to true and returns. Otherwise it eats the food on the square and returns. If it has enough hp, it will create a new ant

Insect:

* Insect inherits EnergyHolder
* Insect’s doSomething does nothing
* rndDirection sets the Insect in a random direction, all Insects call this
* getXYInFrontOfMe passes two variables by reference and changes it to the x and y position in front of the Insect, Insect calls this to move forward
* moveForwardIfPossible updates the Insect’s location if it is not blocked, all Insects need to move forward
* getPoisoned reduces the Insect’s health by 150, all Insects but Adult Grasshopper get poisoned
* getStunned increases the Insect’s stunned time by 2, all Insects but Adult Grasshopper get stunned

Ant:

* Ant inherits Insect
* Actor’s isAnt is set to true
* Ant’s doSomething interprets the commands given by the compiler
* RotateClockwise rotates the Ant clockwise
* RotateCounterClockwise rotates the Ant counterclockwise
* ifCommands takes the condition and evaluates it

Grasshopper:

* Grasshopper inherits Insect
* Grasshopper’s doSomething takes appropriate actions depending on whether the Grasshopper is an Adult or a Baby

BabyGrasshopper:

* BabyGrasshopper inherits Grasshopper
* Actor’s isBabyG is set to true

AdultGrasshopper:

* AdultGrasshopper inherits Grasshopper
* Actor’s isAdultG is set to true

2. I implemented everything, so it should work.

3. I assumed that if there is a tie for the display, the leading Ant should be the one that got there first.

4. StudentWorld: I tested StudentWorld and its helper functions by creating different fields and setting the ants and grasshoppers to execute certain actions. These test fields usually had an area of 4 by 4 open for the insects to interact. I outputted values such as the hp of an insect or how many turns it has left.

Actor: I tested the identities by modifying the field file. I put certain items at certain locations, then I coutted the value.

Pebble: I modified the field file and the Ant and Grasshopper’s doSomething so they move in one direction. As expected, they stopped when they reached the Pebble.

Water/Poison: I modified the field file and the Ant and Grasshopper’s doSomething so they move in one direction. I coutted their hp and break values to ensure that they were performing the correct behavior.

Food: I modified the Ant and Grasshopper’s doSomething so they pickup, put down, eat, and add Food at certain locations.

Pheromone: I modified the Ant’s doSomething and created my own .bug file. I had the ants place it on certain locations. Then checked if they would recognize their own Pheromone while ignoring the others

Anthill: I added Food to the Anthill so it would produce more ants. I coutted to ensure the hp of the Anthill was correct. I also coutted the position of the ants to ensure they were correct.

Insect: I tested Insect by moving Ants and Grasshoppers in certain directions. I also checked the new random direction by spinning Ants and Grasshoppers around. I tested the getPoisoned and getStunned by coutting the appropriate values.

Ant: I created a new .bug file to test each behavior of the Ant. I moved it forward until it got stuck, I emitted Pheromones, picked up, ate, and dropped food. I also rotated it, set it in a random direction, attacked other ants, and generated random numbers;

Grasshopper: I created a unique field files to test both babyGrasshopper and adultGrasshopper

BabyGrasshopper: In the unique field file, I created food so that the babyGrasshopper would eat it and transform. I tested the transform with cout.

AdultGrasshopper: In the unique field file, I allowed the AdultGrasshopper only certain spaces to be able to jump to. I also created stationary enemies so the AdultGrasshoppper would attack them. I tested the bite back feature with cout.