**ROBERT URSUA 604 749 119**

**1. HIGH LEVEL DESCRIPTION OF PUBLIC FUNCTIONS**

class Actor

// Constructor. Sets the proper X and Y coordinates, direction, image ID, and depth by passing those parameter to the GraphObject constructor

// Assigned the StudentWorld\* pointing to the object in which this actor resides

Actor(StudentWorld\* world, int startX, int startY, Direction startDir, int imageID, int depth);

// Destructor. Doesn’t do anything

virtual ~Actor();

// Action to perform each tick. Purely virtual since every actor has a different set of actions to perform.

virtual void doSomething() = 0;

// Returns true if an actor is dead. Returns false by default for this base class so energyless actors don’t get removed after every tick

virtual bool isDead() const;

// returns true if an actor blocks movement

// Returns false by default in this base class since Only pebbles block movement

virtual bool blocksMovement() const;

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

virtual void getBitten(int amt);

// default actor isnt bitable, do nothing

// In general, causes this actor to be poisoned

// Since most actors aren’t poisonable, this base function implementation does nothing.

virtual void getPoisoned();

// Cause this actor to be be stunned.

// Since most actors aren’t stunnable, this base function implementation does nothing.

virtual void getStunned();

// returns true if Actor can be Eaten

// since only food is edible, this base function implementation returns false

virtual bool isEdible() const;

// returns true if Actor is a pheromone belonging to the indicated colony

// since only pheromone objects can return a value of true, this base function implementation returns false

virtual bool isPheromone(int colony) const;

// returns true if this actor is an enemy of an ant of the indicated colony

// returns false by default in this base class

virtual bool isEnemy(int colony) const;

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

// since only triggerable actor and insect objects can return a value of true, this base function implementation returns false

virtual bool isDangerous(int colony) const;

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

// since only anthill can return a value of true, this base function implementation returns false

virtual bool isAntHill(int colony) const;

// Default false

// Get this actor's world.

StudentWorld\* getWorld() const;

// returns true if this actor is stunnable

// this base function implementation returns false since most actors are not stunnable

virtual bool isStunnable() const;

// returns true if this actor is poisonable

// this base function implementation returns false since most actors are not poisonable

virtual bool isPoisonable() const;

// returns true if this actor turns into food when it dies

// this base function implementation returns false since most actors dont become food

virtual bool becomesFoodUponDeath() const;

// returns a random direction, all equally probable

GraphObject::Direction genRandomDir();

class Pebble

// constructor, sets indicated StudentWorld pointer and starting coordinates

// also sets direction to right and assigns the proper image ID

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

// Destructor. Doesn't do too much

virtual ~Pebble();

// Does nothing. its a pebble

virtual void doSomething();

// Returns true since pebbles block movement

virtual bool blocksMovement() const;

class EnergyHolder

// constructor. Sets proper StudentWorld pointer, starting coordinates, direction, energy, imageID, depth, colonial affiliation and Compiler pointer

EnergyHolder(StudentWorld\* sw, int startX, int startY, Direction startDir, int energy, int imageID, int depth, int colony, Compiler\* compiler);

// Destructor. Does nothing;

virtual ~EnergyHolder();

// Supposedly does something but its purely virtual since every EnergyHolder has different actions

virtual void doSomething() = 0;

// Returns true when this EnergyHolders hitpoints or energy reaches 0 (or less)

virtual bool isDead() const;

// Get this actor's amount of energy (for a Pheromone, same as strength).

int getEnergy() const;

// Adjust this actor's amount of energy upward or downward.

void updateEnergy(int amt);

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

// either adds to an existing food object or makes a new one

void addFood(int amt);

// Have this insect pick up up to 400 units of food.

// returns amount of energy successfully picked up

int pickupFood(int amt);

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

// returns amount of energy successfully eaten

int pickupAndEatFood(int amt);

// returns the amount of food carried by this energy holder

int getCarriedFood() const;

// increases or decreases the amount of food carried by this holder

void updateCarriedFood(int amt);

// Sets food carry capacity of this energy Holder

void setCarryCapacity(int cap);

// Does this actor become food when it dies?

// returns false by default since only insects become food

virtual bool becomesFoodUponDeath() const;

// returns the colony to which this EnergyHolder belongs

int getColony() const;

// returns the compiler associated with this Energy Holder

Compiler\* getCompiler() const;

class Food :

// sets StudentWorld pointer, and starting coordinates and energy

Food(StudentWorld\* sw, int startX, int startY, int energy);

// Destructor. Does nothing;

virtual ~Food();

// Does nothing. It's food

virtual void doSomething();

// Returns true since food is edible

virtual bool isEdible()

class AntHill

// constructor. Sets proper StudentWorld pointer, starting coordinates, direction(right), energy(8999), imageID, depth, colonial affiliation and Compiler pointer

AntHill(StudentWorld\* sw, int startX, int startY, int colony, Compiler\* program);

// Destructor. Does nothing;

virtual ~AntHill();

// Decreases queens hp by 1

// checks if it's dead, if so return immediately

// attempts to eat up to 10000 units of food on its tile

// attempts to create ants in its tile with 2000 hp (only if it has 1500 or more hp)

virtual void doSomething();

// true if colony parameter matches anthill colony

virtual bool isAntHill(int colony) const;

class Pheromone

// constructor

// sets indicated StudentWorld, starting coordinates and colony

// sets direction to right and sets proper imageID

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

// Destructor. Does nothing;

virtual ~Pheromone();

// does nothing

virtual void doSomething();

// returns true

virtual bool isPheromone(int colony) const;

// Increase the strength (i.e., energy) of this pheromone.

// caps total strength to 768

void increaseStrength();

class TriggerableActor

// constructor

// sets indicated StudentWorld and starting coordinates

// sets direction to right

// sets imageID to the passed imageID parameter

TriggerableActor(StudentWorld\* sw, int x, int y, int imageID);

// Destructor. Does nothing;

virtual ~TriggerableActor();

// returns true. poison and pools of water are dangerous

virtual bool isDangerous(int colony) const;

// purely virtual because we dont want to create objects of this class

virtual void doSomething() = 0;

class WaterPool

// sets indicated StudentWorld and starting coordinates

// passes WaterPool image ID to TriggerableActor constructor

WaterPool(StudentWorld\* sw, int x, int y);

// Destructor. Does nothing;

virtual ~WaterPool();

// attempts to stun all actors in its current position

virtual void doSomething();

class Poison

// sets indicated StudentWorld and starting coordinates

// passes Poison image ID to TriggerableActor constructor

Poison(StudentWorld\* sw, int x, int y);

// Destructor. Does nothing;

virtual ~Poison();

// attemps to poison all actor in its current tile

virtual void doSomething();

class Insect

// sets indicated StudentWorld, starting coordinates, energy, imageID, colony and compiler

// sets a random starting direction for every insect

Insect(StudentWorld\* world, int startX, int startY, int energy, int imageID, int colony, Compiler\* comp);

// Destructor. Does nothing;

virtual ~Insect();

// purely virtual since all insects have different actions

virtual void doSomething() = 0;

// Makes this insect's energy decrease by amt

virtual void getBitten(int amt);

// Poisons this insect

virtual void getPoisoned();

// Stuns this insect, but checks if it wasn't already previously

// standing on the same WaterPool object

virtual void getStunned();

// true if this insect's colony doesnt match caller's colony

virtual bool isEnemy(int colony) const;

// same function as isEnemy. Needed to be overloaded for proper behavious

virtual bool isDangerous(int colony) const;

// Returns true since all insects turn into food when they die

virtual bool becomesFoodUponDeath() const;

// Set x,y to the coordinates of the spot one step in front of this insect.

void getXYInFrontOfMe(int& x, int& y) const;

// Move this insect one step forward if possible, and return true;

// otherwise, return false without moving.

virtual bool moveForwardIfPossible();

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

void increaseSleepTicks(int amt);

// Returns number of turns left to sleep

int getRemainingSleepTicks() const;

// resets the recorded coordinates of the waterPool object that last stunned this insect

void resetLastStunned();

// makes the insect get hungrier

// checks if insect is dead. If it is, return false

// cehcks if insect is sleeping or stunned

// if it is, decrement sleep counter, return false

// return true if insect isnt dead or sleeping

bool doIDoSomethingThisTick();

class Ant

// Constructor: sets the indicated student world, starting coordinates, colony, compiler program, imageID and reset memory (remembering whether it was bitten or blocked in the tile) also sets the ants carry capacity to 1800

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

// Destructor. Does nothing;

virtual ~Ant();

// In order:

// decreases ant's hp by 1

// Check is the ant is stunned or sleeping

// if yes, decrease sleep count

// if no, interpet a command by calling commandInterpreter()

virtual void doSomething();

// Damages this ant and makes it remember that it got bitten

virtual void getBitten(int amt);

// returns true if the indicated colony differs from this ants colony

virtual bool isEnemy(int colony) const;

// attempts to move the ant forward and remembers if it gets blocked

virtual bool moveForwardIfPossible();

// make this ant emit a pheromone

void emitPheromone();

// the following return true since the base function returned false

// but BabyGrasshoppers are both stunnable and poisonable

virtual bool isStunnable() const;

virtual bool isPoisonable() const;

class Grasshopper

// constructor, sets indicated Student world, starting coordinates, energy and imageID

// sets random distance for grasshopper to travel

Grasshopper(StudentWorld\* sw, int startX, int startY, int energy, int imageID);

// Destructor. Does nothing;

virtual ~Grasshopper();

// Baby and adult grasshopper still have different actions so make this pure virtual

virtual void doSomething() = 0;

// Makes grasshopper attempt to eat 200 units of food

// gives it 50% chance of sleeping if it does eat

// returns true if grasshopper wants to sleep

void attemptToEatHere();

// Makes grasshopper check its distance goal in the direction it wants to travel

// is it still has to walk, make it walk one tile

// otherwise, set a new movement goal

void makeGrassHopperMove();

// travel distance related functions

// sets the distance that the insect has to travel to random number from 2 to 10

void genNewDistanceGoal();

// modifies distance goal by adding num

void adjustDistanceGoal(int num);

// returns distance goal

int getDistanceGoal() const;

class BabyGrasshopper

// constructor: sets indicated StudentWorld and starting coordinates

// sets appropraite imageID and sets energy to 500

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

// Destructor. Does nothing;

virtual ~BabyGrasshopper();

// In order:

// Decreases hp by 1

// Checks if dead and returns immediately if so (StudentWorld will convert it to food)

// Checks if stunned/sleeping.

// Decrements sleep counter if so

// Checks if hp >= 1600

// Creates an Adult Grasshopper in its tile

// kills itself then returns immediately

// Attempts to eat up to 200 units of food in its square (

// Increases sleep duration with a probability of 0.5 if it eats and returns immediately

// Check if finished walking in desired direction

// if yes, pick new direction and distance

// Attempt to move one tile in current distance and direction

// if blocked from moving, set desired distance to zero and increase sleep duration, then return immediately

// if successful, decrement distance counter

// Increase Sleep duration

virtual void doSomething();

// the following return true since the base function returned false

// but BabyGrasshoppers are both stunnable and poisonable

virtual bool isStunnable() const;

virtual bool isPoisonable() const;

class AdultGrasshopper

// constructor: sets indicated StudentWorld and starting coordinates

// sets appropraite imageID and sets energy to 1600

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

// Destructor. Does nothing;

virtual ~AdultGrasshopper();

// In order:

// Decreases hp by 1

// Checks if dead and returns immediately if so (StudentWorld will convert it to food)

// Checks if stunned/sleeping.

// Decrements sleep counter if

// With a 1/3 chance, attempt to bite a random enemy in this tile

// With a 1/10 chance, attempt to skip to an open tile within a radius of 10 units

// Attempts to eat up to 200 units of food in its square

// Increases sleep duration with a probability of 0.5 if it eats and returns immediately

// Check if finished walking in desired direction

// if yes, pick new direction and distance

// Attempt to move one tile in current distance and direction

// if blocked from moving, set desired distance to zero and increase sleep duration, then return immediately

// if successful, decrement distance counter

// Increase Sleep duration

virtual void doSomething();

// Reduces energy by amt then attempts to bite back (1/2 chance)

virtual void getBitten(int amt);

class StudentWorld

// constructor, sets game tick count to 2000,

StudentWorld(std::string assetDir);

// Destructor. calls cleanUp() to delete all Actors

virtual ~StudentWorld();

// loads field files and bug files

// and adds actors to the filed

virtual int init();

// Updates tick count, makes all actors do something,

// removes dead actors, updates the game text,

// and decides if an ant has won or not when the game end

virtual int move();

// deletes all actors

virtual void cleanUp();

// calls doSomething() for all actors

// moves actors in the unordered\_map if they moved in the field

void makeActorsDoSomething();

// Removes dead actors, readjusting the unordered\_map & erasing empty lists

void removeDeadActors();

// Loads all actors into the unordered\_map as specified in the field file

int loadFieldFile();

// Loads bugs files and populates the m\_compilers vector

int loadBugsFiles();

// Updates the score and tick count text at the top of the screen

void updateGameText();

// Can an insect move to x,y?

bool canMoveTo(int x, int y) const;

// Add an actor to the world

void addActor(Actor\* a);

// If an item that can be picked up to be eaten is at x,y, return a

// pointer to it; otherwise, return a null pointer. (Edible items are

// only ever going be food.)

Actor\* getEdibleAt(int x, int y) const;

// If a pheromone of the indicated colony is at x,y, return a pointer

// to it; otherwise, return a null pointer.

Actor\* getPheromoneAt(int x, int y, int colony) const;

// Is an enemy of an ant of the indicated colony at x,y?

bool isEnemyAt(int x, int y, int colony) const;

// Is something dangerous to an ant of the indicated colony at x,y?

bool isDangerAt(int x, int y, int colony) const;

// Is the anthill of the indicated colony at x,y?

bool isAntHillAt(int x, int y, int colony) const;

// Bite an enemy of an ant of the indicated colony at me's location

// (other than me; insects don't bite themselves). Return true if an

// enemy was bitten.

bool biteEnemyAt(Actor\* me, int colony, int biteDamage);

// Poison all poisonable actors at x,y.

bool poisonAllPoisonableAt(int x, int y);

// Stun all stunnable actors at x,y.

bool stunAllStunnableAt(int x, int y);

// Record another ant birth for the indicated colony.

void increaseScore(int colony);

**2. MISSING FUNCTIONALITY AND KNOWN BUGS**

**3. OTHER DESIGN DECISIONS**

**StudentWorld Related**

I used an unordered\_map because I thought that it allowed fast access to the Actors in the field and doesn’t need to waste time reordering elements as a map would.

I made the StudentWorld class deal with turning dead Actors into Food (if indicated) as it seemed easier

**Poison Related**

It wasn’t specified that Poison can’t re-poison an insect that didn’t yet move away from the same tile, but I made the poison only apply when an insect moves into it from another tile (so insects have to move away to be repoisoned by the same tile. WaterPool was explicitly disallowed from restunning targets that didn’t move away from it

**Adult GrassHopper Related**

Not sure if this is a bug but the “jumping” motion of the adult grasshopper doesn’t look instant like it does in the sample Bugs program. The jumping looks smoother in my simulation.

**4. TESTING METHODS**

**Actor Class**

The implementation of the constructor was tested indirectly by testing the derived classes. If this Actor constructor works, then objects must show properly in the field in their correct positions and orientations. I made custom field.txt files to test this. I also checked whether functions return the proper values using cassert() (mostly used for the function that determine certain traits).

**Pebble Class**

Tested to see if BlocksMovement() returns true if the pointer type pointing to the Pebble object is either an Actor pointer or a Pebble pointer. Used the grasshopper class to further see if it works (grasshopper shouldn’t walk over pebbles if all work well)

**EnergyHolder Class**

Tested to see if constructor works properly when creating Ants, Grasshoppers, and other EnergyHolder derived objects. If it does, then the objects must start with the correct HP, colony, carry capacity and compiler objects, so I printed those out manually to check. Again I modified a field.txt file to create a few isolated objects that I would be testing. I also tested the HP related functions to see whether the energy content of this object gets updated correctly using the interface that I created, and if it sets the status to Dead when HP drops to 0

**Food Class**

Tested to see it Food objects start with the correct starting traits and if isEdible() returns true when using either a Food\* or an Actor\*. Food at the beginning of the game should have 6000 HP and new Food objects created later from dead insects must have 100HP.

**AntHill Class**

Tested to see if AntHill objects start with correct starting traits (especially HP, colony, Compiler object) and if isAntHill() returns the proper values when different colonies of ants visit their own or another ant colony’s anthill. Tested doSomething() by monitoring AntHill hp as time progresses and seeing if it updates death status properly, eats food and adds hp properly, and creates ants when it reaches the proper threshold

**Pheromone**

Checked to see if Pheromone start with correct traits. Monitored hp to see if it decreases as indicated in the spec and if it’s capped at 768.

**Triggerable Actor**

Checked to see if TriggerableActor objects start with correct traits. Checked to see if these objects indicate that they were dangerous using isDangerous() when using either Actor\*, TriggerableActor\*, WaterPool\* or Poison\*

**Water Pool**

Checked to see if WaterPool objects start with correct traits. Checked if it called the stunAllStunnable() function in student world properly during every tick.

**Poison**

Checked to see if Poison objects start with correct traits. Checked if it called the poisonAllPoisonable() function in student world properly during every tick.

**Insect**

Check to see if Insect objects start with correct traits (colony, Compiler, random starting direction, imageID etc). Also monitored HP and sleep behavior when using getBItten(), getPoisoned() and incrementSleepTicks() to see if they adjust HP/ sleepticks using the correct values. Tested to see if getStunned() prevented an insect from getting stunned continuously by a WaterPool object by trapping a poor grasshopper inside a pebble cage next to a WaterPool. If the restunning prevention was successful then the grasshopper musn’t get stuck on top of the water (but still sleep longer when it gets stunned by it). Tested getXYInFrontOfMe() and MoveForwardIfPossible() by modifying the field.txt file and observing Grasshopper objects’ movements when in various positions and directions, and when interacting with other objects in the field (i.e. it should be able to walk over most objects except pebbles).

**Ant**

Checked to see if Ants start with the correct traits (colony/ compiler, imageID, etc.). Tested to see if the command interpreter works by modifying both the bugs and field files. For example to test moveForward, pickupFood and eatFood, create a corridor with one ant in the and a walkway filled with food, make the ant repeatedly move forward and eat food as it go, and monitor the food HPs and the ant HP. I Also created similar setups for if\_commands to see it they trigger under the correct circumstances.

**Grasshopper**

Checked starting traits and monitored movement behavior to see if the grasshopper aims to finish walking its desired distance in the correct direction and if it adjusts accordingly when blocked. When testing attemptToEatHere(), I observed both the Food’s HP and the grassHopper’s HP to ensure proper behavior

**Baby Grasshopper**

Tested the same traits in GrassHopper class and also monitored moulting behavior. For example, creating a Grasshopper with 1601 hp must make it turn into an adult immediately I the first tick.

**Adult GrassHopper**

Ran GrassHopper tests and checked biting behavior and jumping. For biting behavior, I trapped an ant and an adult grasshopper in a pebble cage and tested to see if the adult grasshopper bit back occasionally when it was bit (if it was still alive). Also monitored the HP to see if the behavior was correct. For jumping, tested to see if adult grasshopper can jump over pebbles but not go out of bounds. Also recorded distance jumped to see if it was consistent with the radius constraint

**StudentWorld**

First, tested to see if init() has proper behavior when loading legitimate and illegitimate bugs or field files. If it loaded legitimate files, it should place the correct indicated actors with their correct starting traits in the field. For move(), I observed the tick count and game text update functionalities and compared them with the specs. makeActorsDoSomething() was tested by simply populating the field with Actors and observing if their doSomething was called. removeDeadActors was tested by repeatedly populating the field with already dead actors and observing if the function removed them promptly. canMoveTo() was tested with the help of insects (which called this function in its moveForwardIfPossible() function). addActor was tested with the help of ants (which emit pheromones) and the EnergyHolder’s addFood() function

The other functions were basically trait detection functions (some performed action by stunning actors) so to test them I trapped various objects in a pebble cage and explicitly called these functions to test if the trapped object gave the correct characteristics. For example, if an ant is trapped in cell (10, 54), I observed the return value of these trait detection functions when their x and y parameters where set to 10 and 54.