Interplanetary Travel in a Closed Ecosystem

New Mexico Supercomputing Challenge
Final Report
March 30, 2009

Team 15
Aspen Elementary, Los Alamos, NM

Team Members
Colin Redman
Matthew Ticknor
Michael Englert-Erickson
Dhaivat Pandya

Teacher
Zeynep Unal

Project Mentors
Elizabeth Cooper
Jim Redman

Executive Summary

Our team chose to do a model of humans on a journey between planets in our solar system. We concentrated on the environment of the spaceship to make sure the air was good for the plants and humans and enough food was available for the humans. We tried to solve this problem in two ways; an agent based simulation and a mathematical model. We considered several agents for our agent model, but in the end we used two different agents, the humans and some different kinds of plants - strawberries, soybeans, potatoes, tomatoes, and lettuce. We had to do a lot of research for the amount of oxygen and carbon dioxide gases humans and plants need to breathe and produce, and the food needed by humans and produced by plants.

For this agent model we decided to use Madkit, which is an agent-based Java development library. We also started a mathematical model in Java but this was never completed. We ran the simulations in our agent model hundreds of times with different starting parameters. These parameters were the number of humans (1-5), the number of plants (from hundreds to thousands, stepping by 200 plants each run), and the size of the spaceship (by 20 meters length each time). The size of the spaceship determined how much oxygen and carbon dioxide was available to the plants and humans at the start of the simulation. We ran the simulation for up to 500 days. If the humans lived at 500 days then the parameter set was considered successful.

Our results indicated that the size of the spaceship had the biggest influence on the results. There were never enough plants to make enough oxygen for the humans unless there was plenty of oxygen available to start. We always provided 60 days of food for the humans at the start of the simulation, but the plants always generated enough food after they started producing food.

Problem Statement

Our problem was to set up a spacecraft that contains a greenhouse with plants that would provide enough oxygen and food to support the humans. Others have looked into deploying a greenhouse on Mars through a completion by NASA [1], but this is different from our project because this is not an ecosystem. Our ecosystem requires the humans to support the plants with carbon-dioxide as well. We wanted this to be a completely stable closed environment, being essentially a biodome simulation. This means stable supply of oxygen and food for the humans and enough carbon dioxide for the plants. We have also found other similar projects on the requirements for interplanetary travel [2].

Method

First we set up an agent matrix to help us organize our interactions and aid in the implementation. Then, we talked about which agents could be included in the project and which would be the most important to simulate in our project.

Agent	People	Plant	Cricket	Plant bug	People Bug	Rat
People	competing for resources	Plants produce food O2 use co2 water use water and wast use minerals	produce food		kill	
Plant	people use minerals Use food o2 water generate co2 water and wast	competing for resources	eat	kill		eat
Cricket	eat them	food	eat food			eat
Plant Bug	kill	host				
People Bug	kill			spread		
Rat	catch kill	food	food			

When we analyzed the matrix we decided that the humans, the oxygen/carbon-dioxide

balance, and the plants would be the most important components of our simulation. We needed to add monitors to keep track of our resources such as food, oxygen and carbon dioxide. We decided that we could include other agents later after we got a stable system running with the humans and plants. Crickets were considered because they are a good source of protein. It would be funny if the spaceship arrived at its destination with only crickets as the only living creatures.

When we decided to do this project we knew we would have to supply food for the humans. We came up with the idea of a hydroponic system[3]. A hydroponic system word like this.

1.The plants are planted in a hole with their roots dangling in mineral healthy water.
2.The mineral healthy water is pumped up by a pump to the top of the system and then runs down passing through the plants roots for them to soak up.
3.When the water is cycled back to the pump it is given an amount of minerals to take bake up and run back down.
4.This cycle repeats itself over and over
5.When the plants are ready to be picked the pump is shut off momentarily.

We are using some plants like potatoes, tomatoes, soybeans lettuce, and grain. We are using some of these plants because most of them grow quickly and can be grown in a greenhouse. For our simulation hydroponics and the support for this is assumed.

To help us simulate the cycle of plants and the other factors like respiration, light, and the amount cycled out we are using photosynthesis, and the photosynthesis equation[9]. We are also using a general rule of thumb. Photosynthesis is the process in which plants take in Co2, light, and minerals to generate sugar (or biomass) along with O2. The photosynthesis equation (CO_2 + 2 H_2A + photons → $(CH_2O)_n$ + H_2O + 2A) this equation is the general equation carbon dioxide + water + light energy → carbohydrate + oxygen + water.

Photosynthesis Cycle

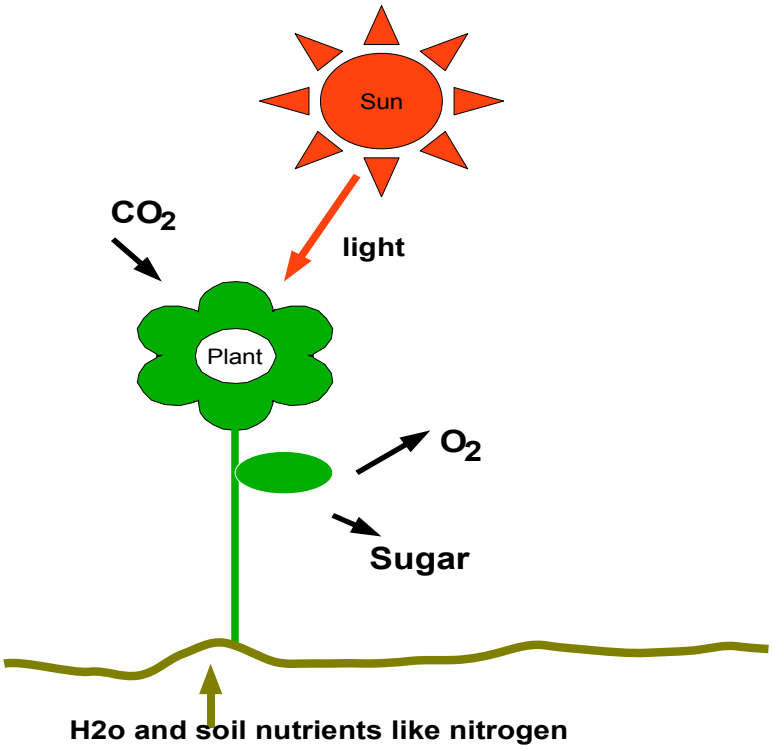


Illustration 1: Photosynthesis - Converting Carbon Dioxide to Food and Oxygen

Co2 and O2 Relationships

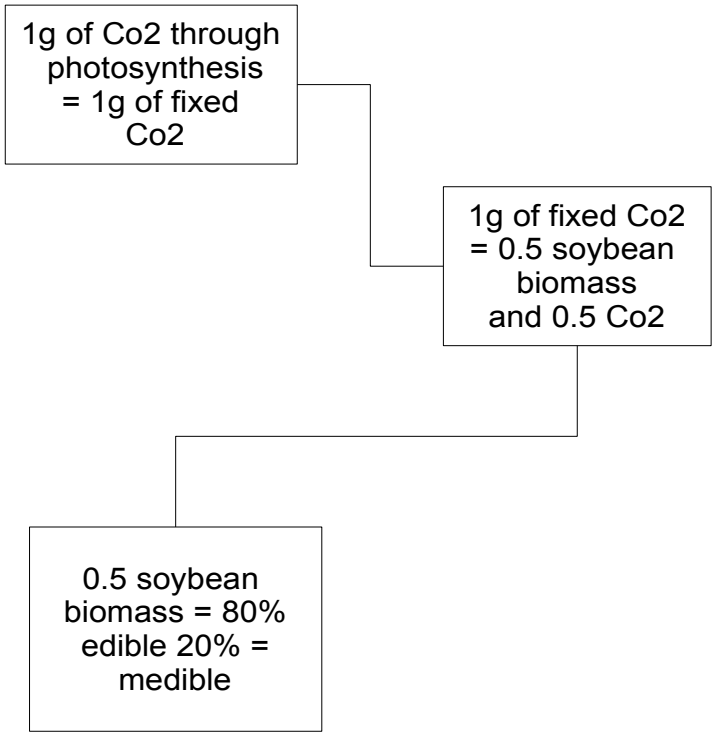


Illustration 2: Conversion of Carbon Dioxide to Food

The general rule of thumb is a CO2 and O2 relationship it is that when
1.1g of CO2 goes through photosynthesis it comes out as 1g of fixed CO2.
2.1g of fixed CO2 equals 0.5 of soybean biomass and 0.5 of CO2 that is recycled
3.0.5 of soybean biomass is 80% edible and 20% percent inedible

This is using soybeans in this example but it works for most plants.

In our simulation we assumed that there was always sufficient water, heat and light for the plants, and water and heating for the humans. We found from our research that humans needed 1.8-2.4 grams O2 per minute [7], and we converted this to 2880 grams per day. Humans also exhale 852.48 grams of Carbon Dioxide[8]. We found that small food plants used about 1 gram of CO2 per day. An interesting rule of thumb we found that 1 gram of CO2 converts to about 10 grams of plant material (including leaves,stems, roots and

fruits/vegetables when producing). Most of this weight is water. The humans need 1650 calories of food per day.

MadKit

We decided as part of our project this year to use the Madkit Java agent library to create our software [5]. MadKit (http://www.madkit.org) was developed mainly by a team from LIRMM (http://www.lirmm.fr/). MadKit is is a free software based on the GPL/LGPL license.

In Eclipse we used the MadKit libraries and create a class for each interactive agent. To make the model reactive we needed to use “Referenceable Agents” [6] so we implemented ReferenceableAgent and subclassed AbstractAgent for each of our Agents. We set up messaging between the agents which makes the model distributable between computers (even over the internet) and provides the way we get agent interactions.

In our MadKit model each agent has three important methods to write. The first one is the “activate” method which is called when the launcher creates the agents. This basically initializes the agent and sets up their group and role.

The second important method in an agent is the method that is called by the activation method of the scheduler. In our agents this is the “behavior” method. The behavior method makes the agent function, such as breathe and eat, etc.

The other important part of the agent is the part that receives the messages. When any other agent sends the agent a message it is received in the “receiveMessage” method. The messages are always StringMessage type objects, so they contain a string to parse. For example if the human agent wants to know how much food is left in the food storage, it will message the food monitor and ask for the amount of food left, and the food monitor will send the reply back to the human.

Our model has an ecosystem launcher which launch all the agents including an ecosystem

scheduler, a gas monitor, a food monitor, a population monitor, a greenhouse monitor, the humans, and the plants. The ecosystem scheduler keeps track of the days, and runs the humans and the plants. The gas monitor stores the gases in the air for both the plants and the humans. The food monitor stores the amount of food given by the plants, and gives it to the humans. The population and greenhouse monitor do practically the same thing, the greenhouse monitor stores the amount of plants that are still active, and the population monitor keeps track of the amount of the humans left alive, and tells the scheduler to quit if they are all dead. The plants reduce the carbon-dioxide from the environment, and they will become more productive if they have enough carbon-dioxide, and will even produce more food and oxygen if there is a lot of carbon-dioxide. The humans need oxygen and food in order to survive, and they produce carbon-dioxide when they breathe. The amounts of carbon dioxide used by plants and generated by humans was found with our research (see references). Also the amount of oxygen and food needed by humans was determined from research.

We start our simulation by reading a file of standard parameters. An example of a parameter file is:

```
#Simulation Parameters
#Sat Mar 28 09:57:40 MDT 2009
experimentName=test2
humans=5
plants=6000
travelDays=500
maxSpaceshipLength=1000
spaceshipLength=225
plantIncrease=200
minPlantsPerCubicMeter=1
timeDelay=10
spaceshipSizeIncrease=20
startingPlants=1000
maxPlantsPerCubicMeter=5
```

So we can run many simulations with different parameters we have the program change the parameters for the next run. The run number is increased each time we start the simulation over. The data that is generated for each run is saved in a separate data file so that each set of data can be analyzed later if needed or graphed. The data file contains the status of the humans, plants, current O2, current CO2 and food each day, day by day.

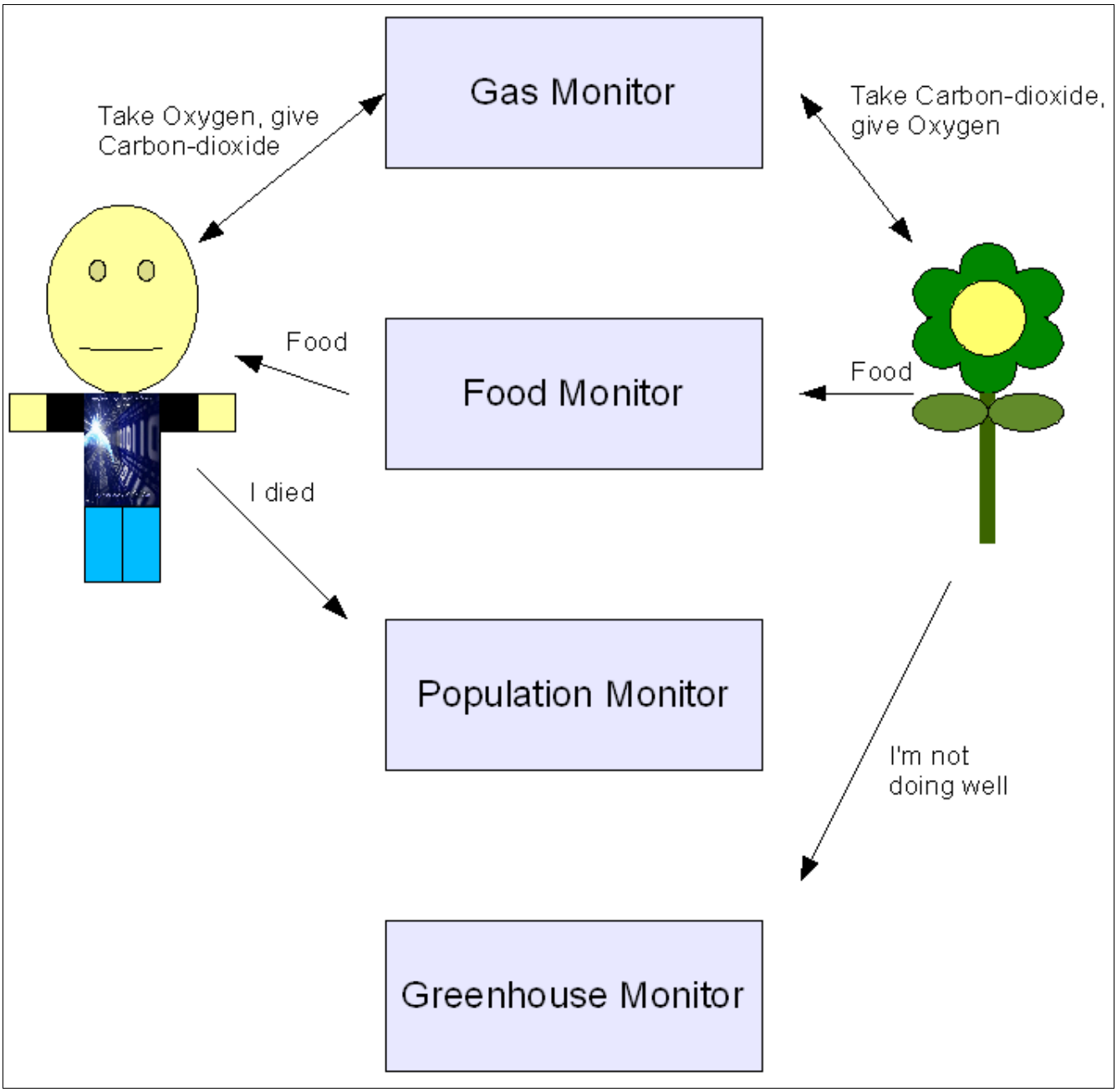
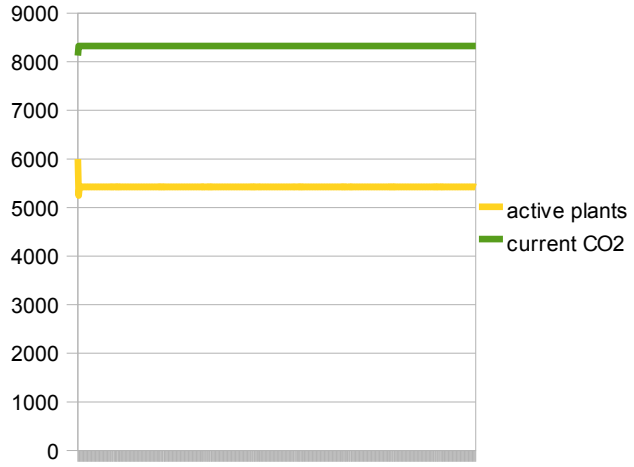
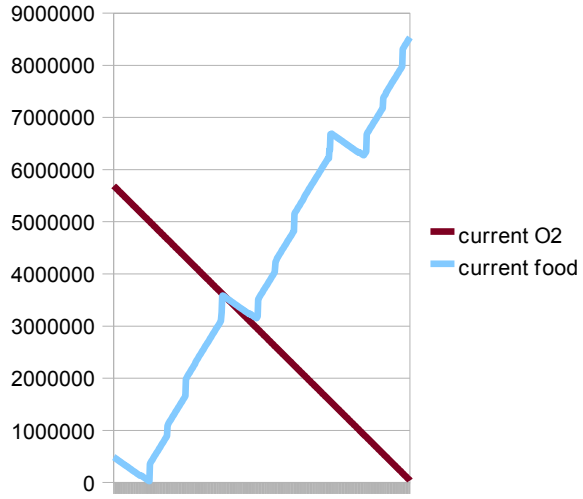


Illustration 3: This picture shows how we message between our agents in the MadKit model

Results





The graphs above show a run of the simulation where we started with 6,000 plants and 5 humans and a spaceship size of 10m x 10m x 250m. The carbon-dioxide (CO2) levels off pretty quickly, because it is basically just the amount breathed out by humans minus the amount breathed in by the living plants. The amount of active plants will level out based on the amount of carbon-dioxide, because that is what they need to stay active, and only a few of the plants will get all the CO2 that they need, because the other plants will breathe in their required amount and the rest will have less, maybe not enough for them. There is never enough oxygen (O2) to support the humans with the number of plants the carbon-dioxide levels will allow, and the only way to provide more carbon-dioxide is to put in more humans into the simulation, but those humans will require more oxygen, thus worsening the problem. In this case, there was enough oxygen to support the humans for the 500 day time period, but there was only enough air left for about a week or two longer. The food goes up erratically, because the plants only produce for a certain time before they must be replanted, causing a decline in the amount of food for a few days before the plants start producing again.

The food supply was a combined effect of all the different kinds of plants. The food is measured in calories. In our simulation we used 15% strawberries, 15% potatoes, 15% tomatoes, 15% lettuce, and 40% soybeans. Each plant had a slightly different behavior. The strawberries lasted a long time (90 days) but produced a little food every day. The lettuce

grew for 45 days then produced some food (then needed 45 days more to be replanted and grow to produce food again). The soybeans grew for 60 days then were harvested for a lot of calories. The potatoes grew for 45 days and were harvested for some food (not as much as the soybeans).

These results are typical of our simulation. If we changed the number of starting plants they would just level off to match the CO2 produced by the humans. The biggest factor for human survival was the amount of oxygen in the spaceship. The plants would never produce enough to stabilize the system, so it continuously decreased. If the spaceship size was too small to start then the humans would die of lack of oxygen before the 500 days were over.

Conclusions and Significant Achievements:

If we ran the model with a few humans, we could support only enough plants that the CO_2 the humans breathed each day would allow. The humans always needed much more oxygen than the plants could provide. In any real world trip like the one that we simulated, we would need to provide more oxygen for the humans in the spacecraft, or . The food generated by the plants (plus the supply we started with) was always enough.

One of the goals of this project was to learn how to use MadKit and create a running simulation. This was a significant achievement since we had very little examples to use. Plus we started to distribute the agents between two networked computers to create a distributed model. This work was started but not yet completed by the time this final report was written.

Teamwork & Acknowledgments

Our project for the supercomputing challenge was very dependent upon teamwork. Dhaivat is in Florida, but he still was part of our project because he was attempting a mathematical model. Colin did the programing with Java for our agent-based simulation. Michael & Matthew did all of the research that was needed for the simulations. Matthew also did some of graphics for the project.

We acknowledge these people who helped us with feedback, coding and model development:

Liz Cooper – our main mentor, helped us with everything
Jim Redman – helped us with distributing the model and getting the MadKit programming started
Robert Robey – helped us get some references on interplanetary space travel
Nick Bennett – gave us feedback on our interim report
Fabien Michel – one of the developers of MadKit that gave us some help on the program structure
Zeynep Unal – our GATE teacher
Cheryl Kuske – helped us with understanding photosynthesis and biomass

References

[1] MarsPort 2002 Competition, NASA; http://projects.olin.edu/marsport/

[2] Biological constraints on interstellar travel,
Lecture Notes in Physics, Springer Berlin / Heidelberg,
Volume 390/1991, in "Bioastronomy The Search for Extraterrestial Life — The Exploration Broadens", Pages 333-337

[3] Wikipedia article on hydroponics

[4] Rising Carbon Dioxide is great for plants by Sylvan H. Wittwer Published in the Fall 1992 issue of Policy Review

[5] MadKit http://www.madkit.org/

[6] MadKit Developer Guide

[7] Oxygen wikipedia article

[8] Carbon Dioxide wikipedia article

[9] Photosynthesis wikipedia article

Code

EcosystemLauncher

```
package net.laschools.team15.ecosystem;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Properties;

import madkit.kernel.AgentAddress;
import madkit.kernel.Scheduler;
import madkit.kernel.StringMessage;

public class EcosystemLauncher extends Scheduler {

		/**
		 *
		 */
		private static final long serialVersionUID = -5149200639469027319L;

		/** initial number of humans */
		public int initialNumberOfHumans = Integer.parseInt(getProperty("SpaceshipParameters.properties", "humans"));

		/** initial number of plants */
		public int initialNumberOfPlants = Integer.parseInt(getProperty("SpaceshipParameters.properties", "plants"));

		/** Calories to start in the storage. */
		public double startingFood = initialNumberOfHumans*59.0*1650.0;// 59 day supply ....
		//public double startingFood = 10000;


		/** minimum grams of O2 in the air allowed for human survival */
		protected double minO2 = 0;

		/** name of our world */
		public static final String SPACESHIP_COMMUNITY = "SpaceShip";

		static double spaceShipLength = Double.parseDouble(getProperty("SpaceshipParameters.properties", "spaceshipLength"));

		//public double percentGreenhouse = 90; //initially make the spaceship 90% greenhouse

		/** run number is incremented each time we run the simulation */
		public static long runNumber = 0;

		/** days to run */
		public static int days = Integer.parseInt(getProperty("SpaceshipParameters.properties", "travelDays"));

		/** pause between days for a number of ms*/
		public static int timeDelay = Integer.parseInt(getProperty("SpaceshipParameters.properties", "timeDelay"));

		public static String summaryFileName = "SimulationSummary.csv";
		public static String runDataFileName = "SpaceShipSimulation."+ runNumber + ".csv";
		public static String directory = "." +  System.getProperty("file.separator")+ System.getProperty("file.separator") +
getProperty("SpaceshipParameters.properties","experimentName") + System.getProperty("file.separator")+ System.getProperty("file.separator");

		public static long spaceshipSize = 0;

		/** Activate the simulation. Create the plants and humans.
		 * Also read the run number and increment it (and write it back to the run.property file)
		 * */
		public void activate() {
				IncrementRun();
```

```
                                WriteSimulationParameters();
                                System.out.println("Simulation run number is " + runNumber);
                                runDataFileName = "SpaceShipSimulation."+ runNumber + ".csv";
                                //gases = new GasMoniter();
                                //food = new FoodMoniter();
                                createGroup(true, SPACESHIP_COMMUNITY,null ,null);
                                requestRole(SPACESHIP_COMMUNITY, "launcher", null);

                                //createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, getAddress().getKernel().toString(), null, null);
                                //requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, getAddress().getKernel().toString(), "launcher",
null);

                                createGroup(true, SPACESHIP_COMMUNITY, "launcher", null, null);
                                requestRole(SPACESHIP_COMMUNITY, "launcher", "launcher", null);

                                System.out.println("Spaceship size is " + spaceShipLength *10 *10 + " cubic meters");
                                spaceshipSize = Long.parseLong(""+Math.round(spaceShipLength *10 *10)); //cubic meters of spaceship
                                System.out.println("Spaceship size is " + spaceShipLength *10 *10 *35.3146667 + " cubic feet");
                                System.out.println("This spaceship is about 30 ft x 30 ft x " + spaceShipLength *10 *10 *35.3146667/900.0 + " feet");

                                GasMonitor gM = new GasMonitor();
                                launchAgent(gM, "Gas Moniter", false);
                                launchAgent(new FoodMonitor(), "Food Moniter", false);
                                launchAgent(new PopulationMonitor(), "Population Monitor",false);
                                launchAgent(new GreenhouseMonitor(), "Greenhouse Monitor",false);
                                AgentAddress foodMonitorAgent = getAgentWithRole(SPACESHIP_COMMUNITY,"monitor","monitorfood");
                                sendMessage(foodMonitorAgent,new StringMessage("addfood " + startingFood));
                                AgentAddress populationMonitor = getAgentWithRole(SPACESHIP_COMMUNITY,"monitor","monitorpopulation");
                                sendMessage(populationMonitor, new StringMessage("addPerson=" + initialNumberOfHumans));
                                AgentAddress greenhouseMonitor =
getAgentWithRole(SPACESHIP_COMMUNITY,"monitor","monitorgreenhouse");
                                sendMessage(greenhouseMonitor, new StringMessage("initPlants=" + initialNumberOfPlants));


                                launchHumans(initialNumberOfHumans);
                                launchPlants(initialNumberOfPlants);
                                launchScheduler();
                                //launchAgent(new EcosystemWatcher(), "Watcher", false);
                                //TurboMethodActivator oberserversDoIt = new
TurboMethodActivator("watch",SPACESHIP_COMMUNITY,"monitor","watcher");
                                //addActivator(oberserversDoIt);


                }

                /** writes all the important STARTING simulation parameters for each run */
                public void WriteSimulationParameters() {
                                //to start the file has the text "run,travel days,starting humans,starting plants,starting food,spaceship
length,survivors,ending active plants,ending CO2, ending O2,ending food"
                                //append these startup parameters to the run data
                                try {
                                                FileWriter fw;
                                                fw = new FileWriter(directory + summaryFileName, true);
                                                BufferedWriter out = new BufferedWriter(fw);
                                                out.write(runNumber + "," + days+ "," +
initialNumberOfHumans+","+initialNumberOfPlants+","+startingFood+","+spaceShipLength+",");
                                                out.close();
                                } catch (FileNotFoundException fnf) {
                                                //create the file
                                                try {
                                                                //make the directory
                                                                File f = new File(directory);
                                    f.mkdir();
                                    File file = new File(directory + summaryFileName );
                                    // Create file
                                    boolean success = file.createNewFile();
                                    if (success) {
                                                FileWriter fw;
                                                                fw = new FileWriter(directory + summaryFileName, true);
                                                                BufferedWriter out = new BufferedWriter(fw);
```

```
                                                                        out.write("run,travel days,starting humans,starting plants,starting
food,spaceship length,survival days,survivors,ending active plants,ending CO2, ending O2,ending food\n\r");
                                                                        out.write(runNumber + "," + days+ "," +
initialNumberOfHumans+","+initialNumberOfPlants+","+startingFood+","+spaceShipLength+",");
                                                                        out.close();
                                        } else {
                                           // File already exists, something is wrong here
                                        }

                                      } catch (IOException e) {
                                                System.err.println(e);
                                      }

                                    }catch (Exception e){

                                    }
                        }



                        /** Reads the run number and increments it */
                        public void IncrementRun() {
                                    Properties propFile = new Properties();

      try {
         // load in the property file
                        File runFile = new File("simulation_run.properties");
         FileInputStream in = new FileInputStream(runFile);
         propFile.load(in);
         in.close();
         try {
            // get the old run value and set the new one.
            String strValue = propFile.getProperty("NextRunNumber");
            long longValue = Long.parseLong(strValue);
            runNumber = longValue;
            longValue++;
            Long newIncrementedValue = new Long(longValue);
            FileOutputStream out = new FileOutputStream("simulation_run.properties");
            propFile.setProperty("NextRunNumber", newIncrementedValue.toString());
            propFile.store(out,"Simulation Run Number");
            out.close();
         } catch (Exception ex) {
            ex.printStackTrace();
         }

         in.close();
      } catch (IOException ioe) {
         ioe.printStackTrace();
      }

   }



                        /** Create the humans. */
                        public synchronized void launchHumans(int numberOfHumans) {
                                    for (int i = 0; i < numberOfHumans; i++) {
                                                Human human = new Human();
                                                launchAgent(human, "Human " + i, false);
                                    }
                                    System.err.println(numberOfHumans + " humans launched");

                        }

                        /** Create the plants. */
                        public synchronized void launchPlants(int numberOfPlants) {
                                    //divide crops into 5 sections for tomatos,potatos,soybeans,strawberries, and lettuce
```

```
                        //15%strawberries,15%lettuce,40%soybeans,15%tomatoes,15%potatoes
                        for (int i = 0; i < numberOfPlants*0.15; i++) {
                                Strawberry strawberry = new Strawberry();
                                launchAgent(strawberry, "Strawberry Plant " + i, false);
                        }
                        for (int i = 0; i < numberOfPlants*0.15; i++) {
                                Tomato tomato = new Tomato();
                                launchAgent(tomato, "Tomato Plant " + i, false);
                        }
                        for (int i = 0; i < numberOfPlants*0.15; i++) {
                                Lettuce lettuce = new Lettuce();
                                launchAgent(lettuce, "Lettuce Plant " + i, false);
                        }
                        for (int i = 0; i < numberOfPlants*0.15; i++) {
                                Potato potato = new Potato();
                                launchAgent(potato, "Potato Plant " + i, false);
                        }

                        for (int i = 0; i < numberOfPlants*0.40; i++) {
                                Soybean soybean = new Soybean();
                                launchAgent(soybean, "Soybean Plant " + i, false);
                        }
                        System.err.println(numberOfPlants + " plants launched");

        }

        protected void launchScheduler() {

                        System.err.println("launching Ecosystem Scheduler...");
                        launchAgent(new EcosystemScheduler(), "EcoystemScheduler", false);
                        AgentAddress schedulerAgent = getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"simulation",
"scheduler");
                        sendMessage(schedulerAgent,new StringMessage("Start Sim"));

                        // }
        }
/**
        protected void launchWatcher() {
                        System.err.println("launching Ecosystem Watcher...");
                        launchAgent(new EcosystemWatcher(), "Ecosystem Watcher", false);
                        //addProbe(co2Observer);
                        //a2 = new TurboMethodActivator("observeCO2",SPACESHIP_COMMUNITY,"plants", "foodProducer");
                        //addActivator(a2);
                        //a1 = new TurboMethodActivator("observeCO2",SPACESHIP_COMMUNITY,"monitor", "gasmonitor");
                        //addActivator(a1);
        }

        */
        public double getSpaceShipLength() {
                        return spaceShipLength;
        }

/*
        public double getPercentGreenhouse() {
                        return percentGreenhouse;

        }
*/


public void live() {
        exitImmediatlyOnKill();
}


public void end()
{
        disposeMyGUI();
        super.end();
}
```

```
                    //public static double getFood() {
                    //                  return 50;
                    //}

                    //public static void addFood(double changeOfFood) {
                    //                  //food  = food + changeOfFood;
                    //}
static String getProperty(String file, String propertie){
                    Properties propFile = new Properties();
                    String retrieved = "";

   try {
      // load in the property file
                    File runFile = new File(file);
      FileInputStream in = new FileInputStream(runFile);
      propFile.load(in);
      retrieved = propFile.getProperty(propertie);
      in.close();
   } catch (Exception ex) {
      ex.printStackTrace();
   }
   return retrieved;
}
}
```

EcosystemScheduler

```
package net.laschools.team15.ecosystem;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Properties;

import madkit.kernel.Activator;
import madkit.kernel.AgentAddress;
import madkit.kernel.Message;
import madkit.kernel.ReferenceableAgent;
import madkit.kernel.Scheduler;
import madkit.kernel.StringMessage;
import madkit.simulation.activators.TurboMethodActivator;

public class EcosystemScheduler extends Scheduler implements ReferenceableAgent {

		Activator humanActivator, plantActivator;
		/** Enables the simulation to stop. */
		public boolean run = true;
		public boolean cO2Known,foodKnown,o2Known,humansKnown,activePlantsKnown,humanStatusKnown = false;

		boolean start = false;

		public double currentCO2;
		public double currentO2;
		public double currentFood;
		public int humans; //the number of humans still alive
		public long activePlants; //the number of active plants
		//public long plantIncrease = 100; //increase plants by this amount each simulation run
		//public long spaceshipSizeIncrease = 35; // increase spaceship size by this amount each simualation
		//public long maxPlants = 10; // maximum number of plants per cubic meter in any spaceship
		//public long maxSpaceShipSize = 10000; // cubic meters




		public void activate() {
				createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, getAddress().getKernel().toString(), null, null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, getAddress().getKernel().toString(), "scheduler", null);

				createGroup(true, EcosystemLauncher.SPACESHIP_COMMUNITY, "simulation", null, null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "simulation", "scheduler", null);
		}


		/** sets up the agent activators */
		public synchronized void setActivators() {
				run = false;
				humanActivator = new TurboMethodActivator("behavior", EcosystemLauncher.SPACESHIP_COMMUNITY,
"human", "foodConsumers");
				addActivator(humanActivator);
				humanActivator.initialize();

				plantActivator = new TurboMethodActivator("behavior", EcosystemLauncher.SPACESHIP_COMMUNITY, "plants",
"foodProducer");
				addActivator(plantActivator);
				plantActivator.initialize();

				run = true;
		}

		public void live() {
```

```
                        int steps = 0;
                        setActivators();
                        //while (true || steps > 1000) {
                        getCO2();
                        getFood();
                        getO2();
                        getHumans();
                getActivePlants();

                try {
                   BufferedWriter out = new BufferedWriter(new FileWriter(EcosystemLauncher.directory+EcosystemLauncher.runDataFileName,
true));
                   out.write("day,live humans,active plants,current CO2,current O2,current food\r\n");
                   out.write("0"+","+humans+","+activePlants+","+currentCO2+","+currentO2+","+currentFood+"\n\r");
                   out.close();
                } catch (IOException e) {

                }
                        while (steps < EcosystemLauncher.days){ //one step per day

                                //exitImmediatlyOnKill();
                                // if (delay==0)
                                // Thread.yield(); // So we avoid locking other threads on
                                // cooperative JVM
                                // else
                                //pause(30);
                                //if(start){
                                //System.err.println("Running!");
                                plantActivator.execute();
                                humanActivator.execute();

                                steps++;

                                getCO2();
                                getFood();
                                getO2();
                                getHumans();

                          getActivePlants();
                          //AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorfood");
                          //sendMessage(gasMonitorAgent,new StringMessage("updatechart"));

                                System.out.println("End of Day " + steps + " CO2 = "+ currentCO2 + " O2 = " + currentO2 + " food =
" + currentFood +" humans = " + humans + " active plants = " + activePlants);
                                //write this to the run data file
                                try {
                             BufferedWriter out = new BufferedWriter(new
FileWriter(EcosystemLauncher.directory+EcosystemLauncher.runDataFileName, true));
                             out.write(steps+","+humans+","+activePlants+","+currentCO2+","+currentO2+","+currentFood+"\r\n");
                             out.close();
                          } catch (IOException e) {

                          }
                          exitImmediatlyOnKill();
                          if (humans<1) {//everyone is dead !
                                                System.out.println("All humans are dead, restart simulation");
                                                writeEndSummary(steps);
                                                 endValueUp();
                                                 end();
                                                 System.exit(199);
                                }

                          //pause a little to slow down computations (let cpu cool)
                          try {
                                                Thread.sleep(EcosystemLauncher.timeDelay);
                                } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                }
```

```
                                        //}else{
                                        //                      System.out.println("Waiting for a start message.");
                                        //}
                                        }
                                         writeEndSummary(steps);
                                         endValueUp();
                                         end();
                                         //System.exit(99);

                        }

/** writes the values of the ending parameters to the run details file */
                        public void writeEndSummary(int maxSteps) {
                                        try {
                                    BufferedWriter out = new BufferedWriter(new FileWriter(EcosystemLauncher.directory+EcosystemLauncher.summaryFileName,
true));
                                    out.write(maxSteps+","+humans + "," + activePlants + "," + currentCO2+","+currentO2+","+currentFood+"\r\n");
                                    out.close();
                                } catch (IOException e) {

                                }

                        }


                        public void end()
                        {
                                        disposeMyGUI();
                                        super.end();
                        }

                        // //// ACCESSOR METHODS ////////////////////

                        /** sets the Run state */

                        public void setRun(boolean run) {
                                        this.run = run;
                        }

                        /** gets the run state */
                        public boolean getRun() {
                                        return run;
                        }

                        /** this messages the gas monitor to get the current CO2 in the spaceship */
                        public void getCO2() {
                                         cO2Known = false;
                                        //message to gas monitor to get the current CO2 in the air
                                        AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","gasmonitor");
                     sendMessage(gasMonitorAgent,new StringMessage("GetCO2 "));
                                        //wait for the reply
                    while(!cO2Known){
                        try {
                                                                        Thread.sleep(50);
                                                                        System.out.println("Ecosystem waiting for CO2 reading");
                                                        } catch (InterruptedException e) {
                                                                        // TODO Auto-generated catch block
                                                                        e.printStackTrace();
                                                        }
                    }

                        }

                        /** this messages the gas monitor to get the current O2 in the spaceship */
                        public void getO2() {
                                         o2Known = false;
                                        //message to gas monitor to get the current O2 in the air
                                        AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","gasmonitor");
```

```
      sendMessage(gasMonitorAgent,new StringMessage("GetO2 "));
                                    //wait for the reply
      while(!o2Known){
                  try {
                                                Thread.sleep(50);
                                                System.out.println("Ecosystem waiting for O2 reading");
                                          } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                          }
      }

                  }

                  /** this messages the food monitor to get the current food supply */
                  public void getFood() {
                                     foodKnown = false;
                                    //message to gas monitor to get the current O2 in the air
                                    AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorfood");
      sendMessage(gasMonitorAgent,new StringMessage("How much food is left? "));
                                    //wait for the reply
      while(!foodKnown){
                  try {
                                                Thread.sleep(50);
                                                System.out.println("Ecosystem waiting for food reading");
                                          } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                          }
      }

                  }

                  /** this messages the food monitor to get the current food supply */
                  public void getHumans() {
                    humansKnown = false;
                   //message to gas monitor to get the current O2 in the air
                     AgentAddress populationAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorpopulation");
      sendMessage(populationAgent,new StringMessage("Get Population"));
                                    //wait for the reply
      while(!humansKnown){
                  try {
                                                Thread.sleep(50);
                                                System.out.println("Ecosystem waiting for population");
                                          } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                          }
      }

                  }

/** gets the number of active (not dead or dying) plants that are photosynthesizing */
                  public void getActivePlants() {
                                    //ask the plant monitor for this information
                                      activePlantsKnown = false;
                                                  //message to the greenhouse monitor to get the current number of active plants
                                                   AgentAddress greenhouseMonitor =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorgreenhouse");
                                             sendMessage(greenhouseMonitor,new StringMessage("Get Active Plants"));
                                                //wait for the reply
                                           while(!activePlantsKnown){
                                                try {
                                                                                                Thread.sleep(50);
                                                                                                System.out.println("Ecosystem waiting for active
plants count");
                                                                                      } catch (InterruptedException e) {
                                                                                                // TODO Auto-generated catch block
```

```
                                                                                e.printStackTrace();
                                                                        }
                                }
        }

        public void receiveMessage (Message m){
                String messageString = ((StringMessage)m).getString();
                if (messageString.contains("CurrentCO2")) {
                        String[] s = messageString.split("=");
                        currentCO2 = Double.parseDouble(s[1]);
                        cO2Known = true;
                }
                if (messageString.contains("CurrentO2")) {
                        String[] s = messageString.split("=");
                        currentO2 = Double.parseDouble(s[1]);
                        o2Known = true;
                }
                if (messageString.contains("Food Left is ")) {
                        String[] s = messageString.split(" ");
                        currentFood = Double.parseDouble(s[3]);
                        foodKnown = true;
                }
                if (messageString.contains("Human Died")) {
                        humans--;
                }
                if (messageString.contains("Population")) {
                        String[] s = messageString.split("=");
                        humans = Integer.parseInt(s[1]);
                        humansKnown=true;
                }
                if (messageString.contains("Active Plants")) {
                        String[] s = messageString.split("=");
                        activePlants = Long.parseLong(s[1]);
                        activePlantsKnown=true;
                }
                if (messageString.contains("Start Sim")) {
                        start = true;
                        System.out.println("Got the start message");
                }

        }
        public void endValueUp() {
                Properties propFile = new Properties();

try {
   // load in the property file
        File runFile = new File("SpaceshipParameters.properties");
   FileInputStream in = new FileInputStream(runFile);

   propFile.load(in);

   String strValue = propFile.getProperty("plants");
   long plants = Long.parseLong(strValue);

   strValue = propFile.getProperty("maxPlantsPerCubicMeter");
   long maxPlantsPerCubicMeter = Long.parseLong(strValue);

   strValue = propFile.getProperty("minPlantsPerCubicMeter");
   long minPlantsPerCubicMeter = Long.parseLong(strValue);

   strValue = propFile.getProperty("spaceshipSizeIncrease");
   long spaceShipSizeIncrease = Long.parseLong(strValue);

   strValue = propFile.getProperty("plantIncrease");
   long plantIncrease = Long.parseLong(strValue);

   strValue = propFile.getProperty("maxSpaceshipLength");
   long maxSpaceshipLength = Long.parseLong(strValue);

   strValue = propFile.getProperty("startingPlants");
```

```
        long startingPlants = Long.parseLong(strValue);

        strValue = propFile.getProperty("spaceshipLength");
        double spaceshipLength = Double.parseDouble(strValue);

        in.close();
        try {
                // get the old run value and set the new one.
                plants = plants + plantIncrease;//increase plants for the next run

                double maxPlantsForSpaceship = maxPlantsPerCubicMeter * EcosystemLauncher.spaceShipLength*10*10;
           System.out.println("Next number of plants ="+plants+" maximum plants for spacehip size is "+maxPlantsForSpaceship);
                if (plants <= maxPlantsForSpaceship ) {
                        FileOutputStream out = new FileOutputStream("SpaceshipParameters.properties");
                        propFile.setProperty("plants", plants +"");
                        propFile.store(out,"Simulation Parameters");
                        out.close();
                } else { //exceeded plant capacity, start over with a new spacehip size
                        FileOutputStream out = new FileOutputStream("SpaceshipParameters.properties");
                        double newSpaceShipLength = spaceshipLength + spaceShipSizeIncrease;
                        double spaceShipSize = 10 * 10 * newSpaceShipLength;
                        long minPlants = (long)(spaceShipSize * minPlantsPerCubicMeter);
                        if (startingPlants > minPlants) {
                                minPlants = startingPlants;
                        }
                        System.out.println("Plants have exceeded greenhouse capacity. Minimum number of plants for new spacehip size is " +
minPlants);
                        propFile.setProperty("plants", minPlants +"");
                        propFile.setProperty("spaceshipLength", newSpaceShipLength +"");
                        propFile.store(out,"Simulation Parameters");
                        out.close();
                }
        } catch (Exception ex) {
                ex.printStackTrace();
        }

        in.close();
      } catch (IOException ioe) {
        ioe.printStackTrace();
      }

   }

}
```

Plant Class

```
package net.laschools.team15.ecosystem;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Properties;

import madkit.kernel.Activator;
import madkit.kernel.AgentAddress;
import madkit.kernel.Message;
import madkit.kernel.ReferenceableAgent;
import madkit.kernel.Scheduler;
import madkit.kernel.StringMessage;
import madkit.simulation.activators.TurboMethodActivator;

public class EcosystemScheduler extends Scheduler implements ReferenceableAgent {

		Activator humanActivator, plantActivator;
		/** Enables the simulation to stop. */
		public boolean run = true;
		public boolean cO2Known,foodKnown,o2Known,humansKnown,activePlantsKnown,humanStatusKnown = false;

		boolean start = false;

		public double currentCO2;
		public double currentO2;
		public double currentFood;
		public int humans; //the number of humans still alive
		public long activePlants; //the number of active plants
		//public long plantIncrease = 100; //increase plants by this amount each simulation run
		//public long spaceshipSizeIncrease = 35; // increase spaceship size by this amount each simualation
		//public long maxPlants = 10; // maximum number of plants per cubic meter in any spaceship
		//public long maxSpaceShipSize = 10000; // cubic meters




		public void activate() {
				createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, getAddress().getKernel().toString(), null, null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, getAddress().getKernel().toString(), "scheduler", null);

				createGroup(true, EcosystemLauncher.SPACESHIP_COMMUNITY, "simulation", null, null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "simulation", "scheduler", null);
		}


		/** sets up the agent activators */
		public synchronized void setActivators() {
				run = false;
				humanActivator = new TurboMethodActivator("behavior", EcosystemLauncher.SPACESHIP_COMMUNITY,
"human", "foodConsumers");
				addActivator(humanActivator);
				humanActivator.initialize();

				plantActivator = new TurboMethodActivator("behavior", EcosystemLauncher.SPACESHIP_COMMUNITY, "plants",
"foodProducer");
				addActivator(plantActivator);
				plantActivator.initialize();

				run = true;
		}

		public void live() {
```

```
                        int steps = 0;
                        setActivators();
                        //while (true || steps > 1000) {
                        getCO2();
                        getFood();
                        getO2();
                        getHumans();
                getActivePlants();

                try {
                   BufferedWriter out = new BufferedWriter(new FileWriter(EcosystemLauncher.directory+EcosystemLauncher.runDataFileName,
true));
                   out.write("day,live humans,active plants,current CO2,current O2,current food\r\n");
                   out.write("0"+","+humans+","+activePlants+","+currentCO2+","+currentO2+","+currentFood+"\n\r");
                   out.close();
                } catch (IOException e) {

                }
                        while (steps < EcosystemLauncher.days){ //one step per day

                                //exitImmediatlyOnKill();
                                // if (delay==0)
                                // Thread.yield(); // So we avoid locking other threads on
                                // cooperative JVM
                                // else
                                //pause(30);
                                //if(start){
                                //System.err.println("Running!");
                                plantActivator.execute();
                                humanActivator.execute();

                                steps++;

                                getCO2();
                                getFood();
                                getO2();
                                getHumans();

                          getActivePlants();
                          //AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorfood");
                          //sendMessage(gasMonitorAgent,new StringMessage("updatechart"));

                                System.out.println("End of Day " + steps + " CO2 = "+ currentCO2 + " O2 = " + currentO2 + " food =
" + currentFood +" humans = " + humans + " active plants = " + activePlants);
                                //write this to the run data file
                                try {
                             BufferedWriter out = new BufferedWriter(new
FileWriter(EcosystemLauncher.directory+EcosystemLauncher.runDataFileName, true));
                             out.write(steps+","+humans+","+activePlants+","+currentCO2+","+currentO2+","+currentFood+"\r\n");
                             out.close();
                          } catch (IOException e) {

                          }
                          exitImmediatlyOnKill();
                          if (humans<1) {//everyone is dead !
                                                System.out.println("All humans are dead, restart simulation");
                                                writeEndSummary(steps);
                                                 endValueUp();
                                                 end();
                                                 System.exit(199);
                                }

                          //pause a little to slow down computations (let cpu cool)
                          try {
                                                Thread.sleep(EcosystemLauncher.timeDelay);
                                } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                }
```

```
                                        //}else{
                                        //                      System.out.println("Waiting for a start message.");
                                        //}
                                        }
                                         writeEndSummary(steps);
                                         endValueUp();
                                         end();
                                         //System.exit(99);

                        }

/** writes the values of the ending parameters to the run details file */
                        public void writeEndSummary(int maxSteps) {
                                        try {
                                    BufferedWriter out = new BufferedWriter(new FileWriter(EcosystemLauncher.directory+EcosystemLauncher.summaryFileName,
true));
                                    out.write(maxSteps+","+humans + "," + activePlants + "," + currentCO2+","+currentO2+","+currentFood+"\r\n");
                                    out.close();
                                } catch (IOException e) {

                                }

                        }


                        public void end()
                        {
                                        disposeMyGUI();
                                        super.end();
                        }

                        // //// ACCESSOR METHODS ////////////////////

                        /** sets the Run state */

                        public void setRun(boolean run) {
                                        this.run = run;
                        }

                        /** gets the run state */
                        public boolean getRun() {
                                        return run;
                        }

                        /** this messages the gas monitor to get the current CO2 in the spaceship */
                        public void getCO2() {
                                         cO2Known = false;
                                        //message to gas monitor to get the current CO2 in the air
                                        AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","gasmonitor");
                     sendMessage(gasMonitorAgent,new StringMessage("GetCO2 "));
                                        //wait for the reply
                    while(!cO2Known){
                        try {
                                                                        Thread.sleep(50);
                                                                        System.out.println("Ecosystem waiting for CO2 reading");
                                                        } catch (InterruptedException e) {
                                                                        // TODO Auto-generated catch block
                                                                        e.printStackTrace();
                                                        }
                    }

                        }

                        /** this messages the gas monitor to get the current O2 in the spaceship */
                        public void getO2() {
                                         o2Known = false;
                                        //message to gas monitor to get the current O2 in the air
                                        AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","gasmonitor");
```

```
      sendMessage(gasMonitorAgent,new StringMessage("GetO2 "));
                                    //wait for the reply
      while(!o2Known){
                  try {
                                                Thread.sleep(50);
                                                System.out.println("Ecosystem waiting for O2 reading");
                                          } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                          }
      }

                  }

                  /** this messages the food monitor to get the current food supply */
                  public void getFood() {
                                     foodKnown = false;
                                    //message to gas monitor to get the current O2 in the air
                                    AgentAddress gasMonitorAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorfood");
      sendMessage(gasMonitorAgent,new StringMessage("How much food is left? "));
                                    //wait for the reply
      while(!foodKnown){
                  try {
                                                Thread.sleep(50);
                                                System.out.println("Ecosystem waiting for food reading");
                                          } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                          }
      }

                  }

                  /** this messages the food monitor to get the current food supply */
                  public void getHumans() {
                     humansKnown = false;
                    //message to gas monitor to get the current O2 in the air
                     AgentAddress populationAgent =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorpopulation");
      sendMessage(populationAgent,new StringMessage("Get Population"));
                                    //wait for the reply
      while(!humansKnown){
                  try {
                                                Thread.sleep(50);
                                                System.out.println("Ecosystem waiting for population");
                                          } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                          }
      }

                  }

/** gets the number of active (not dead or dying) plants that are photosynthesizing */
                  public void getActivePlants() {
                                    //ask the plant monitor for this information
                                      activePlantsKnown = false;
                                                //message to the greenhouse monitor to get the current number of active plants
                                                 AgentAddress greenhouseMonitor =
getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorgreenhouse");
                                           sendMessage(greenhouseMonitor,new StringMessage("Get Active Plants"));
                                                //wait for the reply
                                         while(!activePlantsKnown){
                                                try {
                                                                                                Thread.sleep(50);
                                                                                                System.out.println("Ecosystem waiting for active
plants count");
                                                                                      } catch (InterruptedException e) {
                                                                                                // TODO Auto-generated catch block
```

```
                                                                                e.printStackTrace();
                                                                        }
                                }
                }

                public void receiveMessage (Message m){
                                String messageString = ((StringMessage)m).getString();
                                if (messageString.contains("CurrentCO2")) {
                                                String[] s = messageString.split("=");
                                                currentCO2 = Double.parseDouble(s[1]);
                                                cO2Known = true;
                                }
                                if (messageString.contains("CurrentO2")) {
                                                String[] s = messageString.split("=");
                                                currentO2 = Double.parseDouble(s[1]);
                                                o2Known = true;
                                }
                                if (messageString.contains("Food Left is ")) {
                                                String[] s = messageString.split(" ");
                                                currentFood = Double.parseDouble(s[3]);
                                                foodKnown = true;
                                }
                                if (messageString.contains("Human Died")) {
                                                humans--;
                                }
                                if (messageString.contains("Population")) {
                                                String[] s = messageString.split("=");
                                                humans = Integer.parseInt(s[1]);
                                                humansKnown=true;
                                }
                                if (messageString.contains("Active Plants")) {
                                                String[] s = messageString.split("=");
                                                activePlants = Long.parseLong(s[1]);
                                                activePlantsKnown=true;
                                }
                                if (messageString.contains("Start Sim")) {
                                                start = true;
                                                System.out.println("Got the start message");
                                }

                }
                public void endValueUp() {
                                Properties propFile = new Properties();

try {
   // load in the property file
                File runFile = new File("SpaceshipParameters.properties");
   FileInputStream in = new FileInputStream(runFile);

   propFile.load(in);

   String strValue = propFile.getProperty("plants");
   long plants = Long.parseLong(strValue);

   strValue = propFile.getProperty("maxPlantsPerCubicMeter");
   long maxPlantsPerCubicMeter = Long.parseLong(strValue);

   strValue = propFile.getProperty("minPlantsPerCubicMeter");
   long minPlantsPerCubicMeter = Long.parseLong(strValue);

   strValue = propFile.getProperty("spaceshipSizeIncrease");
   long spaceShipSizeIncrease = Long.parseLong(strValue);

   strValue = propFile.getProperty("plantIncrease");
   long plantIncrease = Long.parseLong(strValue);

   strValue = propFile.getProperty("maxSpaceshipLength");
   long maxSpaceshipLength = Long.parseLong(strValue);

   strValue = propFile.getProperty("startingPlants");
```

```
        long startingPlants = Long.parseLong(strValue);

        strValue = propFile.getProperty("spaceshipLength");
        double spaceshipLength = Double.parseDouble(strValue);

        in.close();
        try {
                // get the old run value and set the new one.
                plants = plants + plantIncrease;//increase plants for the next run

                double maxPlantsForSpaceship = maxPlantsPerCubicMeter * EcosystemLauncher.spaceShipLength*10*10;
           System.out.println("Next number of plants ="+plants+" maximum plants for spacehip size is "+maxPlantsForSpaceship);
                if (plants <= maxPlantsForSpaceship ) {
                        FileOutputStream out = new FileOutputStream("SpaceshipParameters.properties");
                        propFile.setProperty("plants", plants +"");
                        propFile.store(out,"Simulation Parameters");
                        out.close();
                } else { //exceeded plant capacity, start over with a new spacehip size
                        FileOutputStream out = new FileOutputStream("SpaceshipParameters.properties");
                        double newSpaceShipLength = spaceshipLength + spaceShipSizeIncrease;
                        double spaceShipSize = 10 * 10 * newSpaceShipLength;
                        long minPlants = (long)(spaceShipSize * minPlantsPerCubicMeter);
                        if (startingPlants > minPlants) {
                                minPlants = startingPlants;
                        }
                        System.out.println("Plants have exceeded greenhouse capacity. Minimum number of plants for new spacehip size is " +
minPlants);
                        propFile.setProperty("plants", minPlants +"");
                        propFile.setProperty("spaceshipLength", newSpaceShipLength +"");
                        propFile.store(out,"Simulation Parameters");
                        out.close();
                }
        } catch (Exception ex) {
                ex.printStackTrace();
        }

        in.close();
      } catch (IOException ioe) {
        ioe.printStackTrace();
      }

   }

}
```

Human Class

```
package net.laschools.team15.ecosystem;

import madkit.kernel.AbstractAgent;
import madkit.kernel.AgentAddress;
import madkit.kernel.Message;
import madkit.kernel.ReferenceableAgent;
import madkit.kernel.StringMessage;

public class Human extends AbstractAgent implements ReferenceableAgent {

		/**
		 *
		 */
		private static final long serialVersionUID = 723372266287987542 9L;

		/** The grams of oxygen used per day. 1.8 - 2.4 grams a minute, I'm using 2 a minute.*/
		public double oxygenPerDay = 2.0 * 60.0 * 24.0; //60 min per hour, 24 hours per day = 2880 grams O2

		public boolean foodKnown,cO2Known, o2Known = false;

		/** The grams of carbon dioxide generated per day. */
		public float co2PerDay = 852.48f;

		/** How much food to be eaten each day. */
		public int caloriesPerDay = 1650;

		/** how much food is available */
		public double foodSupply;

		/** How much O2 is in the air */
		public double currentO2;

 /** min percent O2 in the air required by human*/
		public double minO2=18.0;


		/** percent O2 in the air */
		public double percentO2=0;



		/** whether or not this human has died */
		boolean alive = true;

		boolean start = false;

   public AgentAddress[] foodMonitorAgentAddress;

   AgentAddress gasMonitorAgent ;
   AgentAddress foodMonitorAgent ;
   AgentAddress populationAgent ;

		public void activate() {
				setDebug(true);
				createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, "human", "people", null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "human",  "foodConsumers", null);
				gasMonitorAgent = getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","gasmonitor");
      foodMonitorAgent = getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorfood");
      populationAgent = getAgentWithRole(EcosystemLauncher.SPACESHIP_COMMUNITY,"monitor","monitorpopulation");
				//requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "human",  "oyxgenConsumers", null);
				//requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "human",  "carbonDioxideProducers",
null);
		}

		/** human is eating */
		public void eat() {
    getFood();
```

```
if(foodSupply < caloriesPerDay){ //Uh Oh !!!
                                        System.err.println(this.getName()+ " has starved to death." ); //human cannot eat
                                        alive = false;
                                        //tell population monitor
                                    sendMessage(populationAgent,new StringMessage("Human Died"));
                                } else{                         //Eat your fruits and veggies little human!
                                        sendMessage(foodMonitorAgent,new StringMessage("addfood " + (0 - caloriesPerDay)));
                                }
//if ()
                                //sendMessage(foodMonitorAgent,new StringMessage("Ate "+caloriesPerDay));
                                //If not enough food...
                                //ReferenceableAgent foodStore = getRole("foodmonitor");
                                                        /*
                                if (Agent(EcosystemLauncher).getRole("foodMonitor").getFood() < caloriesPerDay){
                                        //KILL!!!
                                        end();
                                        //But otherwise...
                                }else {
                                        //Eat your fruits and veggies little human!
                                        EcosystemLauncher.addFood(0 - caloriesPerDay);
                                }
                                */
        }

        /** human is breathing */
        public void breathe() {
                                //check that there is enough O2 before breathing
                                getO2();
                                percentO2=currentO2*EcosystemLauncher.spaceshipSize;
                                if (currentO2 < oxygenPerDay) {//Oh Oh !!!
                                        //System.err.println(this.getName()+ " has died from too little oxygen."); //human cannot breathe

                                        alive = false;
                                        //tell population monitor
                                   sendMessage(populationAgent,new StringMessage("Human Died"));
                                                        //wait for the reply
                                } else {
   sendMessage(gasMonitorAgent,new StringMessage("DepleteO2="+ oxygenPerDay));
                                //  EcosystemLauncher.gases.addCo2(0 - co2PerDay);
                                //Breathe out!
   sendMessage(gasMonitorAgent,new StringMessage("AddCO2="+ co2PerDay));
                                //  EcosystemLauncher.gases.addO2(oxygenPerDay);
                                }
        }

        public void behavior() {
                                //if(start){
                                  if (alive) {
                                                eat();
                                  }
                                  if (alive) {
                                                breathe();
                                  }
                                //}
        }

        /** this messages the gas monitor to get the current CO2 in the spaceship */
        public void getCO2() {
                                 cO2Known = false;
                                //message to gas monitor to get the current CO2 in the air
 sendMessage(gasMonitorAgent,new StringMessage("GetCO2 "));
                                //wait for the reply
while(!cO2Known){
        try {
                                                        Thread.sleep(50);
                                                        System.out.println("Human waiting for CO2 reading");
                                        } catch (InterruptedException e) {
                                                        // TODO Auto-generated catch block
                                                        e.printStackTrace();
                                        }
```

```
	}

		}

		/** this messages the gas monitor to get the current O2 in the spaceship */
		public void getO2() {
				o2Known = false;
				//message to gas monitor to get the current O2 in the air
	 sendMessage(gasMonitorAgent,new StringMessage("GetO2 "));
				//wait for the reply
	while(!o2Known){
		try {
					Thread.sleep(50);
					System.out.println("Human waiting for O2 reading");
			} catch (InterruptedException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
			}
	}

		}

		/** this messages the food monitor to get the current food supply */
		public void getFood() {
				foodKnown = false;
				//message to food monitor to get the current food supply
	 sendMessage(foodMonitorAgent,new StringMessage("How much food is left? "));
				//wait for the reply
	while(!foodKnown){
		try {
					Thread.sleep(50);
					System.out.println("Human waiting for food reading");
			} catch (InterruptedException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
			}
	}

		}


		public void receiveMessage (Message m){
				String messageString = ((StringMessage)m).getString();
				//System.out.println("Human Got Message ->" + messageString);
				if (messageString.contains("Food Left is")) {
						String[] s = messageString.split("is ");
						foodSupply = Double.parseDouble(s[1]);
						foodKnown = true;
				}

				if (messageString.contains("CurrentO2")) {
						String[] s = messageString.split("=");
						currentO2 = Double.parseDouble(s[1]);
						o2Known = true;
				}
				if (messageString.contains("Are You Alive")) {
						String reply = "Alive="+alive;
						sendMessage(m.getSender(),new StringMessage(reply));

				}else if(messageString.contains("Start Sim")) {
						start = true;
				}


		}


}
```

GasMonitor

```
package net.laschools.team15.ecosystem;

import madkit.kernel.AbstractAgent;
import madkit.kernel.Message;
import madkit.kernel.ReferenceableAgent;
import madkit.kernel.StringMessage;
import madkit.simulation.activators.TurboMethodActivator;

public class GasMonitor extends AbstractAgent implements ReferenceableAgent {

		public boolean sentStartCo2 = false;

		/** density of air 1.204 kg/cubic meter at 20 C - we need gms/cubic meter */
		protected double densityOfAir = 1.204 * 1000;

		/** The total weight in grams of gases in the atmosphere*/
		public double totalGas;

		/** The co2 in the atmosphere */
		public double co2;

		/** The o2 in the atmosphere */
		public double o2;

		/** The co2 in the atmosphere at the start */
		public double startCo2;

		/** The o2 in the atmosphere at the start */
		public double startO2;

		/** The other gases in the atmosphere */
		public double otherGas = 100 - (co2 + o2);

		public void observeCO2 () {
			System.out.println("xxxCO2 is " + co2);
		}

		public void activate() {
			setDebug(true);
			createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor", "Monitors Resources", null);
			requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor","gasmonitor", null);
			totalGas = EcosystemLauncher.spaceShipLength * 10 * 10 * densityOfAir; //73440.0;
			o2 = (21.0 / 100.0) * totalGas;
			co2 = (0.03 / 100.0) * totalGas;
			startO2 = o2;
			startCo2 = co2;
			System.out.println("Spaceship is starting with " + o2+ " grams of O2 and " + co2 + " grams of CO2");
		}

		public double getO2(){
			return o2;
		}

		public double getCo2(){
			return co2;
		}

		public double getStartO2(){
			return startO2;
		}

		public double getStartCo2(){
			return startCo2;
		}

		public void addCo2(double amount){
			co2 = co2 + amount;
```

```
                        if (co2<0) {
                                co2=0;
                        }
        }

        public void addO2(double amount){
                        o2 = o2 + amount;
                        if (o2<0) {
                                o2=0;
                        }
        }

        public void receiveMessage (Message m){
                        String messageString = ((StringMessage)m).getString();
                        //System.out.println("Gas Monitor Got Message -> " + messageString + " from "+ m.getSender().toString());

                        if (messageString.contains("GetCO2")) {
                                String reply = "CurrentCO2="+getCo2();
                                sendMessage(m.getSender(),new StringMessage(reply));
                        }
                        else if (messageString.contains("GetO2")) {
                                String reply = "CurrentO2="+getO2();
                                sendMessage(m.getSender(),new StringMessage(reply));
                        }
                        else if (messageString.contains("AddO2")) {
                                String[] s = messageString.split("=");
                                double o2 = Double.parseDouble(s[1]);
                                addO2(o2);
                        }
                        else if (messageString.contains("DepleteO2")) {
                                String[] s = messageString.split("=");
                                double o2 = Double.parseDouble(s[1]);
                                addO2(0-o2);
                        }
                        else if (messageString.contains("AddCO2")) {
                                String[] s = messageString.split("=");
                                double co2 = Double.parseDouble(s[1]);
                                addCo2(co2);
                        }
                        else if (messageString.contains("DepleteCO2")) {
                                //System.out.println("Got the message!");
                                String[] s = messageString.split("=");
                                double co2 = Double.parseDouble(s[1]);
                                addCo2(0-co2);
                        }
                        else if (messageString.contains("GetStartCO2")) {
                                String reply = "StartCO2="+getCo2();
                                sendMessage(m.getSender(),new StringMessage(reply));
                                sentStartCo2 = true;
                                //System.out.println("Sent the start Co2");
                        }
                        else if (messageString.contains("GetStartO2")) {
                                String reply = "StartO2="+getO2();
                                sendMessage(m.getSender(),new StringMessage(reply));
                        }

        }

}
```

FoodMonitor

```
package net.laschools.team15.ecosystem;

import madkit.kernel.AbstractAgent;
import madkit.kernel.AgentAddress;
import madkit.kernel.Message;
import madkit.kernel.ReferenceableAgent;
import madkit.kernel.StringMessage;
import madkit.linechart.LineChartAgent;
import madkit.linechart.LineChartMessage;


public class FoodMonitor extends AbstractAgent implements ReferenceableAgent {
		/** Yum! Food! */
		public double food = 0.0;
		AgentAddress linechartAddress;
		LineChartAgent foodChart;
		double time  = 0;

		public void activate() {
				setDebug(false);
				createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor", "Monitors Resources", null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor","monitorfood", null);
				//initGUI();
		}

		public void initGUI() {

				foodChart = new LineChartAgent();
				launchAgent(foodChart, "foodplot",true);
				linechartAddress = foodChart.getAddress();
				sendMessage(linechartAddress, new LineChartMessage("food over time"));
				setGUIObject(foodChart);

		}

		public double getFood(){
				return food;
		}

		public void addFood(double amount){
				food = food + amount;
		}

		public void receiveMessage (Message m){
				String messageString = m.toString();
				//System.out.println("Got Message -> " + ((StringMessage)m).getString());
				if (messageString.contains("How much food is left?")){
						sendMessage(m.getSender(),new StringMessage("Food Left is " + food));
				}
				if (messageString.contains("addfood ")){
						String[] s = messageString.split(" ");
						addFood(Double.parseDouble(s[1]));
				}
				if (messageString.contains("updatechart ")){
						time = time + 1;
						//sendMessage(linechartAddress, new LineChartMessage("food over time",time,food));
				}
				//System.out.println(food + " Food Left");
		}



}
```

Greenhouse Monitor

```
package net.laschools.team15.ecosystem;

import madkit.kernel.AbstractAgent;
import madkit.kernel.Message;
import madkit.kernel.ReferenceableAgent;
import madkit.kernel.StringMessage;

/** This monitors the status of the plants in the greenhouse */

public class GreenhouseMonitor extends AbstractAgent implements ReferenceableAgent {


		public int currentPlants = 0 ; //current number of plants (live,dead, or otherwise) in the greenhouse
		public int currentActivePlants = 0 ; //current number of live plants in the greenhouse


		public void activate() {
				setDebug(true);
				createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor", "Monitors Plants in the Greenhouse",
null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor","monitorgreenhouse", null);

		}

		/** gets the current number of plants on the ship */
		public double getPlantCount(){
				return currentPlants;
		}

		/** gets the current number of plants on the ship */
		public double getActivePlantCount(){
				return currentActivePlants;
		}




		/** adds or removes Plants from the greenhouse */
		public void addPlant(int amount){ //remove plant if amount is negative
				currentPlants = currentPlants + amount;

		}


		/** adds an active (live) Plant to the greenhouse. */
		public void addActivePlant(){
				currentActivePlants ++;
		}

		/** adds an active (live) Plant to the greenhouse. */
		public void addActivePlant(int numberToAdd){
				currentActivePlants = currentActivePlants + numberToAdd;
		}

		/** removes active (live) Plants from the greenhouse. Removing an active plant means it is
		 * either dead or not doing any photosynthesis. It can be reactivated */
		public void removeActivePlant(){
				  currentActivePlants --;
				}



		public void receiveMessage (Message m){
				String messageString = m.toString();
				if (messageString.contains("Get Plants")){
						sendMessage(m.getSender(),new StringMessage("Plants =" + currentPlants));
				}
```

```
                        if (messageString.contains("Get Active Plants")){
                                sendMessage(m.getSender(),new StringMessage("Active Plants =" + currentActivePlants));
                        }
                        if (messageString.contains("Died")){
                                removeActivePlant();
                        }

                        if (messageString.contains("Inactivate")){
                                removeActivePlant();
                        }
                        if (messageString.contains("Activate")){
                                addActivePlant();
                        }
                        if (messageString.contains("addPlant")){
                                String[] s = messageString.split("=");
                                int count = Integer.parseInt(s[1]);
                                addPlant(count);;
                        }
                        if (messageString.contains("initPlants")){
                                String[] s = messageString.split("=");
                                int count = Integer.parseInt(s[1]);
                                addPlant(count);
                                addActivePlant(count);
                        }
                        if (messageString.contains("addActivePlant")){
                                String[] s = messageString.split("=");
                                int count = Integer.parseInt(s[1]);
                                addActivePlant(count);;
                        }
                }


}
```

PopulationMonitor

```
package net.laschools.team15.ecosystem;

import madkit.kernel.AbstractAgent;
import madkit.kernel.Message;
import madkit.kernel.ReferenceableAgent;
import madkit.kernel.StringMessage;

public class PopulationMonitor extends AbstractAgent implements ReferenceableAgent {

		public int startingPopulation = 0; //number of humans on the spaceship
		public int currentPopulation = startingPopulation; //current number (can change due to death or birth)


		public void activate() {
				setDebug(true);
				createGroup(false, EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor", "Monitors Population", null);
				requestRole(EcosystemLauncher.SPACESHIP_COMMUNITY, "monitor","monitorpopulation", null);

		}

		/** gets the current number of humans on the ship */
		public double getHumanCount(){
				return currentPopulation;
		}

		/** adds or removes a person from the population */
		public void addPerson(int amount){ //remove person if amount is negative
				currentPopulation = currentPopulation + amount;
				//what do we want to do if currentPopulation == 0?
		}



		public void receiveMessage (Message m){
				String messageString = m.toString();
				if (messageString.contains("Get Population")){
						sendMessage(m.getSender(),new StringMessage("Population =" + currentPopulation));
				}
				if (messageString.contains("Died")){
						addPerson(-1);
				}
				else if (messageString.contains("addPerson")) {
						String[] s = messageString.split("=");
						startingPopulation = Integer.parseInt(s[1]);
						currentPopulation = startingPopulation;
				}
		}



}
```

Strawberry Class

```
package net.laschools.team15.ecosystem;


public class Strawberry extends Plant {



                /** constructor, sets its own parameters */
   Strawberry (){
                foodDays=0;//start producing food on first day
                maxPlantAge = 90;
                CO2UsedPerDay=1.0;
                caloriesPerDay = 0.4;//Plant makes one ripe strawberry every 5 days, each with 2 calories
   }

}
```

Potato Class

```
package net.laschools.team15.ecosystem;

public class Potato extends Plant {

		/** constructor, sets its own parameters */
   Potato (){
		foodDays=45;
		maxPlantAge = 45;//once harvested it is finished
		CO2UsedPerDay=1.0;
		caloriesPerDay = 30;//  one head of lettuce
   }

}
```

Soybean Class

```
package net.laschools.team15.ecosystem;


public class Soybean extends Plant {



                /** constructor, sets its own parameters */
   Soybean (){
                foodDays=60; //start producing food on last day of plant life
                maxPlantAge = 60;//once harvested it is finished
                CO2UsedPerDay=1.0;
                caloriesPerDay = 200;//Plant makes this much calories of soybeans when it is harvested
   }

}
```

Tomato Class

```
package net.laschools.team15.ecosystem;


public class Tomato extends Plant {



                /** constructor, sets its own parameters */
   Tomato (){
                foodDays=60;
                maxPlantAge = 180;
                CO2UsedPerDay=2.0;
                caloriesPerDay = 45;//Plant makes 1 tomato per day after it starts making food
   }

}
```

Lettuce Class

```
package net.laschools.team15.ecosystem;


public class Lettuce extends Plant {

                /** constructor, sets its own parameters */
   Lettuce (){
                foodDays=90;
                maxPlantAge = 90;//once harvested it is finished
                CO2UsedPerDay=0.5;//slow growing
                caloriesPerDay = 300;//  one potato
   }

}
```