

Notes on Implementing StupidModel in Java Swarm

Steve Railsback

15th February 2006

StupidModel will make you smart!

Abstract

This document provides notes on implementing StupidModel in Java Swarm. StupidModel is a series of template models designed to illustrate common modeling techniques and features. The notes are intended to help teachers and self-taught users understand and explain how models like StupidModel are implemented in Swarm.

StupidModel (documented separately) includes 16 versions, each adding or modifying parts of the model. Our Java Swarm implementation of StupidModel is available separately. This implementation *should not be considered an example of good Swarm programming*; instead, it was designed to illustrate very simple (not robust or efficient) ways to implement basic functions. Example codes distributed with Swarm (e.g., Heatbugs) are better examples of good Swarm programming practices.

Each of the following 16 sections corresponds to a version of StupidModel. These notes and the StupidModel software were developed using Swarm version 2.2.

1 Basic StupidModel

- Following the Swarm modeling paradigm, StupidModel has two swarms: the model swarm (a subclass of SwarmImpl) and an observer swarm (subclassed from GUISwarmImpl). Each swarm has its own objects and a schedule of actions that the objects execute. The simulation itself (the bugs and their space) are created and controlled by the model swarm; the observer swarm creates and controls the model swarm and the observer tools (in this case, a display of the space).
- Each swarm has a “buildObjects” method that instantiates and initializes the swarm’s objects. The swarms also have a “buildActions” method that creates its schedule.
- Following Java Swarm conventions, the Main class is in a separate file, Start-StupidModel.java.

- The StupidModelSwarm schedule includes only one action group, for bug actions. The only action on the action group executes the bugs’ “move” action. The model swarm is “activatedIn” the observer swarm, meaning it is inserted into (at the top) the observer swarm’s schedule.
- The StupidObserverSwarm schedule includes only one action group, for display actions. One action in this group executes the observer’s “_update_” action, which updates the graphical display. A second action “doTkEvents” is needed to tell Swarm to re-draw the screen displays.
- By default in Swarm, schedules execute actions in the order they are scheduled in the code, not (like Repast) in randomized order.
- Swarm does not include a toroidal grid space, so the bugs themselves must figure out when they can jump to the opposite side of the space. To do this, we copied the “xnorm” and “ynorm” methods from Repast’s toroidal space class and put them in the StupidBugs class.
- The observer swarm has an “Object2DDisplayImpl” object (called “bugDisplay”) that draws the bugs on the raster display window (which is an instance of ZoomRasterImpl). Bugs have a “drawSelf” method that draws a point at their location on the display raster window. Execution of this method is controlled by the bugDisplay object.
- In Java Swarm, as in Repast (but not Objective-C Swarm) selectors must be explicitly instantiated before they can be used in scheduling actions (or other applications such as defining the bugDisplay). Defining a selector requires identifying the class it applies to. Sometimes it is convenient to identify the class simply by providing its name in the code. Note that when code is developed in Eclipse, it belongs to a package and the class name used to define a selector must include the package name. See, for example, the selector “moveSel” in the model swarm’s “buildActions” method, which includes: `Class.forName("stupidModel_2.StupidBug")` where `stupidModel_2` is the package name and `StupidBug` is the class to which the selector applies.

2 Bug Growth

- The StupidBug class now includes a new method “grow”.
- “grow” is added to the model schedule in StupidModelSwarm’s buildActions method.

3 Habitat Cells and Resource

- Habitat cells require a new class, HabitatCell.

- The space object (“world” in StupidModelSwarm and “myWorld” in StupidBug) now holds a habitatCell at every location, not bugs. Each habitatCell has a variable (“myBug”) that refers to a bug that is in the cell.
- Food growth is implemented in the HabitatCell method “growFood”.
- Because the space now contains habitatCell objects, the HabitatCell class must now handle drawing. In the observer swarm, the bugDisplay object is modified so that it tells the habitat cells to their “drawSelfOn” method, which in turn simply tells the cell’s bug (if there is one) to execute its “drawSelfOn” method, which draws the bug on the raster.
- The drawSelfOn method in StupidBug now illustrates how color can be used to display the state of individuals: each bug’s color gets more red as the bug gets larger. This requires a color map, which is essentially a numbered list of colors. The color map is created in the observer swarm’s buildObjects method. Starting in this version, the color map’s entries 0 to 63 are shades from white (equal parts red, green, and blue, on a scale of zero to one) to red (red = 1, green and blue = 0).
- (NOTE THAT THIS VERSION RUNS MUCH SLOWER!)

4 Cell and Bug Probes

- To make both bugs and cells probeable, a separate space object is required for each. Now, StupidModelSwarm includes two spaces (“cellWorld” and “bug-World”). StupidObserverSwarm also now includes two display objects (“cellDisplay” and “bugDisplay”), both assigned to the same raster window (see “buildObjects” in the observer swarm). However, because the cells are not actually drawn, cellDisplay is not added to the observer’s “_update_” method.
- The user can now click on habitat cells and bugs in the display window to open a probe to them. Code in the observer’s buildObjects method assigns the left mouse button to cells and the right mouse button to bugs.
- The probes display any variables that are declared public (whether or not they have “get” and “set” methods). Methods do not appear unless the user right-clicks on the object’s name box in the probe display. This opens a full probe display to all public variables and methods.
- By default, Swarm probes are not updated as the model continues to execute. Users will want to override this default so open probes will have their values continuously updated as the model runs. This is done by adding an action to the observer schedule (here, in its “displayActions” group) that tells the global probe display manager to update.

5 Parameters and Parameter Displays

- Model parameters can be displayed and controlled via a probe display to the model swarm. The probe display is defined in the model swarm's constructor method. An `EmptyProbeMapImpl` is instantiated (as "modelProbeMap") and then assigned probes for each parameter to be displayed.
- A statement (`Globals.env.createArchivedProbeDisplay`) is required in the observer swarm to actually display the model swarm probe display.
- Users must be careful where they put creation of the probe display. It is created in the model swarm's constructor so that the display appears when the model stops with the control panel set to stopped, in the observer swarm's "buildObjects" method, so users can modify parameters (e.g., initial number of bugs) that are needed in the model's buildObjects method.
- Note that all Swarm probe displays can be expanded to a full display of all public variables and methods by right-clicking on their class name box.

6 Histogram Output

- In concept, adding a histogram is simple. It requires only instantiating one from Swarm's `EZBin` class, and putting the histogram's update statements in the observer's "_update_" method so they are executed each time step. No changes to the model are necessary.
- However, creating the `EZBin` is a bit complicated because of how Java Swarm works around the "createBegin - createEnd" paradigm of Objective-C Swarm. In Objective-C Swarm, an object is instantiated from a Swarm class (such as `EZBin`) using a "createBegin" method. This method returns the new object, to which a "createEnd" message can then be sent to complete initialization of the new object. To implement this paradigm, Java Swarm includes separate create-phase classes (which have names ending in `CImpl`, e.g., `EZBinCImpl`) and implementation-phase classes (which have names ending in `Impl`, e.g., `EZBinImpl`). If no create-phase methods are needed, the user can simply instantiate an implementation-phase object, as we have so far in `StupidModel`. But creating an `EZBin` histogram requires using the create-phase methods that tell the histogram where to get its data, provide axis labels, etc.
- Therefore, we instantiate an `EZBinCImpl` object ("bugSizeHistogramC"), which requires passing a new `EZBinImpl` object as a parameter. Then we use the `EZBinCImpl` methods to set all the histogram's parameters and tell it which selector to use to get the values to graph. Finally, the createEnd statement returns an `EZBinImpl` object ("bugSizeHistogram") ready to use. It is `bugSizeHistogram` that is actually updated and displayed.

7 Stopping the Model

- A method `checkWhetherToStop` is added to `StupidObserverSwarm` and to its schedule. This method simply calls the model swarm's method `checkToStop`, which returns true if the stopping condition is met. If so, the model is stopped by simply sending a message to the control panel. (Due to a conflict with Swarm's controller mechanisms, the method name `checkToStop` cannot be used in an observer swarm.)
- The `checkToStop` method in `StupidModelSwarm` identifies the maximum bug size from its list of bugs. Swarm has a class `Averager` (in the Analysis library) which obtains statistics such as the average, minimum, and maximum from a list, for a specified variable. Unfortunately, `Averager`'s `getMax` method was left out of version 2.2 of Java Swarm (it exists in Objective-C Swarm) so `checkToStop` must manually loop through the list of bugs and identify the largest.

8 File Output

- While writing file output is an observer function, we put file output in the model swarm, not the observer swarm. This is so file output is still produced if we eventually create a "batch swarm"—a replacement for the observer swarm that runs the model without graphical displays.
- A new method `writeFileOutput` is added to the model swarm. It is scheduled by adding a new action group (`fileoutActions`) to the model swarm schedule. This method calculates the minimum, mean, and maximum bug size; and then outputs them using a Java print statement.
- Now a Swarm `Averager` is created to calculate the minimum and mean bug size. It is instantiated in the model swarm's `buildObjects` method, which requires using both the create- and implementation-phase classes of `Averager`.
- The same output could be obtained by creating a Swarm `EZGraph`, which can output (with or without graphing) these statistics. However, `EZGraph` writes a separate output file for each variable.

9 Randomized Agent Actions

- Swarm's scheduling library provides a convenient method (`setDefaultOrder` in the `ActionGroupImpl` class) to randomize the order in which a list of agents executes a particular action, when the action is defined and put on the schedule.

10 Sorted Agent Actions

- A method `sortBugs` is added to the model swarm. It is scheduled in a new `ActionGroup` called `updateActions` at the beginning of the model swarm schedule.

- Sorting uses Swarm’s QSort class (in the SimTools library). It requires each bug to have a comparison method, which is identified in a selector that QSort uses. Due to a quirk in Java Swarm, this comparison method must be explicitly named but cannot be named “compare”, which is QSort’s default comparison method name. Therefore, the StupidBug class now has a method “compareSize”, which compares two bugs and reports a 1, 0, or -1 depending on which has the highest value of “mySize”.

11 Optimal Movement

- Only the method StupidBug.move is modified.
- Swarm’s grid space class does not have a built-in method for identifying neighbor cells, so the “move” method must create the list itself. Because Swarm’s grid space is not toroidal, the bugs must also use the “xnorm” and “ynorm” methods to identify neighbor cells on the opposite side of the space.

12 Bug Mortality and Reproduction

- The code for reproducing is simply added to the StupidBug.grow method. (Alternatively, it could have been placed in a new “reproduce” method that would be scheduled after “grow”.)
- Bugs also have a new “die” method, which compares a random number to the mortality risk to determine if the bug dies.
- The most interesting and confusing part of this version is that bugs cannot just be removed from the agent list when they die (in StupidBug.die), and new bugs cannot just be added to the agent list as soon as they are produced (in StupidBug.grow). Doing so would cause run-time errors because it alters the agent list while the scheduler is looping through it executing its createActionForEach methods. This problem occurs in any Swarm (or Repast) model that includes reproduction or death. Here we use one common technique: creating separate lists of bugs that have been born and have died during a time step (the StupidModel lists newBugList and deadBugList). A new model method addAndRemoveBugs is scheduled at the end of a time step; it removes the dead bugs from the agent list and adds the new bugs.
- The code to initialize a bug “born” via reproduction is in a new, second constructor method of StupidBug.

13 Population Abundance Graph

- This graph requires no changes to any class except StupidObserverSwarm. It uses Swarm’s EZGraph classes, and is implemented almost exactly the same way that the histogram was in version 6.

14 Random Normal Initial Size

- This version requires creating a random normal distribution (NormalDistImpl, in Swarm’s random library). The distribution is assigned the global random number generator and is parameterized using new model parameters for the bug size mean and standard deviation. The distribution is created in the model’s “buildObjects” method.
- The model swarm’s method “getBugInitSize” pulls a random value from the size distribution when called by a bug during its construction.

15 Habitat Data from File Input

- Swarm does not have a simple tool for reading input files, so this version uses the class DoubleFileGridReader written for the Repast implementation of StupidModel. This class uses Java code to read the input file and provide each set of X, Y, and food production values to the model.
- To make the space non-toroidal, the StupidBug “move” method and the constructor for newly born bugs are modified so grid cells off the edge of the space are excluded as destinations.
- The StupidBug “move” method requires randomly shuffling the list of potential destination cells before the bug selects the one to move to. Swarm’s ListShufflerImpl class is used to do so. (In StupidModel, the bugs create a new ListShuffler each time they move; it would be more efficient for them to keep one ListShuffler as an instance variable and use it each time they move.)

16 Predators

- This change requires a new class, StupidPredator. StupidPredator has two methods, “hunt” and “drawSelfOn”.
- Changes in StupidModelSwarm include a new list of predators (“predList”), a new space object (“predWorld”) to contain the predators, and a new predator action group (“predatorActions”) on the schedule.
- A display object for predators (“predDisplay”) is added to the observer swarm.