StupidModel and Extensions: A Template and Teaching Tool for Agent-based Modeling Platforms

Steve Railsback, Steve Lytinen, and Volker Grimm

20th December 2005

StupidModel will make you smart!

Abstract

This document describes StupidModel, a set of 16 simple template models designed as examples for learning how to use agent-based modeling platforms such as Repast and Swarm. Each section describes a version of StupidModel. Version 1 is the basic StupidModel, which is extremely simple. As the first exercise in a class for nonprogrammers, Version 1 can be implemented in 2-3 hours. The subsequent versions incrementally add features that are commonly used in real models. These features include model elements such as interaction among individuals and their environment, stopping rules, alternative orders in which individuals execute their actions, mortality and reproduction, file input for habitat variables, and multiple kinds of agents. Some versions simply add software tools for observing the model: probes, graphs, and file output.

The documentation for each version states its purpose—what tool or technique is illustrated by the version, describes its formulation, and provides notes on how the changes were implemented in the example software.

Available separately are our implementations of StupidModel in Mason, NetLogo, Repast, Objective-C Swarm, and Java Swarm; and notes on each of these implementations.

1 Basic StupidModel

1.1 Purpose

This is the basic StupidModel, an extremely simple individual-based model used as a starting point for learning Repast (or other IBM platforms).

1.2 Formulation

- The space is a two-dimensional grid of dimensions 100 x 100. The space is toroidal, meaning that if bugs move off one edge of the grid they appear on the opposite edge.

- 100 bug agents are created. They have one behavior: moving to a randomly chosen grid location within +/- 4 cells of their current location, in both the X and Y directions. If there already is a bug at the location (including the moving bug itself—bugs are not allowed to stay at their current location unless none of the neighborhood cells are vacant), then another new location is chosen. This action is executed once per time step.
- The bugs are displayed on the space. Bugs are drawn as red circles. The display is updated at the end of each time step.
- Instead of specifying which random number generation algorithm to use, the default generator for each platform is used.

2 Bug Growth

2.1 Purpose

Illustrate adding instance variables and methods to the agents.

2.2 Formulation

- Add a second bug action, grow. Each time step, a bug grows by a fixed amount, 1.0. So bugs need an instance variable for their size, which is initialized to 1.0. This action is scheduled after the move action.
- The bugs' color on the display is shaded to reflect their size. Bug colors shade from white when size is zero to red when size is 10 or greater.

3 Habitat Cells and Resource

3.1 Purpose

Show how to create cell objects that represent habitat and spatial resources. Illustrate how agents and habitat cells interact.

3.2 Formulation

- A new class, HabitatCell, is added. Habitat cell objects have instance variables for their food availability and maximum food production rate. Cells also have a variable for the bug at their location.
- The grid space object now holds habitat cells, not bugs.
- Food availability is initialized to 0.0, and maximum food production rate is initialized to 0.01.

- Each time step, food availability is increased by food production. Food production is a random floating point number between zero and the maximum food production.
- Food production is scheduled before agent actions.
- Bug growth is modified so growth equals food consumption. Food consumption is equal to the minimum of (a) the bug's maximum consumption rate (set to 1.0) and (b) the bug's cell's food availability.
- The food consumed by each bug is subtracted from the food availability of its cell.

4 Cell and Bug Probes

4.1 Purpose

Show how to make model objects probeable from the display.

4.2 Formulation

Make the bugs, and the cells, so they can be probed via mouse clicks on the display.

5 Parameters and Parameter Displays

5.1 Purpose

Show how to define variables as parameters, and how to put parameters in the parameter settings window that appears at startup.

5.2 Formulation

Make these variables into parameters that can be accessed through the settings window:

- Initial number of bugs (a model parameter)
- The maximum daily food consumption (a bug parameter), and
- The maximum food production (a cell parameter).

6 Histogram Output

6.1 Purpose

Illustrate how to add graphs to the display. Provide the ability to see the size distribution of the agents.

6.2 Formulation

Add a histogram display showing the number of agents in each size class. (It works reasonably well to use 10 bins, set minimum to zero, and set maximum to 10.)

7 Stopping the Model

7.1 Purpose

Show how to cause a model to stop itself upon a certain condition. Show how to "clean up" when a model stops.

7.2 Formulation

- The model stops when the largest bug reaches a size of 100.
- Close the graphic windows (and do any other cleanup steps) when the program stops.

8 File Output

8.1 Purpose

Show how to write results to an output file. Illustrate how to iterate over a list.

8.2 Formulation

Each time step, write the minimum, mean, and maximum bug size on one line of an output file.

9 Randomized Agent Actions

9.1 Purpose

Show how to randomize the order in which agents execute an action.

9.2 Formulation

The bugs' move action is altered so that the order in which bugs execute the action is shuffled each time step.

10 Sorted Agent Actions

10.1 Purpose

Show how to sort a list of agents, and cause an agent action to be executed in size order.

10.2 Formulation

- The list of bugs is sorted by descending size order at the start of each time step.
- The bugs' move action is un-randomized so it is executed in descending size order.

11 Optimal Movement

11.1 Purpose

Show how agents can identify and rank neighbor cells. Illustrate how to iterate over a list.

11.2 Formulation

- In its move method, a bug identifies a list of all cells that are within a distance of 4 grids but do not have another bug in them. (The bug's current cell is included on this list.)
- The bug iterates over the list and identifies the cell with highest food availability. The bug then moves to that cell.

12 Bug Mortality and Reproduction

12.1 Purpose

Show how to "kill" and drop objects from a model, and how to create new objects during a run.

12.2 Formulation

- When a bug's size reaches 10, it reproduces by splitting into 5 new bugs. Each new bug has an initial size of 0.0, and the old bug disappears.
- New bugs are placed at the first empty location randomly selected within +/- 3 cells of their parent's last location. If no location is identified within 5 random draws, then the new bug dies.
- A new bug parameter "survivalProbability" is initialized to 0.95. Each time step, each bug draws a uniform random number, and if it is greater than survivalProbability, the bug dies and is dropped.
- This mortality action is scheduled after the bug moves and grows.
- The model stopping rule is changed: the model stops after 1000 time steps have been executed or when the number of bugs reaches zero.

13 Population Abundance Graph

13.1 Purpose

Show how to add a simple time series graph to a model. This graph is important for understanding results now that reproduction and mortality change the abundance of bugs.

13.2 Formulation

No change is made to the model formulation. A graph is added to display the number of bugs alive at each time step.

14 Random Normal Initial Size

14.1 Purpose

Illustrate use of random number distributions. A common use of them is to induce variability among initial individuals.

14.2 Formulation

- Two new model parameters are added, and put on the parameter settings window: initialBugSizeMean and initialBugSizeSD. Values of these parameters are 0.1 and 0.03.
- Instead of initializing bug sizes to 1.0 (Sect. 2.2), sizes are drawn from a normal distribution defined by initialBugSizeMean and initialBugSizeSD. The initial size of bugs produced via reproduction is 0.0.
- Negative values are very likely to be drawn from normal distributions such as the one used here. To avoid them, a check is introduced to limit initial bug size to a minimum of zero.

15 Habitat Data from File Input

15.1 Purpose

Show how to read spatial data in from a file.

15.2 Formulation

- Instead of assuming the space size and assuming cell food production is random (Sect. 3.2), food production rates are read in from a file. The file also determines the space size.

- The file contains one line per cell, with (a) X coordinate, (b) Y coordinate, and (c) food production rate.
- Food production in a cell is now equal to the production rate read in from the file, and is no longer random.
- Now, because we are representing real habitat with real data, it no longer makes sense for the space to be toroidal. So the space objects and movement-related methods must be modified so bugs cannot move off the edge of their space.
- The input file is Stupid_Cell.Data. It has X, Y, and food production data for a grid space. X ranges from 0 to 250; Y ranges from 0 to 112. The file starts with three lines of header information that is ignored by the model.
- The cells are now displayed and colored to indicate their current food availability. Cell colors scale from black when cell food availability is zero to green when food availability is 0.5 or higher.
- A change to the bug move method is required to avoid a very strong artifact now that cell food production is no longer random. Near the start of a simulation, many cells will have exactly the same food availability, so a bug simply would move to the first cell on its list of neighbor cells. This is always the top-left cell among the neighbors, so bugs move constantly up and left if all the cells available to them have the same food availability. This artifact is removed by randomly shuffling the list of available cells before the bug loops through it to identify the best.

16 Predators

16.1 Purpose

Show how to create multiple classes of agents that interact.

16.2 Formulation

- 200 predator objects are intialized and randomly distributed as the bugs are. A cell can contain a predator as well as a bug. Predators are created after bugs are.
- Predators have one method: hunt. First, a predator looks through a shuffled list of its immediately neighboring cells (including its own cell). As soon as the predator finds a bug in one of these cells it "kills" the bug and moves into the cell. (However, if the cell already contains a predator, the hunting predator simply quits and remains at its current location.) If these cells contain no bugs, the predator moves randomly to one of them.
- Predator hunting is scheduled after all the bug actions.