Full Documentation of GAMA 1.3

Provided by the GAMA development team

http://code.google.com/p/gama-platform/

Table of Contents

Introduction to GAMA 1.3 .... **19**
Information 20
News 21
Display of simulations 21
GIS integration and geometric tools 21
Additions to the language 22
Bugs and regressions (from 1.2) fixes 22
How to convert models from GAMA 1.1 and GAMA 1.2 to GAMA 1.3 22
First Steps with GAMA 1.3 .... **24**
Modeling Guide **34**
Structure of a model **35**
include 35
global 35
entities 36
environment 38
output 39
batch 40
Species Definition **41**
Introduction 41
Species declaration 41
Skills: behavioral plug-ins 43
Aspects: display properties 44

Parent: inheritance of species....45
Control: behavioral architecture....46
Base: Java foundation....46
Behaviors....47
Common behaviors....47
reflex....47
init....48
EMF-based behaviors....48
task....48
FSM-based behaviors....49
state....49
Types....51
Built-in types....51
bool....51
int....51
float....51
point....52
string....52
list....52
map....53
rgb....53
matrix....54
agent....54
species....54
Species as types....55
Defining custom types....56
Commands....58
General syntax....58
ask....58
Attributes....58
Definition....58

Notice....... 60
create....... 60
Attributes....... 60
Definition....... 60
do....... 62
Attributes....... 62
Enclosed tags....... 62
Definition....... 62
let....... 63
Attributes....... 63
Definition....... 63
set....... 63
Attributes....... 63
Definition....... 64
if....... 64
Attributes....... 64
Enclosed tags....... 64
Definition....... 65
loop....... 65
Attributes....... 65
Definition....... 66
return....... 67
Attributes....... 67
Definition....... 67
save....... 68
Attributes....... 68
Definition....... 68
Operators....... 69
Definition....... 69
Operators by categories....... 70
Arithmetic operators....... 70

Comparison operators.......... 70
Boolean operators.......... 70
Casting operators.......... 70
List-related operators.......... 70
String-related operators.......... 70
Location-related operators.......... 70
Agent-related operators.......... 71
Unary operators.......... 71
!.......... 71
acos.......... 71
abs.......... 71
agent.......... 72
any.......... 72
asin.......... 72
atan.......... 72
one_of.......... 73
bool.......... 73
closest_point_to.......... 73
cos.......... 73
empty.......... 74
exp.......... 74
first.......... 74
flip.......... 75
float.......... 75
int.......... 75
intensity (deprecated).......... 76
last.......... 76
length.......... 76
list.......... 76
ln.......... 77
max.......... 77

mean........ 77
min........ 78
mul........ 78
next_point_towards........ 78
reverse........ 78
rgb........ 79
rnd........ 79
round........ 80
shuffle........ 80
sin........ 80
species........ 80
string........ 81
sum........ 81
tan........ 81
towards........ 82
Binary operators........ 82
=........ 82
!=........ 82
<>........ 83
> >= < <=........ 83
@........ 83
at........ 83
+........ 84
-........ 85
*........ 86
/........ 86
//........ 87
div........ 87
%........ 87
mod........ 87
&........ 87

and.....87
|.....88
or.....88
^.....88
accumulate.....88
among.....88
collect.....89
copy_between.....89
count.....89
distance_to.....90
first_with.....90
in.....90
index_of.....91
last_index_of.....91
max_of.....91
min_of.....91
neighbours_at.....91
of_species.....92
sort_by.....92
starts_with.....92
split_using.....93
tokenize.....93
where.....93
select.....93
with_max_of.....93
with_min_of.....94
with_precision.....94
Ternary operators.....94
? :.....94
Variables.....96
Declaration.....96

base....... 96
init....... 96
const....... 97
value....... 97
Special attributes....... 98
parameter....... 98
max, min....... 98
of....... 99
Naming variables....... 99
Reserved Keywords....... 99
Naming conventions....... 100
Accessing variables....... 100
Direct access....... 100
Remote access....... 100
Keywords....... 103
constants....... 103
nil....... 103
true, false....... 103
pseudo-variables....... 103
self....... 103
myself....... 104
context....... 104
each....... 104
units....... 104
length....... 105
time....... 105
mass....... 105
surface....... 106
volume....... 106
Skills....... 107
Overview....... 107

situated.. 108
variables..108
actions.. 109
moving.. 127
variables..127
actions.. 127
carrying.. 129
variables..129
actions.. 129
communicating 130
variables..130
actions.. 131
data types...132
exploring.......................................133
variables..133
actions.. 133
Built-in Agents.. 135
Introduction.................................... 135
cluster_builder................................. 135
actions.. 136
multicriteria_analyzer..........................140
actions.. 140
random_builder................................143
actions.. 143
Built-in Items... 145
Introduction.................................... 145
name... 145
Note... 145
Built-in actions.................................146
write.. 146
debug.. 146

error......146
tell......147
Global built-in variables......147
time......147
step......147
seed......148
places......148
agents......148
Global built-in actions......149
halt......149
pause......149
diffuse......149
Batch......151
Definition......151
The batch element......151
The param elements......152
The method element......153
Batch Methods......153
Exhaustive exploration of the parameter space......153
Hill Climbing......154
Simulated Annealing......155
Tabu Search......156
Reactive Tabu Search......156
Genetic Algorithm......157
How to write a batch method......158
Introduction......158
Step 1......159
Step 2......159
Training sessions GAMA 1.3......160
Training session GAMA 1.3 for epidemiology (St Louis)......161

Context.. 161
List of presentations........................ 161
Training session GAMA 1.3 (Bondy)....................................... 162
Context.. 162
List of presentations........................ 162
Models.. 162
Tutorial 1: The Stupid model...................................163
Stupid Model GAMA 1.3.. 164
Introduction.................................. 164
Preliminary notes............................ 164
Stupid model: models list.................... 164
Basic stupidModel.. 166
Purpose..166
Formulation.................................. 166
Models.. 166
Model 0 : the minimal set..166
Model 1.1 : the environment and display output...................... 167
Model 1.2 : Defining the agents...................................... 167
Model 1.3 : Instantiating bugs...168
Nota bene..168
Complete model 1.............................. 169
Bug growth...170
Purpose..170
Formulation.................................. 170
Model.. 170
Growing.. 170
Variable automatic update..170
Shading color.. 171
Nota bene..171
Complete model................................171

Nota bene......172
Habitat cells and resource......173
Purpose......173
Formulation......173
Models......173
Giving a behaviour to the cells......173
nota bene......174
Increasing cell's food......174
Bug growth......174
Nota bene......175
Complete model......175
Cell and bug probes......177
Purpose......177
Formulation......177
Models......177
Complete model......178
Parameters and parameters displays......180
Purpose......180
Formulation......180
Models......180
Complete model......181
Nota bene......182
Histogram output......183
Purpose......183
Formulation......183
Models......183
Adding the pie chart......183
Nota bene......184
Complete model......184
Nota bene......186
Stopping the model......187

Purpose...... 187
Formulation...... 187
Models...... 187
Nota bene...... 188
Complete model...... 188
File output...... 191
Purpose...... 191
Formulation...... 191
Models...... 191
Complete model...... 192
Randomized agent actions...... 194
Purpose...... 194
Formulation...... 194
Models...... 194
Sorted agent actions...... 195
Purpose...... 195
Formulation...... 195
Models...... 195
Complete model...... 195
Optimal movement...... 198
Purpose...... 198
Formulation...... 198
Models...... 198
Nota Bene...... 199
Complete model...... 199
Bug mortality and reproduction...... 202
Purpose...... 202
Formulation...... 202
Models...... 202
Multiplying...... 202
Introducing survivability...... 203

Changing the stop condition.. 203
Complete model................................203
Population abundance graph.. 207
Purpose...207
Formulation.................................... 207
Models.. 207
Complete model................................208
Random normal initial size.. 211
Purpose...211
Formulation.................................... 211
Models.. 211
Adding new parameters...211
Normal randomization of bug size...................................... 211
Complete model................................212
Habitat data from file input.. 215
Purpose...215
Formulation.................................... 215
Models.. 216
Reading data from a file.. 216
Cell color...216
Modification of the bug behaviour.......................................217
Complete model................................217
Predators.. 221
Purpose...221
Formulation.................................... 221
Models.. 221
Instanciating predators.. 221
Defining predators...221
Scheduling Predators...222
Visualization.. 222
Complete model................................223

Tutorial 2: GIS tutorial.. 227
Introduction to the tutorial...228
Introduction...................................... 228
Road traffic in a city: model list...............229
1. Loading of GIS data (buildings and roads).............................230
Purpose...230
Formulation...................................... 230
Model... 230
Entities...230
Parameters.. 231
Agentification of GIS data.. 231
Environment... 232
Display..232
Complete model.................................232
2. Integration of people agents.. 234
Purpose...234
Formulation...................................... 234
Model... 234
Entities...234
Parameter...235
Creation and placement of the people agents.......................... 235
Display..235
Complete model.................................236
3. Movement of the people agents...238
Purpose...238
Formulation...................................... 238
Model... 239
People agents..239
Parameters.. 240
Initialisation... 241
Complete model.................................243

4. Definition of weight for the road network......247
Purpose......247
Formulation...... 247
Model...... 247
Road agent......247
Weigthed road network...... 248
Complete model......248
5. Dynamic update of the road network...... 252
Purpose......252
Formulation...... 252
Model...... 252
Global section......252
People agents......253
Complete model......254
6. Definition of a chart display......258
Purpose......258
Formulation...... 258
Model...... 258
Chart display...... 258
Complete model......259
7. Automatic repair of roads...... 264
Purpose......264
Formulation...... 264
Model...... 264
Paremeters...... 264
Road repairing...... 264
Complete model......265
8. Complex repair of roads...... 270
Purpose......270
Formulation...... 270
Model...... 270

Paremeters........ 270
Road repairing........ 271
Complete model........272

Tutorial 3: Robot exploration........ 277

Introduction to the tutorial........278
Introduction........ 278
Robot exploration: model list........279
1. Loading of GIS data (background and obstacles)........ 280
Purpose........280
Formulation........ 280
Model........ 280
Entities........280
Parameters........ 281
Agentification of GIS data........ 281
Environment........ 281
Display........282
Complete model........282
2. Computation of the unknown environment........ 284
Purpose........284
Formulation........ 284
Model........ 284
Geometry union........284
Unknown environment........285
Complete model........285
3. Integration of the robot agent........ 287
Purpose........287
Formulation........ 287
Model........ 287
Parameter........287
Robot agent........ 288

Creation and placement of the robot agent.....288
Display.....289
Complete model.....289
4. Dynamic building of the known environment.....291
Purpose.....291
Formulation.....291
Model.....291
Parameters.....291
Known_area agent.....291
Perception of the robot agent.....292
Display.....293
Complete model.....293
5. Use of a pathfinder for the robot movement.....296
Purpose.....296
Formulation.....296
Model.....296
Triangle agents.....296
Triangle agent creation and graph computation.....297
Movement of the robot agent.....297
Complete model.....298
6. Frontier-based approach.....301
Purpose.....301
Formulation.....301
Model.....301
Parameters.....301
Candidate agent.....302
Computation of the candidate agent.....302
Display.....303
Complete model.....304
7. Dynamic update of the pathfinder.....307
Purpose.....307

Formulation...... 307
Model...... 307
Triangle agents...... 307
Weighting of the graph...... 307
Complete model......308
8. Integration of a chart in the display......312
Purpose......312
Formulation...... 312
Model...... 312
Global variable......312
Stopping criterion......313
Complex display......313
Complete model......314
Tutorial 4: Batch tutorial......318
Batch Tutorial...... 319
Introduction...... 319
Principle of the parameter space exploration......319
Parameter space exploration in GAMA...... 319

Introduction to GAMA 1.3

On this page: Links to the documentation:

- First Steps
  - Modeling Guide
    - Structure of a model
    - Species Definition
    - Behaviors
    - Data Types
    - Commands
    - Operators
    - Variables
    - Keywords
    - Skills
    - Built-in Agents
    - Built-in Items
    - Batch
  - Training session GAMA 1.3 for epidemiology (St Louis)
  - Training session GAMA 1.3 (Bondy)
  - Tutorials
    - Stupid Model GAMA 1.3
      - 1. Basic stupidModel
      - 2. Bug growth
      - 3. Habitat cells and ressource
      - 4. Cell and bug probes
      - 5. Parameters and parameters displays
      - 6. Histogram output
      - 7. Stopping the model
      - 8. File output
      - 9. Randomized agent actions
      - 10. Sorted agent actions
      - 11. Optimal movement
      - 12. Bug mortality and reproduction
      - 13. Population abundance graph
      - 14. Random normal initial size
      - 15. Habitat data from file input
      - 16. Predators
    - [GISTutorials13 GIS tutorials]
      - Road traffic models

    - 1. Loading of GIS data (buildings and roads)
    - 2. Integration of people agents
    - 3. Movement of the people agents
    - 4. Definition of weight for the road network
    - 5. Dynamic update of the road network
    - 6. Definition of a chart display
    - 7. Automatic repair of roads
    - 8. Complex repair of roads
  - Robot exploration
    - 1. Loading of GIS data (background and obstacles)
    - 2. Computation of the unkwnow environment
    - 3. Integration of the robot agent
    - 4. Dynamic building of the known environment
    - 5. Use of a pathfinder for the robot movement
    - 6. Frontier-based approach
    - 7. Dynamic update of the pathfinder
    - 8. Integration of a chart in the display
  - Batch tutorial

Information

Release date: August 2010 [Splash13] < http://gama-platform.googlecode.com/files/splash1-3.png> The version 1.3 add numerous operators to manipulate GIS data. Moreover, it integrates new features like the possibility to define custom display and to define multi-level models. It allows to use clustering and decision-making methods. At last, it improves greatly the performance of the platform. Key points :

- Important improvement of the performance of the platform
- Improvement of the simulation display
- Enrichment of the spatial operators/actions
- Integration of multi-level models
- Integration of clustering algorithms
- Integration of decision-making algorithms

News

Display of simulations

- Disparition of the "visible" skill, replaced by a specific "aspect" section in species, where the modeler can precisely (and programmatically) describe how to draw the agents, using a new "draw" command (able to draw shapes, geometries, text, etc. at any position — either relative to the agent or absolute — , with any color, with or without outline, etc.). A good example is in "ants_random.xml" or "fire_tutorial6".
- Displays can now be organized in the display views by precisely describing their position, size, either relatively to the size of the view (using positive or negative percentages) or using absolute positions and sizes expressed in pixels. (a good example in schelling_google_maps.xml)
- Charts can now be displayed in the same view as the other elements, allowing for nice demo screens (with transparent charts on the environment, etc.). For older models, the specific [ChartView] is gone, replaced (by default) by a separate display view containing the chart. (a good example in schelling_google_maps.xml)
- Grids can now optionally draw their horizontal and vertical lines (by adding the "lines" attribute with a color argument).
- A display view is not restricted to one background image, but can now load and display any images (background, icons, etc.) with the addition of several "image" elements.
- As before, all of these elements can be drawn with an individual transparency.
- As before, as well, all of these attributes are evaluated dynamically during the simulation, meaning that elements can change their position, size, transparency, contents, etc. during the run of a simulation.

GIS integration and geometric tools

- The loading of GIS shape files is now much more robust than before, and individual attributes can be read without efforts (thanks to the new "create" command).
- The work on the pathfinder allow to provide a pathfinder for agents, not only for networks of roads (as it is the case in the 1.2), but for any arbitrary combination of agent geometries ! (including but not restricted to, GIS agents). This means, for example, being able to tell an agent to move "on roads and buildings, avoiding rivers and lakes" : an optimal path will be built automatically from this combination, and even dynamically !! Of course, as the operations involved are quite complex, we will probably provide several options : a quick-and-dirty dynamic solution based on grid approximations, an exact (but slow) one based on triangulations and graphs, etc.

Additions to the language

- Beside the new "aspect" and "draw" commands evoked above, agents are now provided with new commands :
  - "create" is now used for every operation when agents are created : from a GIS file, from other agents (for creating "group" agents), from one agent based on a discretization of its geometry, from one species, etc. It defines a new tag "with" that allows to initialize the attributes of the created agents by using a simple map [attribute1::value1, attribute2::value2…] as its value.
  - "enable" and "disable" allow to control individual behaviors (reflexes, states, tasks). For instance, it can be convenient, when agents are "merged" into a group, that this group disables some of their behaviors. As a consequence, disabled behaviors are no longer executed until they are re-enabled.
  - "save" allows to create new shape files from a species of agents.
- In expressions, actions and primitives declared on a species can now be used as functions ! The syntax is rather straigthforward : agent name_of_the_action map_of_parameters. It returns what the action specifies in its "return" statement (or what the Java primitive specifies in "setReturn(…)"). The map of parameters follows the syntax of "with" for "create" : [param1::value1, param2::value2…] . It is now much easier, then, to declare functions reused in many places of the code, or to use some functions provided, for example, for the geometry, i.e.:

Bugs and regressions (from 1.2) fixes

- A serious regression on the inheritance of species has been uncovered recently and should be now fixed.
- Memory leaks still occur (i.e. the patch for the 1.2 version has still not be entirely ported to the 1.3), but most of them have been patched.
- more smaller bug fixes (too numerous to mention).

How to convert models from GAMA 1.1 and GAMA 1.2 to GAMA 1.3

Backward compatibility with existing models is almost perfect, except for the "background" tag in the "display" section, which now designates the background color rather than a background image. The "built-in" agent attributes "color", "shape", "size", "information" have also disappeared, which means that if they are not redefined in a

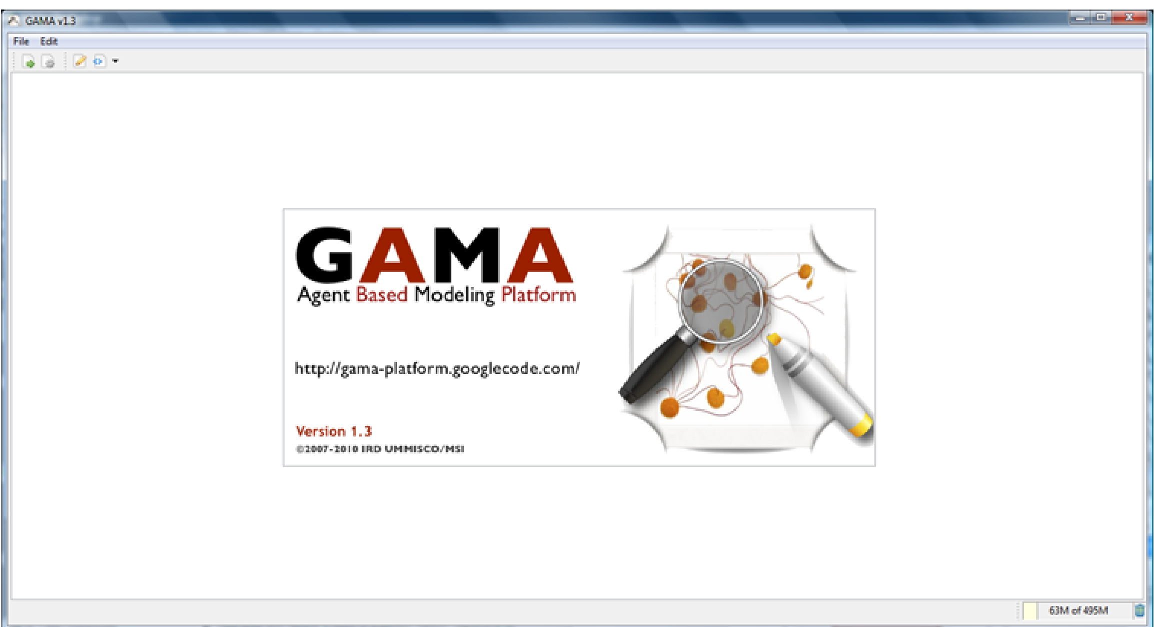
species, they cannot be used implicitly. When no "aspect" is defined in a species, it is assumed that the default geometry for agents is used (i.e. a point or, if a "shape" variable is still there, this shape). In that case, or if a "draw" command does not specify a color, the system looks for a "color" variable and gives a default color if nothing is found. Same thing for the size.

First Steps with GAMA 1.3

The purpose of this section is to show you how to open and "play" with a simple GAMA model. First of all, please run the GAMA platform.

- This is the graphical user interface of GAMA after starting up:

<a href="



" title="GAMA_UI_1"><img alt="GAMA_UI_1" src="

" />

- In GAMA, the models are written in GAML. You can open a model with the GAMA from "running" perspective or an "editing" one.

To open a model from a "running" perspective, you can:

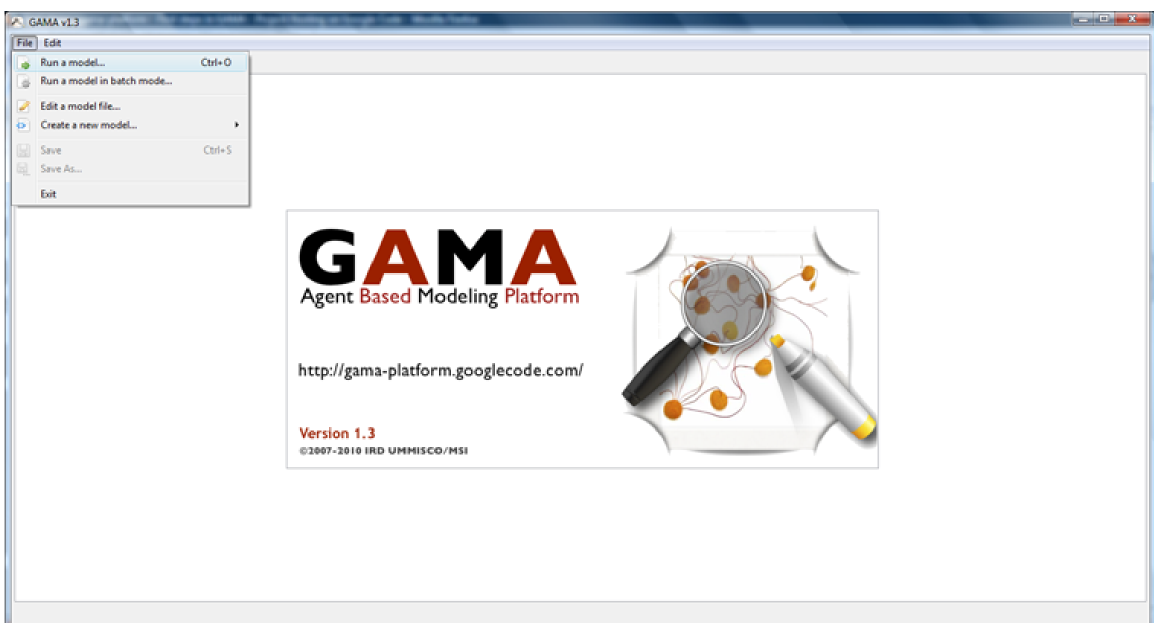
- Click on the “Run a model...” button:

<a href="

" title="GAMA_UI_3">

- Or click on the menu “File/Run a model...”:

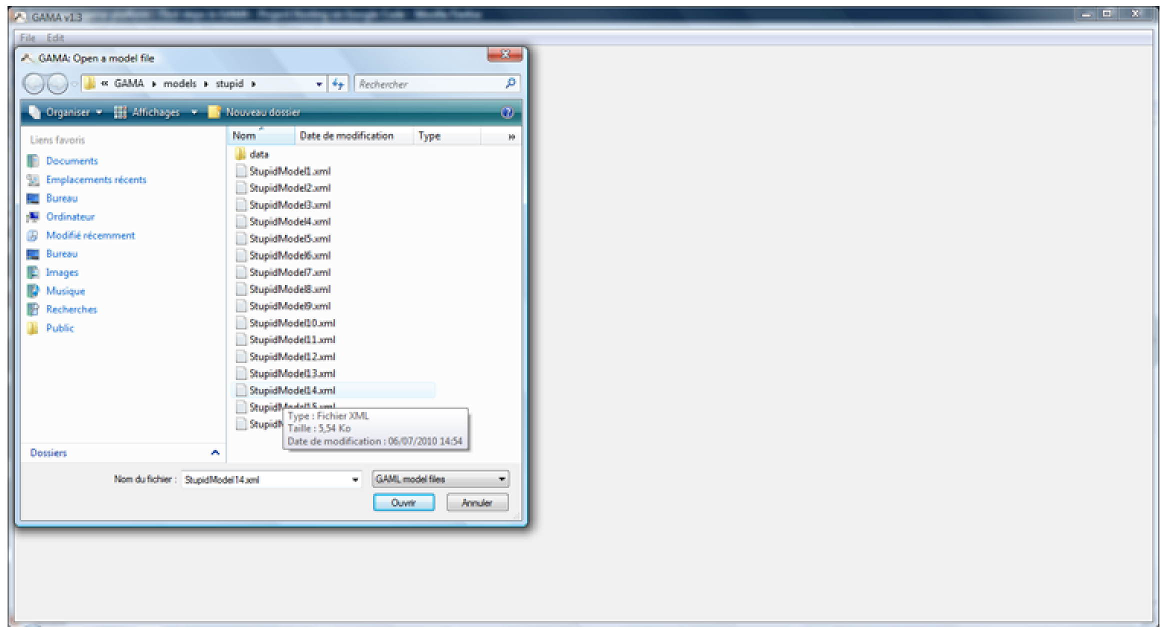
<a href="



" title="GAMA_UI_20">

- Select the [Stupid_Tutorial_Model14 “StupidModel14.xml”] file in "models/stupid:

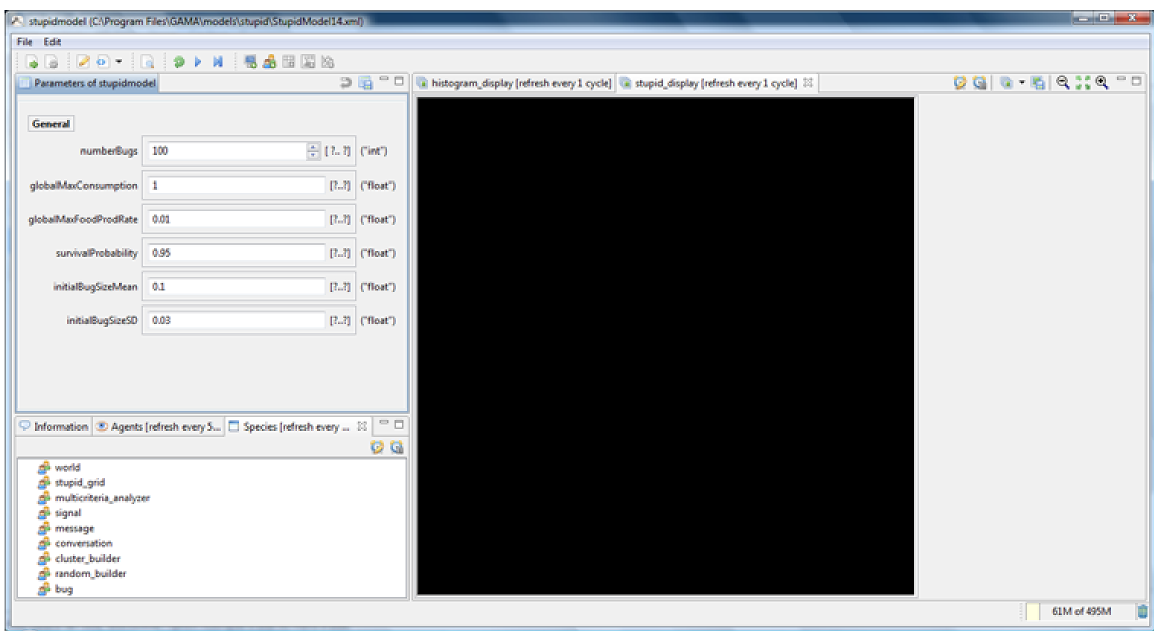
<a href="



" title="GAMA_UI_21">

- You receive:

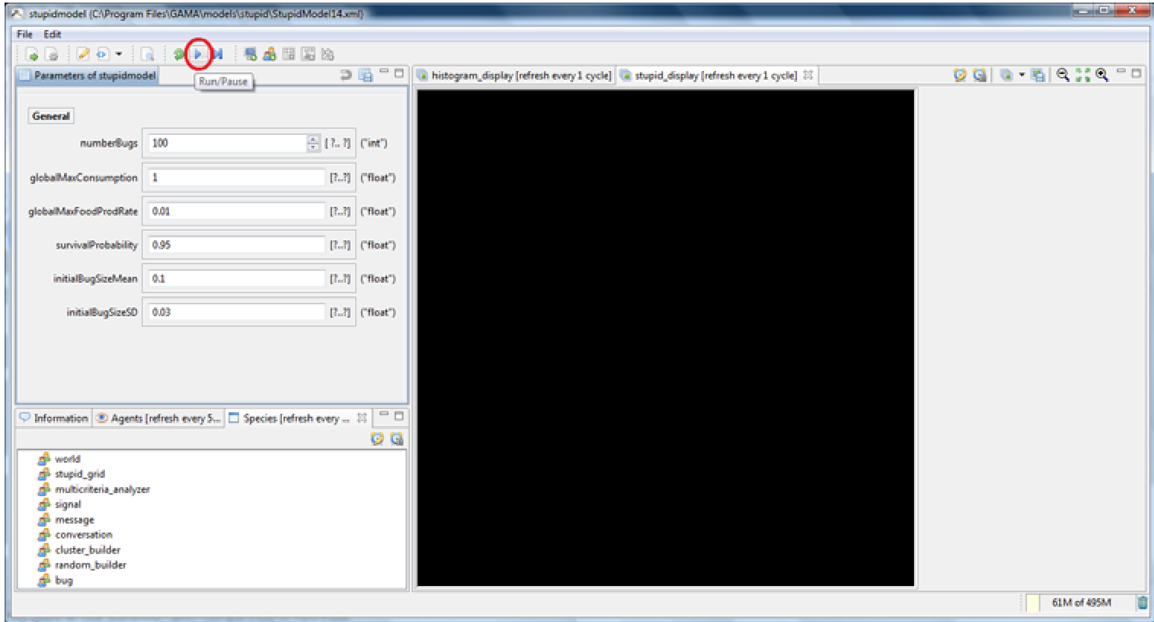
<a href="



" title="GAMA_UI_4">

- To run/pause the simulation, you have to click on the “Run/Pause” button:

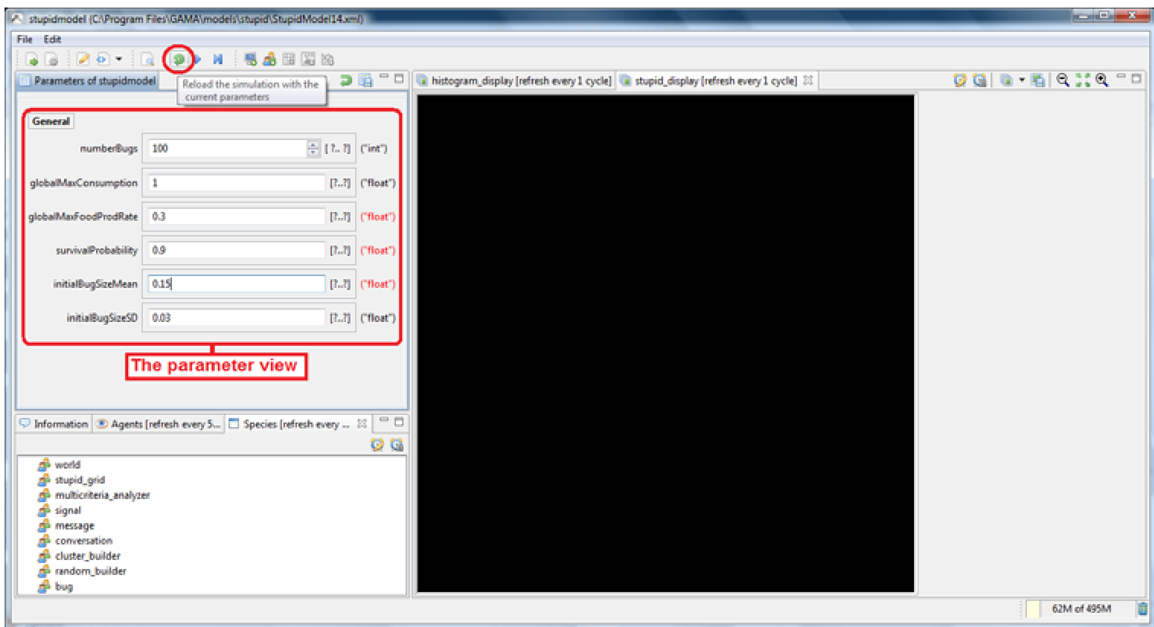
<a href="



" title="GAMA_UI_5">

- To reload the model, you have to click on the “Reload the simulation with the current parameters” button:

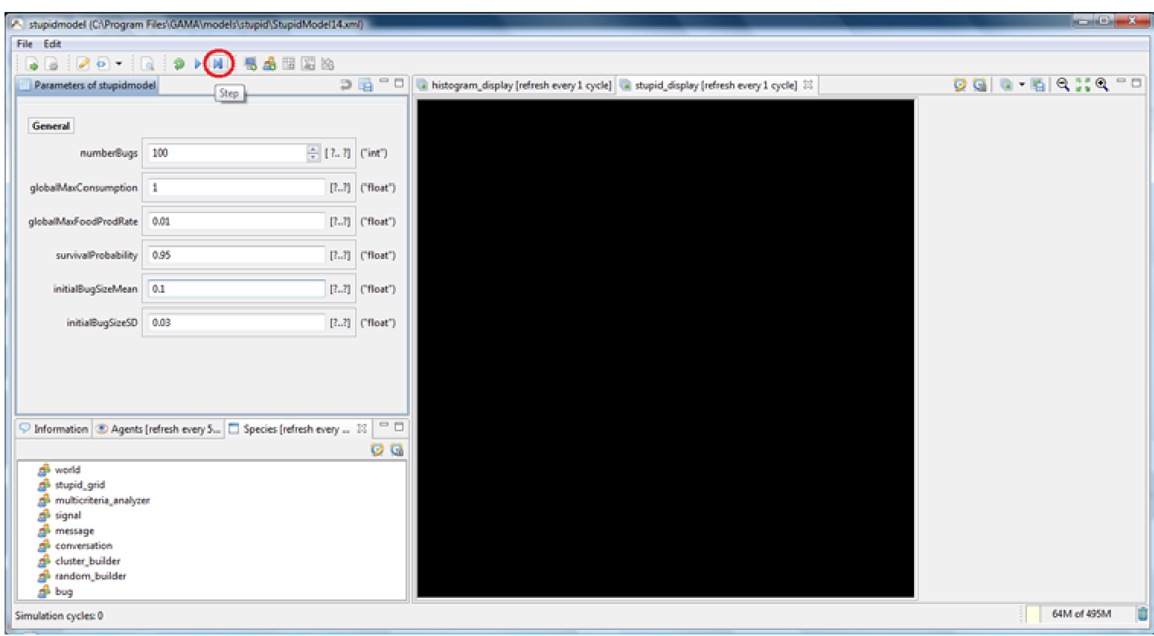
<a href="



" title="GAMA_UI_6">

- If you want to run the simulation step by step, you can click on the “Step” button:

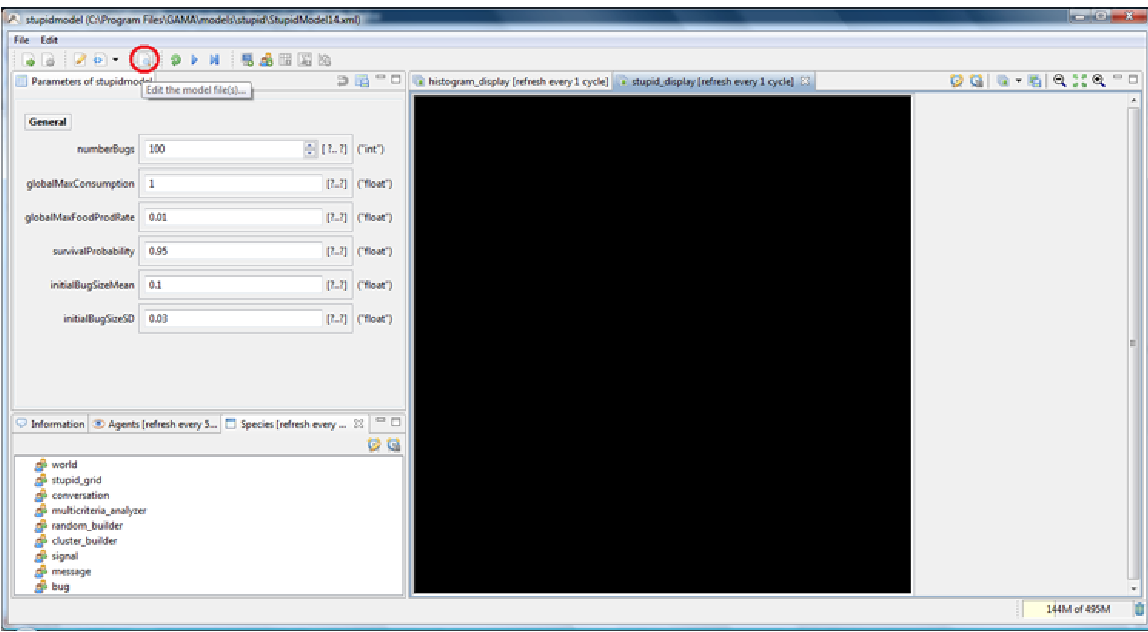
<a href="



" title="GAMA_UI_7">

- If you want to edit the model file, you can click on the “Edit the model file(s)” button:

<a href="



" title="GAMA_UI_10">

- You receive:

<a href="

" title="GAMA_UI_11">

To open a model from a "editing" perspective, you can:

- Click on the “Edit a model file...” button:

<a href="

" title="GAMA_UI_12">

- Or click on the menu “File/Edit a model file...”:

<a href="

" title="GAMA_UI_13">

Modeling Guide

Structure of a model

<wiki:toc max_depth="3" />
 The XML files that compose a model are structured in several sections.

include

This section allows to load another model file before loading the current file. This file can contain anything (whole definition of a model, definition of a species, of an environment, of global variables, etc.) as long as it respects the common structure of GAML models. Note that as many files as needed can be included. Example:

```
<include file="../include/schelling_common.xml" />
<include file="data_global.xml" />
```

global

This "global" section defines the "world" agent, a special agent of a GAMA model. We can define variables and behaviours for the "world" agent. Variables of "world" agent are global variables thus can refered by agents of other species or other places in the model source code. Example:

```
<global>
    <var type="int" name="time" init="0" max="10000" />
    <var type="int" name="step" init="1s" const="true" />
    <var type="float" name="seed" init="675" const="true" />
    <var type="bool" name="use_icons" init="false" parameter="true"/>
    <var type="bool" name="display_state" init="true" parameter="true"/>
    <var type="float" name="evaporation_rate" init="0.1" min="0" max="1"
parameter="true" />
    <var type="float" name="diffusion_rate" init="0.5" min="0" max="1"
parameter="true" />
    <var type="bool" name="fast_draw" init="false" parameter="true" />
    <var type="int" name="gridsize" init="75" const="true"/>
    <var type="int" name="ants_number" init="gridsize + 25" parameter="true"
min="1" max="2000"/>
    <var type="point" name="center" init="{int (gridsize / 2), int (gridsize /
2)}" const="true" />
    <var type="string" name="file" init="'../images/environment' + (string
gridsize) + 'x' + (string gridsize) + '.pgm'" const="true" />
```

```
    <var type="ant_grid" name="nest_place" />
    <var type="rgb" name="black" init="rgb 'black'" const="true" />
    <var type="rgb" name="blue" init="rgb 'blue'" const="true" />
    <var type="rgb" name="green" init="rgb 'green'" const="true" />
    <var type="rgb" name="white" init="rgb 'white'" const="true" />
    <var type="rgb" name="FF00FF" init="rgb 'gray'" const="true" />
    <var type="rgb" name="C00CC00" init="rgb '#00CC00'" const="true" />
    <var type="rgb" name="C009900" init="rgb '#009900'" const="true" />
    <var type="rgb" name="C005500" init="rgb '#005500'" const="true" />
    <var type="rgb" name="yellow" init="rgb 'yellow'" const="true" />
    <var type="rgb" name="red" init="rgb 'red'" const="true" />
    <var type="rgb" name="orange" init="rgb 'orange'" const="true" />
    <var type="string" name="ant_shape_empty" init="use_icons?  '../icons/
ant.png': 'dot' " const="true" />
    <var type="string" name="ant_shape_full" init="use_icons ?  '../icons/
full_ant.png' :'circle' " const="true" />
    <reflex>
        <ask target="places" as="ant_grid">
            <if condition="true">
                <set var="color" value="green"/>
            </if>
        </ask>
    </reflex>
    <reflex>
        <do action="disffuse">
            <arg name="var" value="'road'"/>
            <arg name="proportion" value="diffusion_rate"/>
            <arg name="environment" value="'ant_grid'"/>
        </do>
    </reflex>
</global>
```

entities

Definitions of species are placed int this section. Example:

```
<entities>
    <species name="ant" skills=" moving, visible" control="fsm">
        <var type="rgb" name="color" init="'orange'" const="true" />
        <var type="ant_grid" name="place" value="ant_grid location" />
        <var type="string" name="shape" init="ant_shape_empty" />
        <var type="string" name="information" value="state" />
        <var type="float" name="speed" init="1m/s" const="true" />
        <var type="float" name="size" init="1.0" const="true" />
        <var type="bool" name="hasFood" init="false" />

        <action name="pick">
            <set var="shape" value="ant_shape_full" />
```

```
            <set var="hasFood" value="true" />
            <set var="place.food" value="place.food - 1" />
        </action>

          <action name="drop">
              <set var="hasFood" value="false" />
              <set var="shape" value="ant_shape_empty" />
              <set var="heading" value="heading - 180" />
          </action>

          <action name="choose_best_place">
              <let var="list_places" value="place.neighbours of_species
ant_grid" />
              <if condition="(list_places count (each.food &gt; 0)) &gt; 0 ">
                  <return value="point (list_places first_with (each.food &gt;
0))" />
                  <else>
                      <let var="min_nest" value=" (list_places min_of
(each.nest))" />
                      <set var="list_places" value="list_places sort
((each.nest = min_nest) ? each.road :  0.0)" />
                      <return value="point last list_places" />
                  </else>
            </if >
        </action>
        <state name="init" initial="true">
            <if condition="display_state">
                <do action="display_information">
                    <arg name="color" value="yellow"/>
                </do>
            </if>
            <transition to="wandering" when="true"/>
        </state>
        <state name="wandering">
            <do action="wander">
              <arg name="amplitude" value="120" />
            </do>
            <transition to="carryingFood" when="place.food &gt; 0">
                <do action="pick"/>
            </transition>
            <transition to="followingRoad" when="place.road &gt; 0.05"/>
        </state>

        <state name="carryingFood">
            <set var="place.road" value="place.road+120" />
            <do action="goto">
                <arg name="target" value="nest_place" />
            </do>
            <transition to="wandering" when="place.isNestLocation" >
                <do action="drop" />
            </transition>
```

```
        </state>

        <state name="followingRoad">
            <do action="choose_best_place" return="next_place"/>
            <set var="location" value="next_place"/>
            <transition to="carryingFood" when="place.food &gt; 0">
                <do action="pick"/>
            </transition>
            <transition to="wandering" when="(place.road &lt; 0.05)" />
        </state>
    </species>
</entities>
```

environment

This section contains definitions of environment. Actually, GAMA supports two kinds of environments: GRID based environment and continuous environment. A model can have several GRID environments and one continuous environment. GAMA platform is responsible for assuring the all the necessary synchronizations between these environments. Example:

```
  <environment width="gridsize" height="gridsize">
      <grid name="ant_grid" width="gridsize" height="gridsize"
        neighbours="8">
        <var type="bool" name="multiagent" init="true" const="true" />
        <var type="int" name="type" init="types at {grid_x,grid_y}"
            const="true" />
        <var type="bool" name="isNestLocation" init="(self distance_to center)
&lt; 4"
            const="true" />
        <var type="bool" name="isFoodLocation" init="type = 2" const="true" />
        <var type="rgb" name="color"
            value="isNestLocation ? FF00FF:((food &gt; 0)? blue : ((road &lt;
0.001)? [100,100,100] : ((road &gt; 2)? white : ((road &gt; 0.5)? (C00CC00) :
((road &gt; 0.2)? (C009900) : (C005500))))))" />
        <var type="int" name="food" init="isFoodLocation ? 5 : 0" />
        <var type="int" name="nest" init="300 - (self distance_to center)"
            const="true" />
        <init when=" location = center">
            <create species="ant" number="ants_number"
with="[location::myself.location]"/>
        </init>
      </grid>
  </environment>
```

output

This section defines outputs to display in the simulation mode of GAMA. Several kinds of output are supported.

- display: defines views to display grids, species, charts, texts and images. For each element, it is possible to define a size (defined by a *point*) and a position (also defined by a point). Example of display:

```
<display name="Ants" background="rgb 'white'" refresh_every="2">
    <image file="display_image ? '../images/soil.png' : nil"
            position="{0.05,0.05}" size="{0.9,0.9}" />
    <grid name="ant_grid" transparency="grid_transparency"
position="{0.05,0.05}"
            size="{0.9,0.9}" lines="rgb 'yellow'" />
    <species name="ant" position="{0.05,0.05}" size="{0.9,0.9}"
            aspect="default" />
    <text name="food"
            value="'Food foraged : ' + string (((food_gathered / food_placed)
* 100) with_precision 2) + '%'"
            position="{0.05,0.03}" color="rgb 'black'" size="{1,0.02}" />
    <text name="agents"
            value="'Carrying ants : ' + string (int (list ant count
(each.hasFood)) + int (list ant count (each.state = 'followingRoad')))"
            position="{0.5,0.03}" color="rgb 'black'" size="{1,0.02}" />
</display>
```

- chart: defines a view to display a chart. There are three types of chart: pie, series and histogram.
  - Pie: defines a view to display the pie-based chart.
  - Series: defines a view to display the series-based chart.
  - Histogram: defines a view to display a histogram-based chart.
- inspect: inspectors the species and agents defined in the model.
  - agent inspect: user clicks on the Grid display or Gis display to select an agent. This view will display all the attributes of the selected agents as well as the change of these attributes over the simulation.
  - species inspect: this inspector displays all the species, the corresponding agents and attributes dedined in the model in a tree-based structure.
- monitor: a monitor is used to observe the change in value of an expression over the simulation.

Example:

```
<output>
    <inspect name="Agents" type="agent" refresh_every="1"/>
```

```
    <inspect name="Species" type="species" refresh_every="1"/>
    <chart type="pie" name="Carrying Food" background="rgb 'lightGray'"
refresh_every="5" style="exploded">
        <data name="Carrying Food" value=" (list ant) count (each.state =
'carryingFood')" />
        <data name="Not Carrying Food" value=" (list ant) count (each.state !=
'carryingFood')" />
    </chart>
    <chart type="histogram" name="Carrying Food"" background="rgb 'lightGray'"
refresh_every="5" style="3d">
        <data name="Carrying Food" value=" (list ant) count (each.state =
'carryingFood')" />
        <data name="Not Carrying Food" value=" (list ant) count (each.state !=
'carryingFood')" />
    </chart>
    <display name="Ants" background="rgb 'white'" refresh_every="2">
      <image file="display_image ? '../images/soil.png' : nil"
            position="{0.05,0.05}" size="{0.9,0.9}" />
      <grid name="ant_grid" transparency="grid_transparency"
position="{0.05,0.05}"
            size="{0.9,0.9}" lines="rgb 'yellow'" />
      <species name="ant" position="{0.05,0.05}" size="{0.9,0.9}"
            aspect="default" />
      <text name="food"
            value="'Food foraged : ' + string (((food_gathered / food_placed)
* 100) with_precision 2) + '%'"
            position="{0.05,0.03}" color="rgb 'black'" size="{1,0.02}" />
      <text name="agents"
            value="'Carrying ants : ' + string (int (list ant count
(each.hasFood)) + int (list ant count (each.state = 'followingRoad')))"
            position="{0.5,0.03}" color="rgb 'black'" size="{1,0.02}" />
  </display>
</output>
```

batch

Please see [Batch this section] for a detailed description of the batch mode.

Species Definition

<wiki:toc max_depth="3" />

Introduction

The agents' species are defined in the entities section. A model can contain any number of species. Species are used to specify the structure and behaviors of agents. Although the definitions below apply to all the species, some of them require specific declarations: the species of the world and the species of the environmental places.

Species declaration

The simplest way to declare a species is the following:

```
<species name="a_name">
  [variable declarations]
  [action declarations]
  [behaviors]
</species>
```

for example:

```
<species name="foo"/>
```

The agents that will belong to this species will only be provided with some built-in attributes and actions, a basic behavioral structure and nothing more. So, for instance, it is possible (somewhere else in the model) to write something like:

```
<create species="foo" number="10" return="foo_agents"/>
<ask target="foo_agents">
    <do action="write">
      <arg name="message" value="'my name is:' + name"/>
    </do>
</ask>
```

Which will result in the 10 agents writing their name, in turn, on the console. If the species declare variables, the structure of the agents is modified consequently. For instance:

```
<species name="foo">
    <var type="float" name="energy" init="rnd 100" min="0" max="100"
value="energy - 0.001" />
</species>
```

Will give each agent an amount of energy (between 0 and 100), which will decrease over time until it reaches 0. The species can also declare actions that will supplement the built-in ones and extends the possibilities of the agents. Here, we provide two possible actions for agents of species foo, eating and stealing energy:

```
<species name="foo">
    <var type="float" name="energy" init="rnd 100" min="0" max="100"
value="energy - 0.001" />
    <action name="eat">
        <set var="energy" value="energy + rnd 2"/>
    </action>
    <action name="steal">
        <let name="another_agent" value="any (list foo - self)"/>
        <if condition="(another_agent != nil) and ((another_agent.energy) >
0)">
          <set var="another_agent.energy" value="another_agent.energy - 0.01"/
>
          <set var="energy" value="energy + 0.01"/>
        </if>
    </action>
</species>
```

Of course, these actions do nothing unless they are called either by behaviors or by other agents. One might for example extend the previous example like:

```
<create species="foo" number="1000" return="foo_agents"/>
<ask target="100 among foo">
    <do action="eat"/>
</ask>
<ask target="100 among foo">
    <do action="steal"/>
</ask>
```

In this example, we create 1000 foos, ask 100 of them to eat, and another 100 of them to steal energy. If these commands are done repetitively (for example, every turn in the world), they will result in a somewhat complex dynamic distribution of the energy between the foos. Of course, the dynamics of foos can also be declared from within their species. If we change slightly the declaration of foo like this:

```
<species name="foo">
    <var type="float" name="energy" init="rnd 100" min="0" max="100"
value="energy - 0.001" />
    <action name="eat">
        <set var="energy" value="energy + rnd 2"/>
    </action>
    <action name="steal">
        <let name="another_agent" value="any (list foo - self)"/>
        <if condition="(another_agent != nil) and ((another_agent.energy) >
0)">
          <set var="another_agent.energy" value="another_agent.energy - 0.01"/
>
          <set var="energy" value="energy + 0.01"/>
        </if>
    </action>
    <reflex>
        <if condition="flip 0.1">
          <do action="eat"/>
        </if>
        <if condition="flip 0.1">
          <do action="steal"/>
        </if>
    </reflex>
</species>
```

We obtain agents that execute the reflex every turn and decide independently to eat or steal energy. Once they are created using

```
<create species="foo" number="1000"/>
```

they behave by their own.

Skills: behavioral plug-ins

Basic agents like the previous ones cannot, however, do many things. That's what skills are for. Example:

```
<species name="foo" skills="situated">
  ...
</species>
```

makes foos benefit from a set of variables and behaviors declared by the situated skill. Skills are like plug-ins written in Java and can provide a lot of new functionality to the agents.

Aspects: display properties

The aspect section allows to define the display of of the agents. it is possible to define different displays (i.e. different aspect sections) for a same species. In this context, the user will be able to change the display drawn during the simulation execution. The command **draw** allows to draw a shape (line, circle or square), a icon, a text or the agent geometry. this command has several facets:

- shape: optional, can be either "line", "circle", "square" or "geometry (in this case, the geometry of the agent will be drawn).
- text: string, optional, the text to draw.
- image: string, optional, path of the icon to draw (JPEG, PNG, GIF).
- color: rgb, optional, the color to use to display the shape/text/icon/geometry.
- size: float, size of the shape/text/icon (not use in the context of the drawing of a geometry).
- at: point, location where the shape/text/icon is drawn (not use in the context of the drawing of a geometry).
- to: point, terminal location of the line (only use in the in the context of the drawing of a line).
- rotate: int, orientation of the shape/text/icon (not use in the context of the drawing of a geometry).

For example, the following model allows to define three displays for the agent: one named "info", another named "icon" and the last one without a name (and thus named "default").

```
<aspect name="info">
    <draw shape="square" at="location" size="2" rotate="heading" />
    <draw shape="line" at="location" to="destination + ((destination -
location))" color="rgb 'white'" />
    <draw shape="circle" at="location" size="4" empty="true" color="'white'" /
>
    <draw text="my heading" color="rgb 'white'" size="1" />
    <draw text="state" color="rgb 'white'" size="1" at="my location + {1,1}"/>
</aspect>

<aspect name="icon">
    <draw image="shape" at="my location" size="2" rotate="my heading" />
</aspect>
<aspect>
    <draw shape="square" at="my location" empty="!hasFood" color="'yellow'"
size="2" rotate="my heading" />
```

```
</aspect>
```

Parent: inheritance of species

A species can be declared as a child of another species, using the parent property. For instance :

```
<species name="foo" skills="situated" parent="bar">
  ...
</species>
```

will make foo "inherit" from the definition of bar. What does "inherit" precisely mean in this context ?

- skills declared in bar, together with their built-in attributes and actions, are copied to foo and added to the possible new skills defined in foo.
- variables declared in bar, are identically copied to foo unless a variable with the same name is defined in foo, in which case this redefinition is kept. This also applies to built-in variables. The type of the variable can be changed in this process as well (but be careful when doing it, since inherited behaviors can rely on the previous type).
- actions declared in bar are identically copied to foo unless an action with the same name is defined in foo, in which case this redefinition is kept.
- reflexes declared in bar are identically copied to foo unless a reflex with the same name is defined in foo, in which case this redefinition is kept. Unnamed reflexes from both species are kept in the definition of foo.
- behaviors declared in bar are identically copied to foo unless a behavior of the same type with the same name is defined in foo, in which case this redefinition is kept.
- inits are treated differently : each of the init reflexes defined in bar and foo are kept in foo and they are executed in the order of inheritance (ie. bar 's one first, then foo 's one).

Control: behavioral architecture

By default, species are created with a minimal behavioral architecture : they only allow the definition of reflexes as a way to define the agents' behaviors. As reflex-based agents are somewhat limited when it comes to maintaining a state between two steps or enabling the selection of behaviors, GAML provides the modeler with two possible behavioral architectures, EMF (for Etho-Modeling Framework) and FSM (Finite State Machines). Each of them gives the possibility to define new elements in addition to reflexes : respectively tasks and states.

Base: Java foundation

The corresponding class used to initialize agent. An advance feature of the GAMA platform allowing the third party developer to develop their own agent architecture using the Java programming language.

_

Behaviors

<wiki:toc max_depth="3" />

Common behaviors

Basic agents (including the world and grid cells) are provided with a simple behavioral structure, based on reflexes. Species can define any number of reflexes within their body.

reflex

Attributes

- when or if: a boolean expression

Definition

A reflex is a sequence of commands that can be executed, at each time step, by the agent. If no attributes when (or if) are defined, it will be executed every time step. If there is an attribute when (or if), it is executed only if the boolean expression evaluates to true. It is a convenient way to specify the behavior of the agents. Example:

```
<reflex>
    <do action="debug"> Executed every time step
          <arg name="message" value="'Executing the unconditional reflex'"/>
    </do>
</reflex>
```

```
<reflex when="flip 0.5"> Only executed when flip returns true
    <do action="debug">
          <arg name="message" value="'Executing the conditional reflex'"/>
    </do>
</reflex>
```

init

Attributes

- when or if: a boolean expression

Definition

A special form of reflex that is evaluated only once when the agent is created, after the initialization of its variables, and before it executes any reflex. Only one instance of init is allowed in each species (except in case of inheritance, see this section). Useful for creating all the agents of a model in the definition of the world, for instance.

EMF-based behaviors

EMF (Etho Modeling Framework) is task based behavior model. Species can define any number of tasks within their body. At any given time, only one task having the highest priority value is executed.

task

Sub elements

Besides a sequence of commands like reflex, a task contains the following sub elements:

- priority: Mandatory. The activation level of the task.
- durationMin: Optional. The minimal duration of the task. Until it is reached, the task cannot be interrupted.
- durationMax: Optional. The maximal duration of the task. Beyond this duration, the task is automatically halted.
- whileCondition: Optional. The "while" watchdog : a condition tested every step to ensure that the task can still be executed. If false, the task is stopped (except if the min duration is not reached)

- untilCondition: Optional. The "until" watchdog, a condition tested every step to ensure that the task can still be executed. If true, the task is stopped (except if the min duration is not reached)
- endAction: Optional. The action to execute in case of failure/interruption of the task. No repeat is allowed.

Definition

Like reflex, a task is a sequence of commands that can be executed, at each time step, by the agent. If an agent owns several tasks, the scheduler choses a task to execute basing on its current priority value. For an example of task, please have a look the definition of "foodCarrier" species in this Task exemple.

FSM-based behaviors

FSM (Finite State Machine) is a finite state machine based behavior model. During its life cycle, agent possesses several states. At any given time step, an agent is in one state.

state

Attributes

- initial: a boolean expression, indicates the initial state of agent.
- final: a boolean expression, indicates the final state of agent.

Sub elements

- enter: a sequence of commands to execute upon entering the state.
- exit: a sequence of commands to execute right before exiting the state.
- transition: specifies the next state of the life cycle.

Definition

A state like a reflex can contains several commands that can be executed, at each time step, by the agent. For an exemple of state, please have a look at the definition of "ant" species in this FSM exemple.

Types

<wiki:toc max_depth="3" />

Built-in types

bool

- Definition: primitive datatype providing two values : true or false.
- Litteral declaration: both true or false are interpreted as boolean constants.
- Other declarations: expressions that require a boolean operand often directly apply a casting to bool to their operand. It is a convenient way to directly obtain a bool value.

```
bool 0 = bool {} -> true;
```

int

- Definition: primitive datatype holding integer values comprised between -231 and 231 - 1
- Comments: this datatype is internally backed up by the Java int datatype.
- Litteral declaration: decimal notation like 1, 256790 or hexadecimal notation like #1209FF are automatically interpreted.
- Other declarations: expressions that require an integer operand often directly apply a casting to int to their operand. Using it is a way to obtain an integer constant.

```
int 234.5 -> 234.
```

float

- Definition: primitive datatype holding floating point values comprised between -(2-252) **21023 and -(2-252)** 21023.
- Comments: this datatype is internally backed up by the Java double datatype.
- Litteral declaration: decimal notation 123.45 or exponential notation 123e45 are supported.
- Other declarations: expressions that require an integer operand often directly apply a casting to float to their operand. Using it is a way to obtain a float constant.

```
float 12 -> 12.0;
```

point

- Definition: a datatype normally holding two positive float values. Represents the absolute coordinates of agents in the model.
- Comments:
  o although points are intended to only represent coordinates, they can serve other purposes (like holding a couple of values). In that sense, they are allowed to hold values of any datatype. Note, however, that point-specific operations will try to convert the two elements to float values before applying. o when used as coordinates, points bound their elements between 0 and, respectively, the width and height of the environment.
- Litteral declaration: two numbers, separated by a comma, enclosed in braces, like {12.3, 14.5}
- Other declarations: points can be build litteraly from a list, or from an integer or float value by using the point casting operator.

```
point [12,123.45] -> {12.0, 123.45};
point 2 -> {2.0, 2.0}
```

string

- Definition: a datatype holding a sequence of characters.
- Comments: this datatype is internally backed up by the Java String class. However, contrary to Java, strings are considered as a primitive type, which means they do not contain character objects. This can be seen when casting a string to a list using the list operator: the result is a list of one-character strings, not a list of characters.
- Litteral declaration: a sequence of characters enclosed in quotes, like 'this is a string' . If one wants to literally declare strings that contain quotes, one has to double these quotes in the declaration. Strings accept escape characters like \n (newline), \r (carriage return), \t (tabulation), as well as any Unicode character (\uXXXX).
- Other declarations: see string
- Example: see string operators.

list

- Definition: a composite datatype holding an ordered collection of values.
- Comments: lists are more or less equivalent to instances of [ArrayList] in Java (although they are backed up by a specific class). They grow and shrink as

needed, can be accessed via an index (see @ or index_of), support set operations (like union and difference), and provide the modeller with a number of utilities that make it easy to deal with collections of agents (see, for instance, shuffle, reverse,where,sort_by,...).
- Remark: lists can contain values of any datatypes, including other lists. Note, however, that due to limitations in the current parser, lists of lists cannot be declared litteraly; they have to be built using assignments.
- Litteral declaration: a set of expressions separated by commas, enclosed in square brackets, like [12, 14, 'abc', self] . An empty list is noted [].
- Other declarations: lists can be build litteraly from a point, or a string, or any other element by using the list casting operator.

```
list 1 -> [1];
list 'abc' -> ['a','b','c'];
```

map

- Definition: a composite datatype holding an ordered collection of pairs (a key, and its associated value).
- Comments: maps are more or less equivalent to instances of Hashtable in Java (although they are backed up by a specific class).
- Remark: maps can contain values of any datatypes, including other maps or lists.
- Litteral declaration: a set of pair expressions separated by commas, enclosed in square brackets; each pair is represent by a key and a value sperarated by '::'. An example of map is [agentA::'big', agentB::'small', agentC::'big'] . An empty map is noted [].
- Other declarations: lists can be build litteraly from a point, or a string, or any other element by using the map casting operator.

```
map 1 -> [1::1];
map {1,5} -> [x::1, y::5];
```

rgb

- Definition: a datatype that represents a color in the RGB space.
- Litteral declaration: rgb colors can only be built literally using the rgb casting operator applied to a string, a list or an int. See the definition of this operator for some examples.
- Comments: to get the red, green and blue components of a color, apply the list casting operator to the color.
  o Example (get, in the temporary variables red, green and blue, the three integer components of the color, and change only the red component of the color) :

```
<let var="color" value="rgb 'green'"/>
<let var="components" value="list color"/>
<let var="red" value="first components"/>
<let var="green" value="components @ 1"/>
<let var="blue" value="last components"/>
<set var="red" value="red + 100"/>
<set var="color" value="rgb [red, green, blue]"/>
```

matrix

- Definition: a composite datatype that represents either a two-dimension array (matrix) or a one-dimension array (vector), holding any type of data (including other matrices).
- Comments: Matrices are fixed-size structures that can be accessed by index (point for two-dimensions matrices, integer for vectors).
- Litteral declaration: Matrices cannot be defined literally. One-dimensions matrices can be built by using the matrix casting operator applied on a list. Two-dimensions matrices need to to be declared as variables first, before being filled.

```
<let var="mm" value="matrix [10, 20, 30, 40, 50]" /> (builds a one-dimension
matrix, of size {10,10], where each cell is initialized to zero)
<matrix name="mat" size="{10,5}" />  (builds a two-dimensions matrix with 10
columns and 5 lines)
```

agent

- Definition: a generic datatype that represents an agent whatever its actual species.
- Comments: This datatype is barely used, since species can be directly used as datatypes themselves.
- Declaration: the agent casting operator can be applied to an int (to get the agent with this unique index), a string (to get the agent with this name) or to a point (to get the agent closest to this location). Example of this latter use :

```
<let name="trial" value="agent {100,100}"/>
<agent name="one_agent" init="nil" value="first neighbours"/> (declares a
variable whose value will always be the first agent in the neighbourhood.
Since the variable neighbours returns all the agents, we cannot know in
advance what its species could be)
```

species

- Definition: a generic datatype that represents a species
- Comments: this datatype is actually a "meta-type". It allows to manipulate (in a rather limited fashion, however) the species themselves as any other values.

- Litteral declaration: the name of a declared species is already a litteral declaration of species.
- Other declarations: the species casting operator, or its variant called species_of can be applied to an agent in order to get its species.

If we want to refine the previous example in setting one_agent to the first neighbour iff it belongs to the same species:

```
<agent name="one_agent" init="nil"/>
<reflex>
    <let name="first_agent" value="first neighbours"/>
    <if condition="(first_agent != nil) and (species first_agent) = (species 
self)">
          <set var="one_agent" value="first_agent"/>
          <else>
              <set var="one_agent" value="nil"/>
          </else>
    </if>
</reflex>
```

An easiest way to do the same:

```
<agent name="one_agent" init="nil" value="first (neighbours of_species 
(species self))" />
```

Species as types

Once a species has been declared in a model, it automatically becomes a datatype. This means that :

- It can be used to declare variables, parameters or constants,
- It can be used as an operand to commands or operators that require species parameters,
- It can be used as a casting operator (with the same capabilities as the built-in type agent)

In the simple following example, we create a set of "humans" and initialize a random "friendship network" among them. See how the name of the species, human, is used in the create command, as an argument to the list casting operator, and as the type of the variable named friend.

```
<global>
```

```
    <init>
        <create species="human" number="10"/>
        <ask target="list human">
              <set var="friend" value="one_of (list human - self)"/>
        </ask>
    </init>
</global>
<species name="human">
    <human name="friend" init="nil"/>
</species>
```

Defining custom types

Sometimes, besides the species of agents that compose the model, it can be necessary to declare custom datatypes. Species serve this purpose as well, and can be seen as "classes" that can help to instantiate simple "objects". In the following example, we declare a new kind of "object", bottle, that lacks the skills habitually associated with agents (moving, visible, etc.), but can nevertheless group together attributes and behaviors within the same closure. The following example demonstrates how to create the species:

```
<species name="bottle">
    <float name="volume" init="0.0" max="1 liter" min="0.0" />
    <bool name="is_empty" value="volume = 0.0"/>
    <action name="fill">
        <set var="volume" value="1 liter"/>
    </action>
</species>
```

How to use this species to declare new bottles :

```
<create species="bottle" number="1">
    <set var="volume" value="0.5 liter" />
</create>
```

And how to use bottles as any other agent in a species (a drinker owns a bottle; when he gets thirsty, it drinks a random quantity from it; when it is empty, it refills it):

```
<species name="drinker">
    ...
    <bottle name="my_bottle" init="nil" />
    <float name="quantity" init="(rnd 100) / 100"/>
    <bool name="thirsty" init="false" value="flip 0.1"/>
    ...
```

```
    <action name="drink">
        <if condition="! bottle.is_empty">
              <set var="bottle.volume" value="bottle.volume - quantity"/>
              <set var="thirsty" value="false"/>
        </if>
    </action>
    ...
    <init>
          <create species="bottle" number="1" return="created_bottle">
              <set var="volume" value="0.5 liter" />
          </create>
          <set var="my_bottle" value="created_bottle"/>
    </init>
    ...
    <reflex when="bottle.is_empty">
        <ask target="bottle">
              <do action="fill"/>
        </ask>
    </reflex>
    ...
    <reflex when="thirsty">
        <do action="drink"/>
    </reflex>
</species>
```

—

Commands

<wiki:toc max_depth="3" />

General syntax

A command is an XML tag, followed by specific attributes, some of them mandatory (in bold), some of them optional (in italic).

```
<command_tag attribute1="expression1" attribute2="expression2" ... />
```

If the command encloses other commands, they are declared between the opening tag and a closing tag, as in:

```
<command_tag1 attribute1="expression1" attribute2="expression2" ... >
    <command_tag2 attribute1="expression1" attribute2="expression2" ... >
      ...
    </command_tag2>
    <command_tag3 attribute1="expression1" attribute2="expression2" ... />
</command_tag1>
```

ask

Attributes

- target: an expression that evaluates to an agent or a list of agents
- as: an expression that evaluates to a species

Definition

Allows an agent, the sender agent, to ask another (or other) agent(s) to perform a set of commands. It obeys the following syntax, where the target attribute denotes the receiver agent(s):

```
<ask target="receiver_agent(s)">
    [commands]*
</ask>
```

If the value of the target attribute is nil or empty, the command is ignored. The species of the receiver agents must be known in advance for this command to compile. If not, it is possible to cast them using the as attribute, like :

```
<ask target="receiver_agent(s)" as="a_species_expression">
    [command_set]
</ask>
```

Alternative forms for this casting are :

- if there is only a single receiver agent:

```
<ask target="species_name receiver_agent">
    [command_set]
</ask>
```

- if receiver_agent(s) is a list of agents:

```
<ask target="receiver_agents of_species species_name">
    [commands]*
</ask>
```

Any command can be declared in command_set, except ask itself (although an ask can be performed in an action called within the command_set). All the commands will be evaluated in the context of the receiver agent(s), as if they were defined in their species, which means that an expression like self will represent the receiver agent and not the sender. If the sender needs to refer to itself, some of its own attributes (or temporary variables) within the command_set, it has to use the keyword myself.

```
<species name="animal">
    <float name="energy" init="rnd 1000" min="0.0"/>
    <reflex when="energy > 500"> executed when the energy is above the given
threshold
        <let name="others" value="(self neighbours_at 5) of_species animal"/>
find all the neighbouring animals in a radius of 5 meters
        <let name="shared_energy" value="(energy - 500) / length others"/>
compute the amount of energy to share with each of them
        <ask target="others"> no need to cast, since others has already been
filtered to only include animals
              <if condition="energy &lt; 500">  refers to the energy of each
animal in others
                  <set var="energy" value="energy + myself.shared_energy"/>
increases the energy of each animal
```

```
                  <set var="myself.energy" value="myself.energy -
myself.shared_energy"/> decreases the energy of the sender
              </if>
        </ask>
    </reflex>
</species>
```

Notice

If the target is an addition of list like "target = (list speciesA) + (list speciesB)", the temporary built list will use the default cast from the first list and won't add the second list as the elements are from a different type.

create

Attributes

- species: an expression that evaluates to a species
- number: an expression that evaluates to an int
- from: an expression that evaluates to a localized entity, a list of localized entities or a string
- with: an expression that evaluates to a map
- type: an expression that evaluates to a string
- size: an expression that evaluates to a float
- return: a temporary variable name

Definition

Allows an agent to create **number** agents of species **species** , to create agents of species **species** from a shapefile or to create agents of species **species** from one or several localized entities (discretization of the localized entity geometries). Its simple syntax is :

- To create **an_int** agents of species **a_species** :

```
<create species="a_species" number="an_int"/>
```

If **number** equals 0 or species is not a species, the command is ignored

- To create agents of species **a_species** from a shapefile **the_shapefile** while reading the attribute 'TYPE_OCC' and 'NATURE' of the shapefile:

```
<create species="a_species" from="the_shapefile" with="[type::read 'TYPE_OCC',
nature::'NATURE']"/>
```

One agent agent will be created by object contained in the shapefile. In this example, we assume that for the species **a_species** , two variables **type** and **nature** are declared and that their **type** corresponds to the types of the shapefile attributes.

- To create agents of species **a_species** by discretizing the geometry of one or several localized entities:

```
<create species="a_species" from="[agentA, agentB, agentC]"/>
```

Two types of discretization exists:

- **'Triangles'** : default discretization. The agent geometries are decomposed into triangles; for each triangle, an agent is created. if a size is declared by attribute **size** , the geometries are first decomposed into squares of size **size** , then each square is decomposed into triangles.

```
<create species="a_species" from="[agentA, agentB, agentC]" type="'Triangles'"
size="10.0"/>
```

- **'Squares'** : The agent geometries are decomposed into squares of size **size** ; for each square, an agent is created:

```
<create species="a_species" from="[agentA, agentB, agentC]" type="'Squares'"
size="10.0"/>
```

The agents created are initialized following the rules of their species. If one wants to refer to them after the command is executed, the return (or result) keyword has to be defined: the agents created will then be referred to by the temporary variable it declares. For instance, the following command creates 0 to 4 agents of the same species as the sender, and puts them in the temporary variable children for later use.

```
<create species="species self" number="rnd 4" return="children"/>
<ask target="children">
...
</ask>
```

If one wants to specify a special initialization sequence for the agents created, create provides the same possibilities as ask. This extended syntax is:

```
<create species="a_species" number="an_int">
```

```
    [commands]*
</create>
```

The same rules as in ask apply. The only difference is that, for the agents created, the assignments of variables will bypass the initialization defined in species. For instance :

```
<create species="species self" number="rnd 4" return="children">
    <set var="location" value="myself.location + {rnd 2, rnd 2}" /> tells the children to be initially located close to me
    <set var="parent" value="myself" /> tells the children that their parent is me (provided the variable parent is declared in this species)
</create>
```

do

Attributes

- action: the name of an action or a primitive
- return: a temporary variable name

Enclosed tags

- arg or a datatype name : specify the arguments expected by the action/primitive to execute.

Definition

Allows the agent to execute an action or a primitive. For a list of primitives available in every species, see this page; for the list of primitives defined by the different skills, see this page. Finally, see this page to know how to declare custom actions. The simple syntax (when the action does not expect any argument and the result is not to be kept) is :

```
<do action="name_of_action_or_primitive" />
```

In case the result of the action needs to be made available to the agent, the return keyword has to be defined; the result will then be referred to by the temporary variable declared in this attribute:

```
<do action="name_of_action_or_primitive" return="temp_var" />
```

In case the action expects one or more arguments to be passed, they are defined by using enclosed tags, either arg or the name of a datatype if one wants to specify their type. For instance:

```
<do action="name_of_action_or_primitive" return="temp_var" >
    <arg name="arg1" value="expression1"/>
    <var type="the_datatype" name="arg2_of_type_datatype"
value="expression2_of_type_datatype"/>
    ...
</do>
```

let

Attributes

- name: the name of the temporary variable
- type: the datatype of the temporary variable
- value: an expression

Definition

Allows the agent to declare a temporary variable, local to the scope in which it is defined. The naming rules follow those of the |variables declarations. In addition, a temporary variable cannot be declared twice in the same scope. The generic syntax is :

```
<let name="temp_var1" type="a_datatype" value="an_expression"/>
```

If the datatype of the variable is not specified, it is inferred from that of the expression (which can be enforced using casting operators if necessary). After it has been declared this way, a temporary variable can be used like regular variables (for instance, the set command should be used to assign it a new value within the same scope).

set

Attributes

- name or var: an expression that either returns a variable or an element of a composite type

- value: an expression

Definition

Allows the agent to assign a value to a variable or an element of a composite variable (Types#string, list, matrix, point). See this section to know how to access variables.
To access an element within a composite variable, the operator at has to be used.
Examples:

```
<set name="my_var" value="expression"/>
<set name="temp_var" value="expression" />
<set name="global_var" value="expression" />
<set name="composite_var at index" value="expression" />
```

The variable assigned can be accessed in the value atribute. In that case, it represents the value of the variable (or the element) before it has been modified. Examples (with temporary variables):

```
<let name="my_list" value="[1,2,3]" />
<set name="my_list at 2" value="(my_list at 2) + 10" /> my_list now equals
[1,2,13]
<let name="my_int" value="1000"/>
<set name="my_int" value="my_int + 1"/> my_int now equals 1001
```

Assignments are the only way to create lists composed of list elements. For instance :

```
<let name="list_of_lists" value="[]"/>
<set name="list_of_lists at 0" value="[1,2,3]"/>
<set name="list_of_lists at 1" value="[4,5,6]"/>
<set name="list_of_lists at 2" value="[7,8,9]"/>
```

creates the list : [[1,2,3], [4,5,6], [7,8,9]].

if

Attributes

- condition: a boolean expression

Enclosed tags

- else : encloses alternative commands

Definition

Allows the agent to execute a sequence of commands if and only if the condition evaluates to true. The generic syntax is:

```
<if condition="bool_expr">
    [command]*
</if>
```

Optionally, the commands to execute when the condition evaluates to false can be defined in an enclosed command else. The syntax then becomes:

```
<if condition="bool_expr">
    [command]*
    <else>
        [command]*
    </else>
</if>
```

ifs and elses can be imbricated as needed. For instance:

```
<if condition="bool_expr">
    [command]*
    <else>
        <if condition="bool_expr2">
              [command]*
              <else>
                    [command]*
              </else>
        </if>
    </else>
</if>
```

loop

Attributes

- times: an int expression, or
- while: a boolean expression, or
- over: a list expression, together with
- var: a temporary variable name

Definition

Allows the agent to perform the same set of commands either a fixed number of times, or while a condition is true, or by progressing in a collection of elements or along an interval of integers. The basic syntax for each of these usages are:

```
<loop times="an_int_expression">
    [command]*
</loop>
```

Or:

```
<loop while="a_bool_expression">
    [command]*
</loop>
```

Or:

```
<loop over="a_list_expression" var="a_temp_var">
    [command]*
</loop>
```

Or:

```
<loop from="int_expression_1" to="int_expression_2" var="a_temp_var">
    [command]*
</loop>
```

In these latter two cases, the var attribute designates the name of a temporary variable, whose scope is the loop, and that takes, in turn, the value of each of the element of the list (or each value in the interval). For example, in the first instance of the "loop over" syntax :

```
<let name="a" value="0"/>
<loop over="[10, 20, 30]" var="i">
    <set var="a" value="a + i"/>
</loop> a now equals 60
```

The second (quite common) case of the "loop" syntax allows one to use an interval of integers. The from and to attributes take a a integer expression as arguments, with the first (resp. the last) specifying the beginning (resp. end) of the interval. The step is assumed equal to 1.

```
<let name="the_list" value="list (species_of self)"/>
<loop from="0" to="length the_list" var="i">
    <ask target="the_list at i"/>
```

```
        ...
    </ask>
</loop> every  agent of the list is asked to do something
```

Be aware that there are no prevention of infinite loops. As a consequence, open loops should be used with caution, as one agent may block the execution of the whole model.

return

Attributes

- value: an expression

Definition

Allows to specify which value to return from the evaluation of the surrounding command. Usually used within the declaration of an action. Contrary to other languages, using return does not stop the evaluation of the surrounding command (for instance, a loop). It simply indicates what value to return: if it is inside a loop, then, only the last evaluation of return will be returned. Example:

```
<action name="foo">
    <return value="'foo'"/>
</action>
...
<reflex>
    <do action="foo" return="foo_result"/>
    <do action="write">
        <arg name="message" value="foo_result"/>
    </do>
</reflex>
... the agent will print  foo  on the console at each step
```

In the specific case one wants an agent to ask another agent to execute a command with a return, it should be done similarly to:

```
Species A:
<action name="foo_different">
    <return value="'foo_not_same'"/>
</action>
---
Species B
<reflex>
    <let var="temp" value="nil"/>
```

```
    <ask target="some_agent_A">
        <do action="foo_different" return="foo_result">
            <set var="myself.temp" value="foo_result"/>
        </do>
    </ask>
    <do action="write">
        <arg name="message" value="temp"/>
    </do>
</reflex>
... the agent will print  foo_not_same  on the console at each step
```

save

Attributes

- species: an expression that evaluates to a species
- to: an expression that evaluates to an string

Definition

Allows to save the localized entities of species **species** into a shapefile. Its simple syntax is :

```
<save species="a_species" to="the_shapefile"/>
```

—

Operators

<wiki:toc max_depth="3" />

Definition

An operator performs a function on one, two, or three operands. An operator that only requires one operand is called a unary operator. An operator that requires two operands is a binary operator. And finally, a ternary operator is one that requires three operands. The GAML programming language has only one ternary operator, ?:, which is a short-hand if-else statement. Unary operators are written using a prefix notation. Prefix notation means that the operator appears before its operand:

```
operator operand
```

All the binary operators use an infix notation, which means that the operator appears between its operands:

```
op1 operator op2
```

The ternary operator is also infix; each component of the operator appears between operands:

```
op1 ? op2 : op3
```

In addition to performing operations, operators are functional, i.e. they return a value. The return value and its type depend on the operator and the type of its operands. For example, the arithmetic operators, which perform basic arithmetic operations such as addition and subtraction, return numbers - the result of the arithmetic operation. Moreover, operators are strictly functional, i.e. they have no side effects on their operands. For instance, the shuffle operator, which randomizes the positions of elements in a list, does not modify its list operand but returns a new shuffled list.

Operators by categories

Arithmetic operators

abs cos sin tan exp sqrt float int ln rnd round + - / * // div % mean mod ^ with_precision

Comparison operators

= != <> > < >= <=

Boolean operators

! & and || or flip

Casting operators

agent bool float int string point rgb matrix list species

List-related operators

any accumulate empty first last length list max min mul sum reverse shuffle at + - among collect copy_between count first_with last_with in index_of last_index_of max_of min_of with_max_of with_min_of of_species sort_by starts_with where select

String-related operators

any empty first last length list max min reverse shuffle at + - among collect copy_between count first_with last_with in index_of last_index_of max_of min_of with_max_of with_min_of sort_by starts_with where select > < >= <= tokenize

Location-related operators

closest_point_to next_point_towards towards point distance_to neighbours_at

Agent-related operators

agent species species_of int string neighbours_at of_species

Unary operators

Unary operators take one and only one operand. The syntax is : operator operand. The priority between operators is determined from right to left (i.e. in op1 op2 op3 operand, op3 is evaluated before op2 and op1).

!

- Operand: a boolean expression.
- Result: opposite boolean value.
- Return type: bool.
- Special cases: if the parameter is not boolean, it is casted to a boolean value.
- see also: bool

```
! true # false.
```

acos

- Operand: an int or a float.
- Result: the arccos of the operand (which has to be expressed in decimal degrees).
- Return type: float.
- see also: asin, atan.

```
acos 90 # 0.
```

abs

- Operand: an arithmetic expression
- Result: the absolute value of the operand
- Return type: a positive int or float depending on the type of the operand

```
abs (200 * -1 + 0.5) # 200.5
```

agent

species name

- Operand: an int, agent, point or string.
- Result: casting of the operand to an agent (if **species name** is used, casting to an instance of **species name**).
- Return type: agent (or **species name**).
- Special cases:
  o if the operand is a point, returns the closest agent (resp. closest instance of **species name**) to that point; o if the operand is a string, returns the agent (resp. instance of **species name**) with this name; o if the operand is an agent, returns the agent (resp. tries to cast this agent to **species name** and returns nil if the agent is instance of another species) o if the operand is an int, returns the agent (resp. instance of **species name**) with this unique index;
- see also: of_species, species

any

same signification as one_of operator.

asin

- Operand: an int or a float.
- Result: the arcsin of the operand (which has to be expressed in decimal degrees).
- Return type: float.
- see also: acos, atan.

```
asin 90 # 1.
```

atan

- Operand: an int or a float.
- Result: the arctan of the operand (which has to be expressed in decimal degrees).
- Return type: float.
- see also: acos, asin.

```
atan 45 # 1.
```

one_of

- Operand: a list, string or matrix.
- Result: a random element from the list.
- Return type: any.
- Special cases: returns nil if the list is empty.
- Comment: the operand is casted to a list before the expression is evaluated. Therefore, if foo is the name of a species, any foo will return a random agent from this species (see list).

```
any [1,2,3] # 1, 2, or 3.
one_of [1,2,3] # 1, 2, or 3.
```

bool

- Operand: any type.
- Result: casting of the operand to a boolean value.
- Return type: bool.
- Special cases: if the operand is an int or a float, returns true if it is greater than 0 (or 0.0). If the operand is a list or a string, bool is formally equivalent to not empty (a la Lisp). Otherwise, returns false.

```
bool 0 # false;
bool [1, 2, 3] # true;
```

closest_point_to

- Operand: a localized agent or a point.
- Result: the point, closest to the operand, that can be reached by the agent.
- Return type: point.
- Special cases: returns nil if the argument cannot be reached by the agent, or if the agent is neither localized nor able to move. Uses the path finder if the environment is a GIS.
- see also: next_point_towards, towards.

```
closest_point_to {0,0} # returns the point, closest to the origin, that can be
reached by the agent.
```

cos

- Operand: an int or a float.

- Result: the cosinus of the operand (in decimal degrees).
- Return type: float.
- Special cases: the argument is casted to an int before being evaluated. Integers outside the range "0-359" are normalized.
- see also: sin, tan.

```
cos 0 # 1.
```

empty

- Operand: a list, string or matrix.
- Result: true if the argument is empty, false otherwise.
- Return type: bool.
- Special cases: matrices are never considered as empty, and thus always return false when the operand is a matrix.
- Comments: the operand is casted to a list before the expression is evaluated. Therefore, if foo is the name of a species, empty foo indicates whether or not agents of species foo are present in the model.

```
empty [] # true;
empty 'abced' # false;
```

exp

- Operand: an int or a float.
- Result: returns Euler's number e raised to the power of the operand.
- Return type: float.
- Special cases: the operand is casted to a float before being evaluated.
- see also: ln.

```
exp 0 # 1.
```

first

- Operand: a list, string, point or matrix.
- Result: the first element of the operand
- Return type: any.
- Special cases: if the operand is a string, returns a string composed of its first character.
- see also: last.
- Comments: first is used to grab the x-coordinate of a point;

```
first 'abce' # 'a'; first [1, 2, 3] # 1.
first {10,12} # 10.
```

flip

- Operand: an int or a float (between 0 and 1).
- Result: true or false given the probability represented by the operand.
- Return type: bool.
- Special cases: flip 0 always returns false, flip 1 true.
- see also: rnd.

```
flip 0.66666 # 2/3 chances to return true.
```

float

- Operand: any type.
- Result: casting of the operand to a floating point value.
- Return type: float.
- Special cases: if the operand is an agent, returns its unique index as a float; if the operand is a string, tries to convert its content to a floating point value; if the argument is a list, a point or a matrix, returns its first element converted to a float; if the operand is a boolean, returns 1.0 for true and 0.0 for false; if the argument is a color, returns its RGB value as a float.
- see also: int.

```
float '234.0' # 234.0; float {12,23} # 12.0;
float true # 1.0;
float (rgb 'black') # 0.0;
```

int

- Operand: any type.
- Result: casting of the operand to an integer value.
- Return type: int.
- Special cases: if the operand is a float, returns its value truncated (but not rounded); if the operand is an agent, returns its unique index; if the operand is a string, tries to convert its content to an integer value; if the argument is a list, a point or a matrix, returns its first element converted to an int; if the operand is a boolean, returns 1 for true and 0 for false; if the argument is a color, returns its RGB value as an integer; if the operand is a species, returns the number of its agents.
- see also: round, float.

```
int '234.3' # 234;
int {12,23} # 12;
int true # 1;
int (rgb 'black') # 0;
```

intensity (deprecated)

- Operand: a string.
- Result: the intensity of the signal denoted by the operand at the agent's location.
- Return type: float.
- Special cases: returns 0.0 if the agent is not controlled by EMF, if the agent is not localized or if no grids have been defined as an environment.

```
intensity 'pheromone' # returns the intensity of the 'pheromone' signal on the grid cell where the agent is located.
```

last

- Operand: a list, strin, point or matrix.
- Result: the last element of the operand
- Return type: any.
- Special cases: if the operand is a string, returns a string composed of its last character, or an empty string if the operand is empty.
- see also: first.
- Comments: last is used to grab the y-coordinate of a point; last {10,12} $\rightarrow$ 12.

```
last 'abce' # 'e'; last [1, 2, 3] # 3.
```

length

- Operand: a list, string or matrix.
- Result: the number of elements contained in the list; the number of characters in a string; the size (number of cells) of a matrix.
- Return type: int.

```
length 'I am an agent' # 13; length [12,13] # 2.
```

list

- Operand: any type.
- Result: casting of the operand to a list.
- Return type: list.

- Special cases:
  o if the operand is a point, returns a list containing its two coordinates; o if the operand is a string, returns a list of strings, each containing one character; o if the operand is a matrix, returns a list containing its elements; o if the operand is a rgb color, returns a list containing its three integer components; o if the operand is a species, return a list of its agents; o otherwise returns a list containing the operand.
- see also: +, - (for lists).
- Comment: list always tries to cast the operand except if it is an int, a bool or a float; to create a list, instead, containing the operand (including another list), use the + operator on an empty list (like [] + 'abc').

```
list 'abc' # ['a', 'b', 'c'];
list 123 # '123';
list ['a', 23] # ['a', 23];
list (rgb 'black') # [0, 0, 0];
```

ln

- Operand: an int or a float (greater than 0).
- Result: Returns the natural logarithm (base e) of the operand.
- Return type: float.
- Special cases: an exception is raised if the operand is less than zero.
- see also: exp.

max

- Operand: a list, point or matrix.
- Result: the maximum element found in the operand.
- Return type: float.
- Special cases: the elements of the operand are casted to float values before max is evaluated.
- see also: min.

```
max [100, 23.2, 34.5] # 100.0.
```

mean

- Operand: a list, point or matrix.
- Result: the mean of all the elements of the operand.
- Return type: float.
- Special cases: the elements of the operand are casted to float values before summing up then the sum value is divided by the number of elements.

- see also: sum.

```
mean [4.5, 3.5, 5.5] # 4.5.
```

min

- Operand: a list, point or matrix.
- Result: the minimum element found in the operand.
- Return type: float.
- Special cases: the elements of the operand are casted to float values before min is evaluated.
- see also: max.

```
min [100, 23.2, 34.5] # 23.2.
```

mul

- Operand: a list, point or matrix.
- Result: the product of all the elements of the operand.
- Return type: float.
- Special cases: the elements of the operand are casted to float values before mul is evaluated..
- see also: sum.

```
mul [12,10, 3] # 360.0.
```

next_point_towards

- Operand: an agent or a point.
- Result: the next location of the agent if it heads towards the operand and keeps the same speed.
- Return type: point.
- Special cases: if the agent is localized but cannot move, returns the location of the agent; if the agent is not localized, returns nil. Uses the path finder if the environment is a GIS.
- see also: closest_point_to.

reverse

- Operand: a list, point, string or matrix.
- Result: the operand elements in the reversed order.

- Return type: list, point or string.
- Special cases:
  o if the operand is a point {x,y}, returns a new point {y,x}; o if the argument is a matrix, returns a list of its elements in the reversed order;
- see also: shuffle.

```
reverse 'abcd' # 'dcba';
reverse {10,13} # {13, 10};
reverse [10,12,14] # [14, 12, 10].
```

rgb

- Operand: any type.
- Result: casting of the operand to a rgb color.
- Return type: rgb.
- Special cases:
  o if the operand is nil, returns white; o if the operand is an agent, returns its color if it is visible (otherwise white); o if the operand is a string, returns the color denoted by its contents if it is part of the basic colors understood by java.awt.Color; otherwise tries to cast the string to an int and returns this color; o if the operand is an int or a float, returns the color denoted by this number (decoding the number using the three R, G, and B integer components); o if the operand is a list, and its size is equal or greater than 3, casts the three first elements to integers and interprets them as R, G and B components; o if the operand is a boolean, returns black for true and white for false; o otherwise casts the operand to an int and returns the associated color;

```
rgb 'white' # white color;
rgb #FFFFFF # white color;
rgb [0, 255,0] # green color;
rgb '255' # blue color;
```

rnd

- Operand: an int or a float.
- Result: a random integer in the interval [0, operand] .
- Return type: int.
- Special cases: the argument is casted to an int before being evaluated.
- see also: flip.
- Comments : to obtain a probability between 0 and 1, use the expression (rnd n) / n, where n is used to indicate the precision;

```
rnd 2 # 0, 1 or 2
```

```
(rnd 1000) / 1000 # a float between 0 and 1 with a precision of 0.001
```

round

- Operand: an int or a float.
- Result: the rounded value of the operand.
- Return type: int.
- Special cases: the argument is casted to a float before round is evaluated.
- see also: int.

```
round 0.51 # 1;
round 100.2 # 100.
```

shuffle

- Operand: a list, string or matrix.
- Result: the elements of the operand in random order.
- Return type: list or string.
- Special cases: if the operand is empty, returns an empty list (or string);
- see also: reverse.

```
shuffle [12, 13, 14] # [14,12,13];
shuffle 'abc' # 'bac'.
```

sin

- Operand: an int or a float.
- Result: the sinus of the operand (in decimal degrees).
- Return type: float.
- Special cases: the argument is casted to an int before being evaluated. Integers outside the range "0-359" are normalized.
- see also: cos, tan.

```
sin 0 # 0.
```

species

species_of

- Operand: an agent expression.
- Result: casting of the operand to a species.

- Return type: species.
- Special cases:
  o if the operand is nil, returns nil; o if the operand is an agent, returns its species; o if the operand is a string, returns the species with this name (nil if not found);

```
species_of self # the species of the agent;
species 'world' # the species named 'world';
```

string

- Operand: any type.
- Result: casting of the operand to a string.
- Return type: string.
- Special cases:
  o if the operand is nil, returns 'nil'; o if the operand is an agent, returns its name; o if the operand is a string, returns the operand; o if the operand is an int or a float, returns their string representation (as in Java); o if the operand is a list, returns its string representation; o if the operand is a boolean, returns 'true' or 'false'; o if the operand is a species, returns its name; o if the operand is a color, returns its litteral value if it has been created with one (i.e. 'black', 'green', etc.) or the string representation of its hexadecimal value.

```
string (rgb 'black') # 'black';
string 12.34 # '12.34';
string world # 'world' if world is a species.
```

sum

- Operand: a list, point or matrix.
- Result: the sum of all the elements of the operand.
- Return type: float.
- Special cases: the elements of the operand are casted to float values before sum is evaluated..
- see also: mul.

```
sum [12,10, 3] # 25.0.
```

tan

- Operand: an int or a float.
- Result: the trigonometic tangent of the operand (expressed in decimal degrees).
- Return type: float.

- Special cases: the argument is casted to an int before being evaluated. Integers outside the range "0-359" are normalized.
- see also: sin, cos.

```
tan 180 # 0.
```

towards

- Operand: an agent or a point.
- Result: the direction (in degrees) that the agent must follow to face the operand.
- Return type: int.
- Special cases: if the agent is not localized, returns 0;
- see also: . next_point_towards

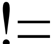
```
towards {0, 0} # an int between 0 and 359.
```

Binary operators

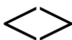
=

- Left-hand operand: any expression.
- Right-hand operand: any expression.
- Result: true if both operands are equal, false otherwise.
- Comments: this operator will return true if the two operands are identical (i.e., the same object) or equal. Comparisons between nil values are permitted. Note that floating point and integer values are always considered as different (i.e., 0.0 is not the same as 0).
- see also: !=.

```
[1, 2, 3.0] = [1,2,3.0] # true;
int 3.0 = 3 # true;
rgb 'black' = rgb #000000 # true;
float 1 = 1.0 # true;
```



same signification as <> operator.



- Left-hand operand: any expression.
- Right-hand operand: any expression.
- Result: true if both operands are different, false otherwise.
- Comments: this operator will return false if the two operands are identical (i.e., the same object) or equal. Comparisons between nil values are permitted. Note that floating point and integer values are always considered as different (i.e., 0.0 is not the same as 0).
- see also: =.

```
[1, 2, 3.0] != [1,2,3.0] # false;
int 3.0 != 3 # false;
rgb 'black' != rgb #000000 # false;
float 1 != 1.0 # false;
```

> >= < <=

- Left-hand operand: any expression.
- Right-hand operand: any expression.
- Result: true if the left-hand operand is respectively greater than, less than, greater than or equal to, less than or equal to the right-hand operand. false otherwise.
- Comments: these operator work with numbers (int and float) as well as strings, for which a lexicographic comparison is performed. Note on the syntax to use in the XML flavor of GAML: XML files do not allow the < and > characters to be used within tags. They have then to be replaced by their counterpart in HTML entities: < and > .

```
12 > 11.0 # true;
'zzz' > 'abc' # true;
int 12.0 < 12 # false;
```

##

same signification as at operator.

at

- Left-hand operand: a list, string, matrix or point.
- Right-hand operand: an integer or a point.

- Result: the element of the left-hand operand located at the index specified by the right-hand operand.
- Comments:
  o strings, lists, and matrices are zero-based implementations, which means that the first index is 0 and the last one length-1. o at can be used in the left member of an assignment to assign a new value to one of the positions of a list, a matrix, a string, a point...

```
<let var="my_list" value="[12, 13, 14, 15]" />
<set var="my_list at 2" value="18" /> # my_list is now equal to [12, 13, 18, 15].
```

```
<let var="my_matrix" size={2,2}" />
<set var="my_matrix at {0,1}" value="42"/> # the element on the first column and second line is now 42. (the rest of the matrix is empty)
```

- Special cases:
  o if the left-hand operand is a string, a list, a point or a one-dimension matrix, and the index is less than 0 or greater than its length, returns nil. o if the left-hand operand is a two-dimensions matrix and the index is a point, the same rule applies for both dimensions. o if the left-hand operand is a one-dimension matrix, a list, a point or a string, and the index is a point, the x-coordinate of the point is used for the index. o if the left-hand operand is a string, returns a one-character string.

```
[1, 10, 3.0] @ 2 # 3.0;
'abcdef' at 0 # 'a';
```

- Left-hand operand: a list, string, matrix, point, int or float.
- Right-hand operand: a list, string, matrix, point, int or float.
- Result: the sum, union or concatenation of the two operands.
- Special cases:

If both operands are numbers, performs a normal arithmetic sum and returns a float if one of them is a float.

```
1 + 1 # 2; 1.0 + 1 # 2.0;
```

If the left-hand operand is an int and the right-hand operand is not a number, casts the right-hand operand to an int.

```
1 + '34' # 35;
```

If the left-hand operand is a float and the right-hand operand is not a number, casts the right-hand operand to a float.

```
1.0 + '34' # 35.0;
```

If the left-hand operand is a string, casts the right-hand operand to a string and returns their concatenation.

```
'abc' + 'def' # 'abcdef';
```

If both operands are lists, returns their union.

```
[1, 2, 3] + [4, 5] # [1, 2, 3, 4, 5];
list 'abc' + list 'def' # ['a','b','c','d','e','f'];
```

If both operands are points, returns their sum.

```
{1, 2} + {4, 5} # {5, 7};
```

If the left-hand operand is a list and the right-hand operand is not a list, returns a new list with the right-hand operand added to the end of the left-hand operand. Matrices, points and strings are not considered as lists and will be added as individual elements.

```
[1,2,3] + 4 # [1,2,3,4];
['a','b','c'] + 'def' # ['a', 'b', 'c', 'def'];
```

Comments: Sum of matrices is not yet implemented but should be available sometime in the future with the same syntax.

-

- Left-hand operand: a list, matrix, point, int or float.
- Right-hand operand: a list, matrix, point, int or float.
- Result: the difference of the two operands.
- Special cases:

If both operands are numbers, performs a normal arithmetic difference and returns a float if one of them is a float.

```
1 - 1 # 0; 1.0 - 1 # 0.0;
```

If the left-hand operand is an int and the right-hand operand is not a number, casts the right-hand operand to an int.

```
1 - '34' # -33;
```

If the left-hand operand is a float and the right-hand operand is not a number, casts the right-hand operand to a float.

```
1.0 - '34' # -33.0;
```

If both operands are lists, returns their difference (removes all the elements of the right-hand operand from the left-hand operand).

```
[1, 2, 3] - [3, 4, 5] # [1, 2];
list 'abc' - list 'abk' # ['c'];
```

If both operands are points, returns their difference.

```
{10, 20} - {4, 5} # {6, 15};
```

If the left-hand operand is a list and the right-hand operand is not a list, returns a new list with the right-hand operand removed from the left-hand operand.

```
[1,2,3] - 3 # [1,2];
```

- Comments: Difference of matrices is not yet implemented but should be available sometime in the future with the same syntax.

*

- Left-hand operand: an int or float.
- Right-hand operand: an int or float.
- Result: a float, equal to the product of the two operands.
- Comments: Product of matrices is not yet implemented but should be available sometime in the future with the same syntax.

/

- Left-hand operand: an int or float.
- Right-hand operand: an int or float.
- Result: a float, equal to the division of the left-hand operand by the rigth-hand operand.
- Special cases: if the right-hand operand is equal to zero, raises no exception and returns the maximum value of float instead.
- Comments: Division of matrices is not yet implemented but should be available sometime in the future with the same syntax.

//

same signification as div operator.

div

- Left-hand operand: a int or float.
- Right-hand operand: a int or float.
- Result: an int, equal to the truncation of the division of the left-hand operand by the rigth-hand operand.
- Special cases: if the right-hand operand is equal to zero, raises no exception and returns the maximum value of int instead.

%

same signification as mod operator.

mod

- Left-hand operand: a int or float.
- Right-hand operand: a int or float.
- Result: an int, equal to the remainder of the integer division of the left-hand operand by the rigth-hand operand.
- Special cases: if the right-hand operand is equal to zero, raises no exception and returns the maximum value of int instead.

same signification as and operator.

and

- Left-hand operand: a boolean expression
- Right-hand operand: a boolean expression
- Result: a bool value, equal to the logical and between the left-hand operand and the rigth-hand operand.
- Comments: both operands are always casted to bool before applying the operator. Thus, an expression like 1 and 0 is accepted and returns false.
- see also: bool.

|

same signification as or operator.

or

- Left-hand operand: a boolean expression
- Right-hand operand: a boolean expression
- Result: a bool value, equal to the logical or between the left-hand operand and the rigth-hand operand.
- Comments: both operands are always casted to bool before applying the operator. Thus, an expression like 1 or 0 is accepted and returns true.
- see also: bool.

^

- Left-hand operand: a int or float expression
- Right-hand operand: a int or float expression
- Result: an int or float value, equal the left-hand operand raised to the power of the right-hand operand.
- Special cases: if the right-hand operand is equal to 0, returns 1; if it is equal to 1, returns the left-hand operand.

accumulate

- Left-hand operand: a list, point or string expression
- Right-hand operand: any.
- Result: a list containing all elements of left-hand expression and right-hand expression.

```
[1,2,3] accumulate [4,5,6] # [1,2,3,4,5,6]
```

among

- Left-hand operand: a int expression
- Right-hand operand: a list, string, point or matrix.
- Result: a list of length the value of the left-hand operand, containing random elements from the right-hand operand.

- Special cases: if the right-hand operand is empty, returns an empty list; if the left-hand operand is greater than the length of the right-hand operand, returns the right-hand operand.

```
3 among [1,2,3,4,5,6,7,8] # [3, 6, 7]
```

collect

- Left-hand operand: a list, matrix, string or point.
- Right-hand operand: an arbitrary expression.
- Result: a list containing the result of the right-hand expressions applied to each of the elements of the left-hand operand.
- Comments: the order of the elements is kept. In the right-hand operand, the keyword each can be used to represent, in turn, each of the elements.

```
[1,2,3,4] collect (each + 10) # [11, 12, 13, 14];
'abcd' collect (each > 'a') # [false, true, true, true];
```

copy_between

- Left-hand operand: a list or a string
- Right-hand operand: a point
- Result: a string or a list containing the elements of the left-hand operand between the indexes provided by the x-coordinate (inclusive) and the y-coordinate (exclusive) of the right-hand operand.
- Comments: the length of the resulting list or string is (second index - first index).
- Special cases: if the first index is less than the second, returns an empty list or string; if the first index is less than 0, it is set to 0; if the second index is greater than the length of the left-hand operand, it is set to this length.

```
[1,2,3,4,5,6,7,8] copy_between {1,3} # [2,3];
'abcdefgh' between {2,5} # 'cde';
```

count

- Left-hand operand: a list, string, matrix or point
- Right-hand operand: a boolean expression
- Result: an int, equal to the number of elements of the left-hand operand that make the right-hand operand evaluate to true.
- Comments: in the right-hand operand, the keyword each can be used to represent, in turn, each of the elements.

```
[1,2,3,4,5,6,7,8] count (each > 3) # 5;
(list species self) count ((first each.location)> 30)  # all the agents of the same species whose x-coordinate is greater than 30.
```

distance_to

- Left-hand operand: a point or an agent
- Right-hand operand: a point or an agent
- Result: a float, equal to the distance (in meters) between the two operands.
- Comments: in case a GIS environment is part of the model, the pathfinder is used to compute the distance; otherwise the distance in straight line is returned.
- Special cases: if one of the agents is not localized, returns 0.0.

```
((list species self) - self) collect (each distance_to self) # collects the distances between self and all the agents of the same species.
```

first_with

- Left-hand operand: a list, string, matrix or point
- Right-hand operand: a boolean expression
- Result: any value, equal to the first element of the left-hand operand that make the right-hand operand evaluate to true.
- Comments: in the right-hand operand, the keyword each can be used to represent, in turn, each of the elements.

```
[1,2,3,4,5,6,7,8] first_with (each > 3) # 4;
(list species self) first_with ((first each.location)> 30)  # the first agent of the same species whose x-coordinate is greater than 30.
```

in

- Left-hand operand: an expression or a list
- Right-hand operand: a list, string, matrix or point
- Result: if the left_hand operand is not a list, true if the left-hand operand is equal to one of the elements of the right-hand operand, otherwise false; if it is a list, returns true if all its elements are present in the right-hand operand, otherwise returns false;

```
3 in [1,2,3,4,5,6,7,8] # true;
[3, 10] in [1,2,3,4,5,6,7,8] # false;
'bc' in 'abcded' # true;
['a','b','cd'] in 'abcdef' # true;
```

index_of

- see: last_index_of.

last_index_of

- Left-hand operand: a list, string, matrix or point
- Right-hand operand: an expression
- Result: an int, equal to the index of the first (resp. last) occurence of the right-hand operand in the left-hand operand.
- Comments: if the right-hand operand is not present, returns -1.

```
[1,2,3,4,5,6,7,8] index_of 3 # 2;
'abcdefgh' index_of 'bcf' # -1;
```

max_of

- see: min_of.

min_of

- Left-hand operand: a list, string, matrix or point
- Right-hand operand: an in or float expression
- Result: an int or float, equal to the maximum (resp. minimum) value of the right-hand expression evaluated on each of the elements of the left-hand operand
- Comments: in the right-hand operand, the keyword each can be used to represent, in turn, each of the elements.

```
[1,2,10,4,7,6,7,8] max_of (each * 100) # 1000;
(list species self) min_of (first each.location))  # the smallest x-coordinate
of the agents of the same species as self.
```

neighbours_at

- Left-hand operand: an agent
- Right-hand operand: an int or float (expressing a distance in meters)
- Result: a list, containing all the agents located in the circle of radius equal to the right-hand operand, and center the location of the left-hand operand.
- Comments: if the left-hand operand is a grid cell, returns only grid cells of the same environment; if it is a regular agent, returns all the agents, whatever their species, save for grid cells.

- Special cases: if the left-hand operand is a not localized agent, returns an empty list.

```
(self neighbours_at (10)) of_species (species self) # all the agents of the same species situated at a distance greater or equal to 10 of self.
```

of_species

- Left-hand operand: a list (of agents)
- Right-hand operand: a species expression
- Result: a list, containing the agents of the left-hand operand whose species is that denoted by the right-hand operand.
- Comments: in the right-hand operand, the keyword each can be used to represent, in turn, each of the elements. The expression agents of_species (species self) is equivalent to agents where (species each = species self); however, the advantage of using the first syntax is that the resulting list is correctly typed with the right species, whereas, in the second syntax, the parser cannot determine the species of the agents within the list (resulting in the need to cast it explicitely if it is to be used in an ask statement, for instance).

```
(self neighbours_at 10) of_species (species self) # all the neighbouring agents of the same species.
```

sort_by

- Left-hand operand: a list, string, point, or matrix
- Right-hand operand: an int, float or string expression.
- Result: a list, containing the agents of the left-hand operand sorted in ascending order by the value of the right-hand operand when it is evaluated on them.
- Comments: the left-hand operand is casted to a list before applying the operator. Therefore, a species value can be used directly to represent all the agents of this species. In the right-hand operand, the keyword each can be used to represent, in turn, each of the elements.

```
(self neighbours_at 10) sort_by (first each.location) # all the neighbouring agents, sorted by their x-coordinate.
```

starts_with

- Left-hand operand: a string
- Right-hand operand: a string
- Result: true is the left-hand operand starts with the right-end operand.

- Comments: equivalent to (left-hand_operand [[[#index_of] right-hand_operand) = 0, but faster.

split_using

- see: tokenize.

tokenize

- Left-hand operand: a string
- Right-hand operand: a string (delimiters)
- Result: a list, containing the sub-strings (tokens) of the left-hand operand delimited by each of the characters of the right-hand operand.
- Comments: delimiters themselves are excluded from the resulting list.

```
'to be or not to be, that is the question' split_using ' ,' #
['to','be','or','not,'to','be','that','is','the','question'].
'to be or not to be, that is the question' tokenize ' ,' #
['to','be','or','not,'to','be','that','is','the','question'].
```

where

- see: select.

select

- Left-hand operand: a list, string, matrix or point
- Right-hand operand: a boolean expression
- Result: a list containing all the elements of the left-hand operand that make the right-hand operand evaluate to true.
- Comments: in the right-hand operand, the keyword each can be used to represent, in turn, each of the elements.

```
[1,2,3,4,5,6,7,8] select (each > 3) # [4, 5, 6, 7, 8];
(list species self) where ((first each.location)> 30)  # all the agents of the
same species whose x-coordinate is greater than 30;
'abcdef' select (each < 'd') # ['a', 'b','c'].
```

with_max_of

- see: with_min_of.

with_min_of

- Left-hand operand: a list, string, matrix or point
- Right-hand operand: an int, float or string expression
- Result: a list containing the elements of the left-hand operand that maximize (resp. minimize) the value of the right-hand operand.
- Comments: in the right-hand operand, the keyword each can be used to represent, in turn, each of the elements.

```
(list species self) with_max_of (each.var)  # all the agents of the same
species whose variable var is equal to the current maximum value of var
(equivalent to (list species self) where (each.var = max (list species self
collect (each.var))) -- only faster and easier to write);
['abc', 'bcde', 'fgh', 'fffde'] with_min_of (each.length) # ['abc','fgh'].
```

with_precision

- Left-hand operand: an int or float.
- Right-hand operand: an int.
- Result: round off the value of left-hand operand to the precision given by the value of right-hand operand.

```
12345.78943 with_precision 2 # 12345.79
123 with_precision 2 # 123.00
```

Ternary operators

? :

- Left-hand operand: a boolean expression
- Middle operand: an expression
- Right-hand operand: an expression
- Result: if the left-hand operand evaluates to true, returns the value of the middle operand, otherwise that of the right-hand operand

```
[10, 19, 43, 12, 7, 22] collect ((each > 20) ? 'above' : 'below') # ['below',
'below', 'above', 'below', 'below', 'above']
```

These functional tests can be combined together (here, the color variable takes three values with respect to the value of food, above 5, between 2 and 5, and below 2):

```
<set var="color" value="rgb ((food > 5) ? 'red' : ((food >= 2)? 'blue' :
'green'))"/>
```

Variables

<wiki:toc max_depth="3" />

Declaration

base

Except temporary variables, which are declared with their own syntax within behaviors or actions, all other variables are declared using the following one:

```
<var type="datatype" name="var_name" [optional_attributes="..."]* />
```

In this declaration, datatype refers to the name of a built-in type or a species declared in the model. The value of name can be any combination of letters and digits (plus the underscore, "_") that does not begin with a digit and that follows certain rules (see "Naming variables"). Examples of valid declarations are:

```
<var type="int" name="i"/>
<var type="list" name="my_list"/>
<var type="agent" name="an_agent"/>
<var type="my_species" name="an_agent_of_my_species" /> if my_species is
declared in the model as a species.
```

These variables are given default values at their creation, depending on their datatype:
Default Value

int	float	bool	string	list	matrix	point	rgb
0	0.0	false	''	[]	nil	nil	black

init

When it is necessary to initialize the variable with another value than its default value, the init attribute can be used.

```
<var type="datatype" name="var_name"
init="initial_expression" [optional_attributes="..."]* />
```

The initial_expression is expected to be of the same type as the variable (otherwise it is casted to the datatype). Its only (obvious) restriction is that it cannot refer to the variable being declared. Examples of valid declarations are:

```
<var type="int" name="i" init="0"/>
<var type="list" name="my_list" init="[i + 1, i + 2, i + 3]" />
<var type="agent" name="an_agent" init="self" />
```

const

If the value of the variable is not intended to change over time, it is preferable to declare it as a constant in order to avoid any surprise (and to optimize, a little bit, the compiler's work). This is done with the const attribute set to true (if const is not declared, it is considered as false by default):

```
<var type="datatype" name="var_name" init="initial_expression"
const="true" [optional_attributes="..."]* />
```

With this declaration, the variable var_name will keep the result of initial_expression as its value for the whole execution of the simulation.

value

What if, on the contrary, the value of the variable is supposed to change over time and the modeller wants to define this evolution ? The value attribute is precisely available for this purpose. Basically, its contents is evaluated every time step and assigned to the variable. It means that, unless the contents of this attribute refers to the variable itself, every modification made in the model to the value of the variable will be lost and replaced with the evaluation of the expression.

```
<var type="datatype" name="var_name" init="initial_expression"
value="value_expression" [optional_attributes="..."]* />
```

All the variables of all the agents are updated at the same time, before they are given a chance to execute behaviors. Some examples of use for value:

- Automatically evolving variables:

```
<var type="int" name="my_int" init="0" value= "my_int + 1"/> -> my_int is
incremented by 1 every time step.
<var type="float" name="my_float" init = "100" value="my_float - (my_float /
100)" /> -> my_float is decremented by 1% every time step.
```

- Sticky variables:

```
<var type="int" name="sticky_int" value="100"/> -> whatever the changes made
in the model to sticky_int, its value returns to 100 at the beginning of every
turn.
```

- Conditionally evolving variables:

```
<var type="int" name="cond_int" value="my_int < 100 ? 0 : my_int // 10" /> ->
the value of cond_int depends on that of my_int.
<var type="float" name="log_my_int" value="ln my_int" /> -> the value of
"cond_int" is always coupled to that of my_int.
```

Special attributes

parameter

This attribute can only be used in the context of global variables, i.e. variables declared in the world species in the global section. Indicates that the value of the variable will (or can) be defined by an external input : either a file or an optimization process (in the case of batch simulations), or the user (in the case of interactive simulations with a user interface). Makes **const** turns to false if it has been defined. For example, declaring:

```
<var type="int" name="max_energy" init="300" parameter="true"/>
```

will translate to this in the user interface: <br clear="all"/>
 [Image:100.png] <br clear="all"/>
 In several cases, this interface will allow the user to change the value of the variable during a simulation. If behaviors depend on it, the outcome of the simulation will then be affected by these changes, which can be a great way to manually and interactively explore the effect of parameters on a model. More on this in the presentation of the interface. The value of parameter can be used to name the variable on the interface. Any sequence of characters will do. If true is used, then the name of the variable itself is used for the label. Example:

```
<var type="int" name="max_energy" init="300" parameter="Maximum energy for the
agents"/>
```

max, min

These two attributes are only available in the context of int or float variables. They allow the modeler to control the values of the variable, by specifying a maximum and

minimum value. The variable will never be allowed to be lower than the minimum or greater than the maximum.

```
<var type="int" name="max_energy" init="300" min="100" max="3000" />
```

min and max combine gracefully with the parameter attribute and allow to control what the user can enter, or the limits between which exploring the values of variables. from For int variables, when declared in a "grid species". Not documented, because it will be added to the declaration of matrices instead (and removed from int). An example of the current use can be found in the "ants_from_file.xml" model.

of

Only defined in the context of matrix and list variables. Allows to define the type/species of values contained in the list. For instance, it can be handy, sometimes, to fix the species of the agents in a list at once rather than having to use the of_species operator every time. An example of that with the re-declaration of the built-in neighbours variable in a model with only one species of agents:

```
<var type="list" name="neighbours" of="species self" />
```

Doing so enables the use of neighbours, in the following expressions, without having to specify which kind of agents are manipulated in it.

Naming variables

Reserved Keywords

In GAML, some keywords are already reserved with a specific meaning and cannot be used for naming variables (and also species, actions, etc.). They are :

- The names of the global built-in variables
- The names of the primitive data types and new species defined in the model.
- The special keywords used by the language.
- The names of the variables found in every species.
- The names of the variables defined in skills when a species declares their use.
- The names of the units that can attached to numeric values.

Naming conventions

Alphanumeric characters can be use to variable name. No space is accepted.

Accessing variables

Direct access

Global variables, species variables and temporary variables declared in the same scope can be accessed directly by agents. For instance:

```
<species name="animal" skills="situated">
  <var type="float" name="energy" init="1000" min="0" max="2000" value="energy
- 0.001"/>
  <var type="int" name="age_in_years" init="1" value="age + int (time / (1
year))"/>
  <action name="eat">
      <arg name="amount" default="0"/>
      <let name="gain" type="float" value="amount / age_in_years"/>
      <set var="energy" value="energy + gain"/>
  </action>
</species>
```

In this declaration, we are able to directly name time, the global built-in variable, energy and age_in_years, which are defined as species variables, amount, which is an argument to the action eat and gain, a temporary variable within the action.

Remote access

When an agent needs to get access to the variable of another agent, a special notation (similar to that used in Java) has to be used:

```
remote_agent.variable
```

where remote_agent can be the name of an agent, an expression returning an agent, self, myself, context or each. For instance, if we modify the previous species by giving its agents the possibility to feed another agent found in its neighbourhood, the result would be:

```
<species name="animal" skills="situated">
  <var type="float" name="energy" init="1000" min="0" max="2000" value="energy
- 0.001"/>
  <var type="int" name="age_in_years" init="1" value="age_in_years + int
(time / (1 year))"/>
  <action name="eat">
      <arg name="amount" default="0"/>
      <let name="gain" type="float" value="amount / age_in_years"/>
      <set var="energy" value="energy + gain"/>
  </action>
  <action name="feed">
      <arg name="target"/> a target agent is passed as the argument of the
action
      <let name="agent_to_feed" type="animal" value="animal target"/> we only
feed agents of the same species
      <if condition="(agent_to_feed != nil) and (agent_to_feed.energy &lt;
energy)"> verifies that the agent exists and that it need to be fed
            <ask target="agent_to_feed">
                <do action="eat">
                    <arg name="amount" value="myself.energy / 10"/> asks the
agent to eat 10% of our own energy
                </do>
            </ask>
            <set var="energy" value="energy - (energy / 10)/> reduces the
energy by 10%
      </if>
  </action>
  <reflex>
      <let name="candidates" type="animal" value="neighbours of_species
(species_of self)"/> gathers all the neighbours
      <set var="agent_to_feed" value="one_of (candidates with_min_of
(each.energy))"/> grabs one agent with the lowest energy
      <do action="feed">
          <arg name="target" value="agent_to_feed"/> tries to feed it
      </do>
  </reflex>
</species>
```

In this example, agent_to_feed.energy, myself.energy and each.energy show different remote accesses to the variable energy. The dotted notation used here can be employed in assignments as well. For instance, an action allowing two agents to exchange their energy could be defined as:

```
<action name="random_exchange"> exchanges our energy with that of a random
neighbour
    <let name="one_agent" type="animal" value="one_of neighbours of_species
(species_of self)"/>
    <if condition="one_agent != nil">
        <let name="temp" type="float" value="one_agent.energy"/> temporary
storage of the agent's energy
```

```
        <set name="one_agent.energy" value="energy"/> assignment of the
agent's energy with our energy
        <set name="energy" value="temp"/>
    </if>
</action>
```

_

Keywords

<wiki:toc max_depth="3" />

constants

nil

Represents the null value (or undefined value). It is the default value of variables of type agent, point or species when they are not initialized.

true, false

Represent the two possible values of boolean variables or expressions. (see bool).

<font color="blue">pseudo-variables</ font>

Pseudo-variables are special variables whose value cannot be changed by agents (but can change depending on the context of execution).

self

self is a pseudo-variable (can be read, but not written) that always holds a reference to the executing agent.

- Example (sets the friend field of another random agent to self and reversely) :

```
<let var="potential_friend" value="((one_of species self) - self)" />
<if condition="potential_friend != nil">
    <set var="potential_friend.friend" value"self"/>
    <set var="friend" value="potential_friend" />
</if>
```

myself

myself is the same as self, except that it needs to be used instead of self in the definition of remotely-executed code (ask and |create commands). myself represents the sender agent when the code is executed by the target agent.

- Example (asks the first agent of my species to set its color to my color) :

```
<ask target="first (list species self)">
    <set var="color" value="myself.color"/>
</ask>
```

- Example (create 10 new agents of my species, share my energy between them, turn them towards me, and make them move 4 times to get closer to me) :

```
<create species="species self" number="10">
  <set var="energy" value="myself.energy / 10"/>
  <loop times="4">
      <set var="heading" value="towards myself" />
      <do action="move"/>
  </loop>
</create>
```

context

context is a pseudo-variable that represents the "environment" of the agent. Using this keyword prevents from explicitly having to access the individual cell(s) or environments on which it is located. The caveat is that an expression like context.var will look for var in all the environments defined, which can be costly.

each

each is a pseudo-variable only used in filter expressions following operators such as where or first_with. It represents, in turn, each of the elements of the target datatype (a list, string, point or matrix, usually).

units

Units can be used to qualify the values of numeric variables. By default, unqualified values are considered as:

- meters for distances, lengths...
- seconds for durations
- cubic meters for volumes
- kilograms for masses

So, an expression like

```
<float name="foo" value="1"/>
```

will be considered as 1 meter if foo is a distance, or 1 second if it is a duration, or 1 meter/second if it is a speed. If one wants to specify the unit, it can be done very simply by adding the unit name after the numeric value, like:

```
<float name="foo" value="1 centimeter"/>
```

In that case, the numeric value of foo will be automatically translated to 0.01 (meter). It is recommended to always use float as the type of the variables that might be qualified by units (otherwise, for example in the previous case, they might be truncated to 0). Several units names are allowed as qualifiers of numeric variables. As they can be used in expressions directly, they are considered as reserved keywords (and therefore cannot be used for naming variables and species). Their complete list is:

length

meter (default), meters, m, centimeters, centimeter, cm, millimeter, millimeters, mm, decimeter, decimeter, dm, kilometer, kilometers, km, mile, miles, yard, yards, inch, inches, foot, feet, ft.

time

second (default), seconds, sec, s, minute, minutes, mn, hour, hours, h, day, days, d, month, months, year, years, y, millisecond, milliseconds, msec.

mass

kilogram (default),kilogram, kilo, kg, ton, tons, t, gram, grams, g, ounce, ounces, oz, pound, pounds, lb, lbm.

surface

square_meter (default), m2, square_meters, square_mile, square_miles, sqmi, square_foot, square_feet, sqft.

volume

m3 (default), liter, liters, l, centiliter, centiliters, cl, deciliter, deciliters, dl, hectoliter, hectoliters, hl. These represent the basic metric and US units. Composed and derived units (like velocity, acceleration, special volumes or surfaces) can be obtained by combining these units using the **and / operators. For instance:**

```
<float name="one_kmh" init="1 km / h" const ="true"/>
<float name="one_microsecond" init="1 sec / 1000"/>
<float name="one_cubic_inch" init="1 sqin * 1 inch" />
... etc ...
```

Skills

<wiki:toc max_depth="3" />

Overview

Skills are built-in modules, written in Java, that provide a set of related built-in variables and built-in actions (in addition to those already provided by GAMA) to the species that declare them. A declaration of skill is done by filling the skills attribute in the species definition:

```
<species name="my_species" skills="skill1, skill2" />
    ...
</species>
```

Skills have been designed to be mutually compatible so that any combination of them will result in a functional species. The list of available skills in GAMA is:

- situated: for agents that are situated in (at least) one environment.
- moving: for agents that need to move.
- carrying: for agents that need to carry other agents.
- communicating: for agents that need to communicate using the FIPA protocols.
- exploring: for agents that need to explore a region of space.

So, for instance, if a species is declared as:

```
<species name="foo" skills="moving, carrying">
```

its agents will automatically be provided with the following variables : range, location, neighbours (r/o), geometry, area, perimeter (r/o), speed, heading, destination (r/o), capacity, contents(r/o) and the following actions: add_point, affinite, ..., triangulate, union, move, goto, wander, drop, load" in addition to those built-in in species and declared by the modeller. Most of these variables, except the ones marked read-only, can be customized and modified like normal variables by the modeller. For instance, one could want to set a maximum for the speed; this would be done by redeclaring it like this:

```
<var type="float" name="speed" max="100" min="0"/>
```

Or, to obtain ever growing agents:

```
<var type="float" name="area" max="100" value="area * 1.01"/>
```

Or, to change their capacity in a behavior:

```
<if condition="capacity = 5">
    <set var="capacity" value="10"/>
</if>
```

situated

variables

range

float, the distance at which agents can see other agents.

location

point, their position (centroid of their geometry) in the environment (see the coordinates system of the environment).

geometry

list, a continuously updated list representing their geometry : the geometry is represented by a list of simple geometries. Each simple geometry is composed of a first list of points that represents the geometry external ring and eventually by other lists of points representing the holes in the geometry. For a simple geometry, if the list of points of the external ring is composed of only 1 point, the resulting geometry is a **Point** , if the first point is equal to the last point, the geometry is a **Polygon** , otherwise the geometry is a **Polyline** .

area

float, area of their geometry.

perimeter

float, read-only, perimeter of their geometry.

neighbours

list, read-only, a continuously updated list of their neighbours (while considering the agent locations), with respect to the value of range: list of the agents of which the location is at a distance lowers than "range" to the agent location.

neighbours_geometry

list, read-only, a continuously updated list of their neighbours (while considering the agent geometries), with respect to the value of range: list of the agents of which the geometry is situated at a distance lowers than "range" to the agent geometry (minimal distance between the geometries).

actions

add_point

Adds a point to the agent geometry

- → point: point, the new point.

```
<do action="add_point">
    <arg name="point" value="{15,20}"/>
</do>
```

affinite

Applies a affinite operation (of a given angle in degrees and a given coefficient) to a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action). If a geometry is passed with the parameter **geometry** , this action returns the resulting geometry; otherwise, this action directly modifies the agent geometry.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.

- → coefficient: float, the coefficient of the scaling operation.
- → angle: float, the rotation angle.
- ← return: list, the resulting geometry (only in the context of a geometry passed with the parameter **geometry**).

```
<do action="affinite">
    <arg name="coefficient" value="0.9"/>
    <arg name="angle" value="45."/>
</do>
```

area_sum

Returns the sum of localized entity geometry areas (passed with the parameter **agents**).

- → agents: list, a list of localized entity, optional, a localized entity.
- ← return: float, the sum of the areas of the localized entity geometries.

```
<do action="area_sum" return="area_tot">
    <arg name="agents" value="[AgentA, AgentB, AgentC]"/>
</do>
```

area_union

Return the area of the geometry resulting from the union of localized entity geometries (passed with the parameter **agents**).

- → agents: list, a list of localized entity, optional, a localized entity.
- ← return: float, the area of the geometry resulting from the union of the localized entity geometries.

```
<do action="area_union" return="area_covered">
    <arg name="agents" value="[AgentA, AgentB, AgentC]"/>
</do>
```

basic_geometry

Returns a basic geometry which centroid is located at a given location.

- → geometry_type: string, 'circle', 'square', 'rectangle' or 'triangle'.
- → radius: float, the circle radius (for circle shape).
- → side_size: float, size of the side (for square and triangle shapes).
- → height: float, height of the rectangle(for rectangle shape).
- → width: float, width of the rectangle(for rectangle shape).

- → location: point, location of the geometry centroid.
- ← return: list,the basic geometry.

```
<do action="basic_geometry" return="geom">
    <arg name="geometry_type" value="'circle'"/>
    <arg name="radius" value="5."/>
    <arg name="location" value="{50,50}"/>
</do>
```

bounds_without

Returns a geometry resulting from the difference between the environnement bounds (minus a interior buffer of size buffer_in) and the geometries of the localized entities of the specified species (application of a buffer on these geometries of size buffer_others)

- → species: species, optional, a list of species (of localized entities);
- → buffer_others: float, optional, size of the buffer applied to the localized entity geometries;
- → buffer_in: float, optional, size of the "interior" buffer applied to the bounds
- ← return: list, geometry resulting from the action

```
<do action="bounds_without" return="geom">
    <arg name="species" value="[speciesA, speciesB]"/>
    <arg name="buffer_others" value="10.5"/>
    <arg name="buffer_in" value="2.5"/>
</do>
```

buffer

Returns the geometry resulting from the application of a buffer on a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action).

- → buffer_size: float, size of the buffer.
- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: list, the geometry resulting from the buffer

```
<do action="buffer" return="geom_buff">
    <arg name="buffer_size" value="2.5"/>
</do>
```

build_basic_geometry

Sets the geometry of the agent with a basic geometry

- → geometry_type: string, 'circle', 'square', 'rectangle' or 'triangle'.
- → radius: float, the circle radius (for circle shape).
- → side_size: float, size of the side (for square and triangle shapes).
- → height: float, height of the rectangle(for rectangle shape).
- → width: float, width of the rectangle(for rectangle shape).
- → location: point, location of the geometry centroid.

```
<do action="build_basic_geometry">
    <arg name="geometry_type" value="'circle'"/>
    <arg name="radius" value="5."/>
    <arg name="location" value="{50,50}"/>
</do>
```

build_geometry_from_agents

Sets the geometry of the agent with a geometry built from a list of localized entity locations (if the number of localized entities is equal 1, then the geometry is a point, if the number of localized entities is equal to 2, the geometry is a polyline, otherwise, the geometry is a polygon).

- → agents: list, the list of localized entities.
- → convex_hull: bool, optional, true: sets the agent geometry as the convex hull of the built geometry.
- → coordinates: float, optional, applies a buffer of **buffer_size** to the built geometry.

```
<do action="build_geometry_from_agents">
    <arg name="agents" value="[agentA, agentB, agentC]"/>
    <arg name="convex_hull" value="true"/>
    <arg name="buffer_size" value="2.5"/>
</do>
```

build_geometry_from_places

Sets the geometry of the agent with a geometry built from a list of grid cells (if the number of grid cells is equal 1, then the geometry is a point, if the number of grid cells is equal to 2, the geometry is a polyline, otherwise, the geometry is a polygon).

- → places: list, the list of grid cells.

- → convex_hull: bool, optional, true: sets the agent geometry as the convex hull of the built geometry.
- → coordinates: float, optional, applies a buffer of **buffer_size** to the built geometry.

```
<do action="build_geometry_from_places">
    <arg name="places" value="[cell1, cell2, cell8, cell10]"/>
    <arg name="convex_hull" value="true"/>
    <arg name="buffer_size" value="2.5"/>
</do>
```

build_simple_geometry_from_list

Sets the geometry of the agent with a geometry built from a list of points.

- → coordinates: list, the list of points.
- → convex_hull: bool, optional, true: sets the agent geometry as the convex hull of the built geometry.
- → coordinates: float, optional, applies a buffer of **buffer_size** to the built geometry.

```
<do action="build_simple_geometry_from_list">
    <arg name="coordinates" value="[{5,5}, {5,95}, {95,95}, {95,5}, {5,5}]"/>
    <arg name="convex_hull" value="true"/>
    <arg name="buffer_size" value="2.5"/>
</do>
```

closest_point_in

Returns the point of a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**) that is the closest to the agent location.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: point, the closest point of the geometry to the agent location

```
<do action="closest_point_in" return="close_pt">
    <arg name="agent" value="agentA"/>
</do>
```

compute_graph

Computes and saves graph built from one or several geometry. If the geometries are polygons, the graph is built from their triangulation. For one geometry: the geometry considered is the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action. For several geometries: the geometries considered are the ones of the localized entities passed with the parameters **triangles** , **agents** or **network** (**triangles** for localized entities with triangle geometries (used for optimization), **agents** for localized entities with polygon geometries; **network** for localized entities with polylines geometries).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → agents: list, optional, a list of localized entities (with polygon geometries).
- → triangles: list, optional, a list of localized entities (with triangle geometries).
- → network: list, optional, a list of localized entities (with polylines geometries).
- → split_lines: boolean, optional (true if not specified), only used in the case of a network of lines: cut the lines at their intersections.
- → name: string, label of the matrix (see action **goto** of the **moving** skill).
- → optimizer_type: string, optional, type of optimizer used to computed the shortest pathes : 'dynamic', 'progressive' or 'static', by default, 'static' (all of the shortest pathes are computed during the **compute_graph** action).
- → weight: string, optional, the name of an attribute (of the agents considered), allows to modify the weight of each edge for the shortest path computation.

```
<do action="compute_graph">
    <arg name="networks" value="[roadA, roadB, roadC, roadD, roadE]"/>
    <arg name="name" value="'graph_road'"/>
    <arg name="optimizer_type" value="'progressive'"/>
    <arg name="weight" value="'speed_coeff'"/>
</do>
```

compute_matrix

Computes and saves the matrix resulting from the square discretisation of a geometry (the one passed with the parameter **geometry** , or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → name: string, label of the matrix (see action **goto** of the **moving** skill).

- → square_size: float, side size of the squares used for the discretisation.

```
<do action="compute_matrix">
    <arg name="agent" value="agentA"/>
    <arg name="name" value="'matrix_agA'"/>
    <arg name="square_size" value="20."/>
</do>
```

convex_hull

Returns the convex hull of a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: list, the convex hull (geometry)

```
<do action="convex_hull" return="convex_hull"/>
```

covers_pt

Indicates if the agent geometry covers the point passed in parameter

- → point: point, the point.
- ← return: bool, true: the agent geometry covers the point

```
<do action="covers" return="is_covered">
    <arg name="point" value="{10,20}"/>
</do>
```

crosses

Indicates if the agent geometry crosses a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: bool, true: the geometries cross

```
<do action="crosses" return="is_crossing">
    <arg name="agent" value="agentA"/>
</do>
```

difference

Returns the geometry resulting from the difference of two geometries (geometry1 - geometry2 or agent1.geometry - agent2.geometry). If geometry1 and agent1 are not specified, agent1 = agent applying the action; If geometry2 and agent2 are not specified, agent2 = agent applying the action.

- → geometry1: optional, the first geometry.
- → agent1: optional, the first localized entity.
- → geometry2: optional, the second geometry.
- → agent2: optional, the second localized entity.
- ← return: list, the resulting geometry

```
<do action="difference" return="geom_diff">
    <arg name="agent1" value="agentA"/>
    <arg name="agent2" value="agentB"/>
</do>
```

disjoint

Indicates if the agent geometry is disjoint with a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: bool, true: the geometries are dijoint

```
<do action="disjoint" return="is_disjoint">
    <arg name="agent" value="agentA"/>
</do>
```

distance_geometry

Returns the (minimal) distance between the agent geometry and ta geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: float, the minimal distance between the two geometries

```
<do action="distance_geometry" return="dist">
```

```
    <arg name="agent" value="agentA"/>
</do>
```

distance_graph

Returns the distance of the shortest path between the agent and a target according to a saved graph (see action **compute_graph**).

- → target: point or localized entity, the target location.
- → graph_name: string, the name of the saved graph.
- ← return: float, the distance of the shortest path.

```
<do action="distance_graph" return="min_dist">
    <arg name="target" value="AgentA"/>
    <arg name="graph_name" value="'road_network'"/>
</do>
```

exterior_ring

Returns the exterior ring of a geometry (the one passed with the parameter **geometry** , or if not specified, the geometry of the localized entity passed with the parameter **agent** or if not specified, the geometry of the agent applying the action). If the geometry is not a **Polygon** , the action returns the geometry.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: list, geometry, the exterior ring (a polyline).

```
<do action="exterior_ring" return="exterior_ring">
    <arg name="agent" value="AgentA"/>
</do>
```

interior

Returns the interior geometry that at least at a distance **buffer_in** of the geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** or if not specified, the bounds geometry) exterior ring and of the interior rings

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → buffer_in: float, size of the "interior" buffer applied to the geometry (exterior and interior rings)

- ← return: list, geometry resulting from the action

```
<do action="interior" return="geom">
    <arg name="agent" value="AgentA"/>
    <arg name="buffer_in" value="2.5"/>
</do>
```

intersection

Returns the geometry resulting from the intersection between the agent geometry and a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**); return null if the resulting geometry area is lower than **min_area**

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → min_area: float, optional, equals to 0 if not specified.
- ← return: float, the minimal distance between the two geometries

```
<do action="intersection" return="geom_intersect">
    <arg name="agent" value="agentA"/>
    <arg name="min_area" value="100"/>
</do>
```

is_contained_in

Indicates if the agent geometry is contained in a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, the localized entity.
- ← return: bool, true: the agent geometry is contained in the localized entity geometry

```
<do action="is_contained_in" return="is_conainted">
    <arg name="agent" value="agentA"/>
</do>
```

is_in_bounds

Indicates if a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified

a point passed with the parameter **point** , or if not specified, the geometry of the agent agent applying the action) is (totally) contained in the bounds.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, the localized entity.
- → point: point, optional, the point.
- ← return: bool, true: the geometry is totally contained in the bounds

```
<do action="is_in_bounds" return="is_contained">
    <arg name="agent" value="agentA"/>
</do>
```

neighbourhood_exclusive

Returns a geometry resulting from the difference between a geometry representing the exterior ring of the agent geometry (ring : geometry.buffer(distance) - geometry.buffer(buffer_in)) and the geometries of the localized entities of the specified species (application of a buffer on these geometries of size buffer_others)

- → distance: float, distance considered for the neighborhood
- → species: list, optional, a list of species (of localized entities);
- → buffer_others: float, optional, size of the buffer applied to the localized entity geometries;
- → buffer_in: float, optional, size of the "interior" buffer applied to the geometry
- ← return: list, the resulting geometry

```
<do action="neighbourhood_exclusive" return="overlapping_ags">
    <arg name="distance" value="20."/>
    <arg name="species" value="[speciesA, speciesB]"/>
    <arg name="buffer_in" value="5."/>
    <arg name="buffer_others" value="2.5"/>
</do>
```

overlapping_agents

Returns all the agents of the specified species (if no specified species, all agents) of which the geometry overlaps the agent geometry

- → species : species, optional, a species of localized entities.
- → buffer_size: float, optional, size of the buffer applied to the agent geometry
- ← return: list, the list of agents (of the specified species) overlapping the agent geometry

```
<do action="overlapping_agents" return="overlapping_ags">
```

```
    <arg name="species" value="speciesA"/>
    <arg name="buffer_size" value="5"/>
</do>
```

overlapping_area

Returns the area of the geometry resulting from the intersection between the agent geometry and a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: float, area of the geometry resulting from the intersection

```
<do action="overlapping_area" return="area_overlap">
    <arg name="agent" value="agentA"/>
</do>
```

overlaps

Indicates if the agent geometry overlaps a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: bool, true: the geometries overlap

```
<do action="overlaps" return="is_overlapping">
    <arg name="agent" value="agentA"/>
</do>
```

partially_overlaps

Indicates if the agent geometry partially overlaps a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: bool, true: the geometries partially overlap (if one of the geometries is contained in the other, the action returns false).

```
<do action="partially_overlaps" return="is_partially_overlapping">
```

```
    <arg name="agent" value="agentA"/>
</do>
```

path_graph

Returns the best path between the agent location and the target location using a saved graph (see action **compute_graph**). The path can takes the form of a list of agents crosses or of a list of points (if the parameter returns_points is set as true).

- → graph_name: string, mandatory, the name of the saved graph.
- → target: point or agent, mandatory, the location or entity corresponding to the final destination of the path built.
- → return_points: bool, optional, by default equals false, if true the function returns a list of points instead of a list of localized entities.
- ← return: list, the list of agents crossed or the list of points composing the best path.

```
<do action="path_graph">
    <arg name="graph_name" value="'background'"/>
    <arg name="target" value="agentA"/>
    <arg name="returns_points" value="true"/>
</do>
```

percieved_area

Returns the geometry percieved by an agent consiering a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**): the agent can not see through the holes of the geometry.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → range: float, optional, perception range. if not specified, this parameter is equal to the value of the **range** variables.
- → precision: int, precision of the percieved area computation. The higher, the more precise the computed percieved area is, but the more computation the action requires.
- ← return: list, geometry, the percieved area.

```
<do action="percieved_area" return="geom_percieved">
    <arg name="agent" value="backgroundAg"/>
    <arg name="range" value="10.0"/>
    <arg name="precision" value="30"/>
</do>
```

place_in

Returns a point situated in the geometry (at least at a distance buffer_in of the geometry exterior and interior rings) resulting from the difference of the geometry passed in parameter (or geometry of the given localized entity or if not specified of the agent applying the action) and the geometry of the localized entities of the specified species (application of a buffer on these geometries of size **buffer_others**)

- $\rightarrow$ geometry: list, optional, a geometry.
- $\rightarrow$ agent: localized entity, optional, a localized entity.
- $\rightarrow$ species: species, optional, a list of species (of localized entities).
- $\rightarrow$ buffer_others: float, optional, size of the buffer applied to the localized entity geometries (of the specified species);
- $\rightarrow$ buffer_in: float, optional, size of the "interior" buffer applied to the geometry (exterior and interior rings)
- $\leftarrow$ return: point, a point situated inside the resulting geometry

```
<do action="place_in" return="point_inside">
    <arg name="species" value="[speciesA, speciesB]"/>
    <arg name="buffer_others" value="2.5"/>
    <arg name="buffer_in" value="5"/>
</do>
```

points_at

Returns a list of equidistant points located at a distance **distance** of the agent location

- $\rightarrow$ distance: float, distance to the agent location
- $\rightarrow$ nb_points: int, number of points returns
- $\rightarrow$ check_in_bounds: bool, optional, true: remove the points that are not contained in the environment bounds
- $\leftarrow$ return: list, list of points

```
<do action="points_at" return="points">
    <arg name="distance" value="2.5"/>
    <arg name="nb_points" value="10"/>
    <arg name="check_in_bounds" value="true"/>
</do>
```

points_exterior_ring

Returns a list of equidistant points situated on the exterior ring of a geometry (the one passed with the parameter **geometry** , or if not specified, the geometry of the localized

entity passed with the parameter **agent** or if not specified, the geometry of the agent applying the action). The distance between two points is given by the parameters **distance** .

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → distance: float, distance between two points (distance computed while considering the exterior ring; not the euclidian distance).
- ← return: list, the list of points.

```
<do action="points_exterior_ring" return="pts_ext">
    <arg name="agent" value="AgentA"/>
    <arg name="distance" value="10.0"/>
</do>
```

remove_geometry_parts

Applies a difference operation between the agent geometry and the geometries of a list of localized entities (with on optional buffer) buffer_size ->

- → agents: list, list of localized entities.
- → buffer_size: float, optional, size of the buffer applied on the geometries.
- → simple_geom: bool, optional, true: if the resulting geometry is complex (list of simple geometries), set the agent geometry as the simple geometry with the biggest area.

```
<do action="remove_geometry_parts" return="points">
    <arg name="agents" value="[agentA, agentB, agentC]"/>
    <arg name="buffer_size" value="5."/>
    <arg name="simple_geom" value="true"/>
</do>
```

rotation

Applies a rotation operation (of a given angle in degrees) to a geometry (the one passed with the parameter **geometry** , or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action). If a geometry is passed with the parameter **geometry** , this action returns the resulting geometry; otherwise, this action directly modifies the agent geometry.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → angle: float, the rotation angle.

- ← return: list, the resulting geometry (only in the context of a geometry passed with the parameter **geometry**).

```
<do action="rotation">
    <arg name="angle" value="45."/>
</do>
```

skeletonization

Returns a list of lines (geometries) resulting from the skeletonization of the agent geometry (or of a given geometry or localized entity geometry)

- → geometry: list, optional, a geometry;
- → agent: localized entity, optional, a localized entities;
- ← return: list, list of lines (geometries)

```
<do action="skeletonization" return="list_lines">
    <arg name="agent" value="agentA"/>
</do>
```

scaling

Applies a scaling operation (of a given coefficient) to a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action). If a geometry is passed with the parameter **geometry** , this action returns the resulting geometry; otherwise, this action directly modifies the agent geometry.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → coefficient: float, the coefficient of the scaling operation.
- ← return: list, the resulting geometry (only in the context of a geometry passed with the parameter **geometry**).

```
<do action="scaling">
    <arg name="coefficient" value="0.9"/>
</do>
```

touches

Indicates if the agent geometry touches a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- ← return: bool, true: the geometries touch

```
<do action="overlaps" return="is_touching">
    <arg name="agent" value="agentA"/>
</do>
```

translation

Applies a translation operation (of a given vector (dx, dy)) geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent** , or if not specified the geometry of the agent applying the action). If a geometry is passed with the parameter **geometry** , this action returns the resulting geometry; otherwise, this action directly modifies the agent geometry.

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → dx: float, x composant of the translation vector.
- → dy: float, y composant of the translation vector.
- ← return: list, the resulting geometry (only in the context of a geometry passed with the parameter **geometry**).

```
<do action="translation ">
    <arg name="dx" value="30."/>
    <arg name="dy" value="-5."/>
</do>
```

triangulate

Returns a list of triangles (geometries) resulting from the triangulation (or of a given geometry or localized entity geometry)

- → geometry: list, optional, a geometry;
- → agent: localized entity, optional, a localized entities;
- ← return: list, list of triangles(geometries)

```
<do action="triangulate" return="list_triangles">
    <arg name="agent" value="agentA"/>
</do>
```

union

Returns the geometry resulting from the union of geometries (agent geometry union with a a geometry (the one passed with the parameter **geometry** or if not specified, the geometry of the localized entity passed with the parameter **agent**) or union of a list of agent geometries (passed with the parameter **agents**).

- → geometry: list, optional, a geometry.
- → agent: localized entity, optional, a localized entity.
- → agents: list, optional, a list of localized entities.
- → convex_hull: boolean, optional, true: returns the convex hull of the resulting geometry
- → without_holes: boolean, optional, true: removes the holes of the resulting geometry
- → buffer_size: float, optional, add a buffer of **buffer_size** size to the agent geometries
- ← return: list, the resulting geometry

```
<do action="union" return="geom_union">
    <arg name="agent" value="agentA"/>
    <arg name="convex_hull" value="true"/>
    <arg name="without_holes" value="true"/>
    <arg name="buffer_size" value="2.5"/>
</do>
```

update_graph

Updates the edge value of a saved graph (see action **compute_graph**).

- → graph_name: string, the name of the saved graph.
- → agents: list, optional, a list of localized entities (with polygon geometries) used to build the graph.
- → network: list, optional, a list of localized entities (with polylines geometries).
- → optimizer_type: string, optional, type of optimizer used to computed the shortest pathes : 'dynamic', 'progressive' or 'static', by default, 'static' (all of the shortest pathes are computed during the **compute_graph** action).
- → weight: string, optional, the name of an attribute (of the agents considered), allows to modify the weight of each edge for the shortest path computation.

```
<do action="update_graph">
    <arg name="graph_name" value="'background'"/>
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="optimizer_type" value="'progressive'"/>
    <arg name="weight" value="'congestion_coeff'"/>
```

```
</do>
```

moving

This skill implicitly adds the situated skill to the species.

variables

speed

float, the speed of the agent, in meter/second.

heading

int, the absolute heading of the agent in degrees (in the range 0-359).

destination

point, read-only, continuously updated destination of the agent with respect to its speed and heading.

actions

move

moves the agent forward, the distance being computed with respect to its speed and heading. The value of the corresponding variables are used unless arguments are passed.

- → speed: float, optional, the speed to use for this move (replaces the current value of speed).
- → heading: int, optional, the direction to take for this move (replaces the current value of heading).
- → distance: float, optional, the distance at which to move (is checked against the speed of the agent for compatibility, may change speed if necessary).
- → geometry: list, optional, list of points, the geometry that restrains this move (the agent moves inside this geometry).

- → agent: localized entity, optional, the geometry (the localized entity geometry) that restrains this move (the agent moves inside this geometry).
- ← return: point, the new location of the agent (or nil if an error occurred).

```
<do action="move" return="new_location">
    <arg name="speed" value="speed - 10"/>
    <arg name="heading" value="heading + rnd 30"/>
    <arg name="distance" value="100" />
    <arg name="agent" value="agentA"/>
</do>
```

wander

moves the agent towards a random location at the maximum distance (with respect to its speed). The heading of the agent is chosen randomly if no amplitude is specified. This action changes the value of heading.

- → speed: float, optional, the speed to use for this move (replaces the current value of speed).
- → amplitude: int, optional, a restriction placed on the random heading choice. The new heading is chosen in the range (heading - amplitude/2, heading+amplitude/2).
- → distance: float, optional, the distance at which to move (is checked against the speed of the agent for compatibility, may change speed if necessary).
- → geometry: list, optional, list of points, the geometry that restrains this move (the agent moves inside this geometry).
- → agent: localized entity, optional, the geometry (the localized entity geometry) that restrains this move (the agent moves inside this geometry).
- ← return: point, the new location of the agent (or nil if an error occurred).

```
<do action="wander" return="new_location">
    <arg name="speed" value="speed - 10"/>
    <arg name="amplitude" value="120"/>
    <arg name="distance" value="100" />
    <arg name="agent" value="agentA"/>
</do>
```

goto

moves the agent towards the target passed in the arguments.

- → target: point or agent, mandatory, the location or entity towards which to move.
- → speed: float, optional, the speed to use for this move (replaces the current value of speed).

- → geometry: list, optional, list of points, the geometry that restrains this move (the agent moves inside this geometry).
- → agent: localized entity, optional, the geometry (the localized entity geometry) that restrains this move (the agent moves inside this geometry).
- → triangulation: bool, optional, in the context of a movement restrained inside a polygon, the method used for the decomposition of the geometry in convex sub-parts: **true** : triangulation, **false** : square discretization .
- → square_size: float, optional, the size of the square (in the context of a square discretization).
- → graph_name: string, optional, name of a saved graph (see action **compute_graph** of the **situated** skill).
- → matrix_name: string, optional, name of a saved matrix (see action **compute_matrix** of the **situated** skill).
- ← return: point, the new location of the agent (or nil if an error occurred).

```
<do action="goto" return="new_location">
    <arg name="target" value="one_of (list species self)"/>
    <arg name="speed" value="speed * 2"/>
    <arg name="graph_name" value="'road_network'"/>
</do>
```

carrying

variables

capacity

int, the maximum number of agents that can be carried

contents

list, read-only, the current list of agents carried.

actions

drop

makes the agent leave, at its location, all or some of the agents it was carrying.

- → agents: list, optional, the list of agents to drop. If not specified, all the agents carried are dropped.
- ← return: list, the list of agents actually dropped.

```
<do action="drop">
    <arg name="agents" value="contents of_species foo"/>
</do>
```

load

makes the agent pick & carry all or some of the agents specified by the arguments.

- → agents: list, optional, the list of agents to load. If not specified, the action is ignored.
- → number: int, optional, the number of agents to load from the list. If not specified, all the agents are loaded until the capacity is reached.
- ← return: list, the list of agents actually loaded.

```
<do action="load" result="loaded_agents">
    <arg name="agents" value="neighbours of_species foo"/>
    <arg name="number" value="10"/>
</do>
```

<font color="blue">communicating </ font>

This skill implements the major FIPA communication and negotiation protocols and enable the agents to communicate using them. It brings along two new datatypes : message and conversation. GAMA implements the following interaction protocols of FIPA : FIPA Brokering, FIPA Contract Net, FIPA Propose, FIPA Query, FIPA Request, FIPA Request When, FIPA Subscribe. Besides the standard interaction protocols of FIPA, GAMA support a special interaction protocol named "no-protocol".

variables

conversations

list, read-only, the conversation in which the agent is currently engaged.

messages

list, read-only, the messages received by the agent.

accept_proposals, agrees, cancels, cfps, failures, informs, proposes, queries, refuses, reject_proposals, requests, requestWhens, subscribes

list, read-only, the messages received by the agent, broken down by performative.

actions

send

allows the agent to send an existing message, or to create one and send it immediately. Either use the argument:

- → message: message, optional, the message to send.

Or a combination of the following arguments:

- → receivers: list, mandatory, the list of agents to send the message to.
- → content: list, mandatory, the content of the message. Can be anything.
- → performative: string, mandatory. The performative of the message (see this page)
- → protocol: string, mandatory, the protocol followed by this message (see this page)
- → conversation: conversation, optional, the conversation in which the message should be inserted. If nil, creates a new conversation.
- ← return: message, the message actually sent or nil if an error has occurred.

```
<create species="message" number="1" return="mes">
    <set name="receivers" value="neighbours" />
    <set name="content" value="['Hello !']" />
    <set name="protocol" value="'no-protocol'" />
    <set name="performative" value="'inform'" />
</create>
<do action="send">
    <arg name="message" value="mes"/>
</do>
```

```
<do action="send" return="sent_message">
```

```
    <arg name="receivers" value="neighbours" />
    <arg name="content" value="['Hello !']" />
    <arg name="protocol" value="'no-protocol'" />
    <arg name="performative" value="'inform'" />
</do>
```

end, failure, inform, propose, query, refuse, reject-proposal, request, subscribe

replies a list of messages with the corresponding performatives "end", "failure", "inform", "propose", "query", "refuse", "reject-proposal", "request", "subscribe". The replied messages will be filled automatically with the corresponding performatives.

- $\rightarrow$ message: list, mandatory, messages to reply to.
- $\rightarrow$ content: list, optional, the content of the replied messages.

```
<list name="noProtocolInformMsgs"  value="(informs) where ((species
(each.sender) = foodCarrier) and (each.protocol = 'no-protocol'))"/>
<do action="end">
    <arg name="message" value="noProtocolInformMsgs" />
</do>
```

data types

message

An instance of message represents a piece of information exchanged between agents. It has the following attributes :

- sender : agent, the sender of the message.
- receivers : a list of agents, contains all the receivers of the message.
- performative : string, the performative of the corresponding message. For more details concerning the performative, please refer to the FIPA Communicative Act Library Specification.
- content : list, contains the content of the corresponding message. The modeller can put whatever content in this list.

conversation

An instance of conversation represents a FIPA interaction protocol. It helps ensure that a conversation between agents respects the specification of a corresponding protocol defined by FIPA. It has the following attributes :

- messages : list, the currently unread messages of the conversation.
- protocol : string, the name of the interaction protocol that this conversation respects.
- initiator : agent, the agent which initiates the conversation/the interaction protocol.
- participants : a list of agents, all the participants of the conversation (except for the initiator).
- ended : bool, helps determine if the corresponding conversation/interaction protocol has already finished or not yet.

Notes

GAMA supports a special interaction protocol named "no-protocol". If the modeller finds that no supported FIPA Interaction Protocols fits his modelling case, he can use the "no-protocol". Uppon declaring a conversation following this procotol, the modeller is free to send messages having whatever performative between agents. But it 's upto the modeller to finish up the corresponding conversation by using the "end" primitive.

exploring

This skill implements the patrolling capability of agents, enables the agents to go on patrol in a list of given points.

variables

points

list, the list of points to patrol. This list must not be empty before the agents can begin its patrolling activity.

patrolling

bool, determines if the corresponding agent is going on patrol or not.

actions

patrol

do the patrolling activity if the list of points is not empty.

- → speed: float, optional, the speed of the corresponding agent.

```
<task name="patrol_the_terrain" while="patrolTheTerrain">
  <priority if="patrolTheTerrain" value="10" else="0"/>
  <duration max="2 hours"/>
  <repeat action="patrol">
    <arg name="speed" value="maxSpeed"/>
  </repeat>
</task>
```

—

Built-in Agents

<wiki:toc max_depth="3" />

Introduction

It is possible to use in the models a set of built-in agents. These agents allow to directly use some advance features like clustering, multi-criteria analysis, etc. The creation of these agents are similar as for other kinds of agents:

```
<create species="my_built_in_agent" return="the_agent" />
```

The list of available built-in agents in GAMA is:

- cluster_builder: allows to use clustering techniques on a set of agents.
- multicriteria_analyzer: allows to use multi-criteria analysis methods.
- random_builder: allows to use advance features concerning the generation of random numbers.

So, for instance, to be able to use clustering techniques in the model:

```
<create species="cluster_builder" return="clusterer" />
```

cluster_builder

The **cluster_builder** agent allows to divide a set of agents into different clusters according to the values of some of their attributes. This agents is built from the weka library : most of the clustering algorithms are directlty based on their weka implementation.

actions

simple_clustering_by_distance

Returns groups of agents using hierarchical clustering. The distance between agents are directly computed from the agent locations (euclidean distance). The distance between two groups of agents corresponds to the min of distances between the agents of each group.

- → agents: list, the list of Entities to be divided into groups.
- → dist_min: float, minimal distance between two groups. By default, num_clusters = -1.
- → distance_geom: bool, optional, if true, uses the distance between the agent geometry; otherwise uses the distance between the agent centroid (more optimize). By default, distance_geom = false.
- ← return: list, a list of groups; each group is a list of agents.

```
<do action="simple_clustering_by_distance" return="groups">
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="dist_min" value="10.0"/>
</do>
```

clustering_cobweb

Returns groups of agents using the Cobweb and Classit clustering algorithms (Weka implementation). For more information see: D. Fisher (1987). Knowledge acquisition via incremental conceptual clustering. Machine Learning. 2(2):139-172; J. H. Gennari, P. Langley, D. Fisher (1990). Models of incremental concept formation. Artificial Intelligence. 40:11-61.

- → agents: list, the list of Entities to be divided into groups.
- → attributes: list, the list of the agent attributes (string: the attribute names) used to divide the agents into groups. For the moment, only numeric attributes are considered.
- → acuity: float, optional, minimum standard deviation for numeric attributes. By default, acuity = 1.0.
- → cutt_off: float, optional, set the category utility threshold by which to prune nodes. By default, cutoff = 0.0028209479177387815.
- → seed, int, optional, The random number seed to be used. By default, seed = 42.
- ← return: list, a list of groups; each group is a list of agents.

```
<do action="clustering_cobweb" return="groups">
```

```
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="attributes"
value="['area','food_quantity','proximity_to_roads']"/>
</do>
```

clustering_DBScan

Returns groups of agents using the DBScan algorithm (Weka implementation). For more information see: Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: Second International Conference on Knowledge Discovery and Data Mining, 226-231, 1996.

- → agents: list, the list of Entities to be divided into groups.
- → attributes: list, the list of the agent attributes (string: the attribute names) used to divide the agents into groups. For the moment, only numeric attributes are considered.
- → distance_f: string, optional, The distance function to use : 2 possible distance functions: (by default) euclidean; and 'manhattan'.
- → epsilon: float, optional, radius of the epsilon-range-queries. By default, epsilon = 0.9.
- → min_points: int, optional, minimun number of [DataObjects] required in an epsilon-range-query. By default, min_points = 6.
- ← return: list, a list of groups; each group is a list of agents.

```
<do action="clustering_DBScan" return="groups">
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="attributes"
value="['area','food_quantity','proximity_to_roads']"/>
</do>
```

clustering_em

Returns groups of agents using the EM (expectation maximisation) algorithm (Weka implementation).

- → agents: list, the list of Entities to be divided into groups.
- → attributes: list, the list of the agent attributes (string: the attribute names) used to divide the agents into groups. For the moment, only numeric attributes are considered.
- → max_iterations: int, optional, the maximum number of iterations to perform. By default, max_iterations = 100.

- → num_clusters: int, optional, set number of clusters. if num_clusters equals -1, EM decides how many clusters to create by cross validation. By default, num_clusters = -1.
- → min_std_dev: float, optional, set minimum allowable standard deviation. By default, min_std_dev = 1.0E-6.
- → seed, int, optional, The random number seed to be used. By default, seed = 100.
- ← return: list, a list of groups; each group is a list of agents.

```
<do action="clustering_em" return="groups">
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="attributes"
value="['area','food_quantity','proximity_to_roads']"/>
</do>
```

clustering_farthestFirst

Returns groups of agents using the farthestFirst algorithm (Weka implementation). For more information see: Hochbaum, Shmoys (1985). A best possible heuristic for the k-center problem. Mathematics of Operations Research. 10(2):180-184.

- → agents: list, the list of Entities to be divided into groups.
- → attributes: list, the list of the agent attributes (string: the attribute names) used to divide the agents into groups. For the moment, only numeric attributes are considered.
- → num_clusters: int, optional, set number of clusters. By default, num_clusters = 2.
- → seed, int, optional, The random number seed to be used. By default, seed = 1.
- ← return: list, a list of groups; each group is a list of agents.

```
<do action="clustering_farthestFirst" return="groups">
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="attributes"
value="['area','food_quantity','proximity_to_roads']"/>
</do>
```

clustering_simple_kmeans

Returns groups of agents using the K-Means algorithm (Weka implementation).

- → agents: list, the list of Entities to be divided into groups.
- → attributes: list, the list of the agent attributes (string: the attribute names) used to divide the agents into groups. For the moment, only numeric attributes are considered.

- → distance_f: string, optional, The distance function to use : 4 possible distance functions: (by default) euclidean; otherwise, 'chebyshev', 'manhattan' and 'levenshtein'.
- → dont_replace_missing_values: bool, optional, Replace missing values globally with mean/mode. By default, dont_replace_missing_values = false.
- → max_iterations: int, optional, the maximum number of iterations to perform. By default, max_iterations = 500.
- → num_clusters: int, optional, set number of clusters. By default, num_clusters = 2.
- → preserve_instances_order: bool, optional, Preserve order of instances. By default, preserve_instances_order = false.
- → seed, int, optional, The random number seed to be used. By default, seed = 10.
- ← return: list, a list of groups; each group is a list of agents.

```
<do action="clustering_simple_kmeans" return="groups">
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="attributes"
value="['area','food_quantity','proximity_to_roads']"/>
</do>
```

clustering_xmeans

Returns groups of agents using the X-Means algorithm (Weka implementation). "X-Means is K-Means extended by an Improve-Structure part. In this part of the algorithm the centers are attempted to be split in its region. The decision between the children of each center and itself is done comparing the BIC-values of the two structures. For more information see: Dan Pelleg, Andrew W. Moore: X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In: Seventeenth International Conference on Machine Learning, 727-734, 2000." (Weka documentation).

- → agents: list, the list of Entities to be divided into groups.
- → attributes: list, the list of the agent attributes (string: the attribute names) used to divide the agents into groups. For the moment, only numeric attributes are considered.
- → bin_value: float, optional, Set the value that represents true in the new attributes. By default, bin_value = 1.0.
- → cut_off_factor: float, optional, the cut-off factor to use. By default, cut_off_factor = 0.5.
- → distance_f: string, optional, The distance function to use : 4 possible distance functions: (by default) euclidean; otherwise, 'chebyshev', 'manhattan' and 'levenshtein'
- → max_iterations: int, optional, the maximum number of iterations to perform. By default, max_iterations = 1.

- → max_kmeans: int, optional, the maximum number of iterations to perform in KMeans. By default, max_iterations = 1000.
- → max_kmeans_for_children: int, optional, the maximum number of iterations KMeans that is performed on the child centers. By default, max_kmeans_for_children = 1000.
- → max_num_clusters: int, optional, set maximum number of clusters. By default, max_num_clusters = 4.
- → min_num_clusters: int, optional, set minimum number of clusters. By default, min_num_clusters = 2.
- → seed, int, optional, The random number seed to be used. By default, seed = 10.
- ← return: list, a list of groups; each group is a list of agents.

```
<do action="clustering_xmeans" return="groups">
    <arg name="agents" value="[agentA, agentB, agentC, agentD, agentE]"/>
    <arg name="attributes"
value="['area','food_quantity','proximity_to_roads']"/>
</do>
```

multicriteria_analyzer

The **multicriteria_analyzer** agent allows to make a decision from a set of candidate solutions according to a set of criteria.

actions

weighted_means_DM

Returns the index of the candidate with the highest utility. The utility of each candidate is computed by a weighted means.

- → criteria: list, the list of criteria. A criterion is a map that contains two elements: a name, and a weight.
- → candidates: list, the list of candidates. A candidate is a list of floats; each float representing the value of a criterion for this candidate.
- ← return: int, the index of the selected candidate.

```
<do action="weighted_means_DM" return="index_cand">
```

```
    <arg name="criteria" value="[[name::'proximity', weight::1],
[name::'quality', weight::3], [name::'usefulness', weight::2]]"/>
    <arg name="candidates" value="[[0.8, 0.1, 0.3],[0.5, 0.7, 0.5],[0.1, 0.5,
0.9],[0.9, 0.2, 0.4],[0.6, 0.5, 0.5],[0.7, 0.4, 0.3]]"/>
</do>
```

promethee_DM

Returns the index of the *best* candidate according to the Promethee II method. This method is based on a comparison per pair of possible candidates along each criterion: all candidates are compared to each other by pair and ranked. More information about this method can be found in Behzadian, M., Kazemzadeh, R., Albadvi, A., M., A.: PROMETHEE: A comprehensive literature review on methodologies and applications. European Journal of Operational Research(2009)

- → criteria: list, the list of criteria. A criterion is a map that contains fours elements: a name, a weight, a preference value (p) and an indifference value (q). The preference value represents the threshold from which the difference between two criterion values allows to prefer one vector of values over another. The indifference value represents the threshold from which the difference between two criterion values is considered significant.
- → candidates: list, the list of candidates. A candidate is a list of floats; each float representing the value of a criterion for this candidate.
- ← return: int, the index of the selected candidate.

```
<do action="promethee_DM" return="index_cand">
    <arg name="criteria" value="[[name::'proximity', weight::1, p::0.9,
q::0.1], [name::'quality', weight::3, p::1.0, q::0.0],
    [name::'usefulness', weight::2, p::0.8, q::0.2]]"/>
    <arg name="candidates" value="[[0.8, 0.1, 0.3],[0.5, 0.7, 0.5],[0.1, 0.5,
0.9],[0.9, 0.2, 0.4],[0.6, 0.5, 0.5],[0.7, 0.4, 0.3]]"/>
</do>
```

electre_DM

Returns the index of the *best* candidate according to a method based on the ELECTRE methods. The principle of the ELECTRE methods is to compare the possible candidates by pair. These methods analyses the possible outranking relation existing between two candidates. An candidate outranks another if this one is at least as good as the other one. The ELECTRE methods are based on two concepts: the concordance and the discordance. The concordance characterises the fact that, for an outranking relation to be validated, a sufficient majority of criteria should be in favor of this assertion. The discordance characterises the fact that, for an outranking relation to be validated, none of the criteria in the minority should oppose too strongly this assertion. These two

conditions must be true for validating the outranking assertion. More information about the ELECTRE methods can be found in Figueira, J., Mousseau, V., Roy, B.: ELECTRE Methods. In: Figueira, J., Greco, S., and Ehrgott, M., (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Springer, New York, 133--162 (2005)

- → criteria: list, the list of criteria. A criterion is a map that contains fives elements: a name, a weight, a preference value (p), an indifference value (q) and a veto value (v). The preference value represents the threshold from which the difference between two criterion values allows to prefer one vector of values over another. The indifference value represents the threshold from which the difference between two criterion values is considered significant. The veto value represents the threshold from which the difference between two criterion values disqualifies the candidate that obtained the smaller value.
- → candidates: list, the list of candidates. A candidate is a list of floats; each float representing the value of a criterion for this candidate.
- ← return: int, the index of the selected candidate.

```
<do action="electre_DM" return="index_cand">
    <arg name="criteria" value="[[name::'proximity', weight::1, p::0.9,
q::0.1, v::0.95], [name::'quality', weight::3, p::0.8, q::0.0, v:1.0],
    [name::'usefulness', weight::2, p::0.8, q::0.2, v::0.9]]"/>
    <arg name="candidates" value="[[0.8, 0.1, 0.3],[0.5, 0.7, 0.5],[0.1, 0.5,
0.9],[0.9, 0.2, 0.4],[0.6, 0.5, 0.5],[0.7, 0.4, 0.3]]"/>
</do>
```

evidence_theory_DM

Returns the index of the *best* candidate according to a method based on the Evidence theory. This theory, which was proposed by Shafer (Shafer G (1976) A mathematical theory of evidence, Princeton University Press), is based on the work of Dempster (Dempster A (1967) Upper and lower probabilities induced by multivalued mapping. Annals of Mathematical Statistics, vol. 38, pp. 325--339) on lower and upper probability distributions.

- → criteria: list, the list of criteria. A criterion is a map that contains seven elements: a name, a first threshold s1, a second threshold s2, a value for the assertion "this candidate is the best" at threshold s1 (v1p), a value for the assertion "this candidate is the best" at threshold s2 (v2p), a value for the assertion "this candidate is not the best" at threshold s1 (v1c), a value for the assertion "this candidate is not the best" at threshold s2 (v2c). v1p, v2p, v1c and v2c have to been defined in order that: v1p + v1c < = 1.0; v2p + v2c < = 1.0.
- → candidates: list, the list of candidates. A candidate is a list of floats; each float representing the value of a criterion for this candidate.

- ← return: int, the index of the selected candidate.

```
<do action="evidence_theory_DM" return="index_cand">
    <arg name="criteria" value="[[name::'proximity', s1::0.1, s2::0.8, v1p::0,
v2p::0.3, v1c::0.5, v2c::0],
    [name::'quality',s1::0.0, s2::0.8, v1p::0.05, v2p::0.5, v1c::0.6, v2c::0],
[name::'usefulness', s1::0.0, s2::0.8, v1p::0, v2p::0.2, v1c::0.9, v2c::0]]"/>
    <arg name="candidates" value="[[0.8, 0.1, 0.3],[0.5, 0.7, 0.5],[0.1, 0.5,
0.9],[0.9, 0.2, 0.4],[0.6, 0.5, 0.5],[0.7, 0.4, 0.3]]"/>
</do>
```

random_builder

The **random_builder** agent allows to draw random number according to a specific distribution.

actions

random_gaussian

Returns a random number following a specific gaussian distribution.

- → mean: float, the mean of the distribution.
- → sd: float, the standard deviation.
- ← return: float, a random number.

```
<do action="random_gaussian" return="rand_val">
    <arg name="mean" value="5.0"/>
    <arg name="sd" value="1.5"/>
</do>
```

random_poisson

Returns a random number following a specific poisson distribution.

- → mean: float, the mean of the distribution.
- ← return: float, a random number.

```
<do action="random_poisson" return="rand_val">
    <arg name="mean" value="5.0"/>
```

```
</do>
```

—

Built-in Items

<wiki:toc max_depth="3" />

Introduction

Species, when declared in a model, inherit some of their attributes (variables, actions) from the Java class that back them behind the scene. For "regular" agents, the number of these attributes is voluntarily limited. In a sense, that class acts exactly like a basic skill, and gives the agents a (very) limited number of common capabilities. However, some agents in the model are based on more specialized Java classes: this is the case of the world, which provides some critical global knowledge and functions to the modeller and the agents. This is also the case of the environmental agents defined by grids. In this section, we start by examining the built-in variables and actions common to all the agents. We then describe the global ones defined on the world. For information about the ones defined on grids, follow this link. Built-in variables

name

string. Each agent has a default name (concatenation of its species name and unique index), which can be changed at will to something more useful for the modeler (if needed).

Note

other "built-in" read-only attributes can be accessed through special operators:

- species or species_of, which return the species of the agent,
- int, which returns its index;

or keywords:

- self and myself, which return the agent itself.

Built-in actions

Three built-in actions are provided to the agents.

write

makes the agent output an arbitrary message in the console.

- → message: string, mandatory, the message to display. Modelers can add some formatting characters to the message (carriage returns, tabs, or Unicode characters), which will be used accordingly in the console.

```
<do action="write">
    <arg name="message" value="'This is a message from ' + self"/>
</do>
```

debug

makes the agent output an arbitrary message in the console. The message is automatically prefixed with the cycle of the simulation and followed by a carriage return and postfixed with information concerning the agent that called this action.

- → message: string, mandatory, the message to display.

```
<do action="debug">
    <arg name="message" value="'This is a message from ' + self"/>
</do>
```

error

makes the agent output an error dialog (if the simulation contains a user interface). Otherwise displays the error in the console.

- → message: string, mandatory, the message to display.

```
<do action="error">
    <arg name="message" value="'This is an error raised by ' + self"/>
</do>
```

tell

makes the agent output a dialog (if the simulation contains a user interface). The simulation goes on, but its interface is not accessible until the dialog is closed (use with caution, as it may prevent the user from accessing the interface).

- $\rightarrow$ message: string, mandatory, the message to display.

```
<do action="tell">
    <arg name="message" value="'This is a message dialog raised by ' + self"/>
</do>
```

Global built-in variables

Global variables can be accessed by the world and every other agent in the model.

time

int, read-only, represents the current simulated time in seconds (the default unit). Begins at zero. Although time is a read-only variable, it is possible to control its maximum value by redeclaring it. When the maximum is reached during a simulation, this simulation is automatically stopped.

```
<global>
...
    <int name="time" max="1000"/>
...
</global>
```

step

int, represents the time step between two executions of the set of agents, in seconds. Its default value is 1. Each turn, the value of time is incremented by the value of step. The definition of step must be coherent with that of the agents' variables like speed.

```
<global>
...
    <int name="step" value="10"/>
...
```

```
</global>
```

seed

float, represents the seed used in the computation of random numbers. Keeping the same seed between two runs of the same model ensures that the sequence of events will remain the same, which can be useful when debugging a model. Declaring it as a parameter allows the user or an external process (batch, for instance) to modify it.

```
<global>
...
    <int name="seed" value="354.0" parameter="true"/>
...
</global>
```

places

list, read-only, returns a list of all the "places" of the model (i.e. all the cells of the different grids that have been defined).

```
<species name="foo" skills="moving, visible">
    <agent name="goal" value="one_of places" />
</species>
```

agents

list, read-only, returns a list of all the agents of the model that are considered as "active" (i.e. all the agents with behaviors, excluding the places). Note that obtaining this list can be quite time consuming, as the world has to go through all the species and get their agents before assembling the result. For instance, instead of writing something like:

```
<ask target = "agent of_species my_species">
...
</ask>
```

one would prefer to write (which is much faster):

```
<ask target="list my_species">
...
</ask>
```

Global built-in actions

halt

stops the simulation.

```
<global>
    ...
    <reflex when="length agents <= 0">
            <do action="halt"/>
    </reflex>
</global>
```

pause

pauses the simulation, which can then be continued by the user.

```
<global>
    ...
    <reflex when="time = 100">
            <do action="pause"/>
    </reflex>
</global>
```

diffuse

diffuses the value of a variable (normally of type float) in a grid environment. Diffusion here means that every cell of the grid distributes to its neighbors a proportion of the amount of the variable's value.

- → var: string, mandatory. The name of the variable to diffuse.
- → proportion: float, mandatory. The proportion to diffuse, between 0 and 1. This proportion is divided by the number of neighbors of the cell in order to obtain the amount distributed to each cell.
- → environment: string, mandatory (optional if only one grid environment is defined). The environment in which to diffuse the variable.

```
<global>
    ...
    <reflex>
```

```
            <do action="diffuse"/>
    </reflex>
</global>
```

_

Batch

Definition

The batch allows to execute numerous successive simulation runs. It is used to explore the parameter space of a model or to optimize a set of model parameters. To use the batch, you must add the <batch>...</batch> section in your model file:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<mymodel>
    <modelClass name="msi.gama.kernel.Simulation" />
    <global>
        ...
    </global>
    <environment width="gridsize" height="gridsize">
        ...
    </environment>
    <entities>
        ...
    </entities>
    <output>
        ...
    </output>
    <batch sameSeed="true" repeat="3" stopSimWhen="time > 200">
        <param name="parameter" min="0.05" max="0.7" step="0.01" />
        <param name="another_parameter" values="'a','b','c'" />
        <method name="annealing" temp_init="100" temp_end="1"
temp_decrease="0.5" nb_iter_cst_temp="5" minimize="time">
        <file name="ant" rewrite="false" />
    </batch>
</mymodel>
```

The batch element

The <batch ... > element takes the following three attributes:

- stopSimWhen: (expression) Specifies when to stop each simulations. Its value is a condition on variables defined in the model. The run will stop when the condition is evaluated to true. If omitted, the first simulation run will go forever, preventing any subsequent run to take place (unless a halt command is used in the model itself).
- repeat: (integer) A parameter configuration corresponds to a set of values assigned to each parameter. The attribute repeat specifies the number of times each

configuration will be repeated, meaning that as many simulations will be run with the same parameter values. Different random seeds are given to the pseudo-random number generator. This allows to get some statistical power from the experiments conducted. Default value is 1.
- sameSeed: (boolean) If true, the same series of random seeds will be used from one parameter configuration to another. Default value is false.

The param elements

The <param ... /> elements specifies which model parameters will change through the successive simulations. The attribute name is mandatory and must refer to a variable declared as a parameter in the model (a variable with the attribute parameter present. Note: it seems that the value of the attribute doesn't matter.). There are 2 ways to define a number type (int or float) parameter:

- Explicit List: < param name="parameter" values="'0.6, 0.75, 0.8, 0.85, 1'" / >
- List with step: < param name="parameter" min="0.05" max="0.7" step="0.01" / >

For Strings and Booleans, you can only use the Explicit List. For every List definition, you may add a random attribute as follows:

```
<param name="parameter" values="'0.6, 0.75, 0.8, 0.85, 1'" random="3"/>
```

This defines the parameter as a list of length 3 built with elements taken randomly from the list defined either by the value attribute or by the 3 attributes min, max and step. Each Batch methods may accept only some kind of definitions and parameter types. See the description of each of them for details. The file element The <file ... /> element allows to output detailed information about the successive simulations to a file. The file will contain, for each simulation run, the value assigned to the model parameters and the value of the expression specified in the data attribute (if specified). The file format used for the encoding is the CSV (See the wikipedia article). Here is how the attributes go:

- name: (string) The data will be output to a file named name.txt. What to do when such a file already exists is specified by the attribute rewrite.
- rewrite: (boolean) If true, a pre-existing file with an identical name will be overwritten. Otherwise, the date will be added to the filename and the data will be output to this new file.
- data: (Expression) Optional; the value of this expression is added to each line of the log file (i.e. each successive simulation). In the context of an optimization process, by default, this expression is equal to the fitness expression.

The method element

If this element is omitted, the batch will run in a classical way, changing one parameter at each step until all the possible combinations of parameter values have been covered. See #Exhaustive exploration of the parameter space for details. The optional method element controls the algorithm which drives the batch. It must contain at least a name attribute to specify the algorithm to use, and for an optimization method, a minimize or a maximize attribute defining the expression to be optimized. Each combination of parameter values is tested repeat times. The fitness of one combination is the average of fitness values obtained with those repetitions. There might be additional attributes for tuning the exploration algorithm. See below for a description of the available methods. It is possible to add a new batch method by writing an appropriate new Java class. See How_to_write_a_batch_method.

Batch Methods

A few batch methods are currently available. Each is described below. It is possible to add a new batch method by writing an appropriate new Java class. See How_to_write_a_batch_method.

Exhaustive exploration of the parameter space

Parameter definitions accepted: List with step and Explicit List. Parameter type accepted: all. This is the standard batch method. The exhaustive mode is defined by default when there is no method element present in the batch section. It explores all the combination of parameter values in a sequential way. Example (models/ants/batch/ant_exhaustive_batch.xml)

```
<batch repeat="2" sameSeed="true" stopSimWhen="(food_remaining = 0) or (time >
400)">
    <param name="evaporation_rate" values="'0.1, 0.2, 0.5, 0.8, 1.0'" />
    <param name="diffusion_rate" min="0.1" max="1.0" step="0.3" />
    <file name="ant_exhaustive" data="time" rewrite="false" />
</batch>
```

The order of the simulations depends on the order of the param. In our example, the first combinations will be the followings:

- evaporation_rate = 0.1, diffusion_rate = 0.1, (2 times)
- evaporation_rate = 0.1, diffusion_rate = 0.4, (2 times)
- evaporation_rate = 0.1, diffusion_rate = 0.7, (2 times)
- evaporation_rate = 0.1, diffusion_rate = 1.0, (2 times)
- evaporation_rate = 0.2, diffusion_rate = 0.1, (2 times)
- ...

Note: this method can also be used for optimization by adding an method element with maximize or a minimize attribute:

```
<batch repeat="2" sameSeed="true" stopSimWhen="(food_remaining = 0) or (time > 400)">
    <param name="evaporation_rate" values="'0.1, 0.2, 0.5, 0.8, 1.0'" />
    <param name="diffusion_rate" min="0.1" max="1.0" step="0.3" />
    <method name="exhaustive" minimize="time" />
    <file name="ant_exhaustive" rewrite="false" />
</batch>
```

Hill Climbing

Name: hill_climbing Parameter definitions accepted: List with step and Explicit List. Parameter type accepted: all. This algorithm is an implementation of the Hill Climbing algorithm. See the wikipedia article. Algorithm:

```
Initialization of an initial solution s
iter = 0
While iter <= iter_max, do:
  Choice of the solution s' in the neighborhood of s that maximize the fitness function
  If f(s') > f(s)
    s = s’
  Else
    end of the search process
  EndIf
  iter = iter + 1
EndWhile
```

Method parameters:

- iter_max: number of iterations

Example (models/ants/batch/ant_hill_climbing_batch.xml):

```
<batch sameSeed="true" repeat="3" stopSimWhen="time > 400">
    <param name="evaporation_rate" min="0.05" max="0.7" step="0.01"/>
    <param name="diffusion_rate" min="0" max="1" step="0.01"/>
    <method name="hill_climbing" iter_max="50" />
```

```
    <file name="ant_hill_climbing" rewrite="false" />
</batch>
```

Simulated Annealing

Name: annealing Parameter definitions accepted: List with step and Explicit List. Parameter type accepted: all. This algorithm is an implementation of the Simulated Annealing algorithm. See the wikipedia article. Algorithm:

```
Initialization of an initial solution s
temp = temp_init
While temp > temp_end, do:
  iter = 0
  While iter < nb_iter_cst_temp, do:
    Random choice of a solution s' in the neighborhood of s
    df = f(s’)-f(s)
    If df > 0
      s = s’
    Else,
      rand = random number between 0 and 1
      If rand > exp(df/T)
        s = s’
      EndIf
    EndIf
    iter = iter + 1
  EndWhile
EndWhile
```

Method parameters:

- temp_init: Initial temperature
- temp_end: Final temperature
- temp_decrease: Temperature decrease coefficient
- nb_iter_cst_temp: Number of iterations per level of temperature

Example (models/ants/batch/ant_simulated_annealing_batch.xml):

```
<batch sameSeed="true" repeat="3" stopSimWhen="time > 400">
    <param name="evaporation_rate" min="0.05" max="0.7" step="0.01"/>
    <param name="diffusion_rate" min="0" max="1" step="0.01"/>
    <method name="annealing" temp_init="100" temp_end="1" temp_decrease="0.5"
nb_iter_cst_temp="5" minimize="time" />
    <file name="ant_simulated_annealing" rewrite="false" />
</batch>
```

Tabu Search

Name: tabu Parameter definitions accepted: List with step and Explicit List. Parameter type accepted: all. This algorithm is an implementation of the Tabu Search algorithm. See the wikipedia article. Algorithm:

```
Initialization of an initial solution s
tabuList = {}
iter = 0
While iter <= iter_max, do:
  Choice of the solution s' in the neighborhood of s such that:
    s' is not in tabuList
    the fitness function is maximal for s'
  s = s’
  If size of tabuList = tabu_list_size
    removing of the oldest solution in tabuList
  EndIf
  tabuList = tabuList + s
  iter = iter + 1
EndWhile
```

Method parameters:

- iter_max: number of iterations
- tabu_list_size: size of the tabu list

Example (models/ants/batch/ant_tabu_search_batch.xml):

```
<batch sameSeed="true" repeat="3" stopSimWhen="time > 400">
    <param name="evaporation_rate" min="0.05" max="0.7" step="0.01"/>
    <param name="diffusion_rate" min="0" max="1" step="0.01"/>
    <method name="tabu" iter_max="50" tabu_list_size="5" minimize="time"/>
    <file name="ant_tabu_search" rewrite="false" />
</batch>
```

Reactive Tabu Search

Name: reactive_tabu Parameter definitions accepted: List with step and Explicit List. Parameter type accepted: all. This algorithm is a simple implementation of the Reactive Tabu Search algorithm ((Battiti et al., 1993)). This Reactive Tabu Search is an enhance version of the Tabu search. It adds two new elements to the classic Tabu Search. The first one concerns the size of the tabu list: in the Reactive Tabu Search, this one is not constant anymore but it dynamically evolves according to the context. Thus, when the exploration process visits too often the same solutions, the tabu list is extended in order to favor the diversification of the search process. On the other hand, when the process

has not visited an already known solution for a high number of iterations, the tabu list is shortened in order to favor the intensification of the search process. The second new element concerns the adding of cycle detection capacities. Thus, when a cycle is detected, the process applies random movements in order to break the cycle. Method parameters:

- iter_max: number of iterations
- tabu_list_size_init: initial size of the tabu list
- tabu_list_size_min: minimal size of the tabu list
- tabu_list_size_max: maximal size of the tabu list
- nb_tests_wthout_col_max: number of movements without collision before shortening the tabu list
- cycle_size_min: minimal size of the considered cycles
- cycle_size_max: maximal size of the considered cycles

Example (models/ants/batch/ant_tabu_search_reactive_batch.xml):

```
<batch sameSeed="true" repeat="3" stopSimWhen="time > 400">
    <param name="evaporation_rate" min="0.05" max="0.7" step="0.01"/>
    <param name="diffusion_rate" min="0" max="1" step="0.01"/>
    <method name="reactive_tabu" iter_max="50" tabu_list_size_init="5"
tabu_list_size_min="2" tabu_list_size_max="10" nb_tests_wthout_col_max="20"
cycle_size_min="2" cycle_size_max="20" minimize="time" />
    <file name="ant_tabu_search_reactive" rewrite="false" />
</batch>
```

Genetic Algorithm

Name: genetic Parameter definitions accepted: List with step and Explicit List. Parameter type accepted: all. This is a simple implementation of Genetic Algorithms (GA). See the wikipedia article. The principle of GA is to search an optimal solution by applying evolution operators on an initial population of solutions There are three types of evolution operators:

- Crossover: Two solutions are combined in order to produce new solutions
- Mutation: a solution is modified
- Selection: only a part of the population is kept. Different techniques can be applied for this selection. Most of them are based on the solution quality (fitness).

Representation of the solutions:

- Individual solution: {Param1 = val1; Param2 = val2; …}
- Gene: Parami = vali

Initial population building: the system builds nb_prelim_gen random initial populations composed of pop_dim individual solutions. Then, the best pop_dim solutions are selected to be part of the initial population. Selection operator: roulette-wheel selection: the probability to choose a solution is equals to: fitness(solution)/ Sum of the population fitness. A solution can be selected several times. Ex: population composed of 3 solutions with fitness (that we want to maximize) 1, 4 and 5. Their probability to be chosen is equals to 0.1, 0.4 and 0.5. Mutation operator: The value of one parameter is modified. Ex: The solution {Param1 = 3; Param2 = 2} can mute to {Param1 = 3; Param2 = 4} Crossover operator: A cut point is randomly selected and two new solutions are built by taking the half of each parent solution. Ex: let {Param1 = 4; Param2 = 1} and {Param1 = 2; Param2 = 3} be two solutions. The crossover operator builds two new solutions: {Param1 = 2; Param2 = 1} and {Param1 = 4; Param2 = 3}. Method parameters:

- pop_dim: size of the population (number of individual solutions)
- crossover_prob: crossover probability between two individual solutions
- mutation_prob: mutation probability for an individual solution
- nb_prelim_gen: number of random populations used to build the initial population
- max_gen: number of generations

Example (models/ants/batch/ant_genetic_algorithm_batch.xml):

```
<batch sameSeed="true" repeat="3" stopSimWhen="time > 400">
    <param name="evaporation_rate" min="0.05" max="0.7" step="0.01"/>
    <param name="diffusion_rate" min="0" max="1" step="0.01"/>
    <method name="genetic" pop_dim="5" crossover_prob="0.7"
mutation_prob="0.1" nb_prelim_gen="1" max_gen="20" minimize="time"/>
    <file name="ant_genetic_algorithm" rewrite="false" />
</batch>
```

How to write a batch method

Introduction

Adding new search methods in GAMA is very simple. We described hereafter the two steps to follow.

Step 1

The first step consists in defining a Java class that extends the abstract class [ParamSpaceExploAlgorithm] Two methods must be implemented:

- public void parametrize(Map < String, String > parameters): method called at the initialization of the search method. It allows to load the search method parameter values (according to what the user has defined in GAML). All the parameters are of the String type.
- public Solution findBestSolution(): main search method. It returns the best solution found.

(Protected) Attributes of the [ParamSpaceExploAlgorithm] class:

- simulation: Simulation, current simulation.
- fitnessFct: [FitnessFunction] , object that allows to evaluate the fitness of a solution (fitness(Solution sol))
- testedSolutions: Map < Solution, Double > , map containing all the tested solutions and their associated fitness. Each time the fitness method of the [FitnessFunction] class is triggered, the result is saved in this map.
- variables: [HashMap]< String, [BatchVariable]> , map containing the variables (parameters that we explore). Key: name of the variable - > Value: [BatchVariable] object representing the variable (its current value, its possible values, its neighbor values, etc.).

Step 2

The adding of new search method in GAMA is automatic. It is just necessary to check that the package in which the class is defined is contained in the String table exploAlgoPackages of the [BatchManager] class. The GAML name of the search method is its class name

—

Training sessions GAMA 1.3

Training session GAMA 1.3 for epidemiology (St Louis)

<wiki:toc max_depth="3" />

Context

Training session on agent-based modeling and GAMA 1.3 for epidemiology organized at the Université Gaston Berger (Saint Louis, Senegal) the 26th of July, 2010.

List of presentations

Here the list of the different presentations (in French) given during the training session:

1. Introduction
2. Introduction ABM
3. Tutoriel - base
4. Tutoriel - appronfondissement
5. Tutoriel - donnees SIG
6. Futur de GAMA

Training session GAMA 1.3 (Bondy)

<wiki:toc max_depth="3" />

Context

Training session on GAMA 1.3 at IRD Bondy (France) the 1st and 5th of October, 2010.

List of presentations

Here, the list of the different presentations (in French) given during the training session:

1. Introduction à GAMA
2. GAMA et SIG
3. TP - Epidémiologie
4. TP - Schelling
5. Futur de GAMA

Models

Here , the models used during the training session.

—

Tutorial 1: The Stupid model

Stupid Model GAMA 1.3

<wiki:toc max_depth="3" />

Introduction

This tutorial is based on the article: StupidModel and Extensions: A template and teaching tool for agent-based modeling platforms by Railsback, Lytinen and Grimm. It is particularly well fitted for this purpose as the complexity increase smoothly and it allows to compare GAMA to other well known platforms such as Mason, Netlogo, Swarm (java & objective-C): (see here for implementations).

Preliminary notes

- Any uncertainity that have appeared within the formulation of the previously cited paper have been dealt using the netlogo implementation.

Stupid model: models list

Here the list of the different models that you will build while following the tutorial. This list is the same as in the original paper though some intermediary steps have been added in some cases. For each model we will present its purpose and an explicit formulation (from the original authors) and possibly some disambiguation or speficities due to the GAMA platform.

1. Basic stupidModel
2. Bug growth
3. Habitat cells and ressource
4. Cell and bug probes
5. Parameters and parameters displays

6. Histogram output
7. Stopping the model
8. File output
9. Randomized agent actions
10. Sorted agent actions
11. Optimal movement
12. Bug mortality and reproduction
13. Population abundance graph
14. Random normal initial size
15. Habitat data frome file input
16. Predators

—

Basic stupidModel

<wiki:toc max_depth="3" />

Purpose

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

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.

Models

Model 0 : the minimal set

On the user interface of GAMA, we create a new model, name it stupidmodel.xml:

```
<stupidmodel>
</stupidmodel>
```

Now we have a good starting point for the stupid model which we will incrementally develop, by adding GAML code, in responding to the model formulation.

Model 1.1 : the environment and display output

We are going to defined a 100x100 toroidal environment and a display output to display the environment in the user interface.

```
<stupidmodel>
  <environment>
    <grid name="stupid_grid" width="100" height="100"  torus="true"/>
  </environment>
  <output>
    <display name="stupid_display">
      <grid name="stupid_grid"/>
    </display>
  </output>
</stupidmodel>
```

Model 1.2 : Defining the agents

Here we have to define the structure of the bug agents then their behaviour:

- What is a bug agent?
  o It is a situated agent (on the default grid) thanks to the associated skill. o It has a red color and a circle shape defined using the **aspect** section.

```
<entities>
  <species name="bug" skills="situated">

      <aspect name="basic">
        <draw shape="circle" color="rgb 'red'" size="1" />
      </aspect>
  </species>
</entities>
```

Here we add an entities section containing the definition of a species. A species is the prototype of an agent.

- What is the behaviour of a bug?
  o It select a destination cell or 'place' within a distance of 4 cells where there is no agent. o it stays at the same cell only if there is no neighbour place empty.

```
<reflex name="basic_move">
  <let var="place" with="stupid_grid location"/>
  <let var="destination" with="one_of ((place neighbours_at 4) where (empty 
each.agents))"/>
    <if condition="destination != nil">
        <set var="location" value="destination"/>
```

```
    </if>
</reflex>
```

We add a **reflex** within the **species** section which declare the behaviour of the bug. This **reflex** will be executed at each time step by the agent. First we declare a temporary variable, using the let command, to hold the current cell of the agent calling it 'place', to do so we cast explicitly our location, built-in variable of the situated skill, into a cell by using the name of the environment, "stupid_grid". This cast is quite powerful as it translate a coordinate (location) into a cell, you will discover several powerful cast like this later on. Then we declare the 'destination' as a cell which is one of the empty (agents=[]) neighbour ‘place'. Finally we check that the 'destination' variable is not null and the agent moves, if 'destination' is null (which means that all neighbour are already full) it stays where it is.

Model 1.3 : Instantiating bugs

- How to instantiate the 100 bugs?
  o As we have no information they will be placed randomly by the system. o We introduce here the global section which is responsible to hold global variables and process global action.

```
  <global>
      <init>
        <create species="bug" number="100" />
      </init>
  </global>
```

We add a global section which contains an init subsection where we call the create command. The init section will be executed upon the creation of the entity. Here the entity is the system itself, we call it the "world". Consequently the bugs will be created before the start of the simulation and will be placed randomly on the default environment (the stupidGrid).

Nota bene

In GAMA we cannot choose when to draw the agent thus the "The display is updated at the end of each time step." statement is of no interest (though it is the case).

Complete model 1

```
<stupidmodel>
  <global>
      <init>
        <create species="bug" number="100" />
      </init>
  </global>

  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true" />
  </environment>
  <entities>
      <species name="bug" skills="situated">

        <reflex name="basic_move">
            <let var="place" value="stupid_grid location" />
            <let var="destination" value="one_of ((place neighbours_at 4)
where (empty  each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb 'red'" size="1" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
  </output>
</stupidmodel>
```

—

Bug growth

<wiki:toc max_depth="3" />

Purpose

Illustrate adding instance variables and methods to the agents.

Formulation

- Add a second bug action, grow. Each time step, a bug grows by a #xed 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 re#ect their size. Bug colors shade from white when size is zero to red when size is 10 or greater.

Model

Growing

Similarly to the first reflex we defined, you have to create a reflex that will increase a 'size' variable by 1 (used after in the aspect section).

Variable automatic update

In GAMA we would have a much simplier way to do so: it is possible to define an automatic variable update, for example:

```
<var type="float" name="size" init="1" value="size + 1"/>
```

Indeed the variable parameter 'value' is evaluated at each timestep while the init paremeter is evaluated once only (upon the creation of the holding entity).

Shading color

We saw previously that we can cast a string like 'red' into a color variable. It is also possible use the hexanumeric value:

```
<var type="rgb" name="color" value="rgb '#00CC00' "/>
```

Here '#00CC00' is a string containing the hexanumeric value always starting with the # character. Unfortunately it is still uneasy to scale smoothly the color from white to red. Fortunately it is possible to define one by one the three RGB component of the color using a list of three elements. In our case we would do as follow:

```
<var type="rgb" name="color" value="rgb [255, 255/size, 255/size]"/>
```

Nota bene

As you can see, it is possible to declare a list by using the brackets and the comma as separator: [element_1, element_2, element_3, etc...].

Complete model

```
<stupidmodel>
  <global>
      <init>
        <create species="bug" number="100" />
      </init>
  </global>

  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />

        <reflex name="basic_move">
            <let var="place" value="stupid_grid location" />
```

```
            <let var="destination" value="one_of ((place neighbours_at 4)
where (empty  each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>

        <reflex name="grow">
            <set var="size" value="size + 0.1" />
    </reflex>

        <aspect name="basic">
            <draw shape="circle" color="color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
  </output>
</stupidmodel>
```

Nota bene

We added a color definition within the grid section to make the grid black because bugs can be white and so invisible on a white grid. It is interesting because it shows how to define cells variable. Indeed the grid section is a bit similar to the species one. It defines the prototype of instance (cell in the case of the grid). It is possible to define here not only variables but also reflexes in the very same way as species.

Habitat cells and resource

<wiki:toc max_depth="3" />

Purpose

Show how to give a behaviour to the cells of the grid. Illustrate how agents and cells interact.

Formulation

- The grid cells have now instance variables for their food availability and maximum food production rate. Cells also have a variable for the bug at their location.
- 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 #oating point number between zero and the maximum food production.
- Bug growth is modi#ed 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.

Models

Giving a behaviour to the cells

Here, we simply have to add the following variables to the cells, that is, in the grid section: maximum food production rate, food availability and food production.

```
<var type="float" name="maxFoodProdRate" value="0.01"/>
<var type="float" name="food" init="0.0"/>
<var type="float" name="foodProd" value="(rnd(1000) / 1000) * 0.01"/>
```

nota bene

The (rnd(1000) / 1000) * 0.01 might seem a bit odd but it is to due the random operator. It works only on integer but allows us tu set the precision. In this example, we have a prevision of 10^-3 (as we use a range from 0 to 1000) and generate a number in the [0;0.01] range.

Increasing cell's food

We already defined the needed variables, we need now to update at each time step. We have two (at least) possibilities:

Simple version

Similarly to previously defined reflexes (here or here, you can define a reflex.

Correct version

It is also possible to use the automatic update of the variable using the value parameter:

```
<var type="float" name="food" init="0.0" value="food + foodProd"/>
```

In the value parameter, which is evaluated at each time step, we add the remaining food and the foodProd variable (which is updated at each time step thanks to value parameter too).

Bug growth

Instead of having a constant increase we now want to have a dynamic one. The increase is defined as the minimum of **maxConsumption** (bug's variable to define) and food (current cell's variable). Note that this quantity has to be substracted from the cell... It cannot be done by a variable automatic update so we will use a reflex.

```
<reflex name="grow">
  <let var="transfer" value="min [maxConsumption, (stupid_grid
location).food]"/>
  <set var="size" value="size + transfer"/>
  <set var="(stupid_grid location).food" value="(stupid_grid location).food +
transfer"/>
</reflex>
```

Nota bene

- As you can guess the executionner of this reflex is a bug but we could imagine that the cell do the work: it would check if there is an agent within it then transfer food (substract to its food variable and add to the bug's size variable).
- Instead of repeating _(stupid_grid location)_ we will add a convenient variable as follow:

```
<var type="stupid_grid" name="myPlace" value="stupid_grid location"/>
```

In GAMA, whenever a species (agent or environment like here) is defined it is possible to use at a type and then reference instance of this type. Please check the datatype section for more explanation on this particular type.

Complete model

```
<stupidmodel>
  <global>
      <init>
        <create species="bug" number="100" />
      </init>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate" value="0.01" />
        <var type="float" name="food" init="0.0" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) * 0.01" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="1" />
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination"
              value="one_of ((myPlace neighbours_at 4) where (empty
each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
```

```
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food + transfer" />
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
  </output>
</stupidmodel>
```

_

Cell and bug probes

<wiki:toc max_depth="3" />

Purpose

Show how to make model objects probeable from the display.

Formulation

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

Models

We introduce here the inspectors. Their declarations go into the output section. We will use two kinds, one to inspect species and one to inspect the internal status of a selected agent (by clicking on it or by selecting it from the species inspector).

```
<inspect name="Agents" type="agent" refresh_every="5"/>
<inspect name="Species" type="species" refresh_every="5"/>
```

- The 'refresh_every' parameter is used to set the refresh rate of the view, here views will be updated every 5 steps. You can also change this rate during the simulation.
- Now You can see that the GAMA interface is "tab-based" you can re-arrange them as you please.
- The species inspector to see the structure of the species (variables, reflexes, etc.) and its populations (instances of the species).
- You may note that the environment is also a species.

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <init>
        <create species="bug" number="100" />
      </init>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate" value="0.01" />
        <var type="float" name="food" init="0.0" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) * 0.01" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="1" />
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination"
              value="one_of ((myPlace neighbours_at 4) where (empty
each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food + transfer" />
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
```

```
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5"/>
      <inspect name="Species" type="species" refresh_every="5"/>
  </output>
</stupidmodel>
```

—

Parameters and parameters displays

<wiki:toc max_depth="3" />

Purpose

Show how to de#ne variables as parameters, and how to put parameters in the parameter settings window.

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)
- The maximum food production (a cell parameter).

Models

- We introduce the parametrization of variables with this model. . To do so, we add the parameter="true" state within the variable definition, as follow:

```
<var type="int" name="numberBugs" value="100" parameter="true"/>
```

Do not forget to update the creation command's parameter 'number'.

- Parametrization is only available to the world's variable within the global section thus for bugs' and cells' parameters we have to define global variables that will be used in the their personal variables definition. We add the following statement to the global section

```
<var type="float" name="globalMaxConsumption" value="1" parameter="true"/>
```

```
<var type="float" name="globalMaxFoodProdRate" value="0.01" parameter="true"/>
```

- We change the maxConsumption variable definition within bug like this:

```
<var type="float" name="maxConsumption" value="(rnd(1000) / 1000) *
globalMaxConsumption"/>
```

- We change the foodProd variable definition within cells like this:

```
<var type="float" name="maxFoodProdRate" value="globalMaxFoodProdRate"/>
```

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
```

```
        <reflex name="basic_move">
            <let var="destination"
              value="one_of ((myPlace neighbours_at 4) where (empty
each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>

        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>

        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>

      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
  </output>
</stupidmodel>
```

Nota bene

It seems useless to declare a 'maxFoodProdRate' variable but in the case we want heterogeneous value of maxFoodProdRate though ith a globalMaxFoodProdRate, it will be very easily done.

Histogram output

<wiki:toc max_depth="3" />

Purpose

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

Formulation

Add a histogram reflecting the distribution of bugs' size.

Models

We will now add a histogram subsection to the output one. In order to have a useable view of it we would define 10 classes within the [0;100] range.

Adding the pie chart

We add the following to the output section (note: the bugs list is added in the global section. see below):

```
<display name="histogram_display">
  <chart type="histogram" name="Size distribution" background="rgb
'lightGray'">
      <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
      <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
      <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
      <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
      <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
```

```
      <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
      <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
      <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
      <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
      <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
  </chart>
</display>
```

- The chart section can be of three types: histogram, pie or series
  o In all cases we can name it, define a background color.
- Within the chart section, we can define several input; for each we defined a name and a value.

Nota bene

When using this version you may note that we see, most of the time, only one class represents almost 100% of agents. It would be much more interesting to use adaptive class. It is possible by changing the value expression by taking into account mean, minimum and maximum value of bugs size. To do so you have to define the needed variables in the global section. We will do that in the complete section in order to replace the (list bug) by a global variable that will be compute once a time step.

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
```

```
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination"
              value="one_of ((myPlace neighbours_at 4) where (empty
each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
```

```
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
      </display>
  </output>
</stupidmodel>
```

Nota bene

- Please note again the usefullness of the cast operator in GAMA with the added variable 'bugs':

```
<var type="list" name="bugs" value="list bug"/>
```

Which translate a species into a list of its instanciated agents.

- Some characters used in GAML are special XML characters thus you will have to write them with the XML code. For example: " > " should be written " > ", " & " - > " & ".

Stopping the model

<wiki:toc max_depth="3" />

Purpose

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

Formulation

- The model stops when the largest bug reaches a size of 100.

Models

If you remember, the whole system is what we call the World (global section). Thus, it makes sense that it will responsible to stop the execution of the simulation. Indeed it is a special action of the world,the 'halt' action. We would do that by adding the following statement in the global section of our model:

```
<reflex name="shouldHalt" when="!(empty (bugs where (each.size &gt; 100)))">
  <do action="halt"/>
</reflex>
```

We can see that:

- The world can have reflexes
- They are defined in the same way
- It is possible to add a condition to the execution of the reflex using the 'when' parameter

We could have used the 'pause' command instead of the 'halt' one also.

Nota bene

We could also have define a reflex within the bugs and whenever the bug attain the size of a '100' it will ask the World to stop. That would be something like this:

```
<reflex name="askToHalt" when="size &gt; 100">
  <ask target="world">
    <do action="halt"/>
  <ask/">
</reflex>
```

- For the first time we see the 'ask command' here. As it sounds it allows a agent to "ask" (understand to force) an agent to execute something.
- In order to do so we would have to define a global variable (accessible to anyone) named "world" and with the world object as value.

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
      <reflex name="shouldHalt" when="!(empty (bugs where (each.size &gt;
100)))">
        <do action="halt"/>
      </reflex>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
```

```
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination"
              value="one_of ((myPlace neighbours_at 4) where (empty
each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
```

```
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
      </display>
  </output>
</stupidmodel>
```

_

File output

<wiki:toc max_depth="3" />

Purpose

Show how to write results to an output #le. Illustrate how to work with list.

Formulation

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

Models

Writing a log file is an output thus we have to add a specific file output. In our case it would as follow:

```
<file name="stupid_results" type="text" data="'cycle: '+ (string  time) + ';
minSize: '
+ (string (bugs min_of   each.size)) + '; maxSize: '
+ (string (bugs max_of   each.size)) + '; meanSize: '
+ (string ((sum (bugs collect ((bug   each).size))) / (length bugs)))"/>
```

We define here:

- A name for the file
- A type, text here but it could XML or CSV.
- A data from an expression, here we check the evolution of two variables plus some strings to make the file more readable.

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
      <reflex name="shouldHalt" when="!(empty (bugs where (each.size &gt;
100)))">
        <do action="halt"/>
      </reflex>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination"
              value="one_of ((myPlace neighbours_at 4) where (empty
each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
```

```
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
      </display>
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

—

Randomized agent actions

<wiki:toc max_depth="3" />

Purpose

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

Formulation

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

Models

In GAMA, the order of execution of agents is already shuffled at every time step. So, we have nothing to do for this step.

Sorted agent actions

<wiki:toc max_depth="3" />

Purpose

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

Formulation

- The bugs' execution is un-randomized so it is executed in descending size order.

Models

GAMA offers the possibility for the modeler to fine tune the scheduling strategy. in our context, we just have to modify the built-in global variable "scheduling_targets" in order to take into account the agent size.

```
<var type="list" name="scheduling_targets" value="bugs sort_by (bug
each).size"/>
```

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
```

```
      <var type="list" name="bugs" value="list bug" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
      <reflex name="shouldHalt" when="!(empty (bugs where (each.size &gt;
100)))">
        <do action="halt"/>
      </reflex>
      <var type="list" name="scheduling_targets" value="bugs sort_by (bug
each).size"/>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination"
              value="one_of ((myPlace neighbours_at 4) where (empty
each.agents))" />
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
```

```
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
      </display>
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

_

Optimal movement

<wiki:toc max_depth="3" />

Purpose

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

Formulation

- In its move method, a bug identi#es 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 identi#es the cell with highest food availability. The bug then moves to that cell.

Models

We have to filter cells in two ways here. First we will remove, all the accessible with an agent then we select one with most food. To do so we will only change the definition of the temporary variable destination.

```
<let var="destination" value="last (((myPlace neighbours_at 4) where (empty
each.agents)) sort_by ((stupid_grid each).food))"/>
```

- As before we filter cells without agent (which we will call filtered_List):

```
(myPlace neighbours_at 4) where (empty each.agents)
```

- Then we sort by ascending of food (which we will ordered_list):

```
filtered_List sort_by ((stupid_grid each).food)
```

- Finally we get the last (because we are in ascending order) cell of the list:

```
last ordered_list
```

Nota Bene

This is one possibility to get a cell with the maximum food, it is also possible to use the with_max_of command. We leave this to you as an exercice.

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
      <reflex name="shouldHalt" when="!(empty (bugs where (each.size &gt;
100)))">
        <do action="halt"/>
      </reflex>
      <var type="list" name="scheduling_targets" value="bugs sort_by (bug
each).size"/>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
```

```
        <reflex name="basic_move">
            <let var="destination" value="last (((myPlace neighbours_at 4)
where (empty each.agents)) sort_by ((stupid_grid each).food))"/>
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
      </display>
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
```

```
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

—

Bug mortality and reproduction

<wiki:toc max_depth="3" />

Purpose

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

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 #rst empty location randomly selected within +/- 3 cells of their parent’s last location. If no location is identi#ed 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.

Models

Multiplying

To make an agent dies and create new agent is pretty easy, it is done with the create command and the die primitive. We will do it within a new reflex.

```
<reflex name="multiply">
  <if condition="size &gt; 10">
    <let var="possible_nests" value="(myPlace neighbours_at 3) where (empty
each.agents)"/>
    <loop times="5">
```

```
      <let var="nest" value="one_of possible_nests"/>
      <if condition="nest != nil">
        <set var="possible_nests" value="possible_nests - nest"/>
        <create species="bug" number="1" return="child"/>
        <ask target="child">
          <set var="location" value="nest.location"/>
        </ask>
      </if>
    </loop>
    <do action="die"/>
  </if>
</reflex>
```

Introducing survivability

We have to introduce a new variable 'survivalProbability', we will set it at 0.95. We will also define a reflex that will manage the death of agents using this parameter.

```
<reflex name="shallDie" when="((rnd 100)/100.0) &gt; survivalProbability">
    <do action="die"/>
</reflex>
```

Changing the stop condition

We have to modify the shouldHalt reflex as follow:

```
<reflex name="shouldHalt" when="(time &gt; 1000) or (empty (list bug))">
  <do action="halt"/>
</reflex>
```

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
```

```
      <var type="float" name="survivalProbability" value="0.95"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
      <reflex name="shouldHalt" when="(time &gt; 1000) or (empty (list bug))">
        <do action="halt"/>
      </reflex>
      <var type="list" name="scheduling_targets" value="bugs sort_by (bug
each).size"/>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination" value="last (((myPlace neighbours_at 4)
where (empty each.agents)) sort_by ((stupid_grid each).food))"/>
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <reflex name="shallDie" when="((rnd 100) / 100.0) &gt;
survivalProbability">
            <do action="die"/>
        </reflex>
        <reflex name="multiply">
            <if condition="size &gt; 10">
              <let var="possible_nests" value="(myPlace neighbours_at 3) where
(empty each.agents)"/>
              <loop times="5">
                  <let var="nest" value="one_of possible_nests"/>
```

```
                  <if condition="nest != nil">
                    <set var="possible_nests" value="possible_nests - nest"/>
                    <create species="bug" number="1" return="child"/>
                    <ask target="child">
                      <set var="location" value="nest.location"/>
                    </ask>
                  </if>
              </loop>
              <do action="die"/>
          </if>
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
      </display>
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
```

```
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

Population abundance graph

<wiki:toc max_depth="3" />

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.

Formulation

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

Models

We previously defined a histogram, the time series one follow the same structure. In order to draw to two histograms in the same display, we are going to use the attributes "position" and "size" of the chart structure.

```
  <chart type="histogram" name="Size distribution" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.05}">
...
  </chart>
  <chart name="Population history" type="series" background="rgb 'lightGray'"
size="{1,0.4}" position="{0, 0.5}">
    <data name="Bugs" value="length (list bug)" color="rgb 'blue'" />
  </chart>
```

<font color="blue">Complete model</ font>

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="float" name="survivalProbability" value="0.95"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <init>
        <create species="bug" number="numberBugs" />
      </init>
      <reflex name="shouldHalt" when="(time &gt; 1000) or (empty (list bug))">
        <do action="halt"/>
      </reflex>
      <var type="list" name="scheduling_targets" value="bugs sort_by (bug
each).size"/>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="1" />
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination" value="last (((myPlace neighbours_at 4)
where (empty each.agents)) sort_by ((stupid_grid each).food))"/>
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
```

```
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <reflex name="shallDie" when="((rnd 100) / 100.0) &gt;
survivalProbability">
            <do action="die"/>
        </reflex>
        <reflex name="multiply">
            <if condition="size &gt; 10">
              <let var="possible_nests" value="(myPlace neighbours_at 3) where
(empty each.agents)"/>
              <loop times="5">
                  <let var="nest" value="one_of possible_nests"/>
                  <if condition="nest != nil">
                    <set var="possible_nests" value="possible_nests - nest"/>
                    <create species="bug" number="1" return="child"/>
                    <ask target="child">
                      <set var="location" value="nest.location"/>
                    </ask>
                  </if>
              </loop>
              <do action="die"/>
          </if>
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.05}">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
```

```
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
        <chart name="Population history" type="series" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.5}">
            <data name="Bugs" value="length (list bug)" color="rgb 'blue'" />
    </chart>
      </display>
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

_

Random normal initial size

<wiki:toc max_depth="3" />

Purpose

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

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 gaussian (normal) distribution de#ned by initialBugSizeMean and initialBugSizeSD.
- 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.

Models

Adding new parameters

You already know how to add variables and make them parameters.

Normal randomization of bug size

In order to use a gaussian distribution, we have to create (during the initialization of the world) a random_builder agent. In order to ease the use of this agent, we define a global variable random of type random_builder .

```
<var type="random_builder" name="random"/>
```

```
<init>
  <create species="random_builder" number="1" return="rand"/>
  <set name="random" value="rand"/>
  ...
</init>
```

It is then possible to call the action random_gaussian of the random agent. <var type="float" name="size" value="random random_gaussian [mean::initialBugSizeMean, sd::initialBugSizeSD] "/>

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="float" name="survivalProbability" value="0.95"
parameter="true" />
      <var type="float" name="initialBugSizeMean" value="0.1"
parameter="true" />
      <var type="float" name="initialBugSizeSD" value="0.03"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <var type="random_builder" name="random"/>
      <init>
        <create species="random_builder" number="1" return="rand"/>
        <set name="random" value="rand"/>
        <create species="bug" number="numberBugs" />
      </init>
      <reflex name="shouldHalt" when="(time &gt; 1000) or (empty (list bug))">
        <do action="halt"/>
      </reflex>
      <var type="list" name="scheduling_targets" value="bugs sort_by (bug
each).size"/>
  </global>
  <environment>
      <grid name="stupid_grid" width="100" height="100" torus="true">
        <var type="rgb" name="color" init="rgb 'black'" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
```

```
        <var type="float" name="foodProd" value="(rnd(1000) / 1000) *
maxFoodProdRate" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="random random_gaussian
[mean::initialBugSizeMean, sd::initialBugSizeSD] "/>
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination" value="last (((myPlace neighbours_at 4)
where (empty each.agents)) sort_by ((stupid_grid each).food))"/>
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <reflex name="shallDie" when="((rnd 100) / 100.0) &gt;
survivalProbability">
            <do action="die"/>
        </reflex>
        <reflex name="multiply">
            <if condition="size &gt; 10">
              <let var="possible_nests" value="(myPlace neighbours_at 3) where
(empty each.agents)"/>
              <loop times="5">
                  <let var="nest" value="one_of possible_nests"/>
                  <if condition="nest != nil">
                    <set var="possible_nests" value="possible_nests - nest"/>
                    <create species="bug" number="1" return="child"/>
                    <ask target="child">
                      <set var="location" value="nest.location"/>
                      <set var="size" value="0.0"/>
                    </ask>
                  </if>
              </loop>
              <do action="die"/>
          </if>
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
```

```
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.05}">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
        <chart name="Population history" type="series" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.5}">
            <data name="Bugs" value="length (list bug)" color="rgb 'blue'" />
    </chart>
      </display>
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

Habitat data from file input

<wiki:toc max_depth="3" />

Purpose

Show how to read spatial data in from a file.

Formulation

- Instead of assuming the space size and assuming cell food production is random (model 3), 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 #le, 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 modi#ed 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 #rst 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.

Models

Reading data from a file

GAMA allows to directly read CSV or txt files and to represent them as a matrix using casting operators:

```
<var type="matrix" name="init_data" init="file 'data/Stupid_Cell.Data'"
const="true" />
```

In this model, we have to find the X max (width) and the Y max (height). A way to find them is to use the max_of operators:

```
<var type="int" name="width" init="((init_data column_at 0) copy_between
{3,rows_number init_data - 1}) max_of each" const="true" />
<var type="int" name="height" init="((init_data column_at 1) copy_between
{3,rows_number init_data - 1}) max_of each" const="true" />
```

Remarks that we use the operator copy_between to skip the first three lines of header information. In order to set the foodProd attribute of each cell, we just have to loop over the matrix rows (from the fourth row), and set the foodProd attribute of the right cell using an ask command.

```
<loop from="3" to="(rows_number init_data) - 1" var="i">
  <let type="int" name="ind_i" value="init_data at {0,i}" />
  <let type="int" name="ind_j" value="init_data at {1,i}" />
  <ask target="(matrix stupid_grid) at {ind_i,ind_j}">
      <set name="foodProd" value="init_data at {2,i}" />
  </ask>
</loop>
```

Cell color

In a similar way than for the bugs (model 2), we have to dynamically compute the color of the cell according to the food availability.

```
<var type="rgb" name="color" value="rgb [0,min [255, food * 255 * 0.5], 0]" />
```

We used the min operator to ensure that the value for the green will be lower or equal to 255.

Modification of the bug behaviour

We have to modify the bug behaviour by randomly shuffling the list of available cells before the bug loops through it to identify the best. The shuffling of a list is very simple in GAMA using the shuffle operator.

```
<let var="destination" value="last ((shuffle ((myPlace neighbours_at 4) where
(empty each.agents))) sort_by ((stupid_grid each).food))"/>
```

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="float" name="survivalProbability" value="0.95"
parameter="true" />
      <var type="float" name="initialBugSizeMean" value="0.1"
parameter="true" />
      <var type="float" name="initialBugSizeSD" value="0.03"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <var type="random_builder" name="random"/>
      <var type="matrix" name="init_data" init="file 'data/Stupid_Cell.Data'"
const="true" />
      <var type="int" name="width" init="((init_data column_at 0) copy_between
{3,rows_number init_data - 1}) max_of each" const="true" />
      <var type="int" name="height" init="((init_data column_at 1)
copy_between {3,rows_number init_data - 1}) max_of each" const="true" />

      <init>
        <create species="random_builder" number="1" return="rand"/>
        <set name="random" value="rand"/>
        <create species="bug" number="numberBugs" />
        <loop from="3" to="(rows_number init_data) - 1" var="i">
              <let type="int" name="ind_i" value="init_data at {0,i}"/>
              <let type="int" name="ind_j" value="init_data at {1,i}"/>
              <ask target="(stupid_grid as matrix) at {ind_i,ind_j}">
              <set name="foodProd" value="init_data at {2,i}"/>
```

```
            </ask>
        </loop>
      </init>
      <reflex name="shouldHalt" when="(time &gt; 1000) or (empty (list bug))">
        <do action="halt"/>
      </reflex>
      <var type="list" name="scheduling_targets" value="bugs sort_by (bug
each).size"/>
  </global>
  <environment bounds="{width,height}">
      <grid name="stupid_grid" width="width" height="height">
        <var type="rgb" name="color" value="rgb [0,min [255, food * 255 *
0.5],0]" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="random random_gaussian
[mean::initialBugSizeMean, sd::initialBugSizeSD] "/>
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination" value="last ((shuffle ((myPlace
neighbours_at 4) where (empty each.agents))) sort_by ((stupid_grid
each).food))"/>
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <reflex name="shallDie" when="((rnd 100) / 100.0) &gt;
survivalProbability">
            <do action="die"/>
        </reflex>
        <reflex name="multiply">
            <if condition="size &gt; 10">
              <let var="possible_nests" value="(myPlace neighbours_at 3) where
(empty each.agents)"/>
              <loop times="5">
                  <let var="nest" value="one_of possible_nests"/>
                  <if condition="nest != nil">
```

```
                    <set var="possible_nests" value="possible_nests - nest"/>
                    <create species="bug" number="1" return="child"/>
                    <ask target="child">
                      <set var="location" value="nest.location"/>
                      <set var="size" value="0.0"/>
                    </ask>
                  </if>
              </loop>
              <do action="die"/>
          </if>
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.05}">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
        <chart name="Population history" type="series" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.5}">
            <data name="Bugs" value="length (list bug)" color="rgb 'blue'" />
    </chart>
      </display>
```

```
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

—

Predators

<wiki:toc max_depth="3" />

Purpose

How to create multiple classes of agents that interact.

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 shuf#ed list of its immediately neighboring cells (including its own cell). As soon as the predator #nds 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.

Models

Instanciating predators

We did it already in the first model (here) thus will leave to you as an exercise.

Defining predators

We also did it in the first model (here). Although the definition of the hunting action is a bit tricky thus we will define it:

```
<reflex name="hunting">
    <let type="list" name="the_neighbours" value="myPlace neighbours_at 1"/>
```

```
    <let type="list" name="the_neighbours_bug" value="the_neighbours collect
(first (each.agents of_species bug)) "/>
    <let type="bug" name="chosenPrey" value="one_of the_neighbours_bug" />
    <if condition="chosenPrey != nil">
        <let type="stupid_grid" name="new_loc" value="stupid_grid
chosenPrey.location"/>
        <if condition="empty (new_loc.agents of_species predator)">
            <set var="location" value="new_loc" />
            <ask target="chosenPrey">
                <do action="die" />
            </ask>
        </if>
        <else>
            <set var="location" value="one_of the_neighbours" />
        </else>
      </if>
</reflex>
```

Explanations

- We select cells around of the predator (direct neighbourhood)
- We select the bugs contained inside these cells
- We select one of them randomly.
- If we have a chosen prey.
  - We check it's empty of other predators.
    - If so we go there and kill the bug.
    - If there is another predator already, nothing happens.
- If we have no place with a bug.
  - We move randomly.

Scheduling Predators

You just have to add to the scheduling_targets list (in the global section) the predator agent. Check model number 10 if you do not remember how to do.

```
<var type="list" name="scheduling_targets" value="(bugs sort_by (bug
each).size) + (list predator)" />
```

Visualization

We just have to add the predator species to the stupid display:

```
<display name="stupid_display">
  <grid name="stupid_grid" />
  <species name="bug" aspect="basic" />
```

```
  <species name="predator" aspect="basic" />
</display>
```

Complete model

We obtain the following model:

```
<stupidmodel>
  <global>
      <var type="int" name="numberBugs" value="100" parameter="true" />
      <var type="int" name="numberOfPredators" init="200" parameter="true"/>
      <var type="float" name="globalMaxConsumption" value="1"
parameter="true" />
      <var type="float" name="globalMaxFoodProdRate" value="0.01"
parameter="true" />
      <var type="float" name="survivalProbability" value="0.95"
parameter="true" />
      <var type="float" name="initialBugSizeMean" value="0.1"
parameter="true" />
      <var type="float" name="initialBugSizeSD" value="0.03"
parameter="true" />
      <var type="list" name="bugs" value="list bug" />
      <var type="random_builder" name="random"/>
      <var type="matrix" name="init_data" init="file 'data/Stupid_Cell.Data'"
const="true" />
      <var type="int" name="width" init="((init_data column_at 0) copy_between
{3,rows_number init_data - 1}) max_of each" const="true" />
      <var type="int" name="height" init="((init_data column_at 1)
copy_between {3,rows_number init_data - 1}) max_of each" const="true" />

      <init>
        <create species="random_builder" number="1" return="rand"/>
        <set name="random" value="rand"/>
        <create species="bug" number="numberBugs" />
        <loop from="3" to="(rows_number init_data) - 1" var="i">
              <let type="int" name="ind_i" value="init_data at {0,i}"/>
              <let type="int" name="ind_j" value="init_data at {1,i}"/>
              <ask target="(stupid_grid as matrix) at {ind_i,ind_j}">
              <set name="foodProd" value="init_data at {2,i}"/>
            </ask>
        </loop>
      </init>
      <reflex name="shouldHalt" when="(time &gt; 1000) or (empty (list bug))">
        <do action="halt"/>
      </reflex>
```

```
      <var type="list" name="scheduling_targets" value="(bugs sort_by (bug
each).size) + (list predator)" />
  </global>
  <environment bounds="{width,height}">
      <grid name="stupid_grid" width="width" height="height">
        <var type="rgb" name="color" value="rgb [0,min [255, food * 255 *
0.5],0]" />
        <var type="float" name="maxFoodProdRate"
value="globalMaxFoodProdRate" />
        <var type="float" name="foodProd" />
        <var type="float" name="food" init="0.0" value="food + foodProd" />
      </grid>
  </environment>
  <entities>
      <species name="bug" skills="situated">
        <var type="float" name="size" init="random random_gaussian
[mean::initialBugSizeMean, sd::initialBugSizeSD] "/>
        <var type="rgb" name="color" value="rgb [255, 255/size, 255/size]" />
        <var type="float" name="maxConsumption" value="globalMaxConsumption" /
>
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />
        <reflex name="basic_move">
            <let var="destination" value="last ((shuffle ((myPlace
neighbours_at 4) where (empty each.agents))) sort_by ((stupid_grid
each).food))"/>
            <if condition="destination != nil">
              <set var="location" value="destination" />
            </if>
        </reflex>
        <reflex name="grow">
            <let var="transfer" value="min [maxConsumption, myPlace.food]" />
            <set var="size" value="size + transfer" />
            <set var="myPlace.food" value="myPlace.food - transfer" />
        </reflex>
        <reflex name="shallDie" when="((rnd 100) / 100.0) &gt;
survivalProbability">
            <do action="die"/>
        </reflex>
        <reflex name="multiply">
            <if condition="size &gt; 10">
              <let var="possible_nests" value="(myPlace neighbours_at 3) where
(empty each.agents)"/>
              <loop times="5">
                  <let var="nest" value="one_of possible_nests"/>
                  <if condition="nest != nil">
                    <set var="possible_nests" value="possible_nests - nest"/>
                    <create species="bug" number="1" return="child"/>
                    <ask target="child">
                      <set var="location" value="nest.location"/>
                      <set var="size" value="0.0"/>
                    </ask>
```

```
                  </if>
              </loop>
              <do action="die"/>
          </if>
        </reflex>
        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="size" />
        </aspect>
      </species>

      <species name="predator" skills="situated">
        <var type="rgb" name="color" init="'blue'" />
        <var type="stupid_grid" name="myPlace" value="stupid_grid location" />

        <reflex name="hunting">
            <let type="list" name="the_neighbours" value="myPlace
neighbours_at 1"/>
            <let type="list" name="the_neighbours_bug" value="the_neighbours
collect (first (each.agents of_species bug)) "/>
            <let type="bug" name="chosenPrey" value="one_of
the_neighbours_bug" />
            <if condition="chosenPrey != nil">
              <let type="stupid_grid" name="new_loc" value="stupid_grid
chosenPrey.location"/>
              <if condition="empty (new_loc.agents of_species predator)">
                  <set var="location" value="new_loc" />
                  <ask target="chosenPrey">
                    <do action="die" />
                  </ask>
              </if>
              <else>
                  <set var="location" value="one_of the_neighbours" />
              </else>
            </if>
        </reflex>

        <aspect name="basic">
            <draw shape="circle" color="rgb color" size="2" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="stupid_display">
        <grid name="stupid_grid" />
        <species name="bug" aspect="basic" />
        <species name="predator" aspect="basic" />
      </display>
      <inspect name="Agents" type="agent" refresh_every="5" />
      <inspect name="Species" type="species" refresh_every="5" />
      <display name="histogram_display">
```

```
        <chart type="histogram" name="Size distribution" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.05}">
            <data name="[0;10]" value="bugs count (each.size &lt; 10)" />
            <data name="[10;20]" value="bugs count ((each.size &gt; 10) and
(each.size &lt; 20))" />
            <data name="[20;30]" value="bugs count ((each.size &gt; 20) and
(each.size &lt; 30))" />
            <data name="[30;40]" value="bugs count ((each.size &gt; 30) and
(each.size &lt; 40))" />
            <data name="[40;50]" value="bugs count ((each.size &gt; 40) and
(each.size &lt; 50))" />
            <data name="[50;60]" value="bugs count ((each.size &gt; 50) and
(each.size &lt; 60))" />
            <data name="[60;70]" value="bugs count ((each.size &gt; 60) and
(each.size &lt; 70))" />
            <data name="[70;80]" value="bugs count ((each.size &gt; 70) and
(each.size &lt; 80))" />
            <data name="[80;90]" value="bugs count ((each.size &gt; 80) and
(each.size &lt; 90))" />
            <data name="[90;100]" value="bugs count ((each.size &gt; 90) and
(each.size &lt; 100))" />
        </chart>
        <chart name="Population history" type="series" background="rgb
'lightGray'" size="{1,0.4}" position="{0, 0.5}">
            <data name="Bugs" value="length (list bug)" color="rgb 'blue'" />
    </chart>
      </display>
      <file name="stupid_results" type="text" data="'cycle: '+ (string  time)
        + '; minSize: ' + (string (bugs min_of   each.size))
        + '; maxSize: ' + (string (bugs max_of   each.size))
        + '; meanSize: ' + (string ((sum (bugs collect ((bug   each).size))) /
(length bugs)))"/>
  </output>
</stupidmodel>
```

—

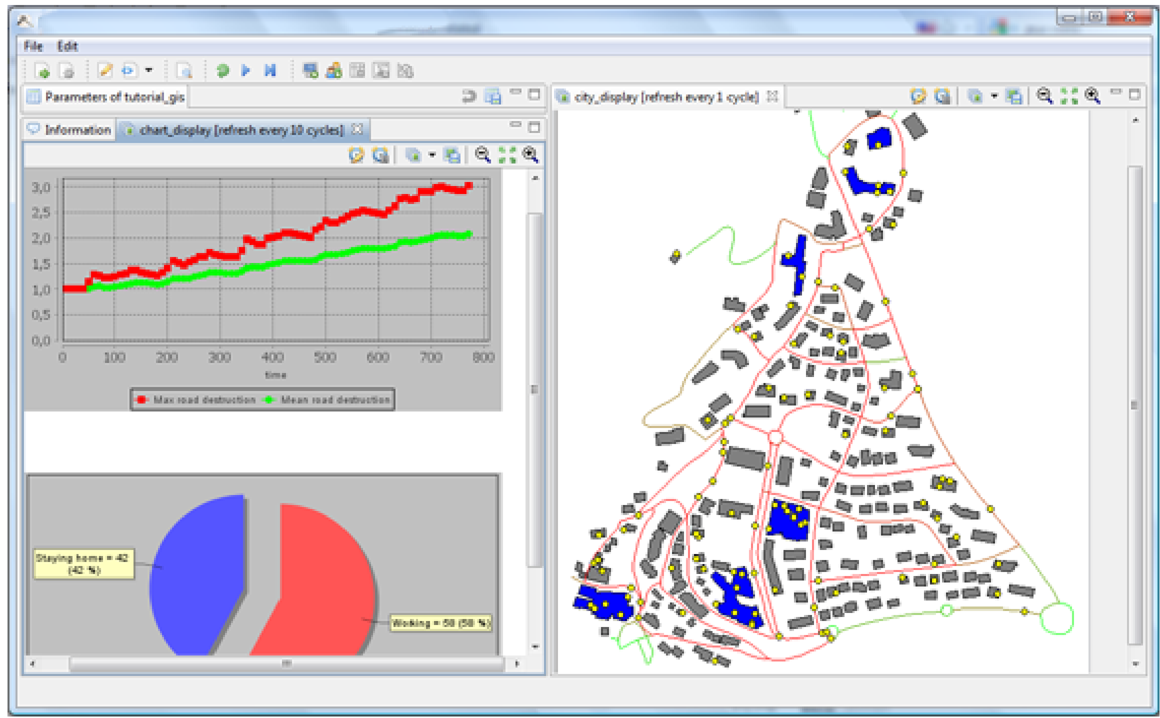
Tutorial 2: GIS tutorial

Introduction to the tutorial

summary Tutorial: Road traffic in a city <wiki:toc max_depth="3" />

Introduction

This tutorial has for goal to present the use of GIS data and complex geometries. In particular, this tutorial shows how to load gis data, to agentify them and to use a network of polylines to constraint the movement of agents. All the files related to this tutorial (shapefiles and models) are available here . The model built in this tutorial concerns the study of the road traffic in a small city. Two layers of GIS data are used: a road layer (polylines) and a building layer (polygons). The building GIS data contain an attribute: the 'NATURE' of each building: a building can be either 'Residential' or 'Industrial'. In this model, people agents are moving along the road network. Each morning, they are going to an industrial building to work, and each night they are coming back home. Each time a people agent takes a road, it wears it out. More a road is worn out, more a people agent takes time to go all over it. The town council is able to repair some roads. <a href="



" title="Road_traffic_image">

Road traffic in a city: model list

This tutorial is composed of 8 models. For each model we will present its purpose, an explicit formulation and the corresponding GAML code.

1. Loading of GIS data (buildings and roads)
2. Integration of people agents
3. Movement of the people agents
4. Definition of weight for the road network
5. Dynamic update of the road network
6. Definition of a chart display
7. Automatic repair of roads
8. Complex repair of roads

1. Loading of GIS data (buildings and roads)

<wiki:toc max_depth="3" />

Purpose

Illustrate how to load GIS data (shapefiles) and to read attributes from GIS data.

Formulation

- Load, agentify and display two layers of GIS data (building and road)
- Read the 'NATURE' attribute of the building data: the buildings of 'Residential' type will be colored in gray, the buildings of 'Industrial' type will be color in blue.

Model

Entities

In this first model, we have to define two species of agents: the **building** agents and the **road** ones. These agents will not have a particular behaviour. They will be just displayed. Thus, we just have to define an aspect for the agents (see [Species Aspects] in the Modeling guide). In this model, we want to represent the geometry of the agent, we then use the value **geometry** of the facet **shape** for the command **draw** . Concerning the **building** agent, we have to add in addition to the color a second attribute: the type of the building ('Residential' or 'Industrial').

```
<entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />

        <aspect name="base">
```

```
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="rgb" name="color" init="rgb 'black'" />

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
  </entities>
```

Parameters

GAMA allows to automatically read GIS data that are formatted as shapefiles. In order to let the user chooses his/her shapefiles, we define three parameters (string). One allowing the user to choose the road shapefile, one allowing him/her to choose the building shapefile, and, at last, one allowing him/her to choose the bounds shapefile. We will come back later on the notion of "bounds" in GAMA.

```
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
    ...
  </global>
```

Agentification of GIS data

In GAMA, the agentification of GIS data is very straightforward: it only requires to use the **create** command with the **from** facet to pass the reference of the shapefile. Each object of the shapefile will be directly used to instantiate an agent of the specified species. The reading of an attribute in a shapefile is also very simple. It only requires to use the **with** facet: the argument of this facet is a dictionary of which the keys are the names of the agent attributes and the value the **read** command followed by the name of the shapefile attribute ('NATURE' in our case).

```
    <global>
    ...
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']">
            <if condition="type='Industrial'">
```

```
            <set name="color" value="rgb 'blue'"/>
          </if>
      </create>
      <create species="road" from="shape_file_roads"/>
    </init>
  </global>
```

Environment

Building a GIS environment in GAMA requires nothing special, just to define the bounds of the environment, i.e. the square envelop of the GIS data. It is possible to use a shapefile to automatically define it. In this model, we use a specific shapefile to define it. However, it would been possible to use the road shapefile to define it.

```
<environment bounds="shape_file_bounds"/>
```

Display

We have to define a diplay for the road and building agents. We use for that the classic **species** markup.

```
  <output>
    <display name="city_display" refresh_every="1">
      <species name="building" aspect="base"/>
      <species name="road" aspect="base"/>
    </display>
  </output>
```

<font color="blue">Complete model</ font>

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']">
            <if condition="type='Industrial'">
```

```
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>
        <create species="road" from="shape_file_roads"/>
      </init>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="rgb" name="color" init="rgb 'black'" />

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
      </display>
  </output>
</model>
```

2. Integration of people agents

<wiki:toc max_depth="3" />

Purpose

Illustrates how to obtain a random point inside a geometry.

Formulation

- Define a new species of agents: the **people** agents. The **people** agents have a point for geometry and are represented by a yellow circle of radius 10m.
- At initialisation, 100 **people** agents are created. Each **people** agent is placed inside a building of type 'Residential' (randomly selected).

Model

Entities

We have to define a new species of agents: the **people** agents. In this model, these agents will have not a specific behaviour. They will be just displayed. Thus, we just have to define an aspect for the agents. We want to represent the **people** agents by a yellow circle of radius 10m. In GAMA, the default reprsentation for a point geometry is a point. Then, we can use the **geometry** value for the **shape** facet of the **draw** command, we just have to specified the expected color and radius size (defined by the facet **size**).

```
<entities>
    <species name="people" skills="situated">
      <var type="rgb" name="color" init="rgb 'yellow'" />

      <aspect name="base">
          <draw shape="geometry" color="color" size="10"/>
      </aspect>
    </species>
</entities>
```

Parameter

We have to add a new parameter: the number of **people** agent created

```
  <global>
    ...
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
    ...
  </global>
```

Creation and placement of the people agents

We have to create **nb_people people** agents. Each **people** is placed inside a buildings of type 'Residential' randomly selected. In order to simplify the GAML code, we use the **return** facet of the **create** command to return the list of **buildings** agents created (list **the_buildings**). To filter this list in order to only keep the **building** agents of which the **type** is 'Redidential', we use the **where** operator. We then obtain a sub-list of **building** agents of type 'Residential'. We use the operator **one_of** to randomly select one of these agents. There are several ways to place a **people** agent inside this building.
In this tutorial, we choose to use the **place_in** action of the **situated** skill. This action returns a random point situated on the geometry passed in parameter (or the geometry of the agent passed in parameter).

```
    <global>
    ...
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>
        <create species="road" from="shape_file_roads"/>
        <create species="people" number="nb_people">
            <set name="location" value="self place_in [agent::one_of
(the_buildings where (each.type='Residential'))]"/>
        </create>
      </init>
  </global>
```

Display

We have to add the **people** agent in the defined diplay.

```
  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
        <species name="people" aspect="base"/>
      </display>
  </output>
```

Complete model

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>
        <create species="road" from="shape_file_roads"/>
        <create species="people" number="nb_people">
            <set name="location" value="self place_in [agent::one_of
(the_buildings where (each.type='Residential'))]"/>
        </create>
      </init>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="rgb" name="color" init="rgb 'black'" />
```

```
        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

      <species name="people" skills="situated">
        <var type="rgb" name="color" init="rgb 'yellow'" />

        <aspect name="base">
            <draw shape="geometry" color="color" size="10"/>
        </aspect>
      </species>
  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
        <species name="people" aspect="base"/>
      </display>
  </output>
</model>
```

—

3. Movement of the people agents

<wiki:toc max_depth="3" />

Purpose

Illustrates how to obtain the area of a polygon and to constraint the movement of an agent according to a network of polylines.

Formulation

- Definition of day_time global variable that will indicate, according to the simulation step, the time of the day: each simulation step will represent 10 minutes, then the day_time variable will be ranged between 0 and 144.
- For each **people** agent: define a living_place(building of type 'Residential') and working place (building of type 'Industrial'). The chance to select a building for the living_place and the working_place will depend on the building area: the probability to assign a building to a **people** agent is equals to its area devided by the sum of the building areas (Residential or Industrial).
- For each **people** agent: define start_work and end_work hours that repectively represent when the agent leaves its house to go to work and when it leaves its working_place to go back home. These hours will be randomly define between 36 and 60 for the start_work and 84 and 132 for the end_work.
- For each **people** agent: define a objective variable: this one can 'go home' or 'working'.
- For each **people** agent: define a speed. The speed will be randomly define between 50 and 100.
- The **people** agents move along the road, taking the shortest path.

Model

People agents

First, we have to change the skill of the **people** agents: as we want to use an action of the **moving** skill (**goto**), we will provide the **people** agents with this skill (that automatically integrate the **situated** skill).

```
<species name="people" skills="moving">
    ...
</species>
```

Then, we have to add new variables to the people agents: living_place, working_place, start_work, end_work, objective and speed. In addition, we will add a "the_target" variable that will represents the point toward which the agent is moving.

```
<species name="people" skills="moving">
    <var type="rgb" name="color" init="rgb 'yellow'" />
    <var type="building" name="living_place" init="nil"/>
    <var type="building" name="working_place" init="nil"/>
    <var type="int" name="start_work"/>
    <var type="int" name="end_work"/>
    <var type="string" name="objectif"/>
    <var type="point" name="the_target" init="nil"/>
    ...
</species>
```

We define two reflex methods that allow to change the objective (and the_target) of the agent at the start_work and en_work hours. Concerning the _target value, we choose a random point in the objective building (working_place or living_place) by using the **place_in** action.

```
<species name="people" skills="moving">
    ...
    <reflex name="time_to_work" when="day_time = start_work">
      <set name="objectif" value="'working'"/>
      <set name="the_target" value="self place_in [agent::working_place]"/>
    </reflex>

    <reflex name="time_to_go_home" when="day_time = end_work">
      <set name="objectif" value="'go home'"/>
      <set name="the_target" value="self place_in [agent::living_place]"/>
    </reflex>
      ...
</species>
```

At last, we define a reflex method that allows to move the agent. If a target point is defined (the_target != nil), the agent moves in direction of this target using the **goto** action. Note that we specified a graph to constraint the moving of the agents on the road network. We will see later how this graph is built. When the agent arrives at destination (the_target = location), the target is set as nil (the agent will stop moving).

```
  <species name="people" skills="moving">
      ...
      <reflex name="move" when="the_target != nil">
        <do action="goto">
            <arg name="target" value="the_target"/>
            <arg name="graph_name" value="'road_network'"/>
        </do>
        <if condition="the_target = location">
            <set name="the_target" value="nil"/>
        </if>
      </reflex>
        ...
  </species>
```

Parameters

We have to add several parameters (min_work_start, max_work_start, min_work_end, max_work_end, min_speed and max_speed) and a global variable (day_time). The value of the day_time variable will be automatically computed at each simulation step and is equals to "time (the simulation step) modulo 144".

```
  <global>
      ...
      <var type="int" name="day_time" value="time%144" />
      <var type="int" name="min_work_start" init="36" parameter="Earliest hour 
to start work" category="People" />
      <var type="int" name="max_work_start" init="60" parameter="Latest hour 
to start work" category="People" />
      <var type="int" name="min_work_end" init="84" parameter="Earliest hour 
to end work" category="People" />
      <var type="int" name="max_work_end" init="132" parameter="Latest hour to 
end work" category="People" />
      <var type="float" name="min_speed" init="50" parameter="minimal speed" 
category="People" />
      <var type="float" name="max_speed" init="100" parameter="maximal speed" 
category="People" />
    ...
  </global>
```

Initialisation

First, we need to compute from the **road** agents, a graph for the moving of the **people** agents. The action **compute_graph** of the **situated** skill allows to do that. In the context of a set of agents which geometries are polylines (like our **road** agent), we have to use the **network** argument and to give the list of **road** agents as value for it. Then, we have to specified a name for this graph that will represent the id of the graph. We choose to call it 'road_network'. GAMA integrates several algorithms to compute the shortest pathes between the nodes of the graph. When no 'optimizer_type' is specified (like in this example), all the shortest pathes are automatically computed with the Floyd Warshall algorithm. Note that, in this example, the agent that is applying the **compute_graph** action is the **World** : the **World** agent is a **situated** agent (so, that can use all the action of the **situated** skill) that has the bounds for geometry.

```
  <global>
      <init>
        ...
        <create species="road" from="shape_file_roads" return="the_roads"/>

        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
        </do>
          ...
      </init>
  </global>
```

We have to assign one working place and one house to each agent. These ones will depends of the area of the buildings. First, to simplify the GAML code, we define two temporary variables: the list of buildings of type 'Residential' and the list of buildings of type 'Industrial' (by using the **where** command). Then, we compute the total areas of buildings of type 'Residential' and of type 'Industrial' by using the **area_sum** action of the **situated** skill and the probability to be selected for each building agent (in the **building** species, we add the **proba_select** variable).

```
  <global>
      <init>
        ...
        <let type="list" name="residential_buildings" value="the_buildings
where (each.type='Residential')"/>
        <let type="list" name="industrial_buildings" value="the_buildings
where (each.type='Industrial')"/>
        <let type="float" name="area_rb_tot" value="self area_sum
[agents::residential_buildings]"/>
        <ask target="residential_buildings">
```

```
            <set name="proba_select" value="area / area_rb_tot"/>
        </ask>
        <let type="float" name="area_ib_tot" value="self area_sum
[agents::industrial_buildings]"/>
        <ask target="industrial_buildings">
            <set name="proba_select" value="area / area_ib_tot"/>
        </ask>
      ...
      </init>
  </global>
```

Now, we have for each **building** agent, a probability to be selected. Now, we search to assign a living_place and a working_place to each **people** agent. But first, we define a speed, a start_work and end_work to each **people** agent (according to the min and max define in the parameters). Concerning the choice of the living_place and working_place, we draw a random number between 0 and 1 and we loop over the list of 'Residential' and 'Industrial' buildings while considering their probability to be selected. When the number drawn is lower to the sum of proba_select of the already enumerate buildings, we assign the building to the corresponding variable.

```
  <global>
      <init>
        ...
        <create species="people" number="nb_people">
            <set name="speed" value="min_speed + rnd (max_speed - min_speed)"/
>
            <set name="start_work" value="min_work_start + rnd (max_work_start
- min_work_start)"/>
            <set name="end_work" value="min_work_end + rnd (max_work_end -
min_work_end)"/>
            <let type="float" name="rand_nb" value="rnd 1000 / 1000"/>
            <let type="float" name="proba_sum" value="0"/>
            <loop over="residential_buildings" var="the_rb">
              <set name="proba_sum" value="proba_sum + the_rb.proba_select"/>
              <if condition="(living_place = nil) and (rand_nb &lt;
proba_sum)">
                    <set name="living_place" value="the_rb"/>
              </if>
            </loop>
            <set name="rand_nb" value="rnd 1000 / 1000"/>
            <set name="proba_sum" value="0"/>
            <loop over="industrial_buildings" var="the_ib">
              <set name="proba_sum" value="proba_sum + the_ib.proba_select"/>
              <if condition="(working_place = nil) and (rand_nb &lt;
proba_sum)">
                    <set name="working_place" value="the_ib"/>
              </if>
            </loop>
            <set name="location" value="self place_in [agent::living_place]"/>
```

```
        </create>
      </init>
  </global>
```

Complete model

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
      <var type="int" name="day_time" value="time%144" />
      <var type="int" name="min_work_start" init="36" parameter="Earliest hour
to start work" category="People" />
      <var type="int" name="max_work_start" init="60" parameter="Latest hour
to start work" category="People" />
      <var type="int" name="min_work_end" init="84" parameter="Earliest hour
to end work" category="People" />
      <var type="int" name="max_work_end" init="132" parameter="Latest hour to
end work" category="People" />
      <var type="float" name="min_speed" init="50" parameter="minimal speed"
category="People" />
      <var type="float" name="max_speed" init="100" parameter="maximal speed"
category="People" />

      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>

        <create species="road" from="shape_file_roads" return="the_roads"/>

        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
        </do>
        <let type="list" name="residential_buildings" value="the_buildings
where (each.type='Residential')"/>
```

```
        <let type="list" name="industrial_buildings" value="the_buildings
where (each.type='Industrial')"/>
        <let type="float" name="area_rb_tot" value="self area_sum
[agents::residential_buildings]"/>
        <ask target="residential_buildings">
            <set name="proba_select" value="area / area_rb_tot"/>
        </ask>
        <let type="float" name="area_ib_tot" value="self area_sum
[agents::industrial_buildings]"/>
        <ask target="industrial_buildings">
            <set name="proba_select" value="area / area_ib_tot"/>
        </ask>

        <create species="people" number="nb_people">
            <set name="speed" value="min_speed + rnd (max_speed - min_speed)"/
>
            <set name="start_work" value="min_work_start + rnd (max_work_start
- min_work_start)"/>
            <set name="end_work" value="min_work_end + rnd (max_work_end -
min_work_end)"/>
            <let type="float" name="rand_nb" value="rnd 1000 / 1000"/>
            <let type="float" name="proba_sum" value="0"/>
            <loop over="residential_buildings" var="the_rb">
              <set name="proba_sum" value="proba_sum + the_rb.proba_select"/>
              <if condition="(living_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="living_place" value="the_rb"/>
              </if>
            </loop>
            <set name="rand_nb" value="rnd 1000 / 1000"/>
            <set name="proba_sum" value="0"/>
            <loop over="industrial_buildings" var="the_ib">
              <set name="proba_sum" value="proba_sum + the_ib.proba_select"/>
              <if condition="(working_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="working_place" value="the_ib"/>
              </if>
            </loop>
            <set name="location" value="self place_in [agent::living_place]"/>
        </create>
      </init>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />
        <var type="float" name="proba_select"/>

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
```

```
      </species>
      <species name="road" skills="situated">
        <var type="rgb" name="color" init="rgb 'black'" />

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

      <species name="people" skills="moving">
        <var type="rgb" name="color" init="rgb 'yellow'" />
        <var type="building" name="living_place" init="nil"/>
        <var type="building" name="working_place" init="nil"/>
        <var type="int" name="start_work"/>
        <var type="int" name="end_work"/>
        <var type="string" name="objectif"/>
        <var type="point" name="the_target" init="nil"/>

        <aspect name="base">
            <draw shape="geometry" color="color" size="10"/>
        </aspect>

        <reflex name="time_to_work" when="day_time = start_work">
            <set name="objectif" value="'working'"/>
            <set name="the_target" value="self place_in
[agent::working_place]"/>
        </reflex>

        <reflex name="time_to_go_home" when="day_time = end_work">
            <set name="objectif" value="'go home'"/>
            <set name="the_target" value="self place_in
[agent::living_place]"/>
        </reflex>

        <reflex name="move" when="the_target != nil">
            <do action="goto">
              <arg name="target" value="the_target"/>
              <arg name="graph_name" value="'road_network'"/>
            </do>
            <if condition="the_target = location">
              <set name="the_target" value="nil"/>
            </if>
        </reflex>
      </species>
  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
```

```
        <species name="people" aspect="base"/>
      </display>
  </output>
</model>
```

4. Definition of weight for the road network

<wiki:toc max_depth="3" />

Purpose

Illustrates how to add weights to a network of polylines.

Formulation

- Add a **destruction_coeff** variable to the **road** agent. The value of this variable is higher or equal to 1. At initialisation, the value of this variable is randomly defined between 1 and 2.
- In the road network graph, more a road is worn out (destruction_coeff high), more a **people** agent takes time to go all over it. Then the value of the arc representing the road in the graph is equal to "length of the road * destruction_coeff".
- The color of the road depends of the **destruction_coeff** . If "destruction_coeff = 1", the road is green, if "destruction_coeff = 2", the road is red.

Model

Road agent

We have to add a **destruction_coeff** variable which initial value is randomly defined between 1 and 2. The color of the agent will depend of this variable.

```
  <species name="road" skills="situated">
      <var type="float" name="destruction_coeff" init="1 + ((rnd 100)/
100.0)"/>
      <var type="rgb" name="color" value="[min [255, 255`*`(destruction_coeff
- 1)],max [0, 255 - (255`*`(destruction_coeff - 1))],0]" />
      ...
```

```
  </species>
```

Weigthed road network

In GAMA, adding a weight for a graph is very simple, we just have to specified with the **weight** argument, the name of the variable that will be used as a weighting coefficient. In our context, it is the **destruction_coeff** variable.

```
  <global>
    ...
      <init>
        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
            <arg name="weights" value="'destruction_coeff'"/>
        </do>
      ...
      </init>
  </global>
```

Complete model

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
      <var type="int" name="day_time" value="time%144" />
      <var type="int" name="min_work_start" init="36" parameter="Earliest hour
to start work" category="People" />
      <var type="int" name="max_work_start" init="60" parameter="Latest hour
to start work" category="People" />
      <var type="int" name="min_work_end" init="84" parameter="Earliest hour
to end work" category="People" />
      <var type="int" name="max_work_end" init="132" parameter="Latest hour to
end work" category="People" />
      <var type="float" name="min_speed" init="50" parameter="minimal speed"
category="People" />
```

```
      <var type="float" name="max_speed" init="100" parameter="maximal speed"
category="People" />

      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>

        <create species="road" from="shape_file_roads" return="the_roads"/>

        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
            <arg name="weights" value="'destruction_coeff'"/>
        </do>
        <let type="list" name="residential_buildings" value="the_buildings
where (each.type='Residential')"/>
        <let type="list" name="industrial_buildings" value="the_buildings
where (each.type='Industrial')"/>
        <let type="float" name="area_rb_tot" value="self area_sum
[agents::residential_buildings]"/>
        <ask target="residential_buildings">
            <set name="proba_select" value="area / area_rb_tot"/>
        </ask>
        <let type="float" name="area_ib_tot" value="self area_sum
[agents::industrial_buildings]"/>
        <ask target="industrial_buildings">
            <set name="proba_select" value="area / area_ib_tot"/>
        </ask>

        <create species="people" number="nb_people">
            <set name="speed" value="min_speed + rnd (max_speed - min_speed)"/
>
            <set name="start_work" value="min_work_start + rnd (max_work_start
- min_work_start)"/>
            <set name="end_work" value="min_work_end + rnd (max_work_end -
min_work_end)"/>
            <let type="float" name="rand_nb" value="rnd 1000 / 1000"/>
            <let type="float" name="proba_sum" value="0"/>
            <loop over="residential_buildings" var="the_rb">
              <set name="proba_sum" value="proba_sum + the_rb.proba_select"/>
              <if condition="(living_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="living_place" value="the_rb"/>
              </if>
            </loop>
            <set name="rand_nb" value="rnd 1000 / 1000"/>
            <set name="proba_sum" value="0"/>
```

```
            <loop over="industrial_buildings" var="the_ib">
              <set name="proba_sum" value="proba_sum + the_ib.proba_select"/>
              <if condition="(working_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="working_place" value="the_ib"/>
              </if>
            </loop>
            <set name="location" value="self place_in [agent::living_place]"/>
        </create>
      </init>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />
        <var type="float" name="proba_select"/>

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="float" name="destruction_coeff" init="1 + ((rnd 100)/
100.0)"/>
        <var type="rgb" name="color" value="[min [255, 255*(destruction_coeff
- 1)],max [0, 255 - (255*(destruction_coeff - 1))],0]" />
        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

      <species name="people" skills="moving">
        <var type="rgb" name="color" init="rgb 'yellow'" />
        <var type="building" name="living_place" init="nil"/>
        <var type="building" name="working_place" init="nil"/>
        <var type="int" name="start_work"/>
        <var type="int" name="end_work"/>
        <var type="string" name="objectif"/>
        <var type="point" name="the_target" init="nil"/>

        <aspect name="base">
            <draw shape="geometry" color="color" size="10"/>
        </aspect>

        <reflex name="time_to_work" when="day_time = start_work">
            <set name="objectif" value="'working'"/>
            <set name="the_target" value="self place_in
[agent::working_place]"/>
        </reflex>

        <reflex name="time_to_go_home" when="day_time = end_work">
```

```
            <set name="objectif" value="'go home'"/>
            <set name="the_target" value="self place_in
[agent::living_place]"/>
        </reflex>

        <reflex name="move" when="the_target != nil">
            <do action="goto">
              <arg name="target" value="the_target"/>
              <arg name="graph_name" value="'road_network'"/>
            </do>
            <if condition="the_target = location">
              <set name="the_target" value="nil"/>
            </if>
        </reflex>
      </species>
  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
        <species name="people" aspect="base"/>
      </display>
  </output>
</model>
```

_

5. Dynamic update of the road network

<wiki:toc max_depth="3" />

Purpose

Illustrates how to obtain a shortest path from a point to another and to update the weights of an existing graph.

Formulation

- At initialisation, the value of the **destruction_coeff** of the **road** agents will be equal to 1.
- Add a new parameter: the **destroy** parameter that represents the value of destruction when a people agent takes a road. By default, it is equal to 0.02.
- When an people arrive at its destination (home or work), it updates the **destruction_coeff** of the **road** agents it took to reach its destination: "destruction_coeff = destruction_coeff - destroy". Then, the graph is updated.

Model

Global section

We have to add the **destroy** parameter. In addition, we add the **update_roads** global variable that indicates if the graph has to be update or not.

```
  <global>
      ...
      <var type="float" name="destroy" init="0.02" parameter="Value of
destruction when a people agent takes a road" category="Road" />
      <var type="bool" name="update_roads" init="false" />
      ...
```

```
</global>
```

We have to update when necessary (when "update_graph = true") the graph. For that, we use the **update_graph** action of the **situated** skill. This action allows to update an existing graph and to compute, if necessary, the new shortest pathes.

```
<global>
    ...
    <reflex name="update_graph" when="update_roads">
      <do action="update_graph">
          <arg name="network" value="list road" />
          <arg name="graph_name" value="'road_network'" />
          <arg name="weights" value="'destruction_coeff'" />
      </do>
      <set name="update_roads" value="false" />
    </reflex>
</global>
```

People agents

We have to integrate the fact that the **people** agents are going to destroy the roads. First, we define a new variable for these agents: the **path** varibale that represents the list of **road** agents that the **people** agent has to take to reach its destination.

```
<species name="people" skills="moving">
    ...
    <var type="list" of="road" name="path" />
    ...
</species>
```

Then, we have to compute the value of this variable, each time the agent is selecting a destination. For that, we use the **path_graph** action of the **situated** skill. This action allows to compute, from a graph, the agents crossed to reach a target.

```
<species name="people" skills="moving">
    ...
    <reflex name="time_to_work" when="day_time = start_work">
      <set name="objectif" value="'working'" />
      <set name="the_target" value="self place_in [agent::working_place]" />
      <set name="path" value="self path_graph [graph_name::'road_network',
target::the_target]" />
    </reflex>

    <reflex name="time_to_go_home" when="day_time = end_work">
      <set name="objectif" value="'go home'" />
      <set name="the_target" value="self place_in [agent::living_place]" />
      <set name="path" value="self path_graph [graph_name::'road_network',
target::the_target]" />
```

```
      </reflex>
      ...
  </species>
```

Finally, when arrived at destination, the **people** agents have to update the value of the **road** agents crossed and to indicate that the graph need to be updated.

```
  <species name="people" skills="moving">
      ...
      <reflex name="move" when="the_target != nil">
        <do action="goto">
            <arg name="target" value="the_target" />
            <arg name="graph_name" value="'road_network'" />
        </do>
        <if condition="the_target = location">
            <set name="the_target" value="nil" />
            <ask target="path">
              <set name="destruction_coeff" value="destroy +
destruction_coeff" />
            </ask>
            <set name="update_roads" value="true" />
        </if>
      </reflex>
      ...
  </species>
```

Complete model

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
      <var type="int" name="day_time" value="time%144" />
      <var type="int" name="min_work_start" init="36" parameter="Earliest hour
to start work" category="People" />
      <var type="int" name="max_work_start" init="60" parameter="Latest hour
to start work" category="People" />
      <var type="int" name="min_work_end" init="84" parameter="Earliest hour
to end work" category="People" />
```

```
      <var type="int" name="max_work_end" init="132" parameter="Latest hour to
end work" category="People" />
      <var type="float" name="min_speed" init="50" parameter="minimal speed"
category="People" />
      <var type="float" name="max_speed" init="100" parameter="maximal speed"
category="People" />
      <var type="float" name="destroy" init="0.02" parameter="Value of
destruction when a people agent takes a road" category="Road" />
      <var type="bool" name="update_roads" init="false" />
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>

        <create species="road" from="shape_file_roads" return="the_roads"/>

        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
            <arg name="weights" value="'destruction_coeff'"/>
        </do>
        <let type="list" name="residential_buildings" value="the_buildings
where (each.type='Residential')"/>
        <let type="list" name="industrial_buildings" value="the_buildings
where (each.type='Industrial')"/>
        <let type="float" name="area_rb_tot" value="self area_sum
[agents::residential_buildings]"/>
        <ask target="residential_buildings">
            <set name="proba_select" value="area / area_rb_tot"/>
        </ask>
        <let type="float" name="area_ib_tot" value="self area_sum
[agents::industrial_buildings]"/>
        <ask target="industrial_buildings">
            <set name="proba_select" value="area / area_ib_tot"/>
        </ask>

        <create species="people" number="nb_people">
            <set name="speed" value="min_speed + rnd (max_speed - min_speed)"/
>
            <set name="start_work" value="min_work_start + rnd (max_work_start
- min_work_start)"/>
            <set name="end_work" value="min_work_end + rnd (max_work_end -
min_work_end)"/>
            <let type="float" name="rand_nb" value="rnd 1000 / 1000"/>
            <let type="float" name="proba_sum" value="0"/>
            <loop over="residential_buildings" var="the_rb">
              <set name="proba_sum" value="proba_sum + the_rb.proba_select"/>
```

```
              <if condition="(living_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="living_place" value="the_rb"/>
              </if>
            </loop>
            <set name="rand_nb" value="rnd 1000 / 1000"/>
            <set name="proba_sum" value="0"/>
            <loop over="industrial_buildings" var="the_ib">
              <set name="proba_sum" value="proba_sum + the_ib.proba_select"/>
              <if condition="(working_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="working_place" value="the_ib"/>
              </if>
            </loop>
            <set name="location" value="self place_in [agent::living_place]"/>
        </create>
      </init>
      <reflex name="update_graph" when="update_roads">
        <do action="update_graph">
            <arg name="network" value="list road" />
            <arg name="graph_name" value="'road_network'" />
            <arg name="weights" value="'destruction_coeff'" />
        </do>
        <set name="update_roads" value="false" />
      </reflex>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />
        <var type="float" name="proba_select"/>

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="float" name="destruction_coeff" init="1"/>
        <var type="rgb" name="color" value="[min [255, 255*(destruction_coeff
- 1)],max [0, 255 - (255*(destruction_coeff - 1))],0]" />
        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

      <species name="people" skills="moving">
        <var type="rgb" name="color" init="rgb 'yellow'" />
        <var type="building" name="living_place" init="nil"/>
        <var type="building" name="working_place" init="nil"/>
        <var type="int" name="start_work"/>
        <var type="int" name="end_work"/>
```

```
        <var type="string" name="objectif"/>
        <var type="point" name="the_target" init="nil"/>
        <var type="list" of="road" name="path" />

        <aspect name="base">
            <draw shape="geometry" color="color" size="10"/>
        </aspect>

        <reflex name="time_to_work" when="day_time = start_work">
            <set name="objectif" value="'working'"/>
            <set name="the_target" value="self place_in
[agent::working_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="time_to_go_home" when="day_time = end_work">
            <set name="objectif" value="'go home'"/>
            <set name="the_target" value="self place_in
[agent::living_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="move" when="the_target != nil">
            <do action="goto">
              <arg name="target" value="the_target" />
              <arg name="graph_name" value="'road_network'" />
            </do>
            <if condition="the_target = location">
              <set name="the_target" value="nil" />
              <ask target="path">
                  <set name="destruction_coeff" value="destroy +
destruction_coeff" />
              </ask>
              <set name="update_roads" value="true" />
            </if>
        </reflex>
      </species>
  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
        <species name="people" aspect="base"/>
      </display>
  </output>
</model>
```

6. Definition of a chart display

<wiki:toc max_depth="3" />

Purpose

Illustrates how to display charts.

Formulation

- Add a chart to display the evolution of the road destruction: the mean value of the **destruction_coeff** of the **road** agents, and its max value (refreshed every 10 simulation steps).
- Add a chart to display the activity of the **people** agent (working or staying home, refreshed every 10 simulation steps).

Model

Chart display

First we have to add a chart of type **series** to display the road destruction evolution. To compute the mean of the **destruction_coeff** , we use the **mean** operator. For the max, we use the **max_of** operator.

```
  <output>
      ...
      <display name="chart_display" refresh_every="10">
        <chart type="series" name="Road Status" background="rgb 'lightGray'"
size="{0.9, 0.4}" position="{0.05, 0.05}">
            <data name="Mean road destruction" value="mean ((list road)
collect each.destruction_coeff)" style="line" color="rgb 'green'" />
            <data name="Max road destruction" value="(list road) max_of
(each.destruction_coeff)" style="line" color="rgb 'red'" />
        </chart>
          ...
      </display>
```

```
  </output>
```

Second, we have to add a chart of type **pie** to display the activity of the **people** agents. We use for that the **objective** variable of the **people** agents.

```
  <output>
      ...
      <display name="chart_display" refresh_every="10">
          ...
        <chart type="pie" name="People Objectif" background="rgb 'lightGray'"
style="exploded"  size="{0.9, 0.4}" position="{0.05, 0.55}">
            <data name="Working" value="length ((list people) where
(each.objectif='working'))" color="rgb 'green'"/>
            <data name="Staying home" value="length ((list people) where
(each.objectif='go home'))" color="rgb 'blue'"/>
        </chart>
      </display>
  </output>
```

Complete model

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
      <var type="int" name="day_time" value="time%144" />
      <var type="int" name="min_work_start" init="36" parameter="Earliest hour
to start work" category="People" />
      <var type="int" name="max_work_start" init="60" parameter="Latest hour
to start work" category="People" />
      <var type="int" name="min_work_end" init="84" parameter="Earliest hour
to end work" category="People" />
      <var type="int" name="max_work_end" init="132" parameter="Latest hour to
end work" category="People" />
      <var type="float" name="min_speed" init="50" parameter="minimal speed"
category="People" />
      <var type="float" name="max_speed" init="100" parameter="maximal speed"
category="People" />
      <var type="float" name="destroy" init="0.02" parameter="Value of
destruction when a people agent takes a road" category="Road" />
```

```
      <var type="bool" name="update_roads" init="false" />
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>

        <create species="road" from="shape_file_roads" return="the_roads"/>

        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
            <arg name="weights" value="'destruction_coeff'"/>
        </do>
        <let type="list" name="residential_buildings" value="the_buildings
where (each.type='Residential')"/>
        <let type="list" name="industrial_buildings" value="the_buildings
where (each.type='Industrial')"/>
        <let type="float" name="area_rb_tot" value="self area_sum
[agents::residential_buildings]"/>
        <ask target="residential_buildings">
            <set name="proba_select" value="area / area_rb_tot"/>
        </ask>
        <let type="float" name="area_ib_tot" value="self area_sum
[agents::industrial_buildings]"/>
        <ask target="industrial_buildings">
            <set name="proba_select" value="area / area_ib_tot"/>
        </ask>

        <create species="people" number="nb_people">
            <set name="speed" value="min_speed + rnd (max_speed - min_speed)"/
>
            <set name="start_work" value="min_work_start + rnd (max_work_start
- min_work_start)"/>
            <set name="end_work" value="min_work_end + rnd (max_work_end -
min_work_end)"/>
            <let type="float" name="rand_nb" value="rnd 1000 / 1000"/>
            <let type="float" name="proba_sum" value="0"/>
            <loop over="residential_buildings" var="the_rb">
              <set name="proba_sum" value="proba_sum + the_rb.proba_select"/>
              <if condition="(living_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="living_place" value="the_rb"/>
              </if>
            </loop>
            <set name="rand_nb" value="rnd 1000 / 1000"/>
            <set name="proba_sum" value="0"/>
            <loop over="industrial_buildings" var="the_ib">
              <set name="proba_sum" value="proba_sum + the_ib.proba_select"/>
```

```
              <if condition="(working_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="working_place" value="the_ib"/>
              </if>
            </loop>
            <set name="location" value="self place_in [agent::living_place]"/>
        </create>
      </init>
      <reflex name="update_graph" when="update_roads">
        <do action="update_graph">
            <arg name="network" value="list road" />
            <arg name="graph_name" value="'road_network'" />
            <arg name="weights" value="'destruction_coeff'" />
        </do>
        <set name="update_roads" value="false" />
      </reflex>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />
        <var type="float" name="proba_select"/>

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="float" name="destruction_coeff" init="1"/>
        <var type="rgb" name="color" value="[min [255, 255*(destruction_coeff
- 1)],max [0, 255 - (255*(destruction_coeff - 1))],0]" />
        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

      <species name="people" skills="moving">
        <var type="rgb" name="color" init="rgb 'yellow'" />
        <var type="building" name="living_place" init="nil"/>
        <var type="building" name="working_place" init="nil"/>
        <var type="int" name="start_work"/>
        <var type="int" name="end_work"/>
        <var type="string" name="objectif"/>
        <var type="point" name="the_target" init="nil"/>
        <var type="list" of="road" name="path" />

        <aspect name="base">
            <draw shape="geometry" color="color" size="10"/>
        </aspect>

        <reflex name="time_to_work" when="day_time = start_work">
```

```
            <set name="objectif" value="'working'"/>
            <set name="the_target" value="self place_in
[agent::working_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="time_to_go_home" when="day_time = end_work">
            <set name="objectif" value="'go home'"/>
            <set name="the_target" value="self place_in
[agent::living_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="move" when="the_target != nil">
            <do action="goto">
              <arg name="target" value="the_target" />
              <arg name="graph_name" value="'road_network'" />
            </do>
            <if condition="the_target = location">
              <set name="the_target" value="nil" />
              <ask target="path">
                  <set name="destruction_coeff" value="destroy +
destruction_coeff" />
              </ask>
              <set name="update_roads" value="true" />
            </if>
        </reflex>
      </species>
  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
        <species name="people" aspect="base"/>
      </display>
      <display name="chart_display" refresh_every="10">
        <chart type="series" name="Road Status" background="rgb 'lightGray'"
size="{0.9, 0.4}" position="{0.05, 0.05}">
            <data name="Mean road destruction" value="mean ((list road)
collect each.destruction_coeff)" style="line" color="rgb 'green'" />
            <data name="Max road destruction" value="(list road) max_of
(each.destruction_coeff)" style="line" color="rgb 'red'" />
        </chart>
        <chart type="pie" name="People Objectif" background="rgb 'lightGray'"
style="exploded"  size="{0.9, 0.4}" position="{0.05, 0.55}">
            <data name="Working" value="length ((list people) where
(each.objectif='working'))" color="rgb 'green'"/>
```

```
            <data name="Staying home" value="length ((list people) where
(each.objectif='go home'))" color="rgb 'blue'"/>
        </chart>
      </display>
  </output>
</model>
```

7. Automatic repair of roads

<wiki:toc max_depth="3" />

Purpose

Illustrates how to select in a list an element that optimise a given function.

Formulation

- add a new parameter, **repair_time** , that is equal to 6.
- Every **repair_time** , the **road** with the highest **destruction_coeff** value is repaired (set its **destruction_coeff** to 1).

Model

Paremeters

We have to add a new parameter: the **repair_time** .

```
  <global>
      ...
      <var type="int" name="repair_time" init="6" parameter="Number of steps
between two road repairs" category="Road" />
      ...
  </global>
```

Road repairing

We have to add a reflex method in the global section that is triggered every **repair_time** . This method selects, thanks to the **with_max_of** operation the **road** agent with the highest **destruction_coeff** value, then sets this value at 1.

```
  <global>
    ...
```

```
      <reflex name="repair_road" when="time%repair_time = 0">
        <let type="road" name="the_road_to_repair" value="(list road)
with_max_of (each.destruction_coeff)"/>
        <ask target="the_road_to_repair">
            <set name="destruction_coeff" value="1"/>
        </ask>
        <set name="update_roads" value="true" />
      </reflex>
    ...
  </global>
```

Complete model

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
      <var type="int" name="day_time" value="time%144" />
      <var type="int" name="min_work_start" init="36" parameter="Earliest hour
to start work" category="People" />
      <var type="int" name="max_work_start" init="60" parameter="Latest hour
to start work" category="People" />
      <var type="int" name="min_work_end" init="84" parameter="Earliest hour
to end work" category="People" />
      <var type="int" name="max_work_end" init="132" parameter="Latest hour to
end work" category="People" />
      <var type="float" name="min_speed" init="50" parameter="minimal speed"
category="People" />
      <var type="float" name="max_speed" init="100" parameter="maximal speed"
category="People" />
      <var type="float" name="destroy" init="0.02" parameter="Value of
destruction when a people agent takes a road" category="Road" />
      <var type="bool" name="update_roads" init="false" />
      <var type="int" name="repair_time" init="6" parameter="Number of steps
between two road repairs" category="Road" />
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
```

```
            </if>
        </create>

        <create species="road" from="shape_file_roads" return="the_roads"/>

        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
            <arg name="weights" value="'destruction_coeff'"/>
        </do>
        <let type="list" name="residential_buildings" value="the_buildings
where (each.type='Residential')"/>
        <let type="list" name="industrial_buildings" value="the_buildings
where (each.type='Industrial')"/>
        <let type="float" name="area_rb_tot" value="self area_sum
[agents::residential_buildings]"/>
        <ask target="residential_buildings">
            <set name="proba_select" value="area / area_rb_tot"/>
        </ask>
        <let type="float" name="area_ib_tot" value="self area_sum
[agents::industrial_buildings]"/>
        <ask target="industrial_buildings">
            <set name="proba_select" value="area / area_ib_tot"/>
        </ask>

        <create species="people" number="nb_people">
            <set name="speed" value="min_speed + rnd (max_speed - min_speed)"/
>
            <set name="start_work" value="min_work_start + rnd (max_work_start
- min_work_start)"/>
            <set name="end_work" value="min_work_end + rnd (max_work_end -
min_work_end)"/>
            <let type="float" name="rand_nb" value="rnd 1000 / 1000"/>
            <let type="float" name="proba_sum" value="0"/>
            <loop over="residential_buildings" var="the_rb">
              <set name="proba_sum" value="proba_sum + the_rb.proba_select"/>
              <if condition="(living_place = nil) and (rand_nb &lt;
proba_sum)">
                    <set name="living_place" value="the_rb"/>
              </if>
            </loop>
            <set name="rand_nb" value="rnd 1000 / 1000"/>
            <set name="proba_sum" value="0"/>
            <loop over="industrial_buildings" var="the_ib">
              <set name="proba_sum" value="proba_sum + the_ib.proba_select"/>
              <if condition="(working_place = nil) and (rand_nb &lt;
proba_sum)">
                    <set name="working_place" value="the_ib"/>
              </if>
            </loop>
            <set name="location" value="self place_in [agent::living_place]"/>
```

```
        </create>
      </init>
      <reflex name="repair_road" when="time%repair_time = 0">
        <let type="road" name="the_road_to_repair" value="(list road)
with_max_of (each.destruction_coeff)"/>
        <ask target="the_road_to_repair">
            <set name="destruction_coeff" value="1"/>
        </ask>
        <set name="update_roads" value="true" />
      </reflex>
      <reflex name="update_graph" when="update_roads">
        <do action="update_graph">
            <arg name="network" value="list road" />
            <arg name="graph_name" value="'road_network'" />
            <arg name="weights" value="'destruction_coeff'" />
        </do>
        <set name="update_roads" value="false" />
      </reflex>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />
        <var type="float" name="proba_select"/>

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="float" name="destruction_coeff" init="1"/>
        <var type="rgb" name="color" value="[min [255, 255*(destruction_coeff
- 1)],max [0, 255 - (255*(destruction_coeff - 1))],0]" />
        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

      <species name="people" skills="moving">
        <var type="rgb" name="color" init="rgb 'yellow'" />
        <var type="building" name="living_place" init="nil"/>
        <var type="building" name="working_place" init="nil"/>
        <var type="int" name="start_work"/>
        <var type="int" name="end_work"/>
        <var type="string" name="objectif"/>
        <var type="point" name="the_target" init="nil"/>
        <var type="list" of="road" name="path" />

        <aspect name="base">
            <draw shape="geometry" color="color" size="10"/>
        </aspect>
```

```
        <reflex name="time_to_work" when="day_time = start_work">
            <set name="objectif" value="'working'"/>
            <set name="the_target" value="self place_in
[agent::working_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="time_to_go_home" when="day_time = end_work">
            <set name="objectif" value="'go home'"/>
            <set name="the_target" value="self place_in
[agent::living_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="move" when="the_target != nil">
            <do action="goto">
              <arg name="target" value="the_target" />
              <arg name="graph_name" value="'road_network'" />
            </do>
            <if condition="the_target = location">
              <set name="the_target" value="nil" />
              <ask target="path">
                  <set name="destruction_coeff" value="destroy +
destruction_coeff" />
              </ask>
              <set name="update_roads" value="true" />
            </if>
        </reflex>
      </species>
  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
        <species name="people" aspect="base"/>
      </display>
      <display name="chart_display" refresh_every="10">
        <chart type="series" name="Road Status" background="rgb 'lightGray'"
size="{0.9, 0.4}" position="{0.05, 0.05}">
            <data name="Mean road destruction" value="mean ((list road)
collect each.destruction_coeff)" style="line" color="rgb 'green'" />
            <data name="Max road destruction" value="(list road) max_of
(each.destruction_coeff)" style="line" color="rgb 'red'" />
        </chart>
        <chart type="pie" name="People Objectif" background="rgb 'lightGray'"
style="exploded"  size="{0.9, 0.4}" position="{0.05, 0.55}">
```

```
            <data name="Working" value="length ((list people) where
(each.objectif='working'))" color="rgb 'green'"/>
            <data name="Staying home" value="length ((list people) where
(each.objectif='go home'))" color="rgb 'blue'"/>
        </chart>
      </display>
  </output>
</model>
```

8. Complex repair of roads

<wiki:toc max_depth="3" />

Purpose

Illustrates how to select in a list an element that optimise a given function.

Formulation

- Replace the parameter **repair_time** by a parameter **repair_capacity_step** which is equal to 10. This parameter represents the number of meters of road that can be repaired of one destruction unit per simulation step.
- Every simulation step, the **road** agents that are the most destroyed are repaired. the process is the following: the **road** agent with the highest **destruction_coeff** is selected. If the repair capacity is high enough ("repair_capacity (initialy equals to repair_capacity_step) > (road length * (destruction_coeff - 1)"), the destruction_coeff of the road is set to "1", and the repair_capacity is updated ("repair_capacity = repair_capacity - road length * (destruction_coeff - 1)"). Another road is then selected, on so on. If the repair capacity is not high enough, the road is partially repaired ("destruction_coeff = destruction_coeff - repair_capacity / road length)".

Model

Paremeters

We have to add a new parameter: **repair_capacity_step** .

```
  <global>
      ...
      <var type="float" name="repair_capacity_step" init="10" parameter="Per
step, number of meters of road that can be repaired of one destruction unit"
category="Road" />
      ...
```

```
</global>
```

Road repairing

We have to add a reflex method in the global section that is triggered every simulation step. We use the attribute **perimeter** contains in the **situated** skill to compute the road length. Note that we remove the update_graph global variable as now the graph is updated at each simulation step.

```
  <global>
    ...
      <reflex name="repair_road">
        <let name="repair_capacity" value="repair_capacity_step" />
        <loop while="repair_capacity &gt; 0">
            <let type="road" name="the_road_to_repair" value="(list road)
with_max_of (each.destruction_coeff)" />
            <let type="float" name="road_requirement"
value="the_road_to_repair.perimeter * (the_road_to_repair.destruction_coeff -
1)" />
            <if condition="road_requirement  = 0">
              <set name="repair_capacity" value="0" />
              <else>
                  <if condition="road_requirement &lt; repair_capacity">
                    <ask target="the_road_to_repair">
                        <set name="destruction_coeff" value="1" />
                    </ask>
                    <set name="repair_capacity" value="repair_capacity -
road_requirement" />
                    <else>
                        <ask target="the_road_to_repair">
                          <set name="destruction_coeff"
value="destruction_coeff - (repair_capacity / perimeter)" />
                        </ask>
                        <set name="repair_capacity" value="0" />
                      </else>
                </if>
                  </else>
              </if>
            </loop>
      </reflex>
    ...
  </global>
```

Complete model

```
<model name="tutorial_gis_city_traffic">
  <global>
      <var type="string" name="shape_file_buildings" init="'../gis/
building.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_roads" init="'../gis/road.shp'"
parameter="Shapefile for the roads:" category="GIS" />
      <var type="string" name="shape_file_bounds" init="'../gis/bounds.shp'"
parameter="Shapefile for the bounds:" category="GIS" />
      <var type="int" name="nb_people" init="100" parameter="Number of people
agents" category="People" />
      <var type="int" name="day_time" value="time%144" />
      <var type="int" name="min_work_start" init="36" parameter="Earliest hour
to start work" category="People" />
      <var type="int" name="max_work_start" init="60" parameter="Latest hour
to start work" category="People" />
      <var type="int" name="min_work_end" init="84" parameter="Earliest hour
to end work" category="People" />
      <var type="int" name="max_work_end" init="132" parameter="Latest hour to
end work" category="People" />
      <var type="float" name="min_speed" init="50" parameter="minimal speed"
category="People" />
      <var type="float" name="max_speed" init="100" parameter="maximal speed"
category="People" />
      <var type="float" name="destroy" init="0.02" parameter="Value of
destruction when a people agent takes a road" category="Road" />
      <var type="float" name="repair_capacity_step" init="10" parameter="Per
step, number of meters of road that can be repaired of one destruction unit"
category="Road" />
      <init>
        <create species="building" from="shape_file_buildings"
with="[type::read 'NATURE']" return="the_buildings">
            <if condition="type='Industrial'">
              <set name="color" value="rgb 'blue'"/>
            </if>
        </create>

        <create species="road" from="shape_file_roads" return="the_roads"/>

        <do action="compute_graph">
            <arg name="network" value="the_roads"/>
            <arg name="name" value="'road_network'"/>
            <arg name="weights" value="'destruction_coeff'"/>
        </do>
        <let type="list" name="residential_buildings" value="the_buildings
where (each.type='Residential')"/>
```

```
        <let type="list" name="industrial_buildings" value="the_buildings
where (each.type='Industrial')"/>
        <let type="float" name="area_rb_tot" value="self area_sum
[agents::residential_buildings]"/>
        <ask target="residential_buildings">
            <set name="proba_select" value="area / area_rb_tot"/>
        </ask>
        <let type="float" name="area_ib_tot" value="self area_sum
[agents::industrial_buildings]"/>
        <ask target="industrial_buildings">
            <set name="proba_select" value="area / area_ib_tot"/>
        </ask>

        <create species="people" number="nb_people">
            <set name="speed" value="min_speed + rnd (max_speed - min_speed)"/
>
            <set name="start_work" value="min_work_start + rnd (max_work_start
- min_work_start)"/>
            <set name="end_work" value="min_work_end + rnd (max_work_end -
min_work_end)"/>
            <let type="float" name="rand_nb" value="rnd 1000 / 1000"/>
            <let type="float" name="proba_sum" value="0"/>
            <loop over="residential_buildings" var="the_rb">
              <set name="proba_sum" value="proba_sum + the_rb.proba_select"/>
              <if condition="(living_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="living_place" value="the_rb"/>
              </if>
            </loop>
            <set name="rand_nb" value="rnd 1000 / 1000"/>
            <set name="proba_sum" value="0"/>
            <loop over="industrial_buildings" var="the_ib">
              <set name="proba_sum" value="proba_sum + the_ib.proba_select"/>
              <if condition="(working_place = nil) and (rand_nb &lt;
proba_sum)">
                  <set name="working_place" value="the_ib"/>
              </if>
            </loop>
            <set name="location" value="self place_in [agent::living_place]"/>
        </create>
      </init>
      <reflex name="repair_road">
        <let name="repair_capacity" value="repair_capacity_step" />
        <loop while="repair_capacity &gt; 0">
            <let type="road" name="the_road_to_repair" value="(list road)
with_max_of (each.destruction_coeff)" />
            <let type="float" name="road_requirement"
value="the_road_to_repair.perimeter * (the_road_to_repair.destruction_coeff -
1)" />
            <if condition="road_requirement  = 0">
              <set name="repair_capacity" value="0" />
```

```
              <else>
                  <if condition="road_requirement &lt; repair_capacity">
                    <ask target="the_road_to_repair">
                        <set name="destruction_coeff" value="1" />
                    </ask>
                    <set name="repair_capacity" value="repair_capacity -
road_requirement" />
                    <else>
                        <ask target="the_road_to_repair">
                          <set name="destruction_coeff"
value="destruction_coeff - (repair_capacity / perimeter)" />
                        </ask>
                        <set name="repair_capacity" value="0" />
                      </else>
                </if>
                  </else>
              </if>
            </loop>
      </reflex>
      <reflex name="update_graph">
        <do action="update_graph">
            <arg name="network" value="list road" />
            <arg name="graph_name" value="'road_network'" />
            <arg name="weights" value="'destruction_coeff'" />
        </do>
      </reflex>
  </global>
  <entities>
      <species name="building" skills="situated">
        <var type="string" name="type"/>
        <var type="rgb" name="color" init="rgb 'gray' " />
        <var type="float" name="proba_select"/>

        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>
      <species name="road" skills="situated">
        <var type="float" name="destruction_coeff" init="1"/>
        <var type="rgb" name="color" value="[min [255, 255*(destruction_coeff
- 1)],max [0, 255 - (255*(destruction_coeff - 1))],0]" />
        <aspect name="base">
            <draw shape="geometry" color="color"/>
        </aspect>
      </species>

      <species name="people" skills="moving">
        <var type="rgb" name="color" init="rgb 'yellow'" />
        <var type="building" name="living_place" init="nil"/>
        <var type="building" name="working_place" init="nil"/>
        <var type="int" name="start_work"/>
```

```
        <var type="int" name="end_work"/>
        <var type="string" name="objectif"/>
        <var type="point" name="the_target" init="nil"/>
        <var type="list" of="road" name="path" />

        <aspect name="base">
            <draw shape="geometry" color="color" size="10"/>
        </aspect>

        <reflex name="time_to_work" when="day_time = start_work">
            <set name="objectif" value="'working'"/>
            <set name="the_target" value="self place_in
[agent::working_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="time_to_go_home" when="day_time = end_work">
            <set name="objectif" value="'go home'"/>
            <set name="the_target" value="self place_in
[agent::living_place]"/>
            <set name="path" value="self path_graph
[graph_name::'road_network', target::the_target]" />
        </reflex>

        <reflex name="move" when="the_target != nil">
            <do action="goto">
              <arg name="target" value="the_target" />
              <arg name="graph_name" value="'road_network'" />
            </do>
            <if condition="the_target = location">
              <set name="the_target" value="nil" />
              <ask target="path">
                  <set name="destruction_coeff" value="destroy +
destruction_coeff" />
              </ask>
            </if>
        </reflex>
      </species>
  </entities>
  <environment bounds="shape_file_bounds"/>

  <output>
      <display name="city_display" refresh_every="1">
        <species name="building" aspect="base"/>
        <species name="road" aspect="base"/>
        <species name="people" aspect="base"/>
      </display>
      <display name="chart_display" refresh_every="10">
        <chart type="series" name="Road Status" background="rgb 'lightGray'"
size="{0.9, 0.4}" position="{0.05, 0.05}">
```

```
            <data name="Mean road destruction" value="mean ((list road) 
collect each.destruction_coeff)" style="line" color="rgb 'green'" />
            <data name="Max road destruction" value="(list road) max_of 
(each.destruction_coeff)" style="line" color="rgb 'red'" />
        </chart>
        <chart type="pie" name="People Objectif" background="rgb 'lightGray'" 
style="exploded"  size="{0.9, 0.4}" position="{0.05, 0.55}">
            <data name="Working" value="length ((list people) where 
(each.objectif='working'))" color="rgb 'green'"/>
            <data name="Staying home" value="length ((list people) where 
(each.objectif='go home'))" color="rgb 'blue'"/>
        </chart>
      </display>
  </output>
</model>
```

—

Tutorial 3: Robot exploration

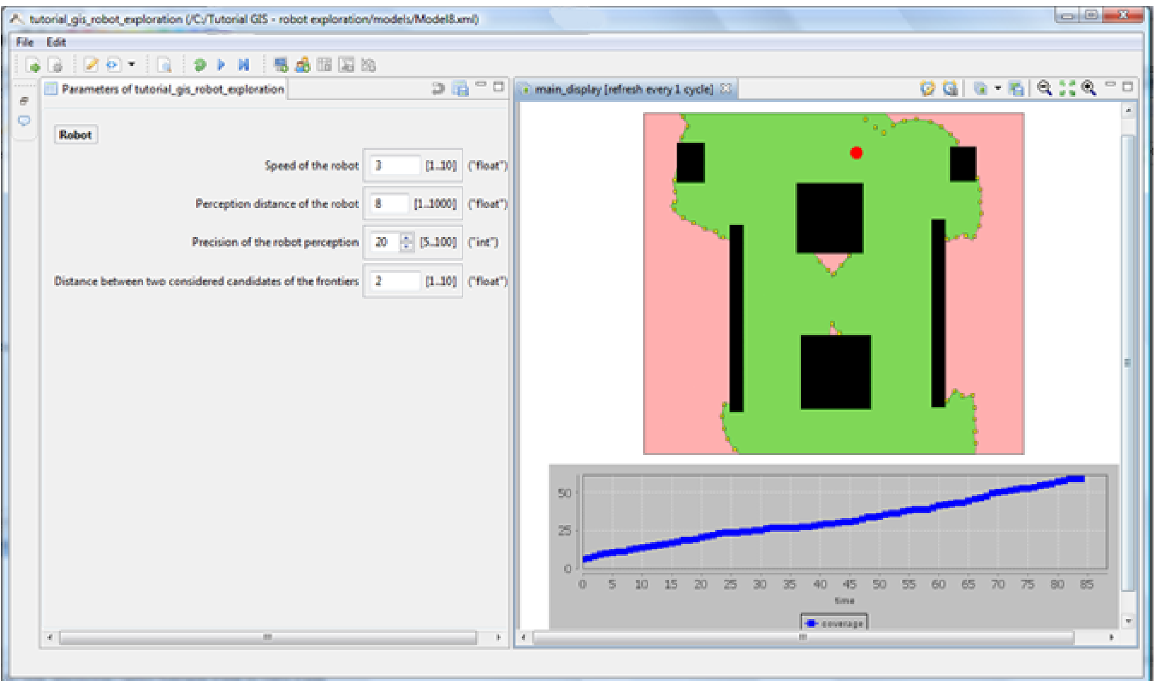
—

Introduction to the tutorial

summary Tutorial: Road traffic in a city <wiki:toc max_depth="3" />

Introduction

This tutorial has for goal to present the use of GIS data and complex geometries. In particular, this tutorial shows how to load gis data, to agentify them and to use a polygon to constraint the movement of agents. All the files related to this tutorial (shapefiles and models) are available here . The model built in this tutorial concerns the exploration of an unknown environment by a robot. Two layers of GIS data are used: a background layer (polygon) and a obstacles layer (polygons). In this model, a robot is moving inside the background, avoiding the obstacles. Its goal is to explore the space while covering the minimum distance. To achieve this goal, we propose to use a frontier-based approach : the robot is going to compute a set of candidate positions located at the frontier of the known environment, then to chose among them the one that maximises an evaluation function. In this tutorial, the evaluation function used is a simple computation of the distance between the current location of the robot and the candidate location: the closer, the better. <a href="



" title="Robot_exploration_image">

Robot exploration: model list

This tutorial is composed of 8 models. For each model we will present its purpose, an explicit formulation and the corresponding GAML code.

1. Loading of GIS data (background and obstacles)
2. Computation of the unkwnow environment
3. Integration of the robot agent
4. Dynamic building of the known environment
5. Use of a pathfinder for the robot movement
6. Frontier-based approach
7. Dynamic update of the pathfinder
8. Integration of a chart in the display

1. Loading of GIS data (background and obstacles)

<wiki:toc max_depth="3" />

Purpose

Illustrate how to load GIS data (shapefiles).

Formulation

- Load, agentify and display two layers of GIS data (background and obstacles). The background will be drawn in pink and the obstacles in black.

Model

Entities

In this first model, we have to define two species of agents: the **background** agents and the **obstacles** ones. These agents will not have a particular behaviour. They will just be displayed. Thus, we just have to define an aspect for the agents (see [Species Aspects] in the Modeling guide). In this model, we want to represent the geometry of the agent, we then use the value **geometry** of the facet **shape** for the command **draw** .

```
<entities>
  <species name="background" skills="situated">
      <aspect name="base">
          <draw shape="geometry" color="rgb 'pink'" />
      </aspect>
    </species>
  <species name="obstacle" skills="situated">
      <aspect name="base">
          <draw shape="geometry" color="rgb 'black'" />
      </aspect>
```

```
    </species>
</entities>
```

Parameters

GAMA allows to automatically read GIS data that are formatted as shapefiles. In order to let the user chooses his/her shapefiles, we define two parameters (string). One allowing the user to choose the background shapefile, one allowing him/her to choose the obstacles shapefile.

```
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
    ...
  </global>
```

Agentification of GIS data

In GAMA, the agentification of GIS data is very straightforward: it only requires to use the **create** command with the **from** facet to pass the reference of the shapefile. Each object of the shapefile will be directly used to instantiate an agent of the specified species.

```
    <global>
    ...
      <init>
        <create species="background" from="shape_file_background"/>
        <create species="obstacle" from="shape_file_obstacles"/>
      </init>
  </global>
```

Environment

Building a GIS environment in GAMA requires nothing special, just to define the bounds of the environment, i.e. the square envelop of the GIS data. It is possible to use a shapefile to automatically define it. In this model, we use the background shapefile to define it.

```
<environment bounds="shape_file_background"/>
```

Display

We have to define a diplay for the background and obstacles agents. We use for that the classic **species** markup.

```
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
      </display>
  </output>
```

Complete model

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />

      <init>
        <create species="background" from="shape_file_background"/>
        <create species="obstacle" from="shape_file_obstacles"/>
      </init>
  </global>
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
      </display>
```

```
  </output>
</model>
```

2. Computation of the unknown environment

<wiki:toc max_depth="3" />

Purpose

Illustrate how to apply basic operations on geometries.

Formulation

- Build a geometry representing the union of all obstacles
- Build the unknown environment: "the background - the obstacles"

Model

Geometry union

First, we have to build the union of all the obstacles. We define a global variable: the_obstacle that will represent this geometry. Note that a geometry in GAML is represented by a list of simple geometries. Each simple geometry is composed of a first list of points that represents the geometry external ring and eventually by other lists of points representing the holes in the geometry. For a simple geometry, if the list of points of the external ring is composed of only 1 point, the resulting geometry is a Point, if the first point is equal to the last point, the geometry is a Polygon, otherwise the geometry is a Polyline. To compute the union of a set of agent geometries, we have to use the **union** action of the **situated** skill and to use its **agents** argument.

```
<global>
    ...
    <var type="list" name="the_obstacles"/>
    ...
    <init>
```

```
    <create species="obstacle" from="shape_file_obstacles" return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
    ...
      </init>
  </global>
```

Unknown environment

In order to build the unknown environment (that will be reprsented by the **background** agent), we have to apply the **difference** action of the **situated skill** to compute the resulting geometries from the difference between the initial background geometry and the geometries of the obstacles. Note that we add a new global variable, **the_bg** , that represents the background to clarify the code.

```
  <global>
      ...
      <var type="background" name="the_bg" />
      ...
      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
        <set name="the_bg" value="first a_bg" />
        ...
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
        </ask>
      </init>
  </global>
```

Complete model

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
      <var type="list" name="the_obstacles"/>
      <var type="background" name="the_bg" />
      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
```

```
        <set name="the_bg" value="first a_bg" />
        <create species="obstacle" from="shape_file_obstacles"
return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
        </ask>
      </init>
  </global>
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
      </display>
  </output>
</model>
```

3. Integration of the robot agent

<wiki:toc max_depth="3" />

Purpose

Illustrate how to randomly place and move an agent inside a given polygon.

Formulation

- Define a new species of agents: the **robot** agent.
- The **robot** agent has a point for geometry.
- At initialisation, the **robot** agent is randomly place inside the background (minus the obstacles).
- At each simulation step (1 second), the **robot** agent moves randomly at a speed of 3m/s inside the background (minus the obstacles).
- In term of display, the **robot** agent is represented by a red circle of radius 2m.

Model

Parameter

We add a new parameter: the speed of the robot.

```
  <global>
      ...
      <var type="float" name="speed_robot" init="3" min="1" max="10"
parameter="Speed of the robot" category="Robot" />
      ...
  </global>
```

Robot agent

We have to define a new species of agents: the **robot** agents. At each simulation step, the **robot** agent is going to move randomly. Thus, we have to provide it with a **moving** skill. Indeed, we have to use the **wander** action of this **skill** for the agent movement. This action use the **speed** attribute (automaticaly assigned to agent with a **moving** skill) to define the speed of the agent. So, we have to initialise this variable with the expected value. Concerning the display of the **robot** agent, we want to represent the **robot** agents by a red circle of radius 2m. In GAMA, the default reprsentation for a point geometry is a point. Then, we can use the **geometry** value for the **shape** facet of the **draw** command, we just have to specified the expected color and radius size (defined by the facet **size**).

```
<entities>
    <species name="robot" skills="moving">
      <init>
          <set name="speed" value="speed_robot" />
      </init>

      <reflex name="move">
          <do action="wander">
        <arg name="agent" value="the_bg"/>
      </do>
      </reflex>

      <aspect name="base">
          <draw shape="geometry" size="2" color="rgb 'red'" />
      </aspect>
    </species>
</entities>
```

Creation and placement of the robot agent

We have to create one **robot** agent. This agent is randomly placed inside the background (minus the obstacles). In order to place the agent, we have to use the **place_in** action of the **situated** skill. This action returns a random point situated on/ inside the geometry passed in parameter (or the geometry of the agent passed in parameter). In order to clarify the code, we add a global variable, **the_robot** , that represent the **robot** agent.

```
<global>
    ...
    <var type="robot" name="the_robot" />
    ...
    <init>
```



```
        ...
        <create species="robot" number="1" return="a_robot">
        <set name="location" value="self place_in [agent::the_bg]" />
    </create>
    <set name="the_robot" value="a_robot"/>
      </init>
  </global>
```

Display

We have to add the people agent in the defined diplay.

```
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="robot" aspect="base"/>
      </display>
  </output>
```

Complete model

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
      <var type="float" name="speed_robot" init="3" min="1" max="10"
parameter="Speed of the robot" category="Robot" />
      <var type="list" name="the_obstacles"/>
      <var type="background" name="the_bg" />
      <var type="robot" name="the_robot" />

      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
        <set name="the_bg" value="first a_bg" />
        <create species="obstacle" from="shape_file_obstacles"
return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
```

```
        </ask>
        <create species="robot" number="1" return="a_robot">
        <set name="location" value="self place_in [agent::the_bg]" />
    </create>
    <set name="the_robot" value="a_robot"/>
      </init>
  </global>
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
      <species name="robot" skills="moving">
        <init>
            <set name="speed" value="speed_robot" />
        </init>

        <reflex name="move">
            <do action="wander">
          <arg name="agent" value="the_bg"/>
        </do>
        </reflex>

        <aspect name="base">
            <draw shape="geometry" size="2" color="rgb 'red'" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="robot" aspect="base"/>
      </display>
  </output>
</model>
```

4. Dynamic building of the known environment

<wiki:toc max_depth="3" />

Purpose

Illustrate how to randomly place and move an agent inside a given polygon.

Formulation

- Definition of a new parameter: the range of perception of the robot. By default, this parameter equals 8m.
- Computation and representation of the known area: definition of a new species, the **known** area agent that will be display in green with a transparency of 0.5.

Model

Parameters

We have to add a new parameter: the range of perception of the robot.

```
  <global>
      ...
      <var type="float" name="robot_perception_range" init="8" min="1"
max="1000" parameter="Perception distance of the robot" category="Robot" />
      ...
  </global>
```

Known_area agent

We have to define a new species of agents: the **known_area** agents. This agent will not have a particular behaviour. It will just be displayed. Thus, we just have to define

an aspect for the agents (see Aspects in the Modeling guide). In this model, we want to represent the geometry of the agent, we then use the value **geometry** of the facet **shape** for the command **draw** .

```
  <entities>
      <species name="known_area" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'green'" />
        </aspect>
      </species>
  </entities>
```

Perception of the robot agent

First, we add the attribute **the_known** to the **robot** agent. This attribute, which is of type **known_area** is the area that the **robot** agent knows. At initialisation, we set the value of the **range** attribute (automaticaly assigned to agent with a **situated** skill) to the expected value. Then, we initialise the **know_area** attribute: we create an agent of species **known_area** that has for geometry the percieved area. In order to compute this percieved area, we have to use the **percieved_area** action of the **situated** skill. This action, which takes as parameters a background geometry (here, the geometry of the **background** agent that already includes the obstacles) and a precision. The precision is used for the computation: higher this value is, more realist will be the perception, but higher will be the computational time. In addition to the initialisation of the known area, we have to add a **reflex** method to update the geometry of the known area. For that, we use the **percieved_area** action to compute at each step the geometry representing the new perception of the robot and the **union** action of the **situated** skill to add this geometry to the previous geometry of the known area.

```
<species name="robot" skills="moving">
        <var name="the_known" type="known_area"/>
        <init>
            ...
            <set name="range" value="robot_perception_range" />
            <create species="known_area" return="a_known_area">
              <set name="geometry" value="myself percieved_area
[agent::the_bg, precision::20]" />
            </create>
            <set name="the_known" value="a_known_area" />
        </init>
          ...
        <reflex name="update_the_known">
            <let type="list" name="percept" value="self percieved_area
[agent::the_bg, precision::20]" />
        <ask target="the_known">
          <set name="geometry" value="self union [geometry::percept]" />
```

```
            </ask>
        </reflex>
        ...
      </species>
  </entities>
```

Display

We have to add the **known_area** agent in the defined diplay. We have to use the facet **transparency** to add the transparency to the agent display.

```
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="known_area" transparency="0.5" aspect="base"/>
        <species name="robot" aspect="base"/>
      </display>
  </output>
```

<font color="blue">Complete model</ font>

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
      <var type="float" name="speed_robot" init="3" min="1" max="10"
parameter="Speed of the robot" category="Robot" />
      <var type="float" name="robot_perception_range" init="8" min="1"
max="1000" parameter="Perception distance of the robot" category="Robot" />

      <var type="list" name="the_obstacles"/>
      <var type="background" name="the_bg" />
      <var type="robot" name="the_robot" />

      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
        <set name="the_bg" value="first a_bg" />
        <create species="obstacle" from="shape_file_obstacles"
return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
```

```
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
        </ask>
        <create species="robot" number="1" return="a_robot">
        <set name="location" value="self place_in [agent::the_bg]" />
    </create>
    <set name="the_robot" value="a_robot"/>
      </init>
  </global>
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
      <species name="robot" skills="moving">
        <var name="the_known" type="known_area"/>
        <init>
            <set name="speed" value="speed_robot" />
            <set name="range" value="robot_perception_range" />
            <create species="known_area" return="a_known_area">
              <set name="geometry" value="myself percieved_area
[agent::the_bg, precision::20]" />
            </create>
            <set name="the_known" value="a_known_area" />
        </init>

        <reflex name="move">
            <do action="wander">
          <arg name="agent" value="the_bg"/>
        </do>
        </reflex>
        <reflex name="update_the_known">
            <let type="list" name="percept" value="self percieved_area
[agent::the_bg, precision::20]" />
        <ask target="the_known">
          <set name="geometry" value="self union [geometry::percept]" />
            </ask>
        </reflex>

        <aspect name="base">
            <draw shape="geometry" size="2" color="rgb 'red'" />
        </aspect>
      </species>
```

```
      <species name="known_area" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'green'" />
        </aspect>
      </species>
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="known_area" transparency="0.5" aspect="base"/>
        <species name="robot" aspect="base"/>
      </display>
  </output>
</model>
```

5. Use of a pathfinder for the robot movement

<wiki:toc max_depth="3" />

Purpose

Illustrate how to automatically compute the shortest path between two points inside a polygon.

Formulation

- Replace the random movement of the agent by a movement in direction of a random point of the background (minus the obstacles).

Model

Triangle agents

GAMA allows to directly extract a graph from one or several polygons with the action **compute_graph** of the **situated** . Thus, it would have been possible to directly compute a graph from the **background** agent. However, in order to ease the futher steps of the tutorial, we will not directly compute a graph from this agent, but first decomposed this gemetry into a set of **triangle** agents (Delaunay triangulation) with the **create** command, and then compute the graph from these **triangle** agents. The **triangle** agents are just situated agents that has for the moment no attribute.

```
<entities>
    ...
    <species name="triangle" skills="situated"/>
</entities>
```

Triangle agent creation and graph computation

In GAMA, decomposing a polygon in a set of triangle is very simple using the **create** command. Indeed, this command, with the facet **from** allows to automatically create a set of agents of the specified species of which the geometry is a triangle. Concerning the computation of a graph from set of agents (of which the geometry is a polygon), the action **compute_graph** of the **situated** skill allows to do that using the facet **geometry** or **agents** . However, in the context of a set of agents of which the geometries are triangles resulting from a Delaunay triangulation (like our **triangle** agent), we can directly use the **triangles** argument that allows to optimise the computation. Then, we just have to specified a name for this graph that will represent the id of the graph. We choose to call it 'background'. GAMA integrates several algorithms to compute the shortest pathes between the nodes of the graph. When no 'optimizer_type' is specified (like in this example), all the shortest pathes are automatically computed with the Floyd Warshall algorithm.

```
<global>
    ...
    <init>
        ...
      <create species="triangle" from="the_bg" type="'Triangles'"/>
      <do action="compute_graph">
      <arg name="name" value="'background'" />
      <arg name="triangles" value="list triangle" />
      </do>
      ...
    </init>
</global>
```

Movement of the robot agent

We have to modify the **move** action of the **robot** agent. If a target point is not defined (target_loc = nil), the agent chooses a target inside the background using the **place_in** action of the **situated** skill; otherwise, the agent moves in direction of this target using the **goto** action. Note that we specified a graph to constraint the moving of the agent inside the background geometry. When the agent arrives at destination (target_loc = location), the target is set as nil (the agent will choose another target).

```
<entities>
    ...
    <species name="robot" skills="moving">
      ...
      <var name="target_loc" type="point" init="nil" />
      ...
```

```
        <reflex name="move">
            <if condition="target_loc = nil">
              <set name="target_loc" value="self place_in [agent::the_bg]" />
            </if>
            <do action="goto">
              <arg name="graph_name" value="'background'" />
              <arg name="target" value="target_loc" />
            </do>
            <if condition="location = target_loc">
              <set name="target_loc" value="nil" />
            </if>
        </reflex>
        ...
      </species>
      ...
  </entities>
```

Complete model

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
      <var type="float" name="speed_robot" init="3" min="1" max="10"
parameter="Speed of the robot" category="Robot" />
      <var type="float" name="robot_perception_range" init="8" min="1"
max="1000" parameter="Perception distance of the robot" category="Robot" />

      <var type="list" name="the_obstacles"/>
      <var type="background" name="the_bg" />
      <var type="robot" name="the_robot" />

      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
        <set name="the_bg" value="first a_bg" />
        <create species="obstacle" from="shape_file_obstacles"
return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
        </ask>
        <create species="robot" number="1" return="a_robot">
```



```
        <set name="location" value="self place_in [agent::the_bg]" />
    </create>
    <set name="the_robot" value="a_robot"/>
        <create species="triangle" from="the_bg" type="'Triangles'"/>
        <do action="compute_graph">
        <arg name="name" value="'background'" />
        <arg name="triangles" value="list triangle" />
        </do>
      </init>
  </global>
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
      <species name="robot" skills="moving">
        <var name="the_known" type="known_area"/>
        <var name="target_loc" type="point" init="nil" />
        <init>
            <set name="speed" value="speed_robot" />
            <set name="range" value="robot_perception_range" />
            <create species="known_area" return="a_known_area">
              <set name="geometry" value="myself percieved_area
[agent::the_bg, precision::20]" />
            </create>
            <set name="the_known" value="a_known_area" />
        </init>

        <reflex name="move">
            <if condition="target_loc = nil">
              <set name="target_loc" value="self place_in [agent::the_bg]" />
            </if>
            <do action="goto">
              <arg name="graph_name" value="'background'" />
              <arg name="target" value="target_loc" />
            </do>
            <if condition="location = target_loc">
              <set name="target_loc" value="nil" />
            </if>
        </reflex>
        <reflex name="update_the_known">
            <let type="list" name="percept" value="self percieved_area
[agent::the_bg, precision::20]" />
        <ask target="the_known">
```

```
          <set name="geometry" value="self union [geometry::percept]" />
            </ask>
        </reflex>

        <aspect name="base">
            <draw shape="geometry" size="2" color="rgb 'red'" />
        </aspect>
      </species>
      <species name="known_area" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'green'" />
        </aspect>
      </species>
      <species name="triangle" skills="situated"/>
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="known_area" transparency="0.5" aspect="base"/>
        <species name="robot" aspect="base"/>
      </display>
  </output>
</model>
```

6. Frontier-based approach

<wiki:toc max_depth="3" />

Purpose

Illustrate how to automatically compute the shortest path between two points inside a polygon.

Formulation

we propose to use a frontier-based approach to improve the robot effectiveness in terms of area explored: the robot is going to compute a set of candidate positions located at the frontier of the known environment, then to chose among them the one that maximises an evaluation function. In this tutorial, the evaluation function used is a simple computation of the distance between the current location of the robot and the candidate location: the closer, the better.

- Computation and representation of the candidate positions. The distance between two considered candidates of the frontier is equal to 2m.

Model

Parameters

We have to add a new parameter: the distance between two considered candidates of the frontier. By default, this parameter is equal to 2m.

```
  <global>
      ...
      <var type="float" name="dist_pt_front" init="2" min="1" max="10"
parameter="Distance between two considered candidates of the frontier"
category="Robot"/>
      ...
  </global>
```

Candidate agent

We have to define a new species of agents: the **candidate** agents. These agents will not have a particular behaviour. they will just be displayed. Thus, we just have to define an aspect for the agents (see Aspects in the Modeling guide). In this model, we want to represent the geometry of the agent, we then use the value **geometry** of the facet **shape** for the command **draw** .

```
  <entities>
      <species name="candidate" skills="situated">
        <aspect name="base">
        <draw shape="geometry" color="rgb 'yellow'" />
    </aspect>
      </species>
  </entities>
```

Computation of the candidate agent

We have to modify the **move** reflex of the **robot** agent in order to select the next target among the computed candidates. First, in order to clarify the code, we propose to define an action, **choose_target** , that is called by the **move** reflex when a new target has to be selected. Concerning the **choose_target** action, first, we kill the existing candidates if there are ones. Then we use the **points_exterior_ring** action of the **situated** skill to computed, on the frontier of a given geometry (here, the geometry of the **known_area** agent), a set of points distant of a given distance. We will use these points to create the **candidate** agents: the location of each created **candidate** agent will be equal to a point of the list. We will give to these agents a square geometry of side size 0.5m in order to test if there are interesting: indeed, we are going to filter the list of **candidate** agents in order to keep only the ones that are not touching the obstacles (condition testes by using the **overlaps** action of the **situated** skill) and that are totally inside the background (condition testes by using the **partially_overlaps** action of the **situated** skill). Once this filter list of **candidate** agents built, we have to choose the **candidate** agent that is the closer to the robot location. To compute this distance (distance for the robot, not the distance as the crow flies), we use the **distance_graph** action of the **situated** skill that allows, using a graph, to compute the distance between two points. To find the **candidate** agent that minimizes this distance, we use the **with_min_of** operator.

```
<entities>
      <species name="robot" skills="moving">
        ...
        <reflex name="move">
            <if condition="target_loc = nil">
```

```
              <do action="choose_target" />
            </if>
            <do action="goto">
              <arg name="graph_name" value="'background'" />
              <arg name="target" value="target_loc" />
            </do>
            <if condition="location = target_loc">
              <set name="target_loc" value="nil" />
            </if>
        </reflex>
        ...
        <action name="choose_target">
        <ask target="list candidate">
          <do action="die" />
        </ask>
        <let type="list" of="candidate" name="good_candidates" value="[]"/>
        <let type="list" name="all_candidates" value="self
points_exterior_ring [distance::dist_pt_front, agent::the_known]" />
        <if condition="(all_candidates != nil) and !(empty all_candidates)">
          <loop over="all_candidates" var="cand">
              <create species="candidate" return="cs">
                <set name="location" value="cand" />
                <do action="build_basic_geometry">
                    <arg name="geometry_type" value="'square'" />
                    <arg name="side_size" value="0.5" />
                </do>
                <if condition="(self partially_overlaps [agent::the_bg]) or
(self overlaps [geometry::the_obstacles])">
                    <do action="die" />
                    <else>
                      <add item="self" to="good_candidates"/>
                    </else>
                </if>
              </create>
          </loop>
        </if>
        <set name="target_loc" value="(candidate (good_candidates with_min_of
(self distance_graph [graph_name::'background', target::each]))).location" />
        </action>
    ...
      </species>
      ...
  </entities>
```

Display

We have to add the **candidate** agent in the defined diplay.

```
  <output>
      <display name="display" refresh_every="1">
```

```
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="known_area" transparency="0.5" aspect="base"/>
        <species name="candidate" aspect="base"/>
        <species name="robot" aspect="base"/>
      </display>
  </output>
```

Complete model

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
      <var type="float" name="speed_robot" init="3" min="1" max="10"
parameter="Speed of the robot" category="Robot" />
      <var type="float" name="robot_perception_range" init="8" min="1"
max="1000" parameter="Perception distance of the robot" category="Robot" />
      <var type="float" name="dist_pt_front" init="2" min="1" max="10"
parameter="Distance between two considered candidates of the frontier"
category="Robot"/>
      <var type="list" name="the_obstacles"/>
      <var type="background" name="the_bg" />
      <var type="robot" name="the_robot" />

      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
        <set name="the_bg" value="first a_bg" />
        <create species="obstacle" from="shape_file_obstacles"
return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
        </ask>
        <create species="robot" number="1" return="a_robot">
        <set name="location" value="self place_in [agent::the_bg]" />
    </create>
    <set name="the_robot" value="a_robot"/>
        <create species="triangle" from="the_bg" type="'Triangles'"/>
        <do action="compute_graph">
        <arg name="name" value="'background'" />
        <arg name="triangles" value="list triangle" />
```

```
        </do>
      </init>
  </global>
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
      <species name="robot" skills="moving">
        <var name="the_known" type="known_area"/>
        <var name="target_loc" type="point" init="nil" />
        <init>
            <set name="speed" value="speed_robot" />
            <set name="range" value="robot_perception_range" />
            <create species="known_area" return="a_known_area">
              <set name="geometry" value="myself percieved_area
[agent::the_bg, precision::20]" />
            </create>
            <set name="the_known" value="a_known_area" />
        </init>

        <reflex name="move">
            <if condition="target_loc = nil">
              <do action="choose_target" />
            </if>
            <do action="goto">
              <arg name="graph_name" value="'background'" />
              <arg name="target" value="target_loc" />
            </do>
            <if condition="location = target_loc">
              <set name="target_loc" value="nil" />
            </if>
        </reflex>
        <reflex name="update_the_known">
            <let type="list" name="percept" value="self percieved_area
[agent::the_bg, precision::20]" />
        <ask target="the_known">
          <set name="geometry" value="self union [geometry::percept]" />
            </ask>
        </reflex>
        <action name="choose_target">
        <ask target="list candidate">
          <do action="die" />
        </ask>
```

```
        <let type="list" of="candidate" name="good_candidates" value="[]"/>
        <let type="list" name="all_candidates" value="self
points_exterior_ring [distance::dist_pt_front, agent::the_known]" />
        <if condition="(all_candidates != nil) and !(empty all_candidates)">
          <loop over="all_candidates" var="cand">
              <create species="candidate" return="cs">
                <set name="location" value="cand" />
                <do action="build_basic_geometry">
                    <arg name="geometry_type" value="'square'" />
                    <arg name="side_size" value="0.5" />
                </do>
                <if condition="(self partially_overlaps [agent::the_bg]) or
(self overlaps [geometry::the_obstacles])">
                    <do action="die" />
                    <else>
                      <add item="self" to="good_candidates"/>
                    </else>
                </if>
              </create>
          </loop>
        </if>
        <set name="target_loc" value="(candidate (good_candidates with_min_of
(self distance_graph [graph_name::'background', target::each]))).location" />
        </action>

        <aspect name="base">
            <draw shape="geometry" size="2" color="rgb 'red'" />
        </aspect>
      </species>
      <species name="known_area" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'green'" />
        </aspect>
      </species>
      <species name="triangle" skills="situated"/>
      <species name="candidate" skills="situated">
        <aspect name="base">
        <draw shape="geometry" color="rgb 'yellow'" />
    </aspect>
      </species>
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="known_area" transparency="0.5" aspect="base"/>
    <species name="candidate" aspect="base" />
        <species name="robot" aspect="base"/>
      </display>
  </output>
</model>
```

7. Dynamic update of the pathfinder

<wiki:toc max_depth="3" />

Purpose

Illustrates how to add weights and to update them in a graph built from polygons.

Formulation

- For the robot movement, favour the pathes that only pass through the known area:
  - Add a **known** attribute to the **triangle** agents: if the triangle overlaps the known area, "known = 1", otherwise, "known = 100000".
  - Use this attribute to weight the graph computed from the **triangle** agents.

Model

Triangle agents

We have to add a new attribute for the **triangle** agents.

```
<entities>
    <species name="triangle" skills="situated">
      <var name="known" type="float" init="100000" />
    </species>
</entities>
```

Weighting of the graph

We have to update the value of the **known** attribute of the **triangle** agents and to integrate this attribute in the computation of the shortest pathes in the graph. We

propose to achieve that in the **choose_target** action of the **robot** agent. For the **known** attribute update, we have to use the **overlaps** action of the **situated** skill. Concerning the update of the graph, we have to use the **update_graph** of the **situated** skill.
This action allows to update an existing graph and to compute, if necessary, the new shortest pathes. It allows as well to specify with the **weight** argument the name of the variable that will be used as a weighting coefficient.

```
  <entities>
      ...
      <species name="robot" skills="moving">
          <action name="choose_target">
            <ask target="list triangle">
              <if condition="(self.known != 1) and (self overlaps
[agent::myself.the_known])">
                  <set name="known" value="1" />
              </if>
            </ask>
            <do action="update_graph">
              <arg name="graph_name" value="'background'" />
              <arg name="agents" value="list triangle" />
              <arg name="weights" value="'known'" />
            </do>
            ...
        </action>
      ...
      </species>
  </entities>
  </entities>
```

Complete model

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
      <var type="float" name="speed_robot" init="3" min="1" max="10"
parameter="Speed of the robot" category="Robot" />
      <var type="float" name="robot_perception_range" init="8" min="1"
max="1000" parameter="Perception distance of the robot" category="Robot" />
      <var type="float" name="dist_pt_front" init="2" min="1" max="10"
parameter="Distance between two considered candidates of the frontiers"
category="Robot"/>
      <var type="list" name="the_obstacles"/>
```

```
      <var type="background" name="the_bg" />
      <var type="robot" name="the_robot" />

      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
        <set name="the_bg" value="first a_bg" />
        <create species="obstacle" from="shape_file_obstacles"
return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
        </ask>
        <create species="robot" number="1" return="a_robot">
        <set name="location" value="self place_in [agent::the_bg]" />
    </create>
    <set name="the_robot" value="a_robot"/>
        <create species="triangle" from="the_bg" type="'Triangles'"/>
        <do action="compute_graph">
        <arg name="name" value="'background'" />
        <arg name="triangles" value="list triangle" />
        </do>
      </init>
  </global>
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
      <species name="robot" skills="moving">
        <var name="the_known" type="known_area"/>
        <var name="target_loc" type="point" init="nil" />
        <init>
            <set name="speed" value="speed_robot" />
            <set name="range" value="robot_perception_range" />
            <create species="known_area" return="a_known_area">
              <set name="geometry" value="myself percieved_area
[agent::the_bg, precision::20]" />
            </create>
            <set name="the_known" value="a_known_area" />
        </init>

        <reflex name="move">
```

```
            <if condition="target_loc = nil">
              <do action="choose_target" />
            </if>
            <do action="goto">
              <arg name="graph_name" value="'background'" />
              <arg name="target" value="target_loc" />
            </do>
            <if condition="location = target_loc">
              <set name="target_loc" value="nil" />
            </if>
        </reflex>
        <reflex name="update_the_known">
            <let type="list" name="percept" value="self percieved_area
[agent::the_bg, precision::20]" />
        <ask target="the_known">
          <set name="geometry" value="self union [geometry::percept]" />
            </ask>
        </reflex>
        <action name="choose_target">
            <ask target="list triangle">
              <if condition="(self.known != 1) and (self overlaps
[agent::myself.the_known])">
                    <set name="known" value="1" />
              </if>
            </ask>
            <do action="update_graph">
              <arg name="graph_name" value="'background'" />
              <arg name="agents" value="list triangle" />
              <arg name="weights" value="'known'" />
            </do>

        <ask target="list candidate">
          <do action="die" />
        </ask>
        <let type="list" of="candidate" name="good_candidates" value="[]"/>
        <let type="list" name="all_candidates" value="self
points_exterior_ring [distance::dist_pt_front, agent::the_known]" />
        <if condition="(all_candidates != nil) and !(empty all_candidates)">
          <loop over="all_candidates" var="cand">
              <create species="candidate" return="cs">
                <set name="location" value="cand" />
                <do action="build_basic_geometry">
                    <arg name="geometry_type" value="'square'" />
                    <arg name="side_size" value="0.5" />
                </do>
                <if condition="(self partially_overlaps [agent::the_bg]) or
(self overlaps [geometry::the_obstacles])">
                    <do action="die" />
                    <else>
                      <add item="self" to="good_candidates"/>
                    </else>
```

```
                </if>
              </create>
          </loop>
        </if>
        <set name="target_loc" value="(candidate (good_candidates with_min_of
(self distance_graph [graph_name::'background', target::each]))).location" />
        </action>

        <aspect name="base">
            <draw shape="geometry" size="2" color="rgb 'red'" />
        </aspect>
      </species>
      <species name="known_area" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'green'" />
        </aspect>
      </species>
      <species name="triangle" skills="situated">
        <var name="known" type="float" init="100000" />
      </species>
      <species name="candidate" skills="situated">
        <aspect name="base">
        <draw shape="geometry" color="rgb 'yellow'" />
    </aspect>
      </species>
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base"/>
        <species name="obstacle" aspect="base"/>
        <species name="known_area" transparency="0.5" aspect="base"/>
    <species name="candidate" aspect="base" />
        <species name="robot" aspect="base"/>
      </display>
  </output>
</model>
```

8. Integration of a chart in the display

<wiki:toc max_depth="3" />

Purpose

Illustrates how to define a stopping criterion and a complex display displaying at the same time the environment/agents and a chart.

Formulation

- Add a new global variable, **coverage** : the percentage of area covered.
- Add a new stopping criterion: when the **coverage** is higher than 95%, stop the simulation.
- Build a complex display allowing to display, in same panel, the environment/agents and a chart of the coverage evolution.

Model

Global variable

We add a new parameter: the range of perception of the robot.

```
  <global>
      ...
      <var type="float" name="coverage" init="0"
value="(the_robot.the_known.area / the_bg.area) * 100.0"/>
      ...
  </global>
```

Stopping criterion

We have a new **reflex** method in the global section in order to add a stopping criterion. To stop the simulation, we have to use the **halt** action.

```
  <global>
      ...
      <reflex when="coverage &gt; 95">
        <do action="halt"/>
      </reflex>
  </global>
```

Complex display

We have to define a complex display. In order to integrate, in the same display, a chart and the environment/agents, we have to use the **position** and **size** facets of the display markup (**species** and **chart**). Concerning the chart, we have to define a chart of type **series** to display the coverage evolution.

```
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />
        <species name="obstacle" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />
        <species name="known_area" transparency="0.5" aspect="base"
position="{0.2,0.05}" size="{0.6,0.6}" />
        <species name="candidate" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />
        <species name="robot" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />

        <chart type="series" name="Percentage of area covered" background="rgb
'lightGray'" axes="rgb 'white'" position="{0.05,0.67}" size="{0.9,0.3}">
        <data name="coverage" color="rgb 'blue'" value="coverage"
style="line" />
        </chart>
      </display>
  </output>
```

Complete model

```
<model name="tutorial_gis_robot_exploration">
  <global>
      <var type="string" name="shape_file_background" init="'../gis/
Background.shp'" parameter="Shapefile for the buildings:" category="GIS" />
      <var type="string" name="shape_file_obstacles" init="'../gis/
Obstacles.shp'" parameter="Shapefile for the roads:" category="GIS" />
      <var type="float" name="speed_robot" init="3" min="1" max="10"
parameter="Speed of the robot" category="Robot" />
      <var type="float" name="robot_perception_range" init="8" min="1"
max="1000" parameter="Perception distance of the robot" category="Robot" />
      <var type="float" name="dist_pt_front" init="2" min="1" max="10"
parameter="Distance between two considered candidates of the frontiers"
category="Robot"/>
      <var type="list" name="the_obstacles"/>
      <var type="background" name="the_bg" />
      <var type="robot" name="the_robot" />
      <var type="float" name="coverage" init="0"
value="(the_robot.the_known.area / the_bg.area) * 100.0"/>

      <init>
        <create species="background" from="shape_file_background"
return="a_bg" />
        <set name="the_bg" value="first a_bg" />
        <create species="obstacle" from="shape_file_obstacles"
return="the_obs"/>
    <set name="the_obstacles" value="self union [agents:: the_obs]"/>
        <ask target="the_bg">
            <set name="geometry" value="self difference
[geometry2::the_obstacles]"/>
        </ask>
        <create species="robot" number="1" return="a_robot">
        <set name="location" value="self place_in [agent::the_bg]" />
    </create>
    <set name="the_robot" value="a_robot"/>
        <create species="triangle" from="the_bg" type="'Triangles'"/>
        <do action="compute_graph">
        <arg name="name" value="'background'" />
        <arg name="triangles" value="list triangle" />
        </do>
      </init>

      <reflex when="coverage &gt; 95">
        <do action="halt" />
      </reflex>
  </global>
```

```
  <environment bounds="shape_file_background" />
  <entities>
    <species name="background" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'pink'" />
        </aspect>
      </species>
    <species name="obstacle" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'black'" />
        </aspect>
      </species>
      <species name="robot" skills="moving">
        <var name="the_known" type="known_area"/>
        <var name="target_loc" type="point" init="nil" />
        <init>
            <set name="speed" value="speed_robot" />
            <set name="range" value="robot_perception_range" />
            <create species="known_area" return="a_known_area">
              <set name="geometry" value="myself percieved_area
[agent::the_bg, precision::20]" />
            </create>
            <set name="the_known" value="a_known_area" />
        </init>

        <reflex name="move">
            <if condition="target_loc = nil">
              <do action="choose_target" />
            </if>
            <do action="goto">
              <arg name="graph_name" value="'background'" />
              <arg name="target" value="target_loc" />
            </do>
            <if condition="location = target_loc">
              <set name="target_loc" value="nil" />
            </if>
        </reflex>
        <reflex name="update_the_known">
            <let type="list" name="percept" value="self percieved_area
[agent::the_bg, precision::20]" />
        <ask target="the_known">
          <set name="geometry" value="self union [geometry::percept]" />
            </ask>
        </reflex>
        <action name="choose_target">
            <ask target="list triangle">
              <if condition="(self.known != 1) and (self overlaps
[agent::myself.the_known])">
                  <set name="known" value="1" />
              </if>
            </ask>
```

```
            <do action="update_graph">
              <arg name="graph_name" value="'background'" />
              <arg name="agents" value="list triangle" />
              <arg name="weights" value="'known'" />
            </do>

        <ask target="list candidate">
          <do action="die" />
        </ask>
        <let type="list" of="candidate" name="good_candidates" value="[]"/>
        <let type="list" name="all_candidates" value="self
points_exterior_ring [distance::dist_pt_front, agent::the_known]" />
        <if condition="(all_candidates != nil) and !(empty all_candidates)">
          <loop over="all_candidates" var="cand">
              <create species="candidate" return="cs">
                <set name="location" value="cand" />
                <do action="build_basic_geometry">
                    <arg name="geometry_type" value="'square'" />
                    <arg name="side_size" value="0.5" />
                </do>
                <if condition="(self partially_overlaps [agent::the_bg]) or
(self overlaps [geometry::the_obstacles])">
                    <do action="die" />
                    <else>
                      <add item="self" to="good_candidates"/>
                    </else>
                </if>
              </create>
          </loop>
        </if>
        <set name="target_loc" value="(candidate (good_candidates with_min_of
(self distance_graph [graph_name::'background', target::each]))).location" />
        </action>

        <aspect name="base">
            <draw shape="geometry" size="2" color="rgb 'red'" />
        </aspect>
      </species>
      <species name="known_area" skills="situated">
        <aspect name="base">
            <draw shape="geometry" color="rgb 'green'" />
        </aspect>
      </species>
      <species name="triangle" skills="situated">
        <var name="known" type="float" init="100000" />
      </species>
      <species name="candidate" skills="situated">
        <aspect name="base">
        <draw shape="geometry" color="rgb 'yellow'" />
    </aspect>
      </species>
```

```
  </entities>
  <output>
      <display name="display" refresh_every="1">
        <species name="background" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />
        <species name="obstacle" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />
        <species name="known_area" transparency="0.5" aspect="base"
position="{0.2,0.05}" size="{0.6,0.6}" />
        <species name="candidate" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />
        <species name="robot" aspect="base" position="{0.2,0.05}"
size="{0.6,0.6}" />

        <chart type="series" name="Percentage of area covered" background="rgb
'lightGray'" axes="rgb 'white'" position="{0.05,0.67}" size="{0.9,0.3}">
        <data name="coverage" color="rgb 'blue'" value="coverage"
style="line" />
        </chart>
      </display>
  </output>
</model>
```

Tutorial 4:
Batch tutorial

Batch Tutorial

<wiki:toc max_depth="3" />

Introduction

This tutorial has for goal to present the batch mode of GAMA 1.3. The Batch mode allows to explore the model parameter space. This exploration is indispensable to:

- **Build:** find missing elements
- **Calibrate:** estimate parameters values
- **Validate:** reproduce observed values, sensitivity analysis
- **Reason:** use model to explain, discover emergent properties or ...predict.

Principle of the parameter space exploration

Explore the parameter space of a model requires to follow 5 steps:

- Define initial condition and an stopping condition for the model
- Define parameters to explore
- Define a strategy for exploration and recording results
- Execute and collect Data
- Analyze Data

Parameter space exploration in GAMA