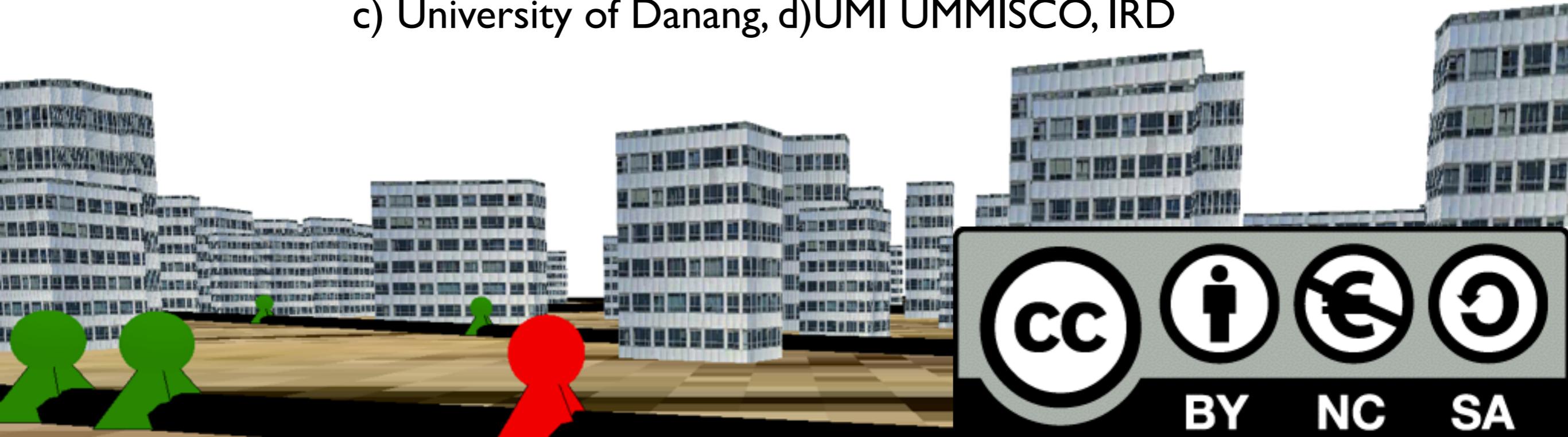




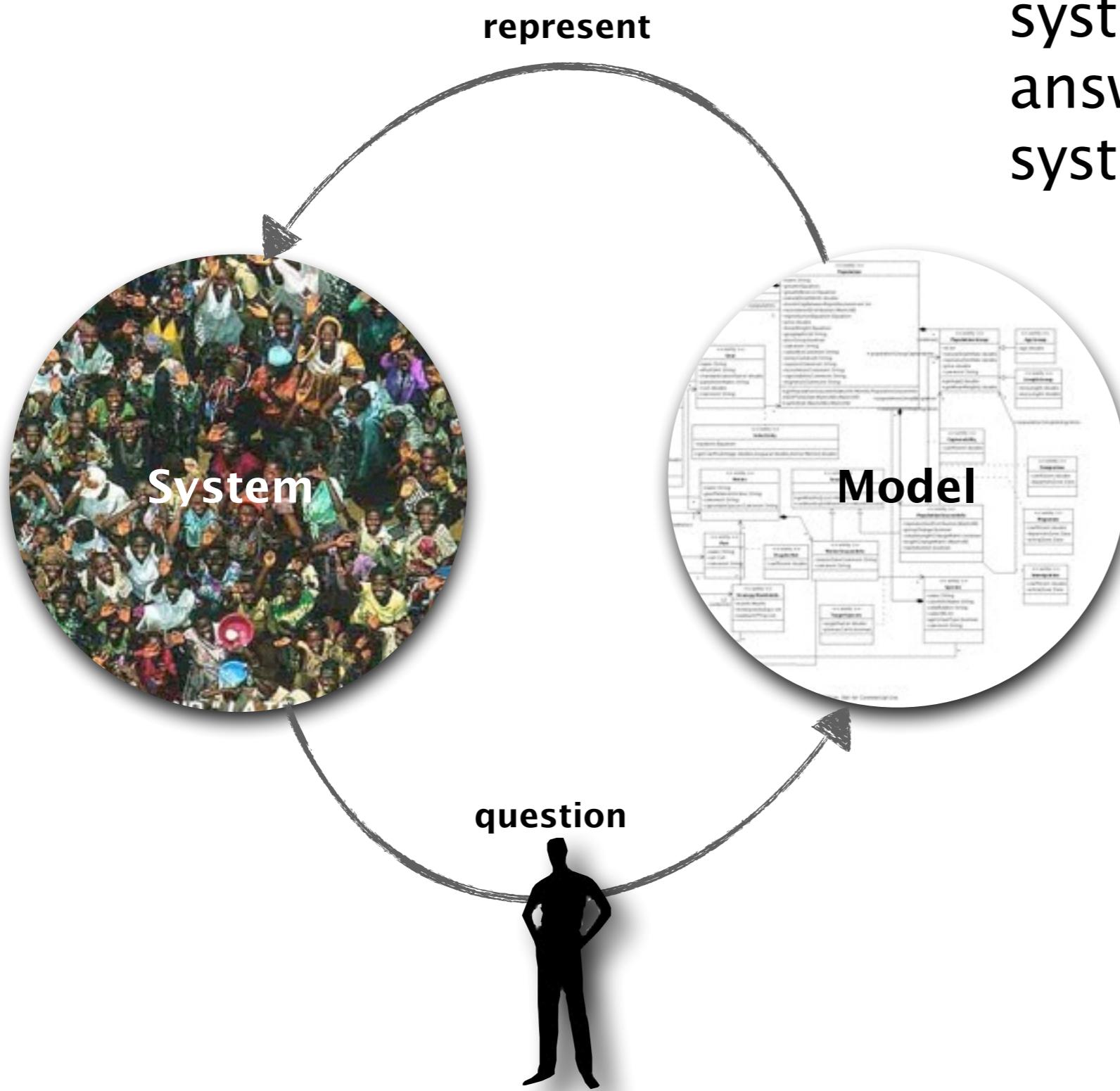
Patrick Taillandier (a), Benoit Gaudou (b), An Duc Vo (c),  
Arnaud Grignard (a), Alexis Drogoul (d)

## A Practical Introduction to the GAMA platform

a) UMR IDEES, University of Rouen, b) UMR IRIT, University of Toulouse I,  
c) University of Danang, d) UMI UMMISCO, IRD



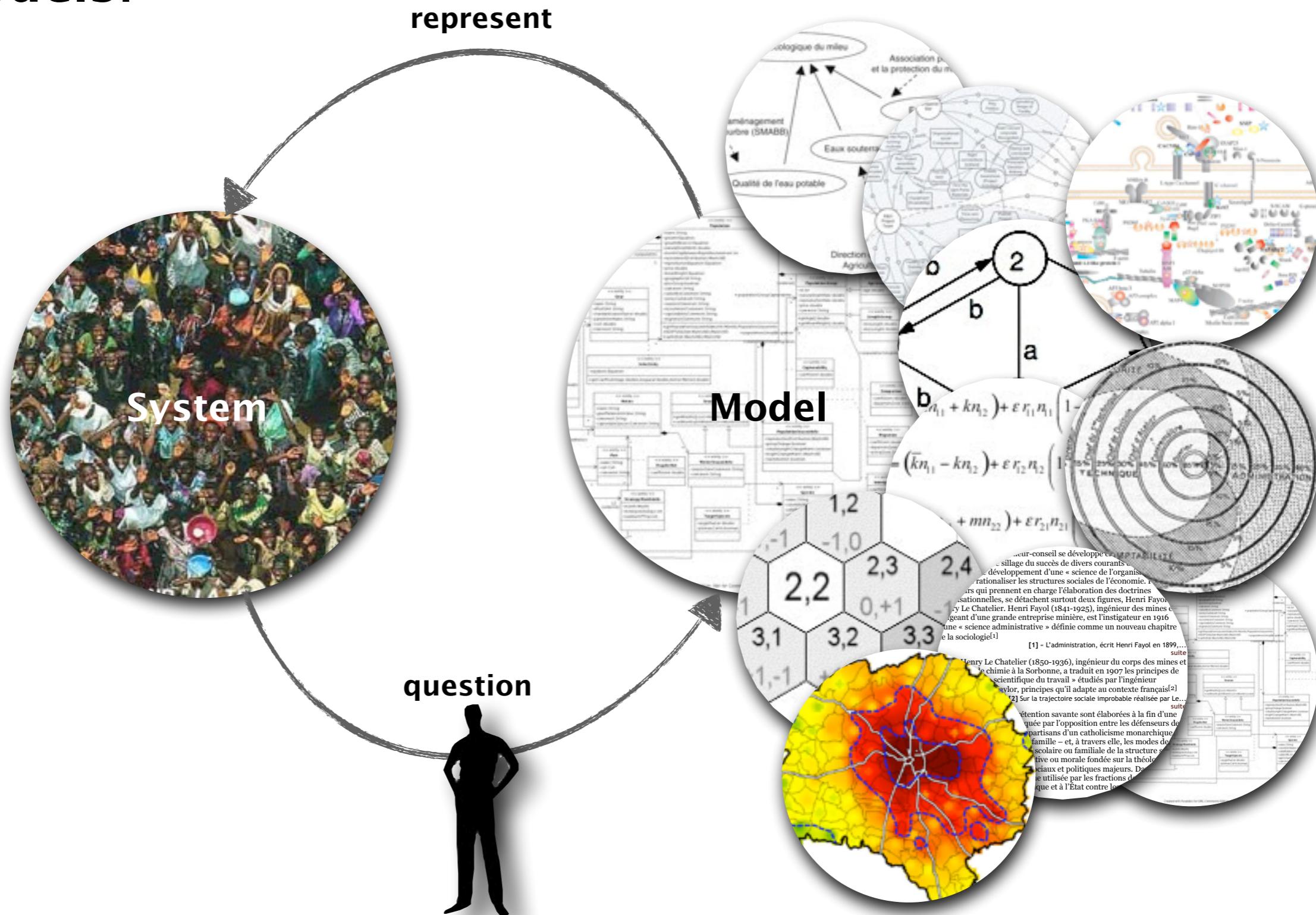
# Model



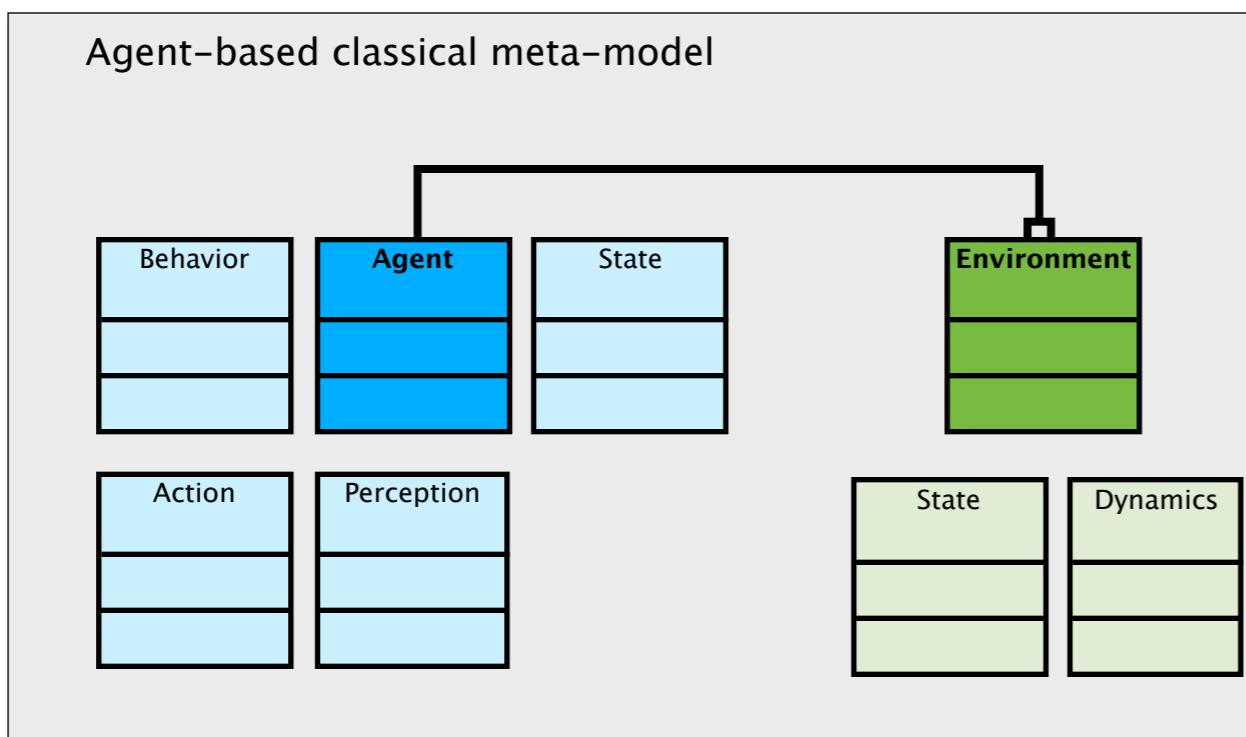
A model is a simplified representation of a reference system, designed to help answering a **question** on this system

# Representation(s)

Representation can use multiple supports and languages, depending on the question, traditions... They are specified by meta-models.



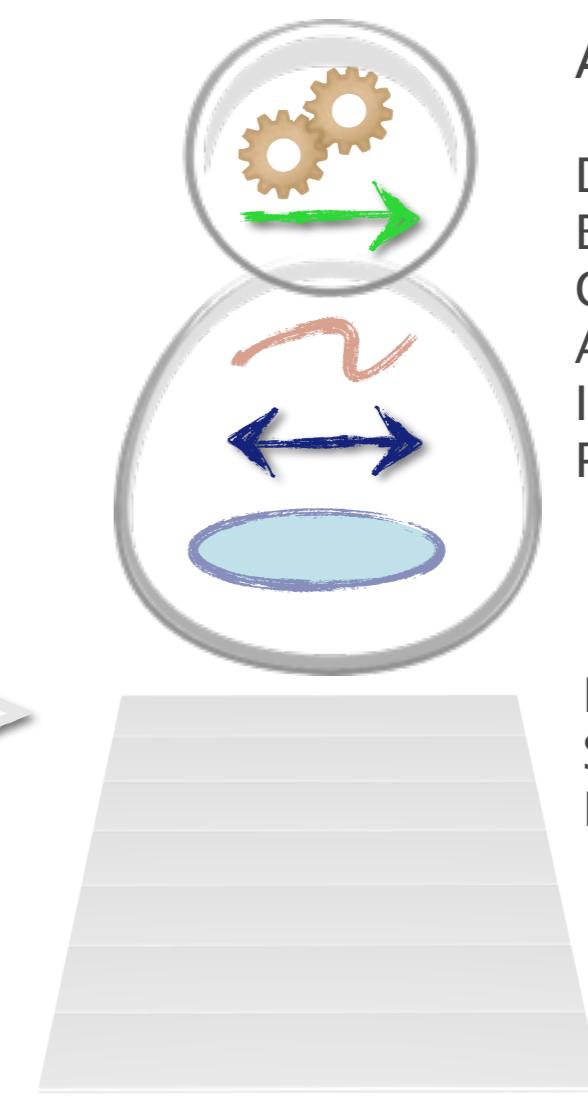
# Structure of agent-based models



```

environment width: width_and_height_of_environment height: width_
ities {
  species cells skills: [moving] {
    const speed type: float <- speed_of_agents ;
    rgb color <- [100 + rnd(155),100 + rnd(155), 100 + rnd(15
    float size min: 1 max: 10 <- 4;
    int strength <- 0 ;
    float range min: range_of_agents max: width_and_height_of_environm
    cells leader <- self ;
    int heading <- rnd(359) update: leader.heading;
    reflex move {
      do move ;
    }

    reflex change_leader when: (leader != self) and (self dis
      if grow_leader {
        set range of mv leader <- (range of mv leader) +
      }
    }
  }
}
  
```



- Agents**
- Decision-making
  - Behavior
  - Communication
  - Actions
  - Interactions
  - Perception

- Environment**
- State(s)
  - Dynamics

# Agent-based models

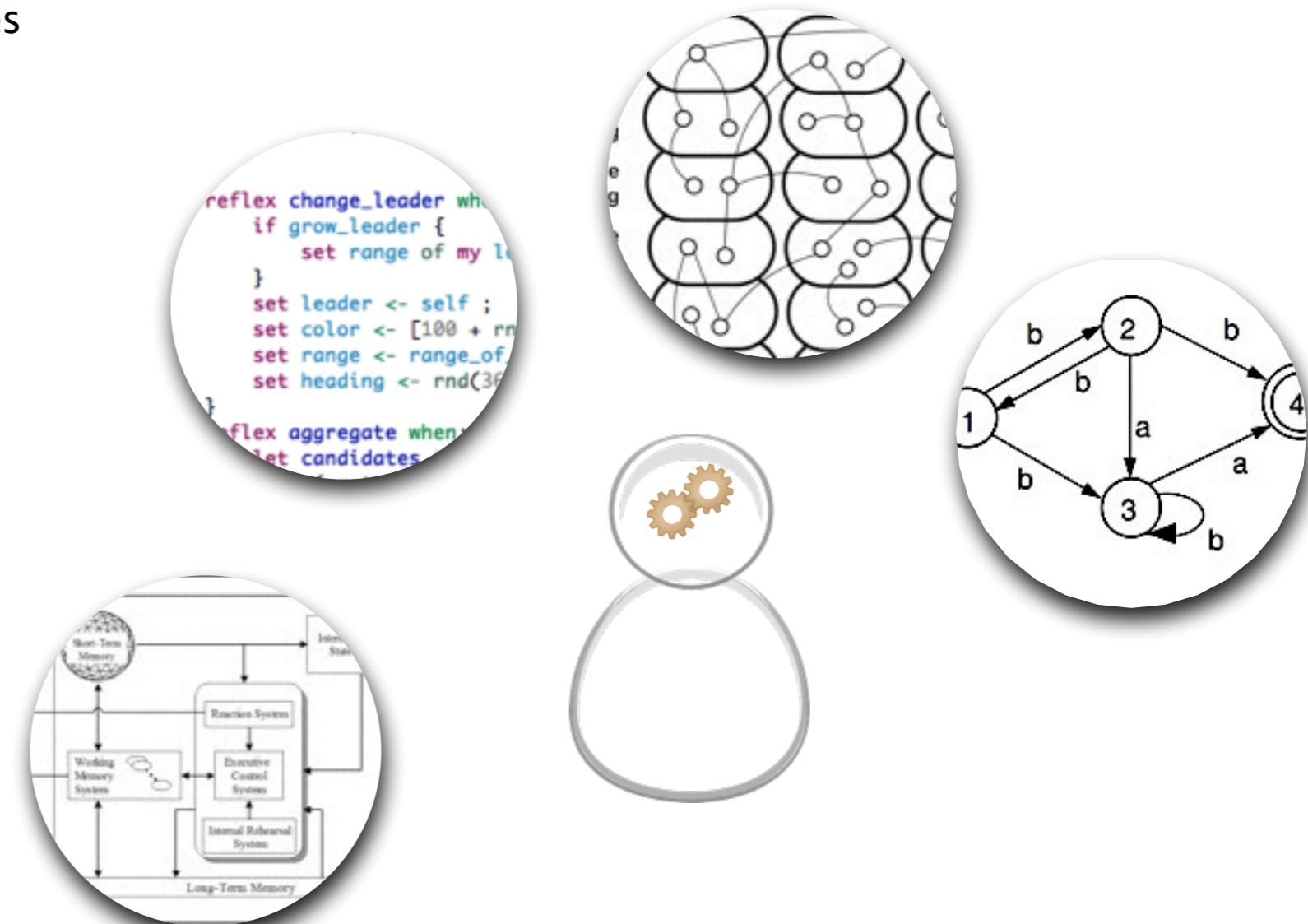
Agent-based models are **versatile** and **heterogeneous**: agents can represent any object or aggregation of objects of the reference system.



# Agent behaviors

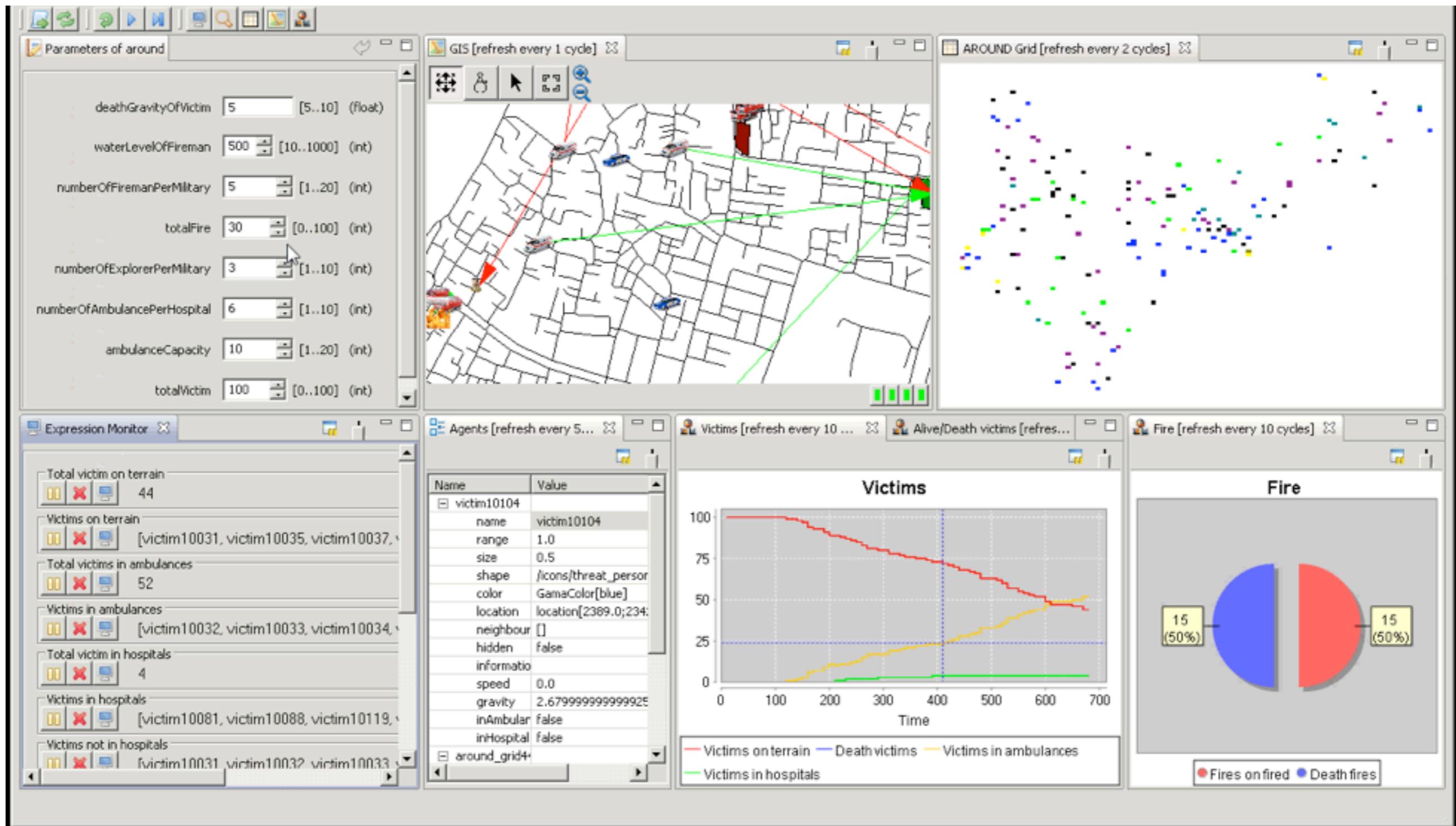
Agent-based models are **agnostic** : agents can be programmed using any language or decision-making architecture

- Any computer program
- Expert systems
- Finite state automata
- Task-based architectures
- Perception–decision–action architectures
- Planning architectures
- Neural networks
- Bayesian networks
- Differential equations



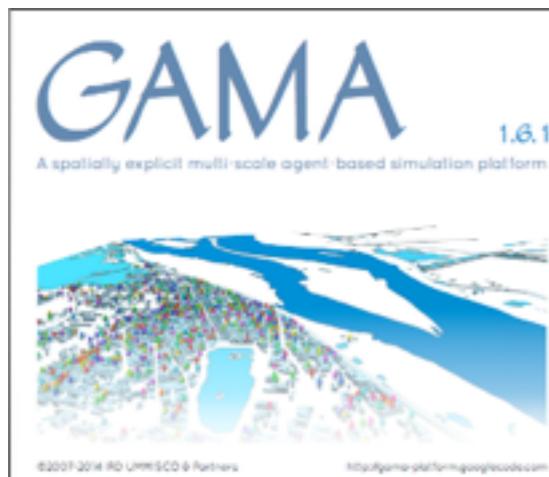
# Miniature laboratories

Agent-based simulations are **miniature laboratories** that support **computational experiments**, which can be interactive



# Generalities on the GAMA platform

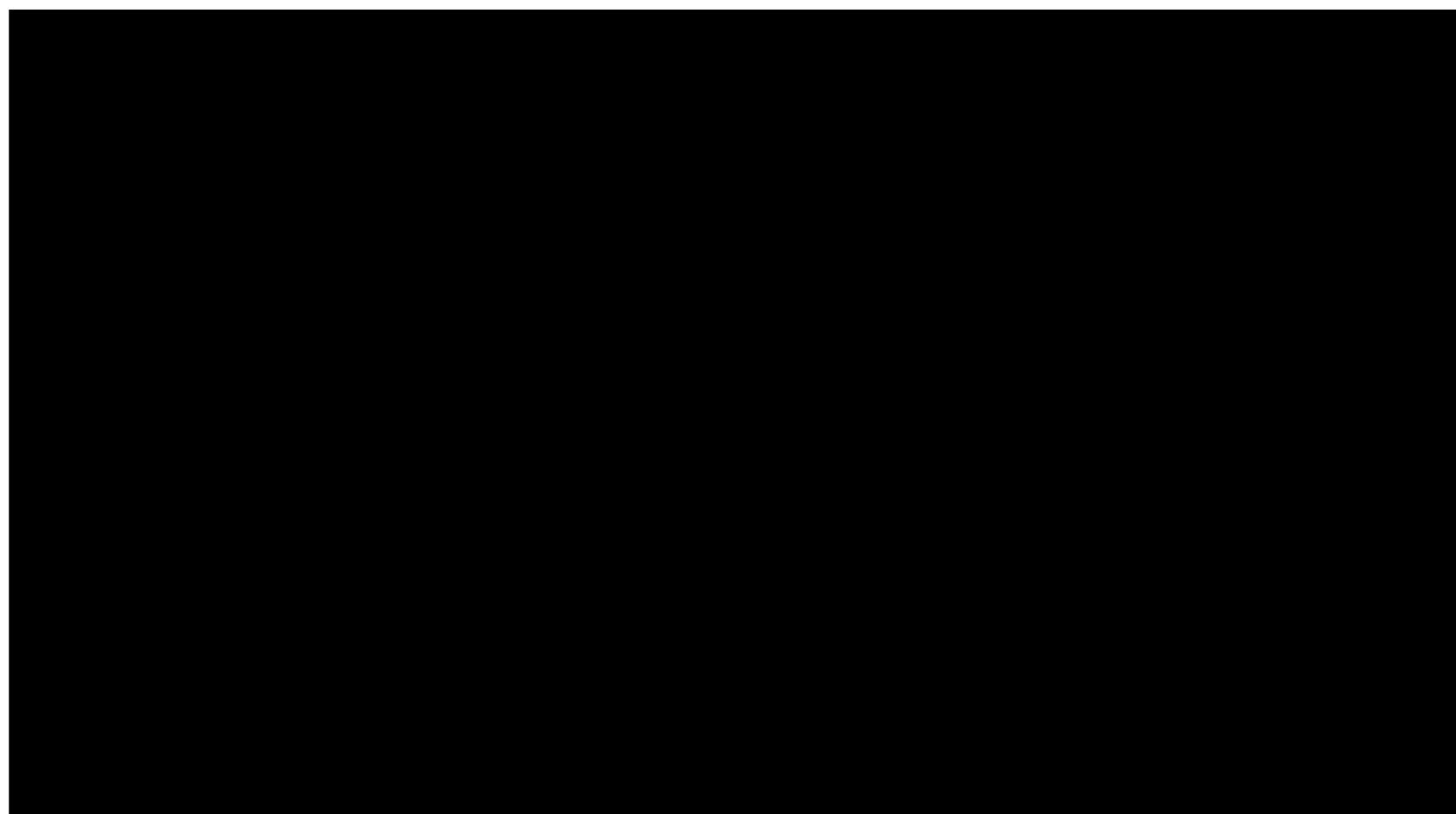
- ❖ Software platform dedicated to building spatially explicit agent-based simulations
  - Developed by a consortium of research team : UMMISCO (IRD - coordinator), DREAMS (Can Tho), IFI (Hanoi), IDEES (Rouen), IRIT (Toulouse), LRI (Orsay), ...
  - Generic : can be used for a wide range of applications
  - Developed under GPL license : **open-source**
  - Integrates a complete modeling language (GAML) and an integrated development environment: **allows modelers (even non computer-scientists) to build models quickly and easily**
  - Developed in JAVA : **easy to extend in order to take specific needs into account**
  - Integrates tools to analyze models: parameters space exploration and is now compatible with **OpenMole**



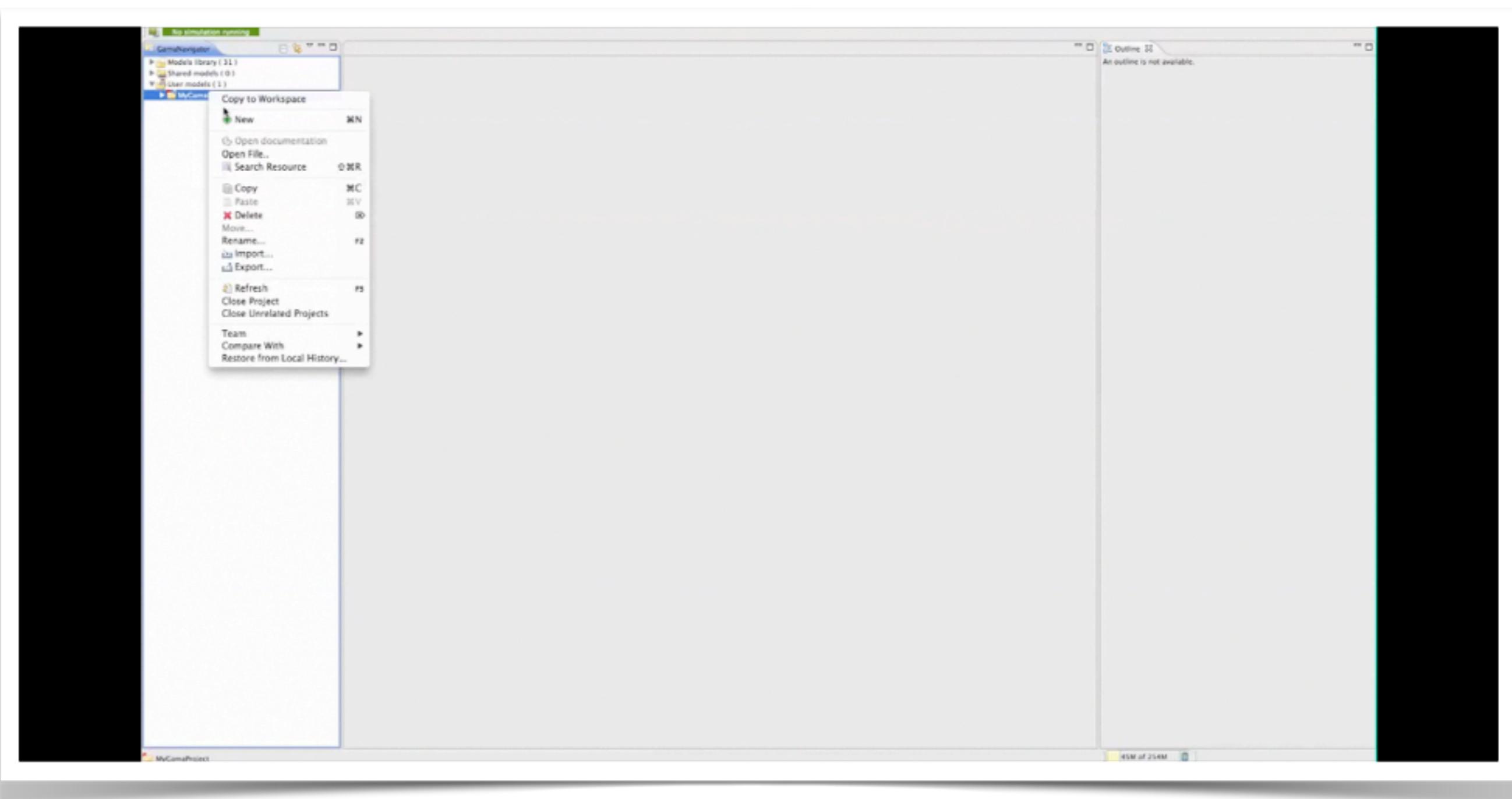
<http://www.gama-platform.org>

# Generalities on the GAMA platform

- ❖ Strengths of GAMA vs other Simulation Frameworks (Netlogo, Repast, Cormas, ...)
  - Supports the development of quite complex models
  - Seamless integration of geographic data and GIS tools with agent-based models
  - Integrates a methodological approach to define multi-level models
  - Integrates high-level tools: multi-criteria decision making tools, clustering functions, statistical operators...
  - Easily extensible thanks to its open architecture, which relies on two legacy Java technologies : OSGI plugin framework and Java annotations



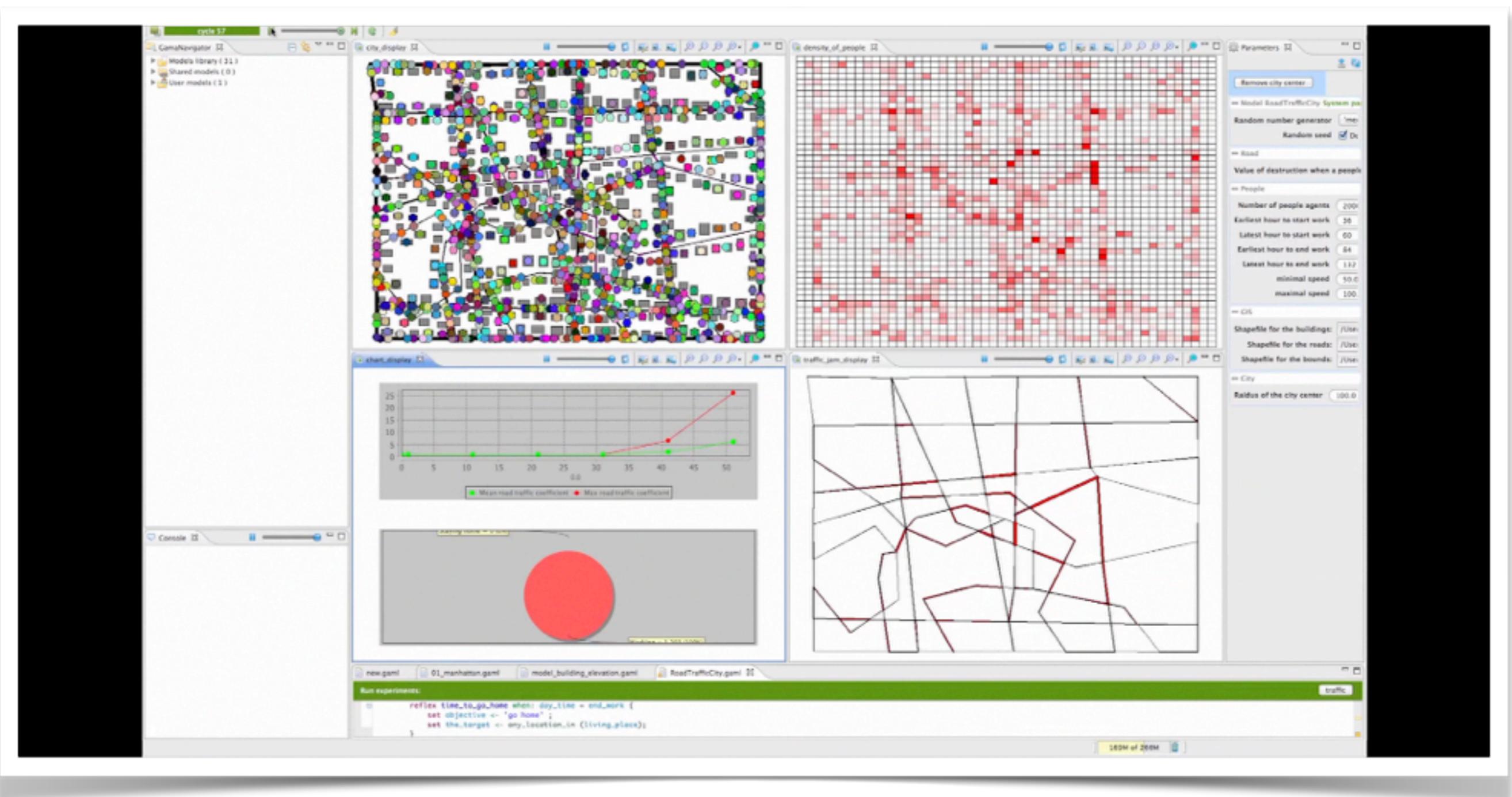
- ❖ GAMA provides a complete Integrated Development environment (IDE) to build models



Dedicated modeling Language (GAML), easy to learn and to extend

# Generalities concerning GAMA

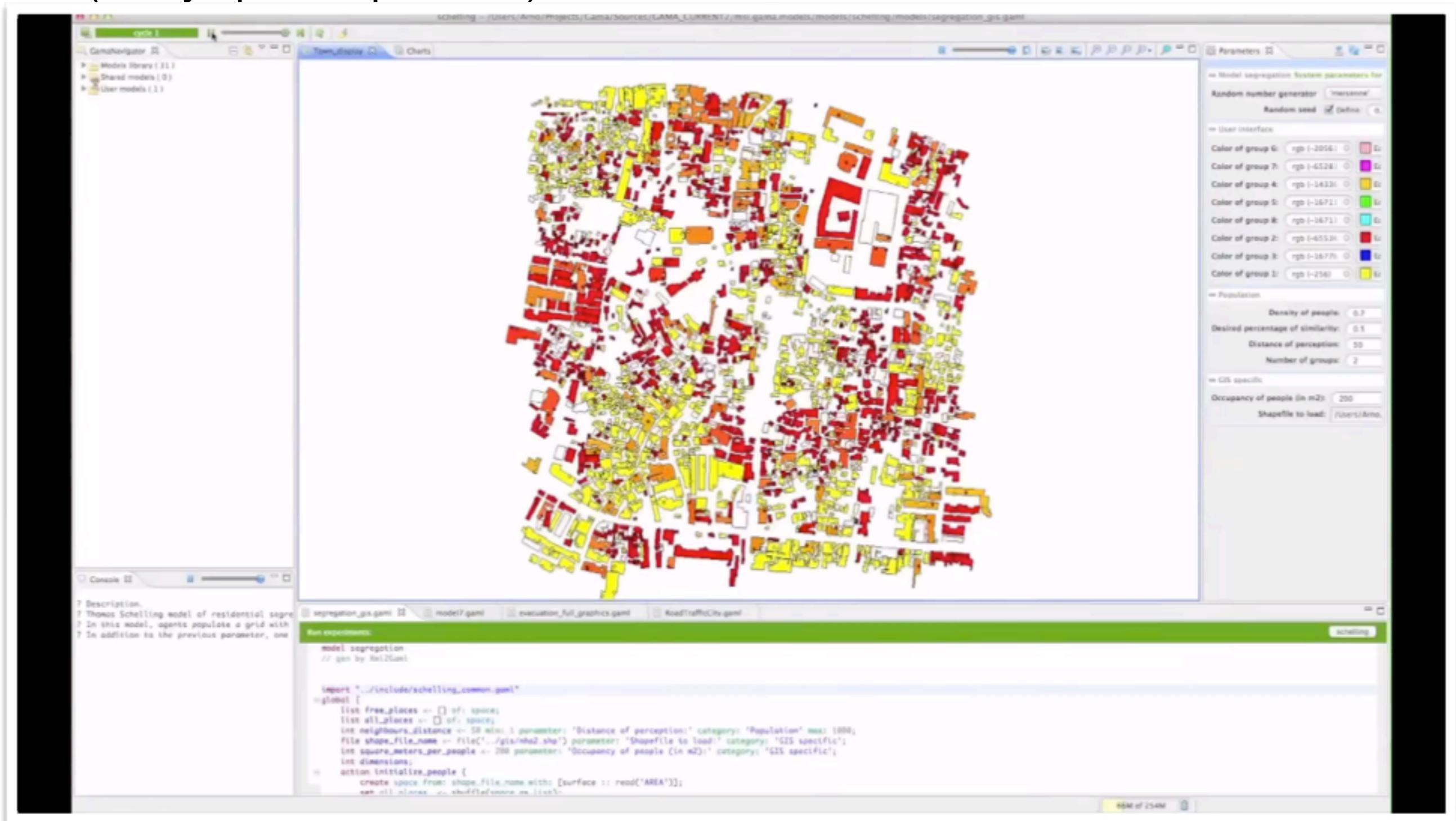
- ❖ Possibility to define as many environments as necessary (3 types of topologies available: continuous, grid and graph)



2D Grid (rectangle, hexagon)  
 Continuous environment, torus environment, graphs

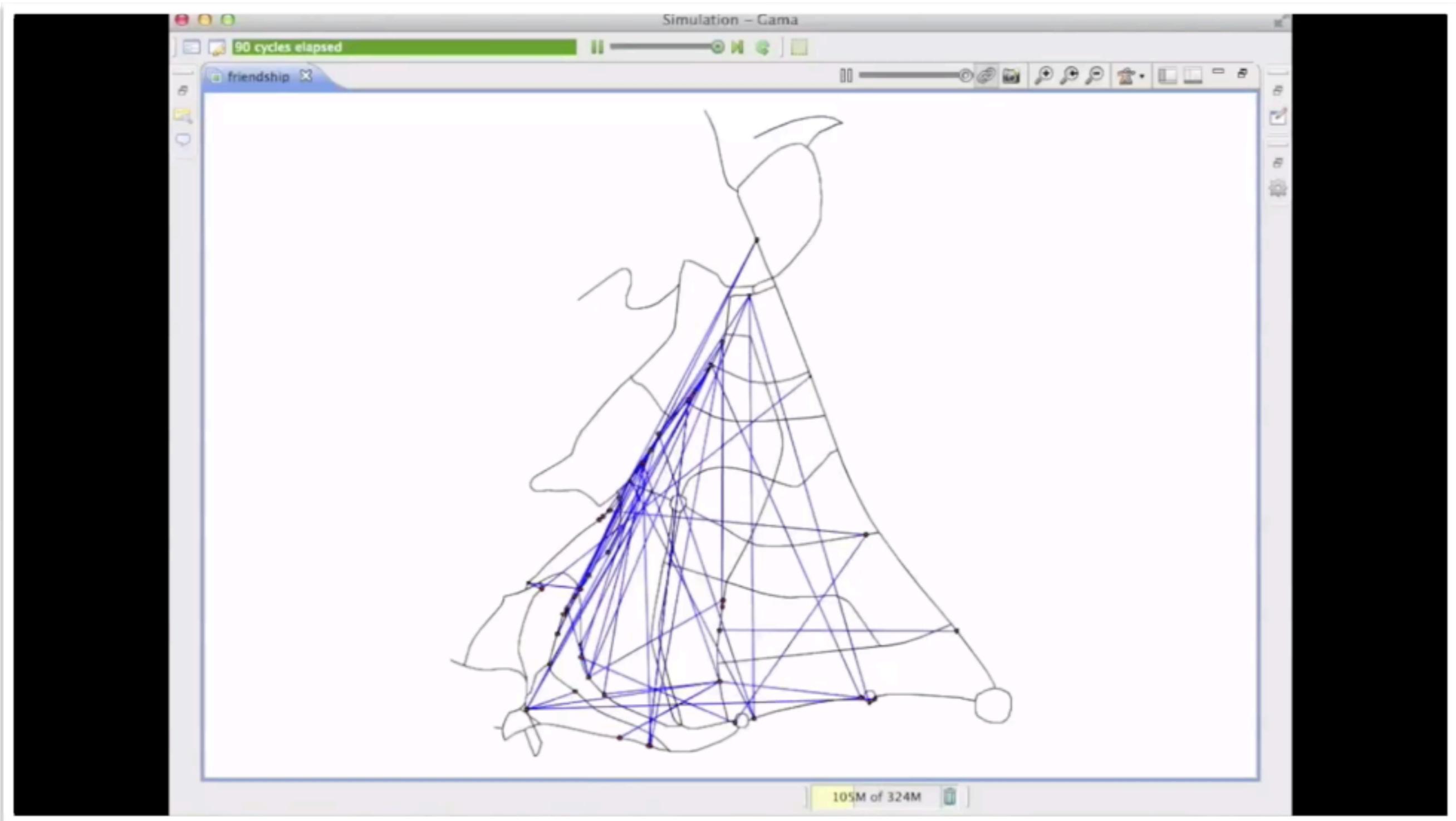
# Generalities concerning GAMA

- ❖ Easy integration of GIS data, powerful features to manage GIS data (many spatial operators)



Transparent agentification of 2D/3D GIS data  
powerful geometrical operations (union, difference, spatial queries....)

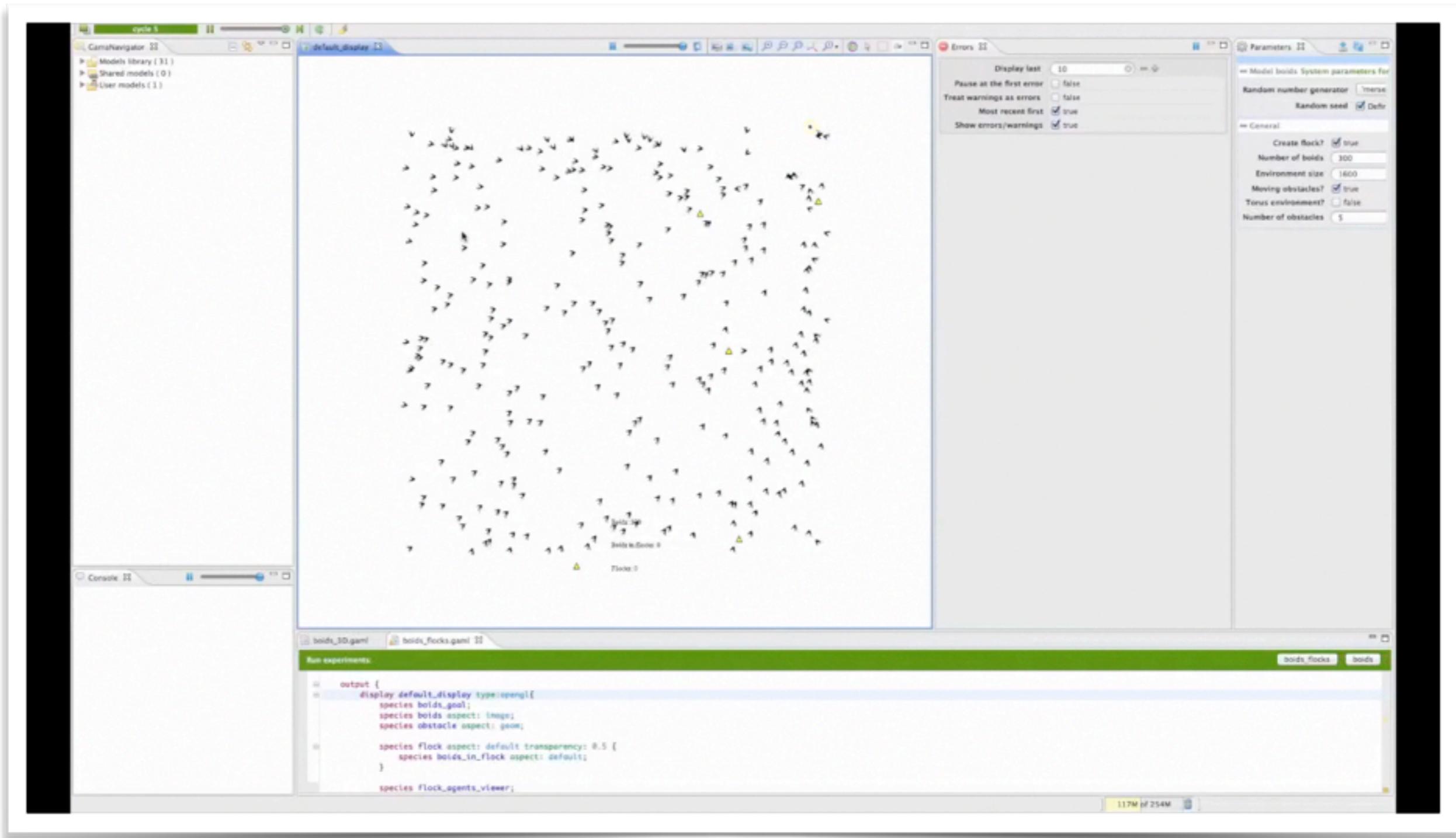
- ❖ Many graph operators



k-shortest paths, centrality betweenness, graph generation...

# Generalities concerning GAMA

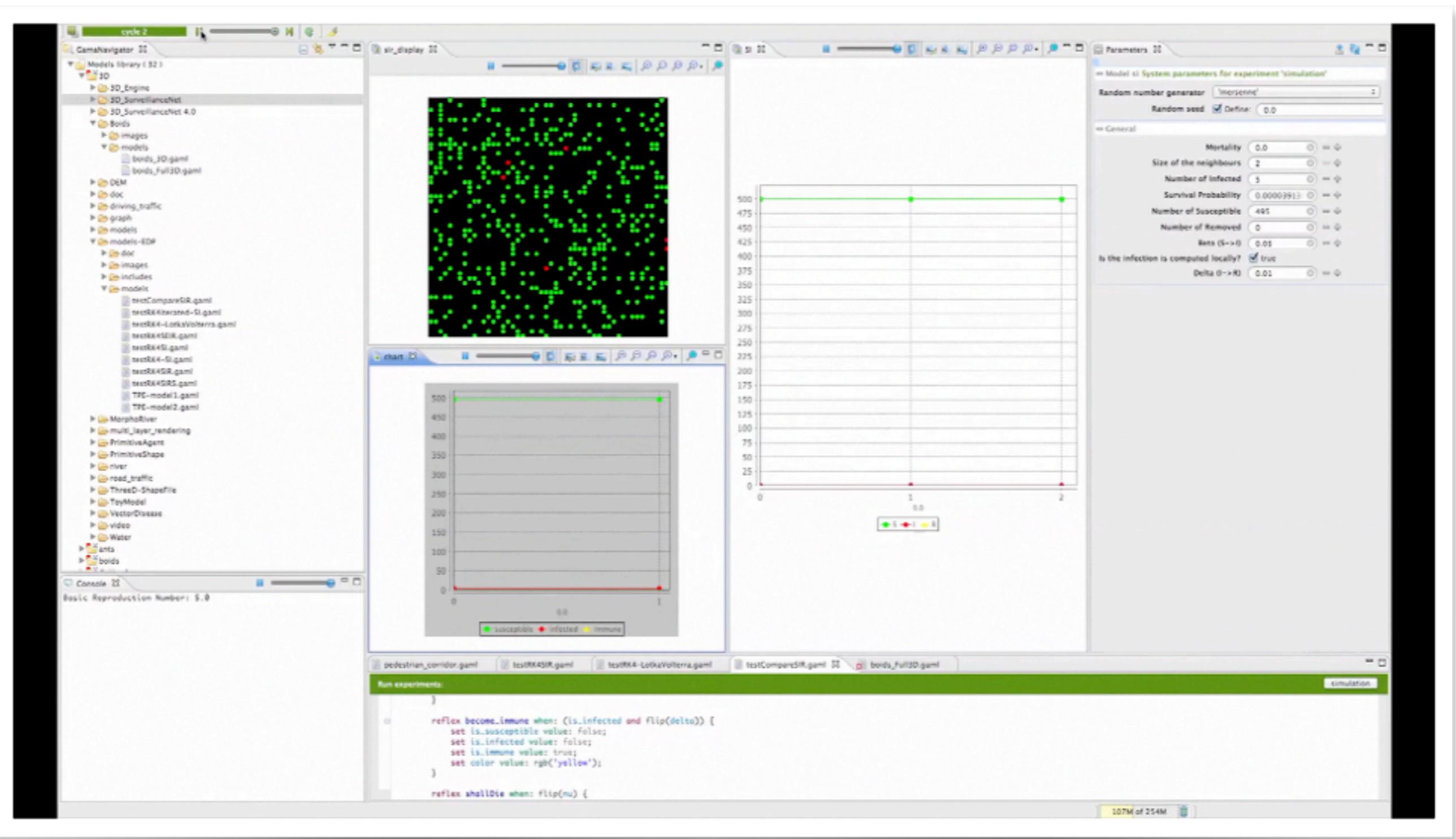
- ❖ Allows to define multi-level models



Different level of abstraction, aggregation operators...

# Generalities concerning GAMA

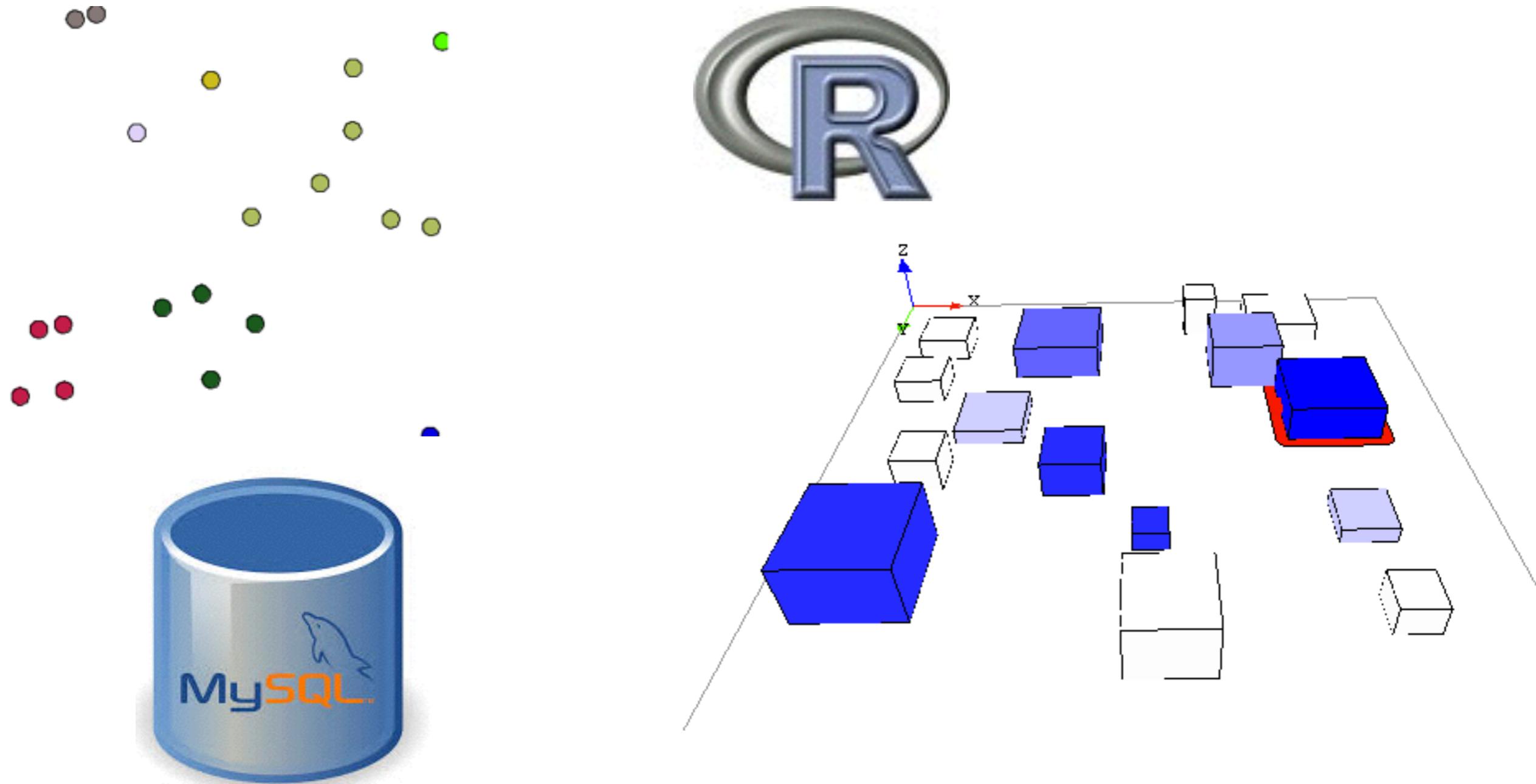
- Allows to use different formalisms to define agent behaviors



Differential equations, Finite state machine, Reflexes, ....

- ❖ 3D modeling and visualization





Link to database, use of R script, clustering, multi-criteria decision making, spatial discretization, spatial interpolation...

# Generalities concerning GAMA

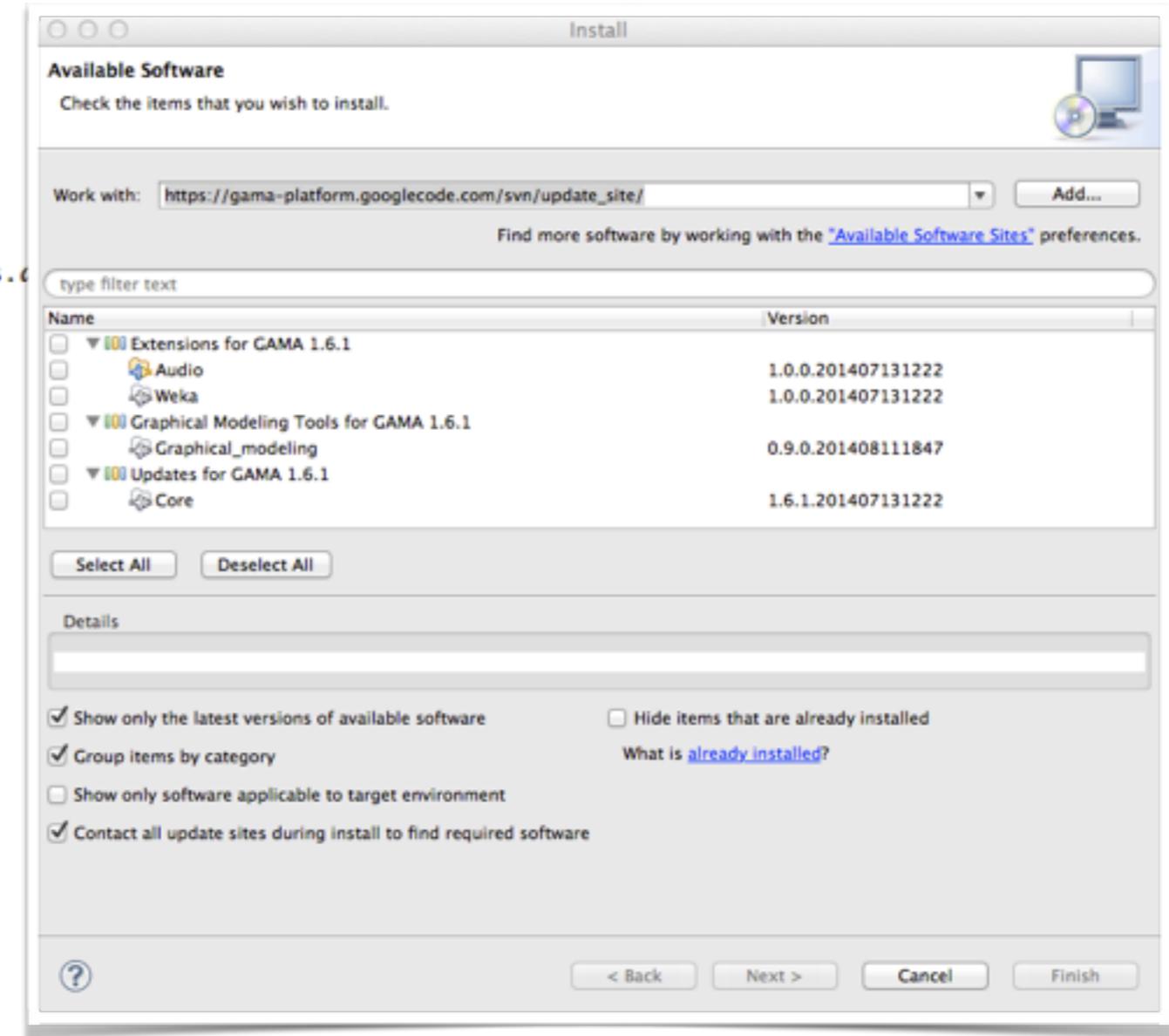
- ❖ Easy development and integration of new plug-ins and features

```

@operator(value = "angle_between", category = { IOperatorCategory.SPATIAL, IOperatorCategory.POINT })
@doc(value = "the angle between vector P0P1 and P0P2",
     examples = { @example(value = "angle_between({5,5},{10,5},{5,10})", equals = "90") })
public static Integer angleInDegreesBetween(final GamaPoint p0, final GamaPoint p1, final GamaPoint p2) {
    final double Xa = p1.x - p0.x;
    final double Ya = p1.y - p0.y;
    final double Xb = p2.x - p0.x;
    final double Yb = p2.y - p0.y;

    final double Na = Maths.sqrt(Xa * Xa + Ya * Ya);
    final double Nb = Maths.sqrt(Xb * Xb + Yb * Yb);
    final double C = (Xa * Xb + Ya * Yb) / (Na * Nb);
    final double S = Xa * Yb - Ya * Xb;
    final double result = S > 0 ? Maths.acos(C) : -1 * Maths.acos(C);
    return Maths.checkHeading((int) result);
}

```



# Generalities concerning GAMA

- ❖ Parameter space exploration tools and compatible with OpenMole

Parameters

Model si\_model Parameters for experiment 'Genetic'

Random number generator: 'mersenne' among [cellular, xor, java, mersenne]  
Random seed:  Define: 0.0

Exploration method

Stop condition: time > 1000  
Best fitness: 1.7976931348623157E308  
Last fitness: 299.0 with {infection\_rate=0.8, speed\_people=5.0}  
Parameter space: infection\_rate (6) \* speed\_people (10) = 60  
Exploration method: Method genetic | fitness = minimize nb\_infected | compute the min of 3 simulations for each solution  
Mutation probability: 0.1  
Crossover probability: 0.7  
Population dimension: 3  
Preliminary number of generations: 1.0  
Max. number of generations: 5.0

Parameters to explore

Infection rate: 0.8 among [0.1, 0.2, 0.5, 0.6, 0.8, 1.0]  
Speed of people: 5.0 between 1.0 and 10.0 every 1.0



# Example of real models: sediment transport

20

- ❖ **Rhône-Alpe, France:** better understand the transport of sediments



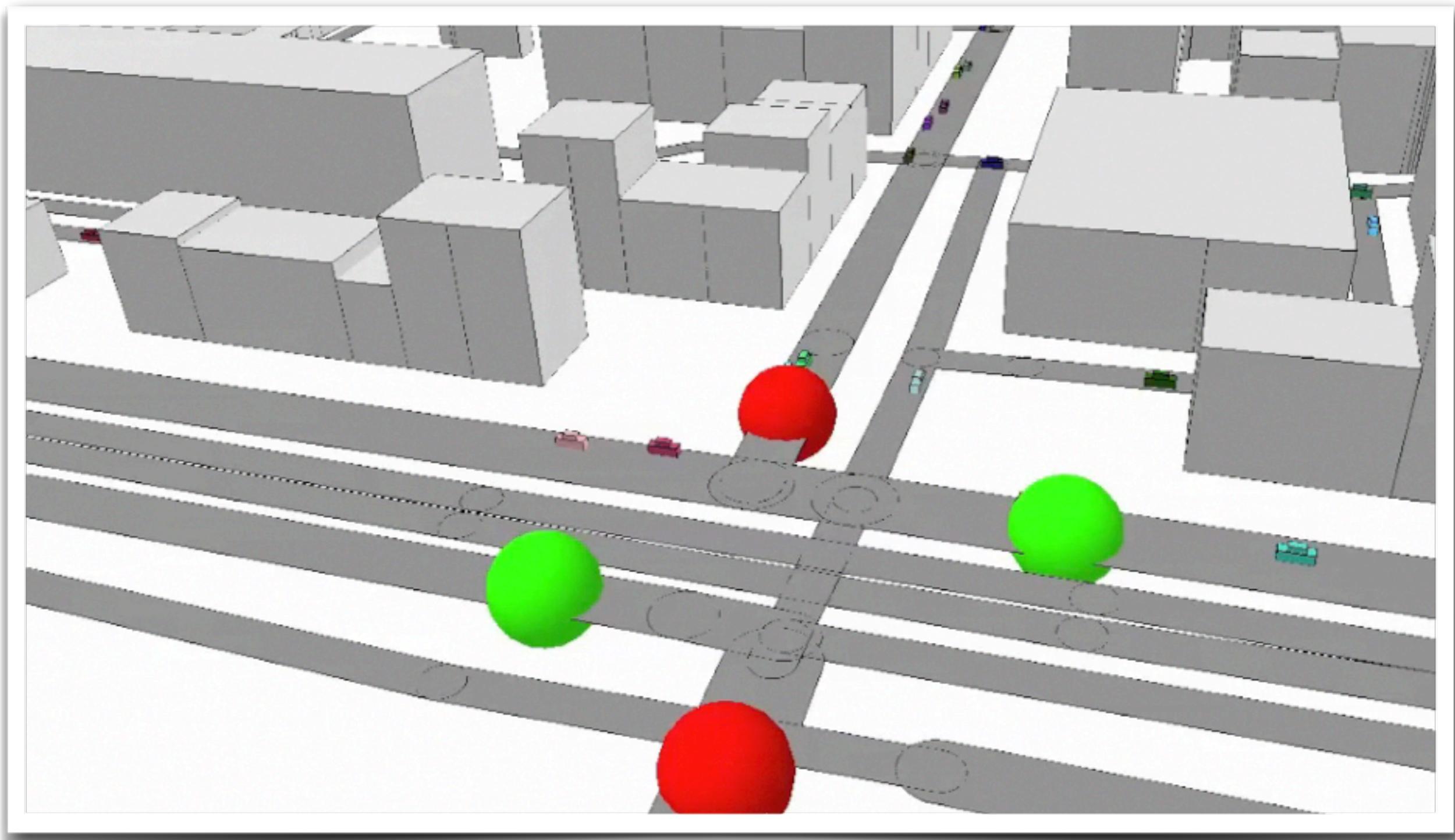
# Example of real models : MAELIA

- ❖ **Adour-Garonne basin, France:** what is the socio-environmental impact of water management norms on water resources?



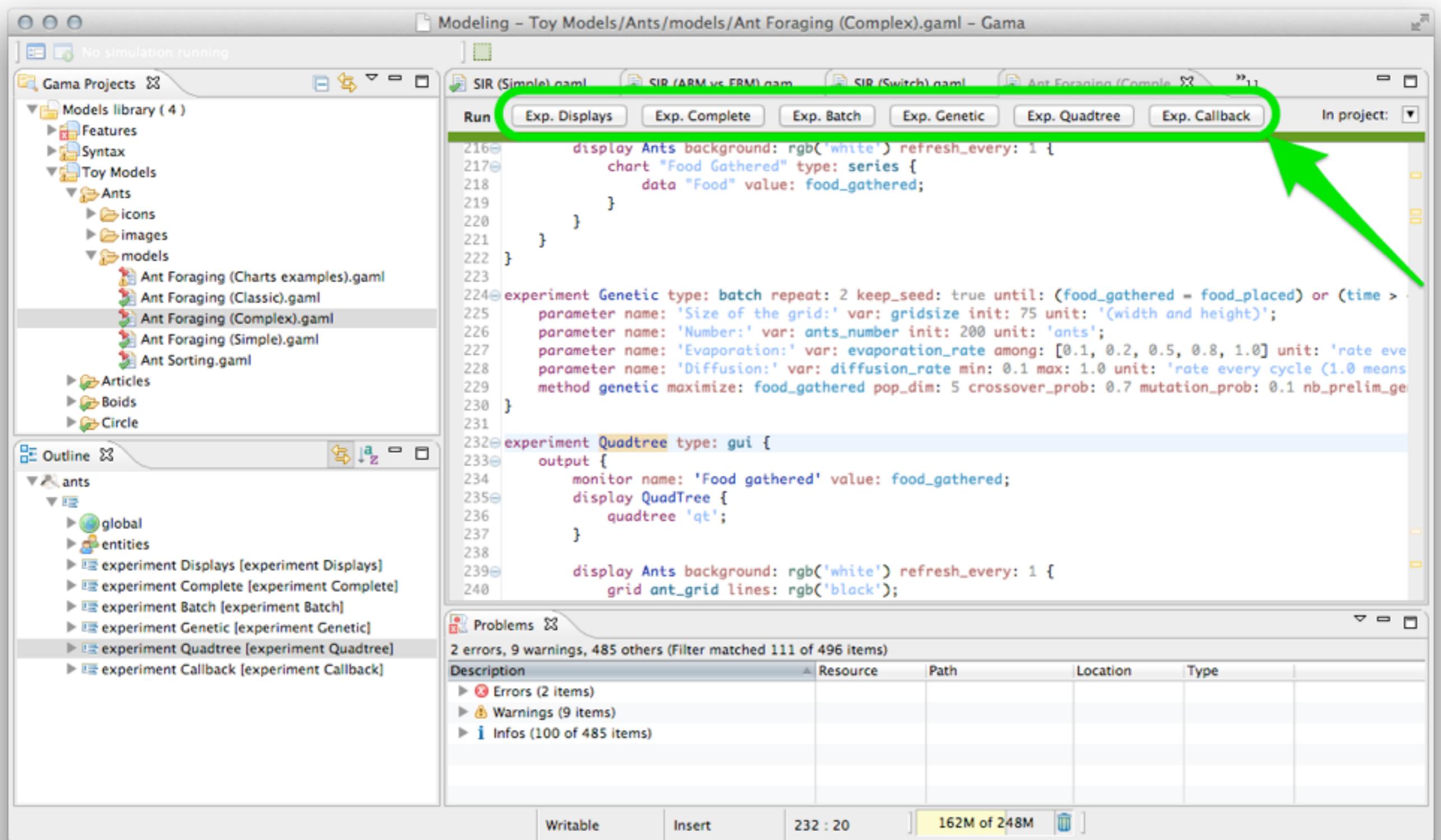
# Example of real models : MOSAIC

- ❖ **Rouen, France:** what is the impact of technological hazards on mobility?



# Loading an experiment

Click on the desired **experiment** button to load it: an experiment define a simulation execution context



# Loading an experiment: model errors

24

The experiment buttons only appear when the experiments can be launched safely, i.e. the model does not contain any error

This screenshot shows the Gama 1.6 IDE interface. The main window displays a code editor with the file "boids.gaml". A red bar at the top of the editor indicates "Error(s) detected. Impossible to run any experiment". The code itself is a Gaml script for a boids model, including parameters like number\_of\_agents, maximal\_speed, and cohesion\_factor. The right side of the interface shows the "Outline" view, which lists various components of the model, including "experiment Boids". The bottom status bar shows "78M of 213M".

```
model boids;
global torus: torus_environment {
    int number_of_agents <- 100 min: 1 max: 1000000;
    int number_of_obstacles <- 5 min: 0;
    float maximal_speed <- 15.0 min: 0.1 max: 15.0;
    int cohesion_factor <- 200;
    int alignment_factor <- 100;
    float minimal_distance <- 10.0;
    int maximal_turn <- 30 min: 0 max: 360;
    int width_and_height_of_environment <- 800;
    bool torus_environment <- false;
    bool apply_cohesion <- true;
    bool apply_alignment <- true;
    bool apply_separation <- true;
    bool apply_goal <- true;
    bool apply_avoid <- true;
    bool apply_wind <- true;
    bool moving_obstacles <- false;
    int bounds <- int(width_and_height_of_environment / 20) depends_on: [width_and_height_of_environment];
    point wind_vector <- {0,0};
    int pool_duration <- 30 update: (pool.duration + 1);
    point goal <- {rnd(width_and_height_of_environment + 20) + 1, rnd(images.of: file <- [file("../images/bird1.png"), file("../images/bird2.png")])};
    int xmin <- bounds depends_on: [bounds];
    int ymin <- bounds depends_on: [bounds];
    int xmax <- (width_and_height_of_environment - bounds);
    int ymax <- (width_and_height_of_environment - bounds);}
```

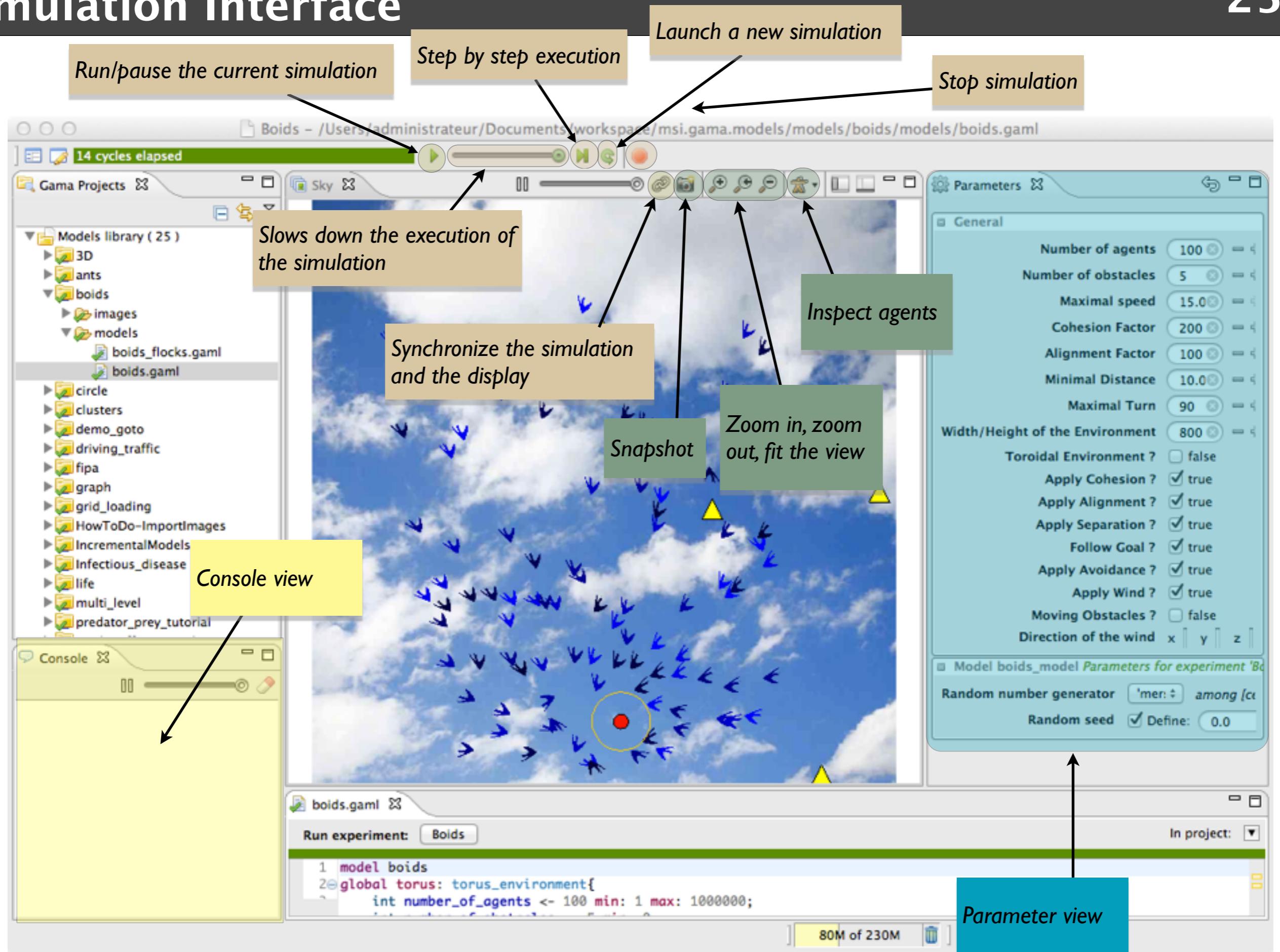
**Red:** Model with error(s)

This screenshot shows the Gama 1.6 IDE interface. The main window displays the same "boids.gaml" file as the previous screenshot, but now it is error-free. The red bar at the top of the editor is gone, replaced by a green bar indicating "Run experiment". The code is identical to the one in the first screenshot. The right side of the interface shows the "Outline" view, which lists various components of the model, including "experiment Boids". The bottom status bar shows "50M of 199M".

```
model boids;
global torus: torus_environment {
    int number_of_agents <- 100 min: 1 max: 1000000;
    int number_of_obstacles <- 5 min: 0;
    float maximal_speed <- 15.0 min: 0.1 max: 15.0;
    int cohesion_factor <- 200;
    int alignment_factor <- 100;
    float minimal_distance <- 10.0;
    int maximal_turn <- 90 min: 0 max: 360;
    int width_and_height_of_environment <- 800;
    bool torus_environment <- false;
    bool apply_cohesion <- true;
    bool apply_alignment <- true;
    bool apply_separation <- true;
    bool apply_goal <- true;
    bool apply_avoid <- true;
    bool apply_wind <- true;
    bool moving_obstacles <- false;
    int bounds <- int(width_and_height_of_environment / 20) depends_on: [width_and_height_of_environment];
    point wind_vector <- {0,0};
    int pool_duration <- 30 update: (pool.duration + 1);
    point goal <- {rnd(width_and_height_of_environment + 20) + 1, rnd(images.of: file <- [file("../images/bird1.png"), file("../images/bird2.png")])};
    int xmin <- bounds depends_on: [bounds];
    int ymin <- bounds depends_on: [bounds];
    int xmax <- (width_and_height_of_environment - bounds);
    int ymax <- (width_and_height_of_environment - bounds);}
```

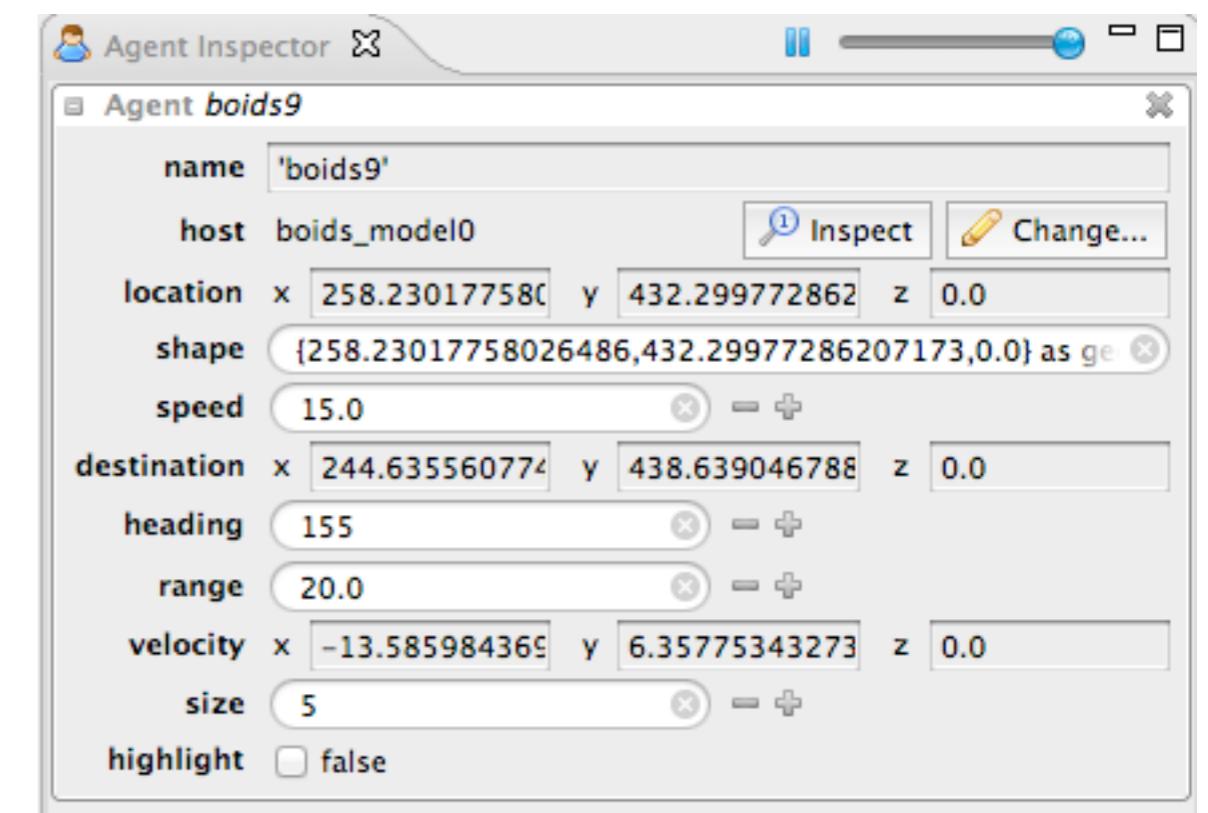
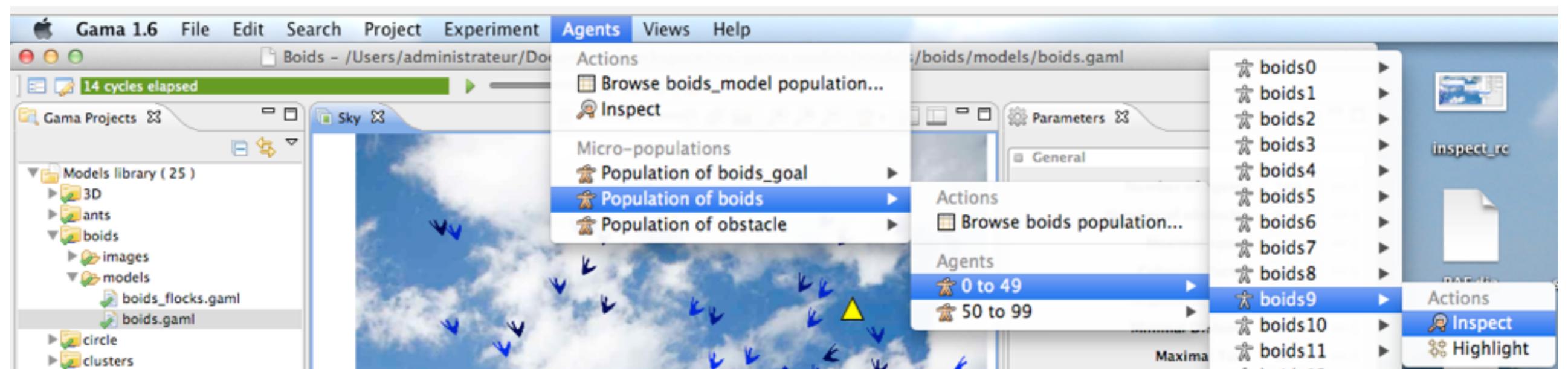
**Green:** Model without error(s) : ready to load !

# Simulation Interface



# Inspector – 1

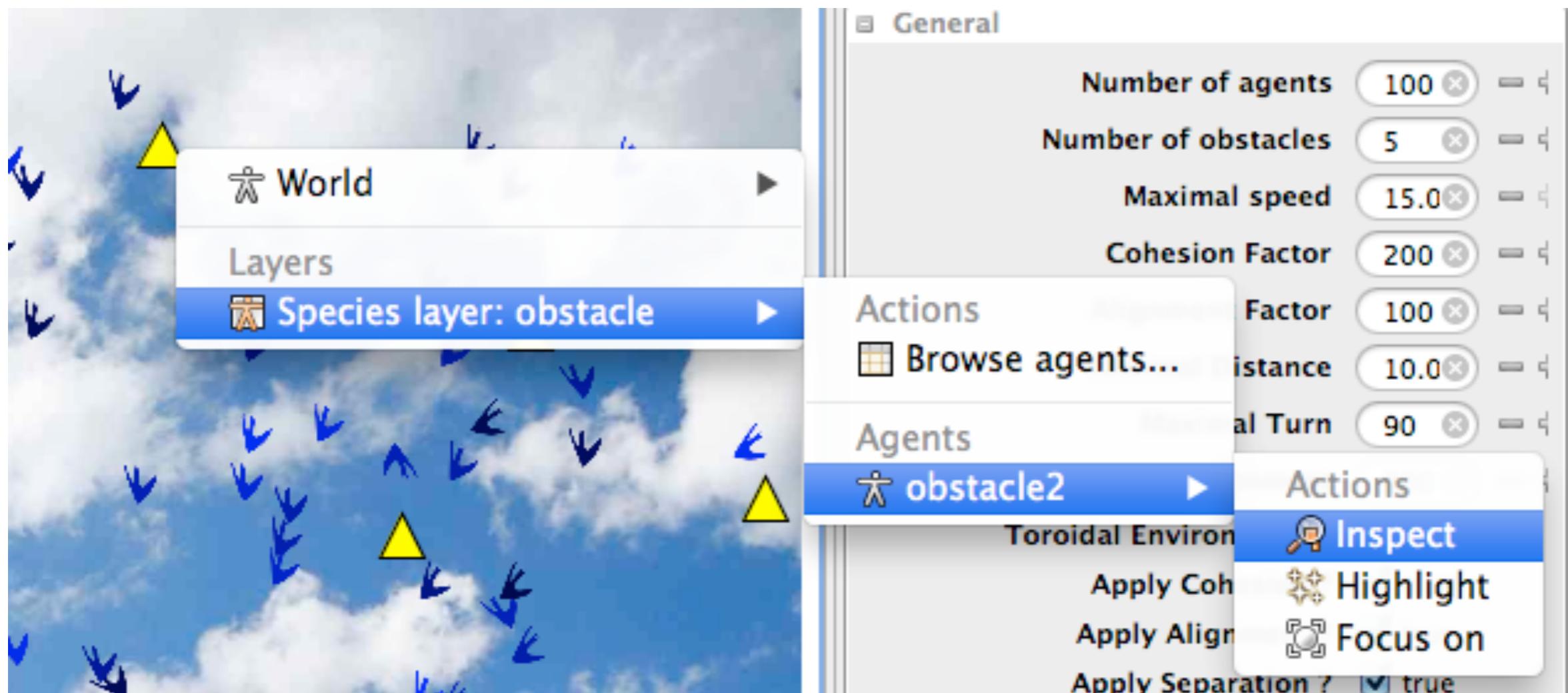
❖ **Inspector:** allows to obtain information on agents



# Inspector – 2

❖ **Inspector:** allows to obtain information on agents

- It is possible to inspect an agent by right clicking on it (in a display)



# Inspector – 3

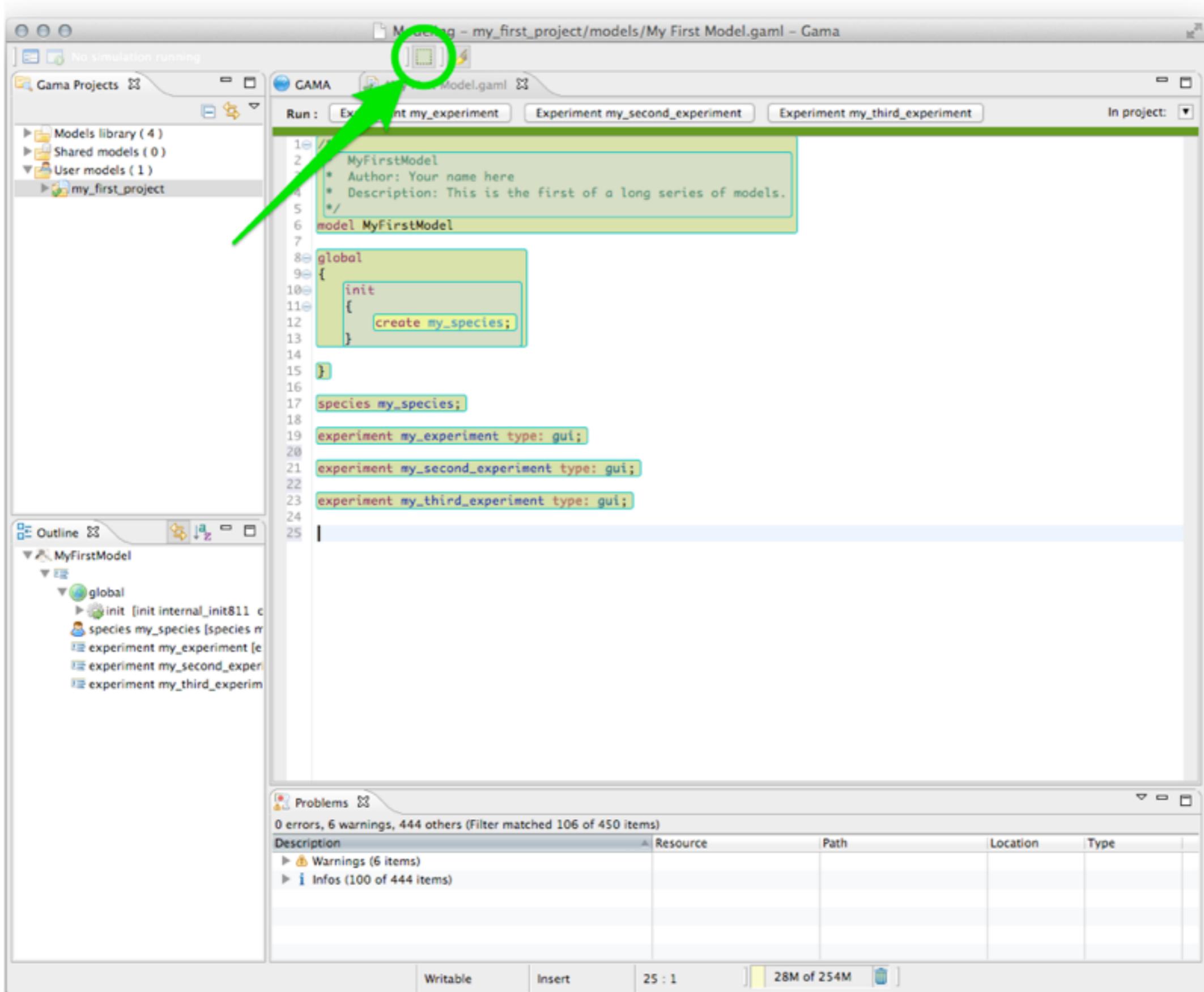
- **Agent browser:** gives information on all agents of a specific species

The screenshot shows the AnyLogic Agent Browser interface. At the top, there's a menu bar with 'Agents', 'Views', 'Window', and 'Help'. Below the menu is a toolbar with icons for actions like 'Browse boids\_model population...', 'Inspect boids\_model0', and 'Micro-populations'. A tree view on the left lists 'Population of boids\_goal' (selected), 'Population of boids' (highlighted in green), and 'Population of obstacle'. A context menu is open over 'Population of boids', with options like 'Actions' and 'Browse boids population...'. Below the tree is a table titled 'Boids population in macro-agent boids\_model0; size: 100 agents'. The table has columns for 'name', 'destination', 'heading', 'host', 'location', and 'range'. The 'Attributes' column on the left lists various agent properties: 'agents', 'destination', 'heading', 'host', 'location', 'members', 'name', 'peers', 'range', 'shape', 'size', 'speed', and 'velocity'. The table contains 100 rows of data, each representing a boid agent.

	name	destination	heading	host	location	range
'boids0'	{260.0411868...	35	0 as boids_...	{247.7539061...	20.0	
'boids1'	{474.4819529...	56	0 as boids_...	{466.0940594...	20.0	
'boids2'	{388.4401281...	19	0 as boids_...	{374.2573494...	20.0	
'boids3'	{320.5702026...	27	0 as boids_...	{307.2051047...	20.0	
'boids4'	{436.9073390...	3	0 as boids_...	{421.9278960...	20.0	
'boids5'	{304.2211317...	22	0 as boids_...	{290.3133738...	20.0	
'boids6'	{325.9359584...	32	0 as boids_...	{313.2152370...	20.0	
'boids7'	{279.2638894...	12	0 as boids_...	{264.5916754...	20.0	
'boids8'	{255.9775343...	24	0 as boids_...	{242.2743524...	20.0	
'boids9'	{382.6303368...	34	0 as boids_...	{370.1947732...	20.0	
'boids10'	{183.6588568...	26	0 as boids_...	{170.1769461...	20.0	
'boids11'	{351.1341535...	23	0 as boids_...	{337.3265807...	20.0	
'boids12'	{450.3780464...	27	0 as boids_...	{437.0129485...	20.0	
'boids13'	{295.7744747...	13	0 as boids_...	{281.1589238...	20.0	
'boids14'	{127.6563099...	193	0 as boids_...	{142.2718609...	20.0	
'boids15'	{220.5039898...	34	0 as boids_...	{208.0684262...	20.0	
'boids16'	{231.5202628...	24	0 as boids_...	{217.8170809...	20.0	
'boids17'	{413.1355617...	9	0 as boids_...	{398.3202366...	20.0	
'boids18'	{429.5359479...	21	0 as boids_...	{415.5322415...	20.0	

# Modeling interface: structural highlighting

29

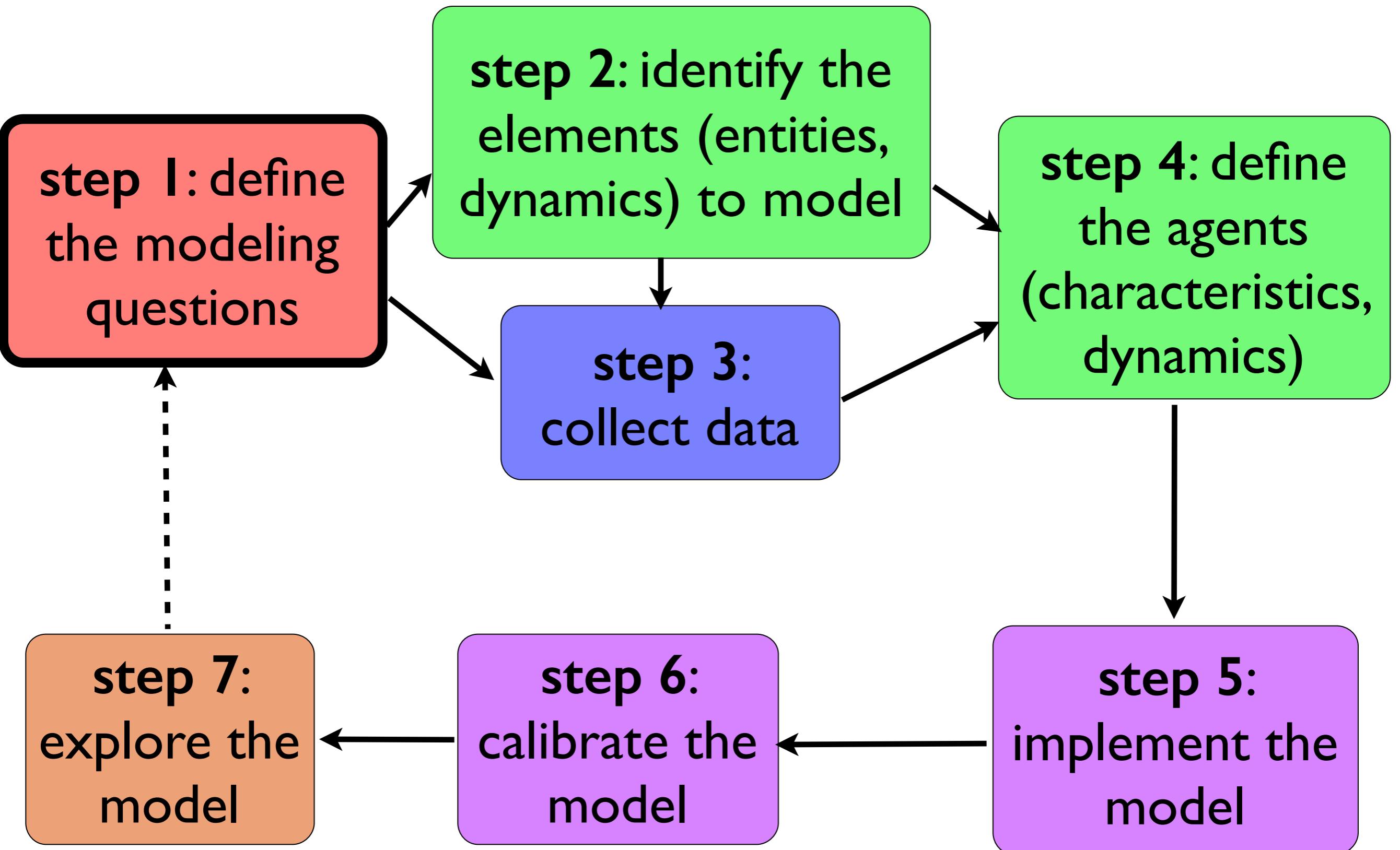


# Phenomena to study: disease outbreak in a small city



Every winters, the flu is spreading in the  
small city of Luneray in Normandie !



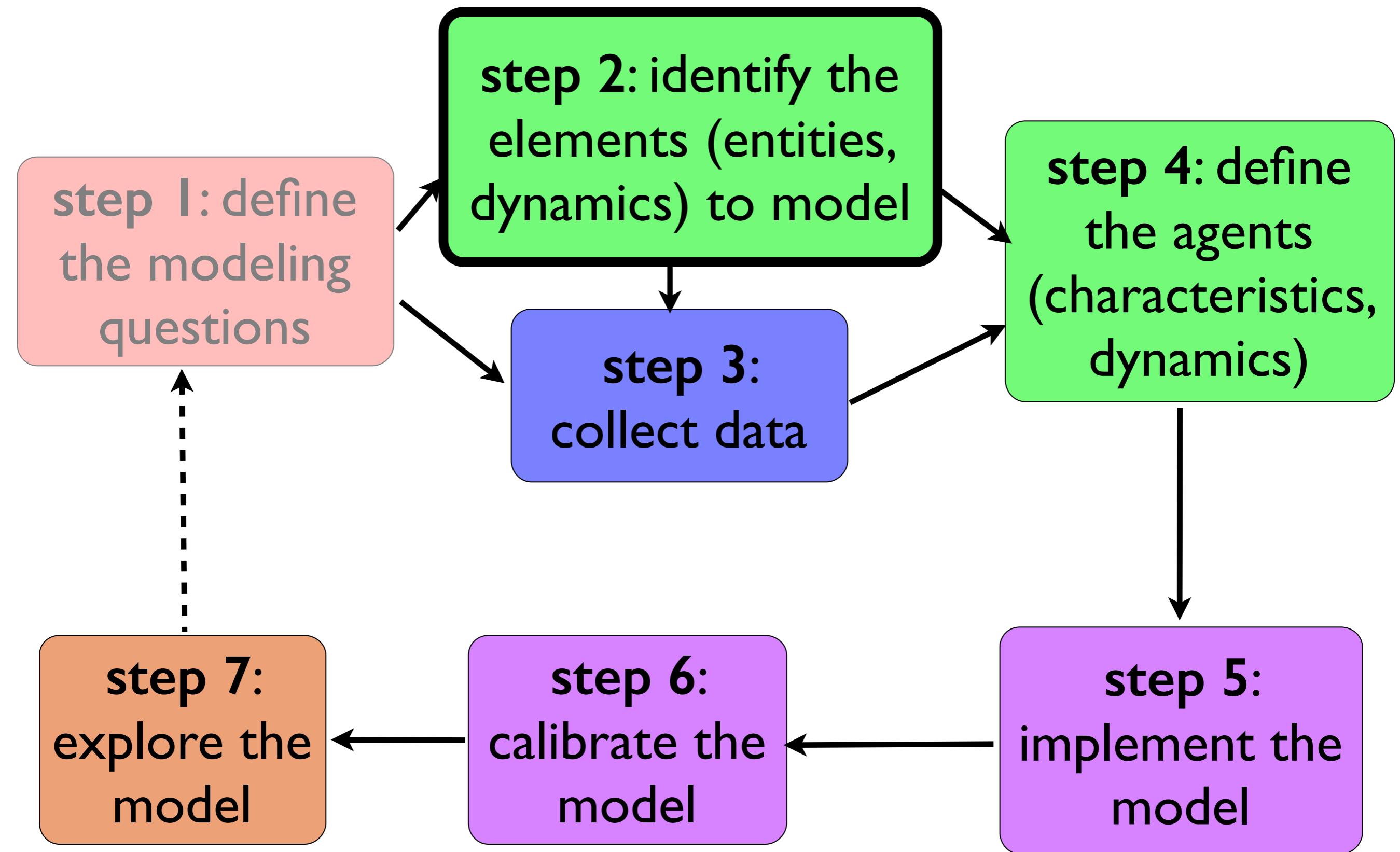


# Step 1: modeling questions

## ► Questions to answer

- How long before the complete infection of the population of Luneray ?





► Study area

- City of Luneray, area = close to 1500m x 1500m

► Duration of the study phenomena

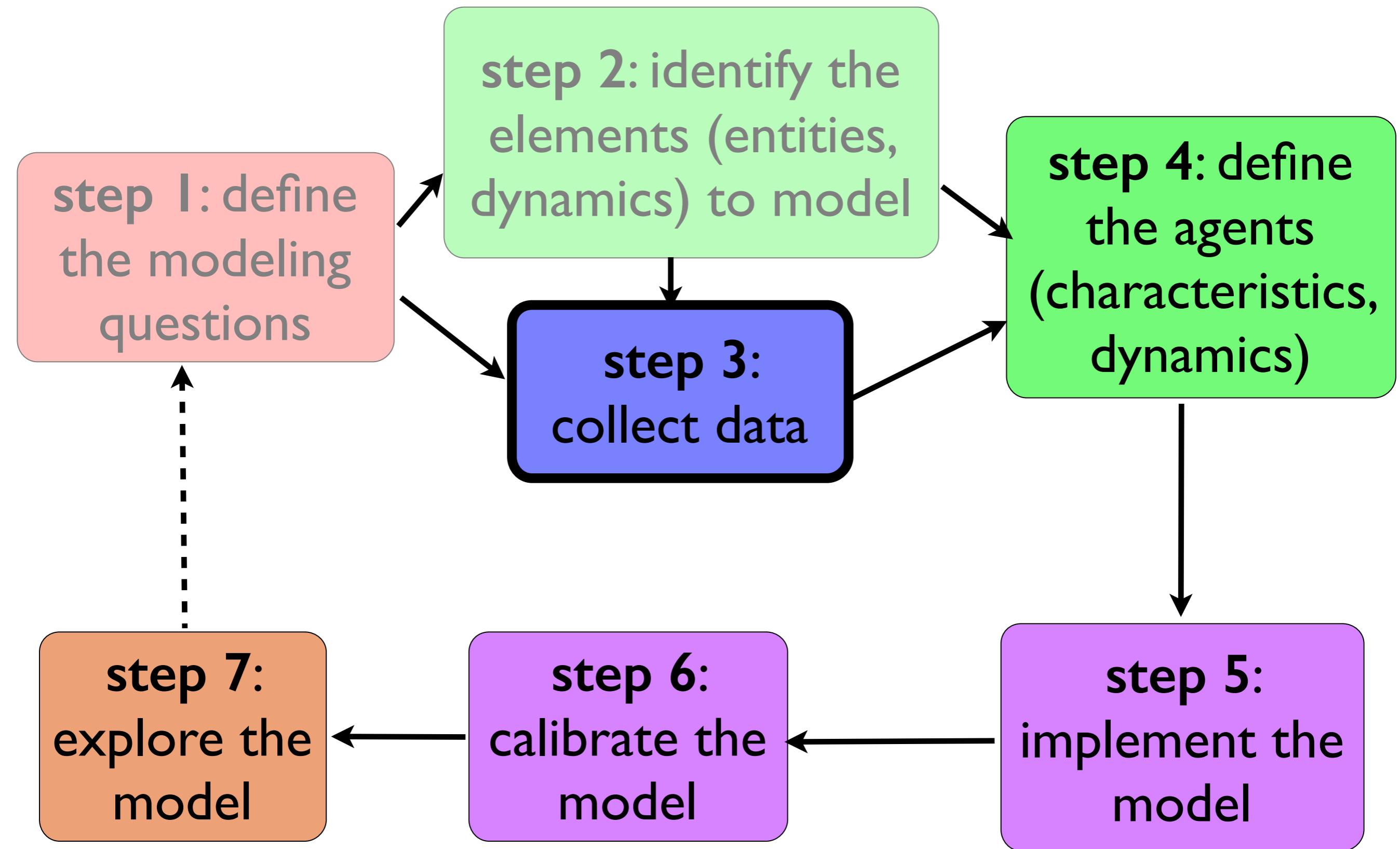
- Several hours

► Dynamic to take into account:

- Movement of people
- Disease spreading

► Elements to take into account:

- The people
- The buildings (where they are carrying out activities)
- The roads used for moving



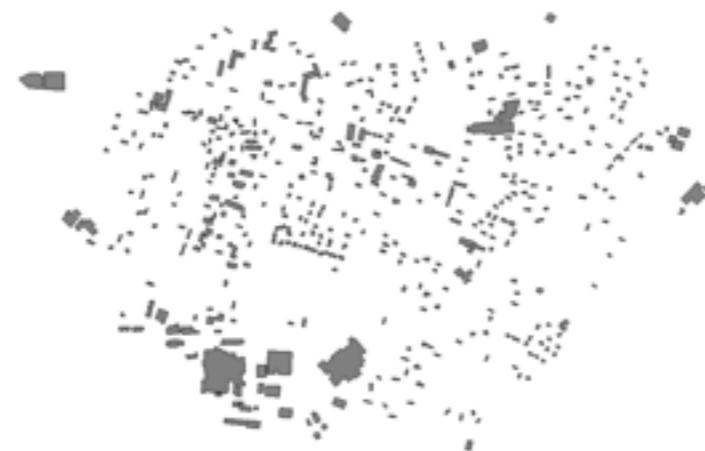
## Step 3: data collection

### ► Data available:

- Shapefile of the roads (OSM)

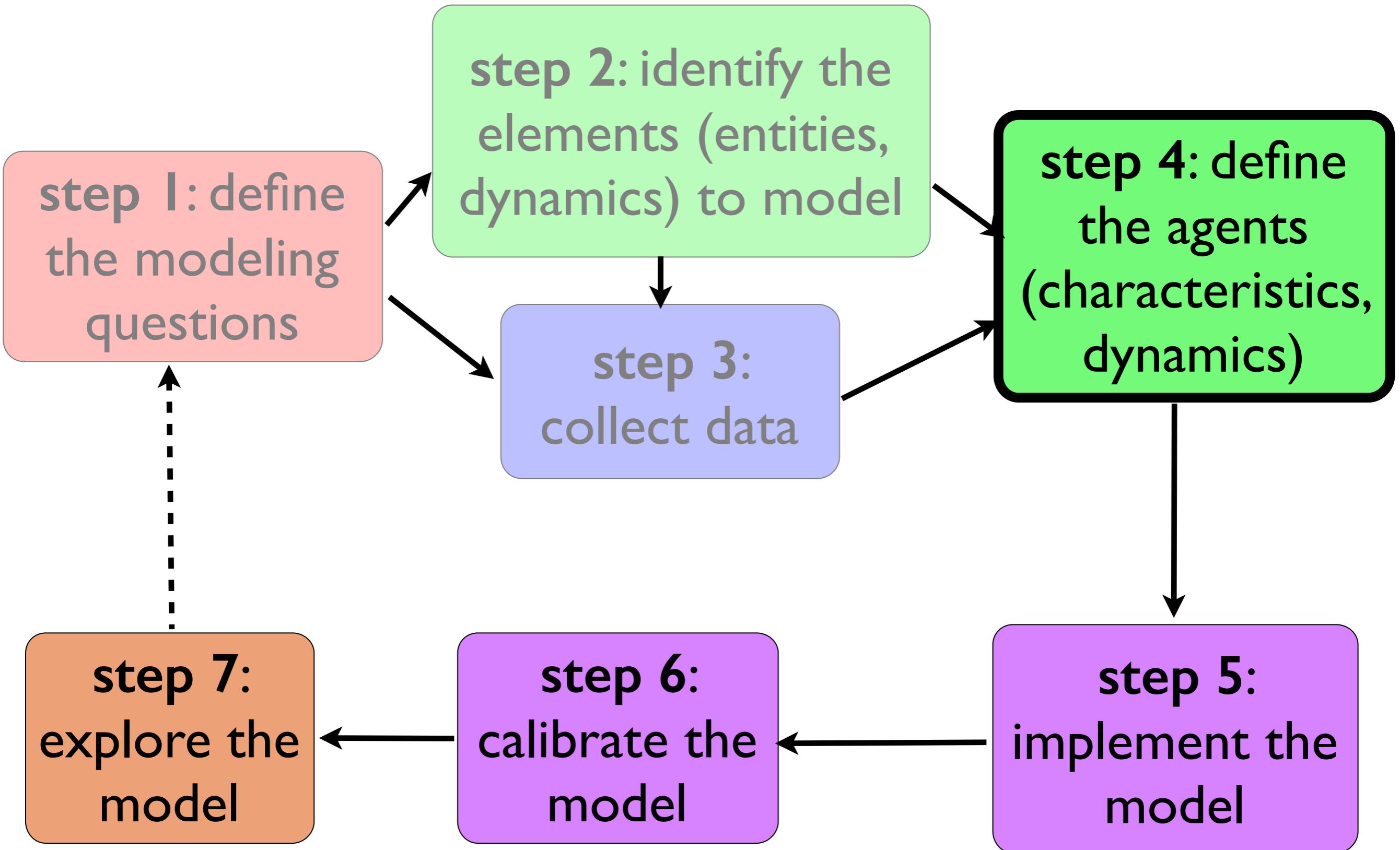


- Shapefile of the buildings (OSM)



- Number of inhabitants of Luneray: 2147 (source : wikipedia)
- Mean speed of the inhabitants (while moving on the road) : 2–5 km/h
- The disease – non lethal – is spreading (by air) from people to people
- Time to cure the disease: more than 100 days
- Infection distance: 10 meters
- Infection probability (when two people are at infection distance) : 0.05/minute

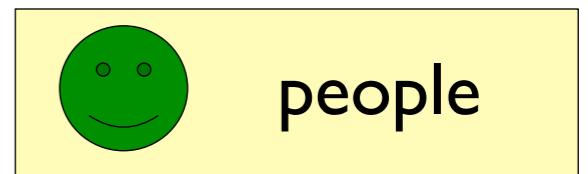
# Modeling steps



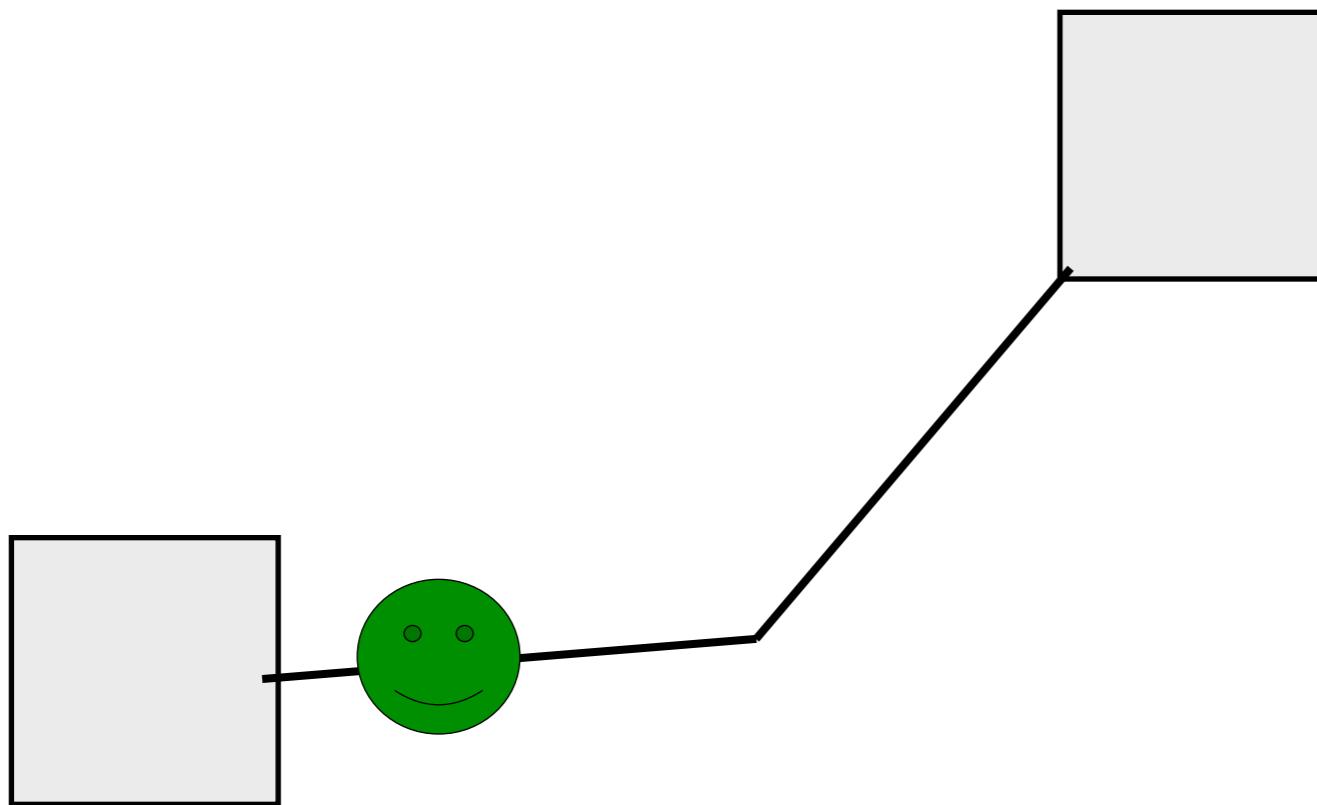
### ► Modeling choices:

- Simulation step: 1 minute
- People are moving on the roads from building to building,
- Most of time people are moving to meet their friend then go back home
- People use the shortest path to move between buildings
- All people move at constant speed
- Each time, people arrived at a building they are staying a certain time : they are staying longer in their home than in their friend houses.
- Infected people are never cured

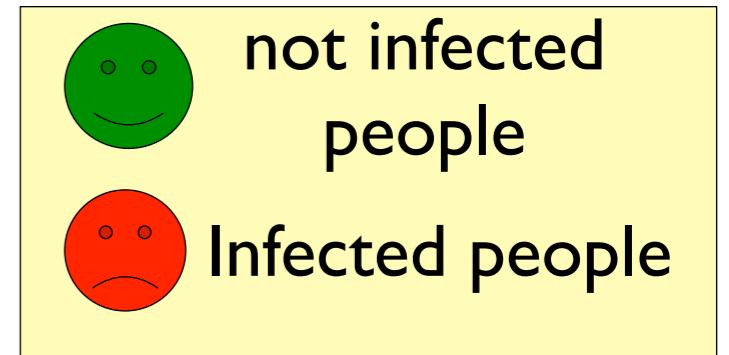
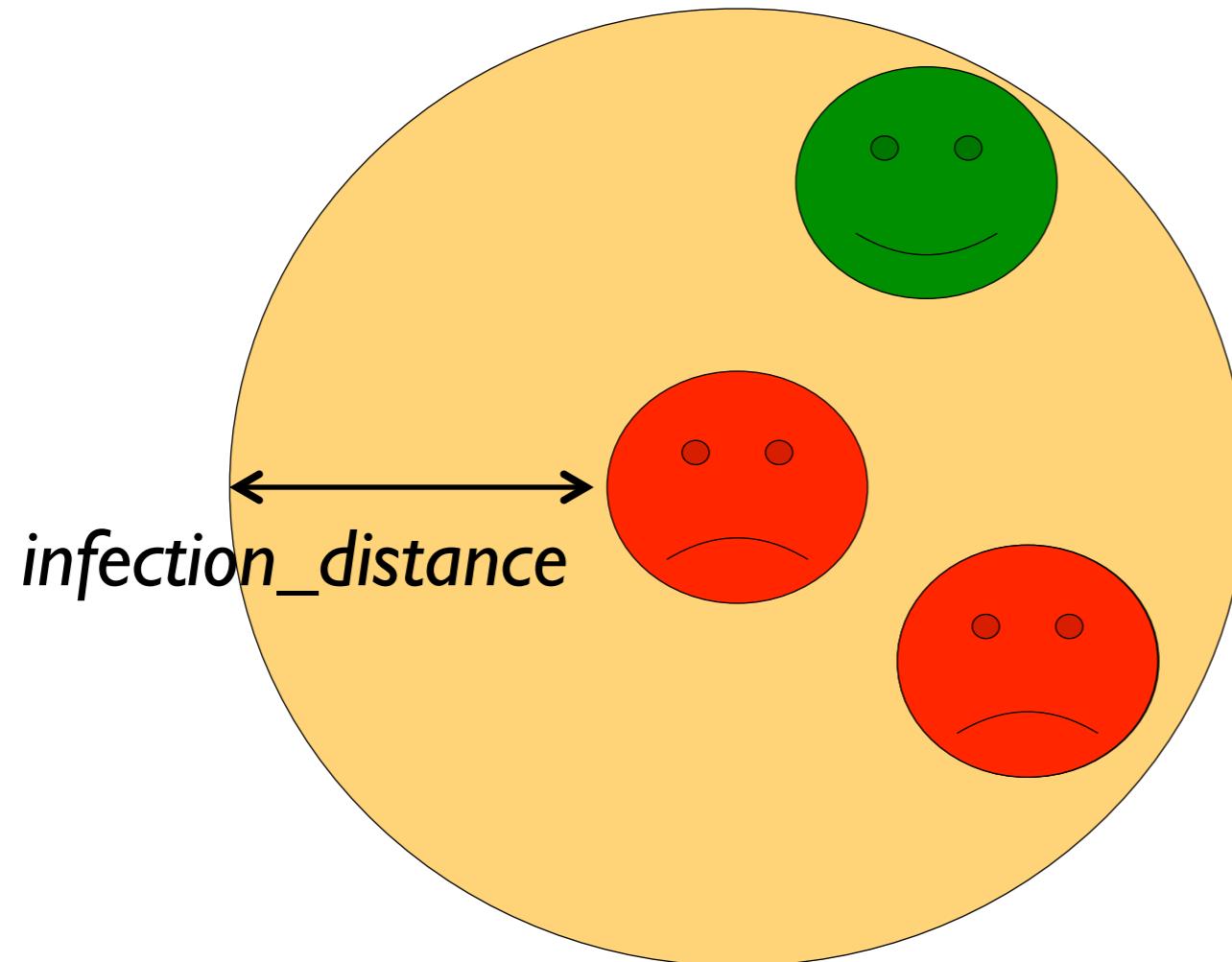
## Moving dynamic



Stay for a certain time: at each simulation step, probability to leave:  
*if in their house: 0.01; otherwise: 0.1*

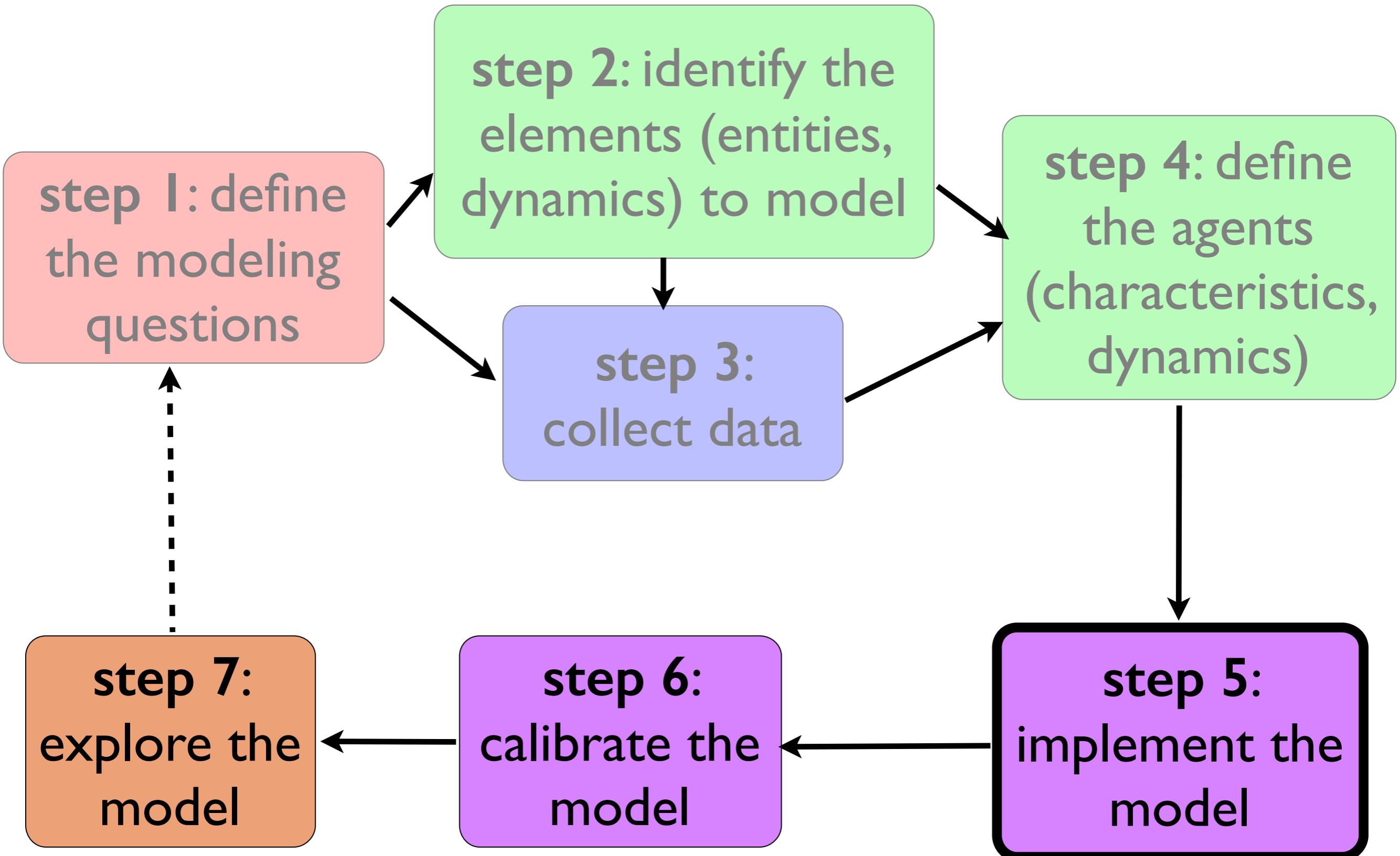


## Infection dynamic



For each potential  
victims, probability to  
be infected:  
*proba\_infection*

# Modeling steps



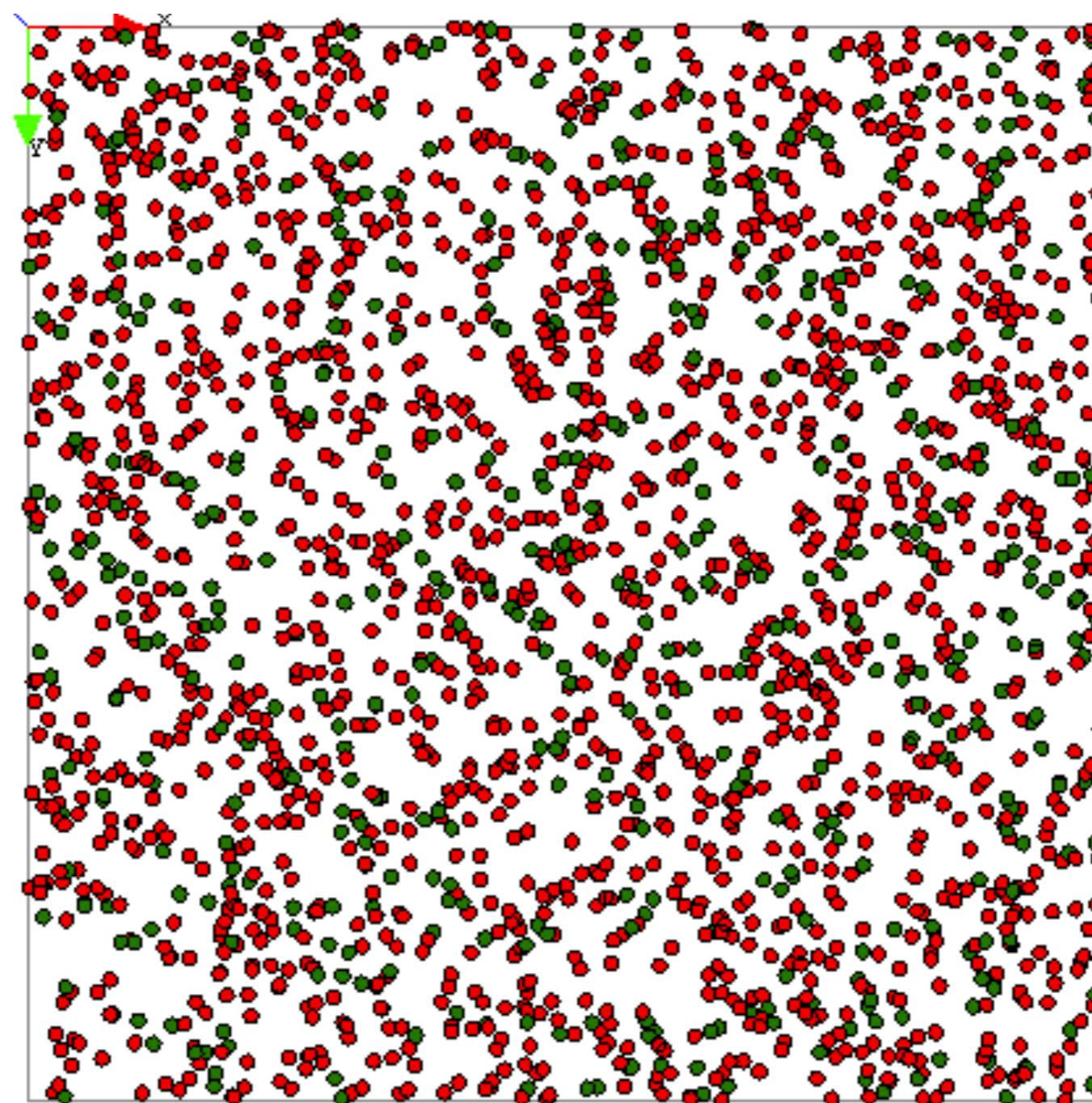
# On-line version of the tutorial

[https://github.com/gama-platform/gama/wiki/Tutorial\\_\\_LuneraysFlu](https://github.com/gama-platform/gama/wiki/Tutorial__LuneraysFlu)

The screenshot shows a GitHub wiki page for the 'Tutorial\_\_LuneraysFlu' repository. The page title is 'Tutorial\_\_LuneraysFlu'. The main content section is titled 'Luneray's flu'. It contains a paragraph about the tutorial's goal to introduce GAMA modeling and GIS data, and a link to a presentation PDF. Below this is a section titled 'Model Overview' with a note about the model concerning the spreading of a flu in Luneray, Normandie, France. A large world map highlights the location of Luneray. To the right is a sidebar with a navigation tree:

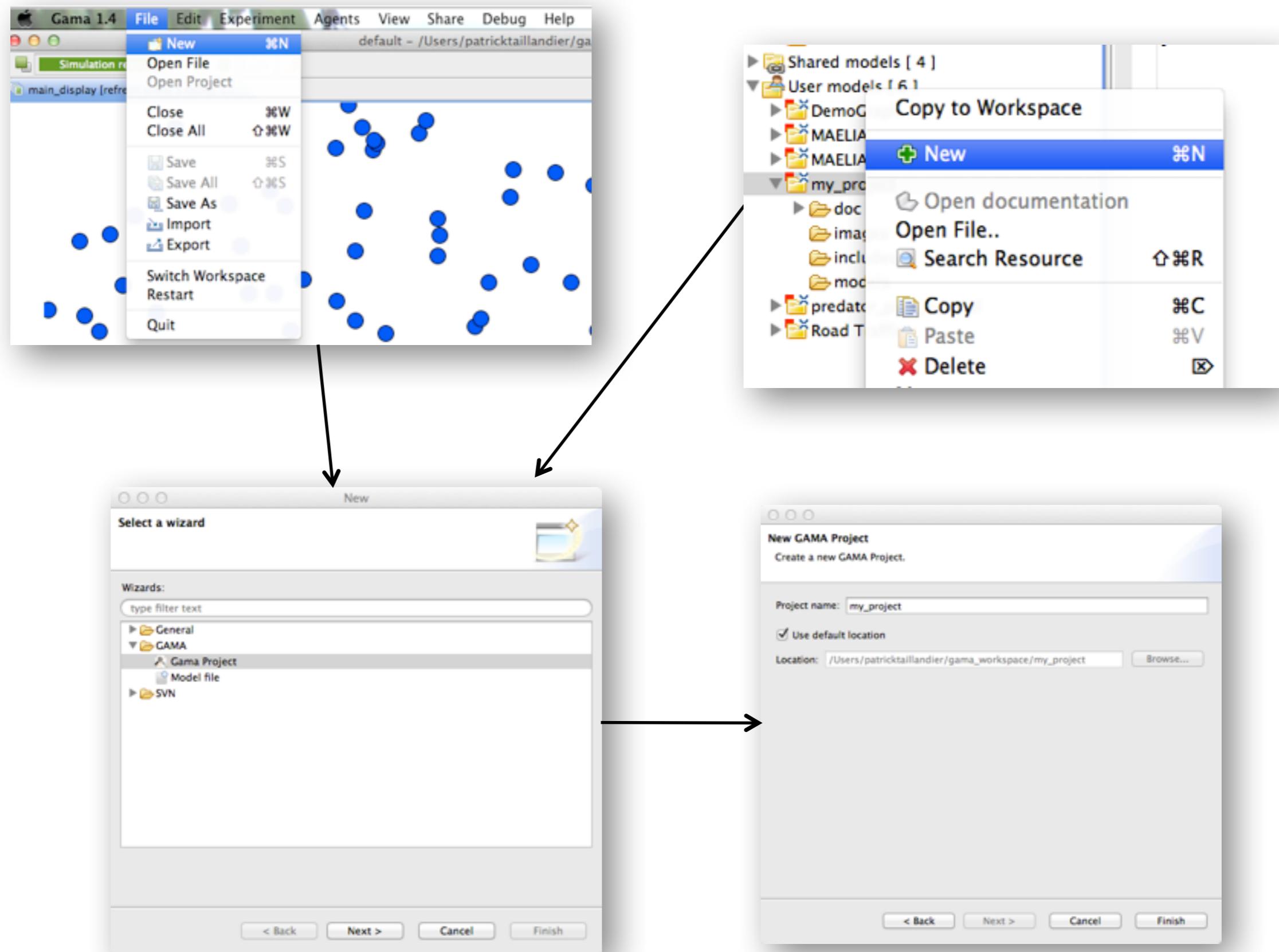
- Home
- Introduction
- Platform
  - 1. Installation and Launching
    - I. Installation
    - II. Launching GAMA
    - III. Headless Mode
    - IV. Updating GAMA
    - V. Installing Plugins
    - VI. Troubleshooting
  - 2. Workspace, Projects and Models
    - I. Navigating in the Workspace
    - II. Changing Workspace
    - III. Importing Models
    - IV. Sharing Models
  - 3. Editing Models
    - I. GAML Editor
    - II. Validation of Models
    - III. Graphical Editor
  - 4. Running Experiments
    - I. Launching Experiments
    - II. Experiments User

Definition of the structure of the model and of the basic behaviors of the people agents



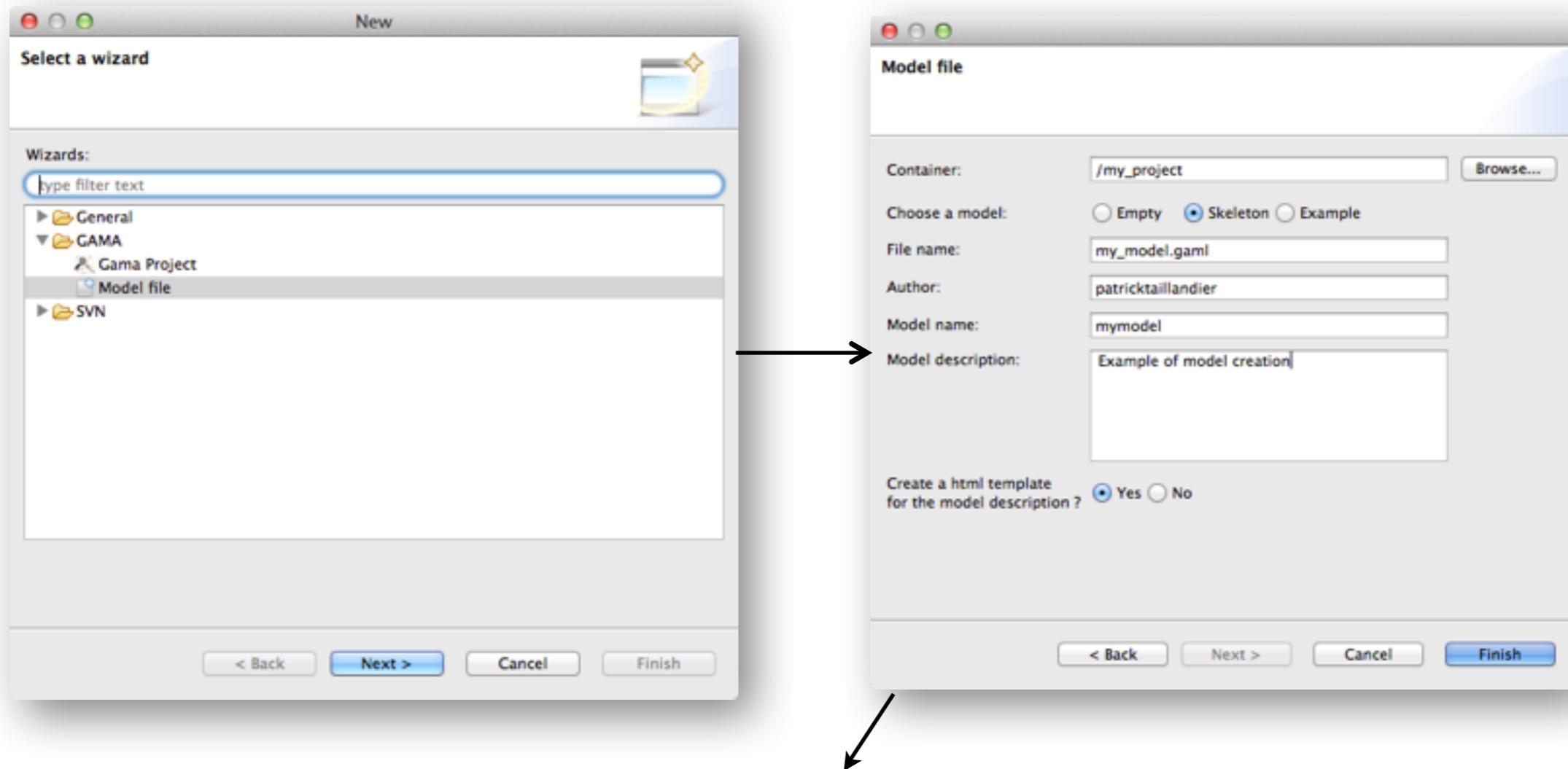
# Creation of a new project

44



# Creation of a new model

45



```
model mymodel

global {
    /** Insert the global definitions, variables and actions here */
}

experiment mymodel type: gui {
    /** Insert here the definition of the input and output of the model */
    output {
    }
}
```

# Structure of a GAMA model

## ❖ 3 types of sections:

- **Global** : global variables, actions, dynamics and initialization.
- **Species and Grid**: agent species. Several species blocks can be defined.
- **Experiment** : simulation execution context, in particular inputs and outputs. Several experiment blocks can be defined.

```

model my_model

global {
    /** Insert the global definitions,
     * variables and actions here
    */
}

species my_species{
    /** Insert here the definition of the
     * species of agents
    */
}

experiment my_model type: gui {
    /** Insert here the definition of the
     * input and output of the model
    */
}

```

2 ways to write commentaries (texts that are not just part of the model but here for information purpose):

- //... : for one line. Example : //this is a commentary
- /\* ... \*/ : can be used for several lines. Example : /\* this is as well a commentary \*/

# species and grid block: Species and Grid

- To define a species, 4 main elements can be defined :
  - the internal state of the agents (attributes - variables)
  - their initialisation (init)
  - their behaviors (methods)
  - their display (aspects)

```
model my_model

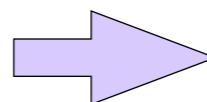
global {
}

species my_species{
}

experiment my_model type: gui {
}
```

## Model : People species

People
+speed: float
+is_infected: bool
+target: point
+staying_counter
+stay()
+move()
+infect()



Species name  
 species people {  
 }

# species block: Species definition – skills

- An agent can have *skills*
- Un *skill* is a module integrating variables and actions coded in Java

```
species people skills:[moving]{  
}  
list of skills given to  
the agents
```

The *people* agents will have some supplementary variables (*speed*, *heading*, *destination*) and actions (*follow*, *goto*, *move*, *wander*)

## Color guide:

statement
statement facet
operator
element (species, variable...) defined by the modeler
reference to an element defined by the user or pre-defined
built-in value (int, float, logic)
other

# species block: Species definition – variables 1/3

- ❖ Variable definition : ***variable type*** + *nom*
  - ***type***: int, float, string, bool (boolean, can be either *true* or *false*), point, list, pair, map, file, matrix, agent species, rgb (Red, Green, Blue - color), graph, path...
  - Optional attributes:
    - ***<-*** : initial value,
    - ***update*** : value computed at each simulation step
    - ***function or -> + {..}*** : value computed each time the variable is called
    - ***min*** : min value
    - ***max*** : max value
- ❖ GAMA allows to define **local variables**, i.e. variables that just exist inside a block. These variables are deleted from the computer memory at the end of the block : ***variable type*** + *nom* ***<- init\_value;***

All GAMA agents are provided with some built-in variables :

- ***name*** (string)
- ***shape*** (geometry) 
- ***location*** (point) : centroid of its shape

## Model : People attributes

```
species people skills:[moving]{  
    float speed <- (2 + rnd(3)) #km/#h;  
    bool is_infected <- false;  
}
```

The symbol # allows to precise the unity of a value

The operator **rnd(an\_int)** returns a random value between 0 and an\_int ([0, an\_int])

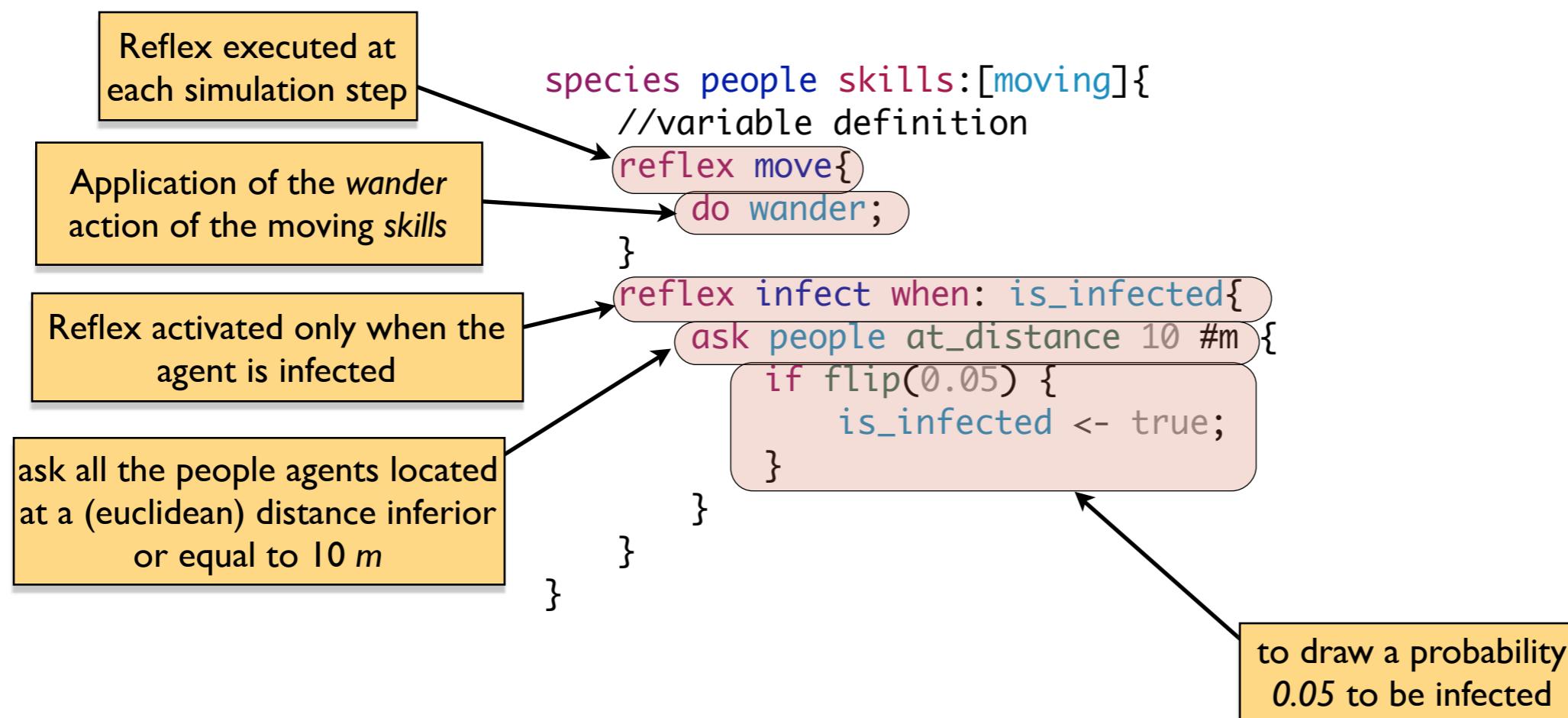
### Color guide:

statement  
statement facet  
operator  
element (species, variable...) defined by the modeler  
reference to an element defined by the user or pre-defined  
built-in value (int, float, logic)  
other

# species block: Species definition – reflex 3/3

- ❖ A reflex is a statement block that will be executed if its attached condition is checked
- ❖ **reflex reflex\_name when: execution\_condition {...}**
- ❖ The **when** facet is optional: if this facet is not defined, the reflex is executed at each simulation step.

## Model : People Reflex



# species block: Species definition – reflex 3/3

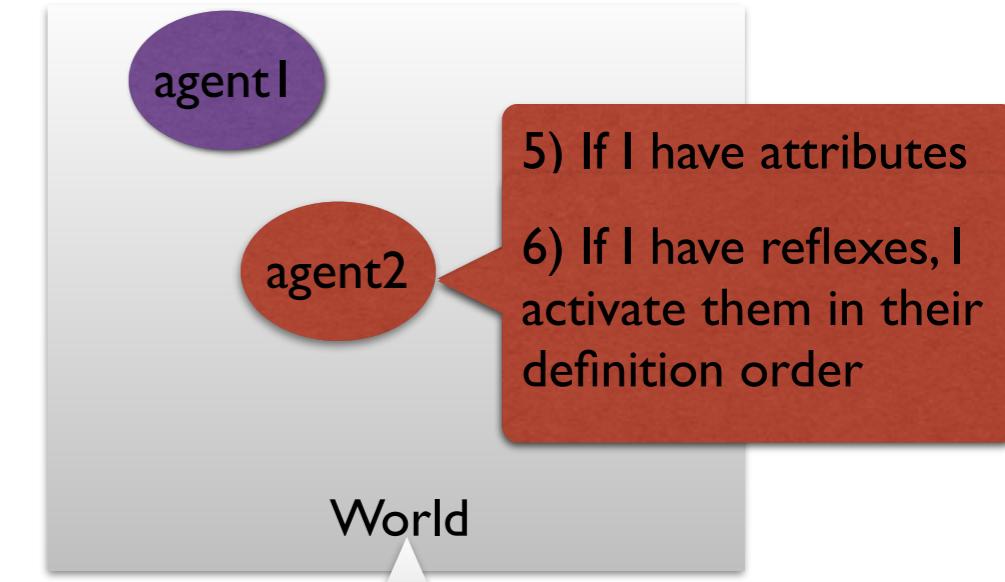
- The basic scheduler of GAMA works as follows:

- GAMA activates the world agent (global) then all the other agents according to their order of creation
- When an agent is executed, first its update its attributes (facet **update** of the attributes), then it activates its reflexes in their definition order

- Of course the Scheduler can be easily tuned through the GAML language:

- modification of the order of activation of the agents (than can be dynamic)
- Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action....)

3) If I have attributes with a  
4) If I have reflexes, I activate them in their definition order



2) If I have reflexes, I activate them in their definition order

1) If I have attributes with a *update* facet, I compute their new value in their definition order

**Note:** GAMA offers some specific control architectures (finite state machine, task-oriented architectures...) that can be added to species

# species block: Species definition – aspect

❖ An *aspect* represent A possible display for a species of agents : ***aspect aspect\_name {...}***

❖ In an *aspect* block, it is possible to display (as layers) :

- a geometry/shape: for example, the agent shape
- an image: for example, an icon
- a text

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
```

## Model : aspect circle of the people agents

```
species people {
    ...//variable and reflex definition
```

```
    aspect circle {
        draw circle(10) color:is_infected ? #red : #green;
    }
}
```

This aspect allow to display each people agent as a red or green (depending on *is\_infected*) circle of radius 10m

The symbol # allows also to define a color

# global block

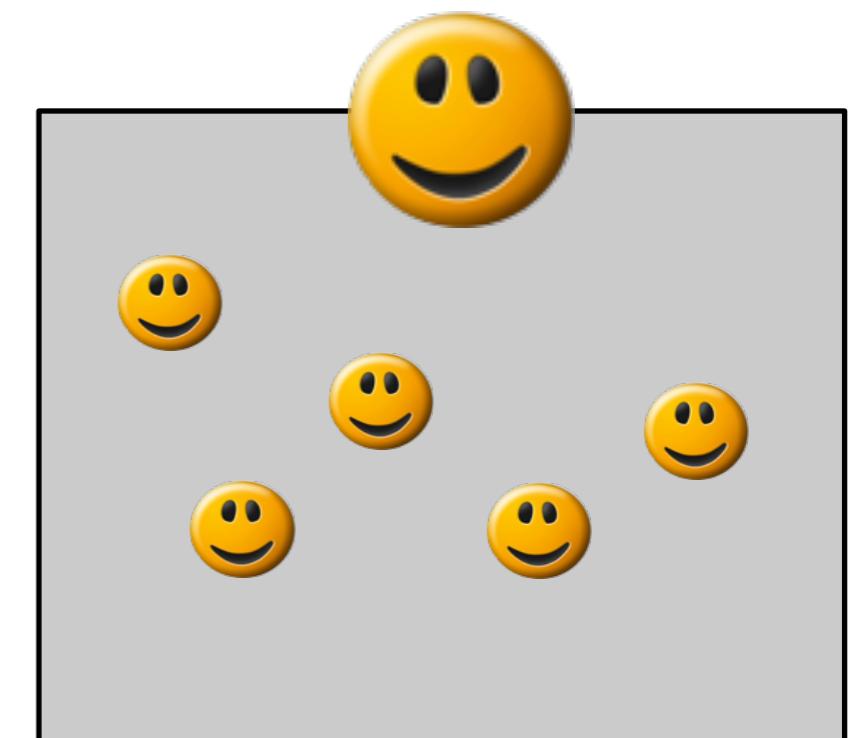
## ❖ Global block

- Define a specific agent (called world)
- Represent all that is global to the model: variables, actions, reflexes....
- Initialize the simulation (init block): when the experiment button is pushed, the world agent is created and then activates its init block
- The geometry of the world agent (shape) is a rectangle that define the size of the environment in which all the agents are localized. By default the shape of the world agent is a square of 100m size
- Define the nature of the environment: torus or not (by default, not torus)

```
model my_model
  global {
  }

  species my_species{
  }

  experiment my_model type: gui {
  }
```



## Model :World attributes

```
model my_model
  global {
  }
species my_species{
}
experiment my_model type: gui { }
```

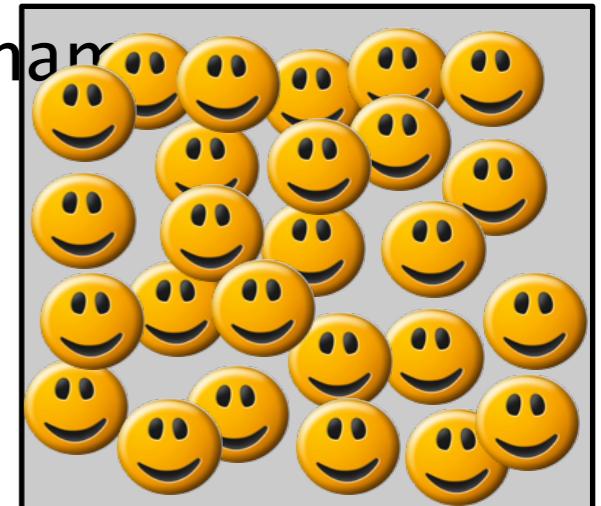
```
global {
  int nb_people <- 2147;
  int nb_infected_init <- 5;
  float step <- 1 #mn;
  geometry shape<-square(1500 #m);
}
```

Built-in variable:  
define the duration  
of simulation step

Define the geometry of the world: a  
square of 1500m side size

## ❖ Creation of agents : use of the statement: **create species\_name**

- number : number of agent to create (int, by default, 1)
- from : GIS or Raster file (string ou file)
- with : allows to give initial values to the agent variables
- returns: list of created agents



## Model :World init

```
global {
    // world variable definition

    init{
        create people number:nb_people;
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Create *nb\_people*  
people agents

Ask *nb\_infected\_init* people  
(randomly chosen) to be  
infected

**Note:** By default,  
agents are **randomly**  
placed in the  
environment (except  
when the facet **from:**  
+ S/G is used)

**Note:** The *create* statement can be used in all  
init/actions/reflex of the model, not only in the  
global section

# experiment block

- ❖ An experiment block define an execution context of simulations
- ❖ Several experiment blocks can be defined
- ❖ Define by : **experiment xp\_name type:** gui/batch  
 {...}
  - gui : one simulation with graphical interface.
  - batch : experiment plan: set of simulations without graphical interface

```

model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}

```

## Model : Main experiment

```

experiment main_experiment type: gui {
}

```

# experiment block: parameter definition

## ❖ Parameter :

**parameter legend var: var\_name category: my\_cat;**

- Allow to give to the user the possibility to define the value of a global variable
- legend: string to display
- var\_name: reference to a global variable
- category: string (use to better organize the parameters) – optional

```
model my_model

global {
}

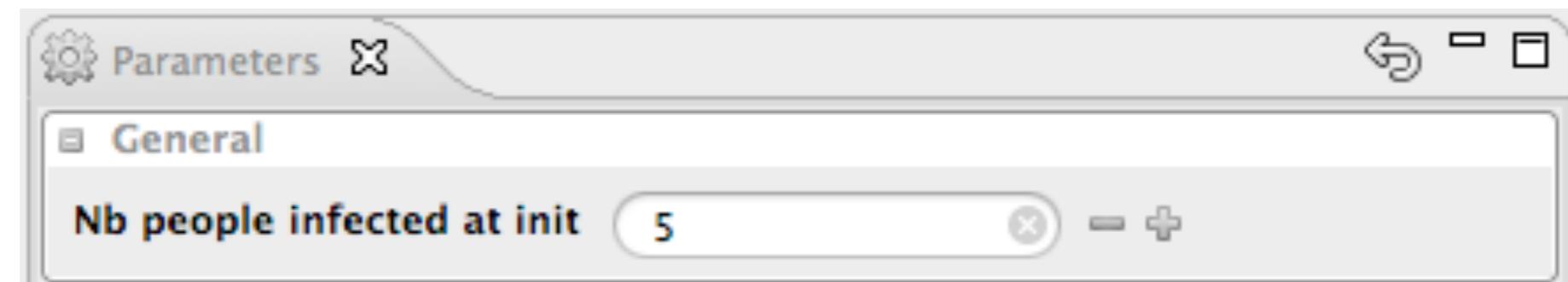
species my_species{

experiment my_model type: gui {
}
```

## Model : Main experiment

```
experiment main_experiment type:gui{
    parameter "Nb people infected at init" var: nb_infected_init min: 1 max: 2147;
```

```
output {
}
```



It is possible to define here the min and max values of a parameter

# experiment block: output definition

- ❖ The *output* block has to be defined in an *experiment* block
- ❖ It allows to define displays:
  - A refreshing rate can be defined: facet **refresh\_every: nb (int)**
  - Each *display* can contain different displays:
    - list of agents : **agents layer\_name value: agents aspect: my\_aspect;**
    - Agent species (all the agents of the species) : **species my\_species aspect: my\_aspect**
    - Grids: optimized display of grids: **grid grid\_name lines: my\_color;**
    - Images: **image layer\_name file: image\_file;**
    - Texts: **texte layer\_name value: my\_text;**
    - Charts: see later

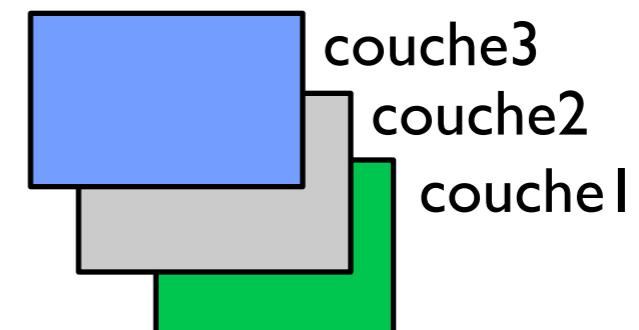
```
model my_model

global {
}

species my_species{

}

experiment my_model type: gui {
}
```



## Model : experiment

```
experiment main_experiment type: gui {
  ... //parameter definition

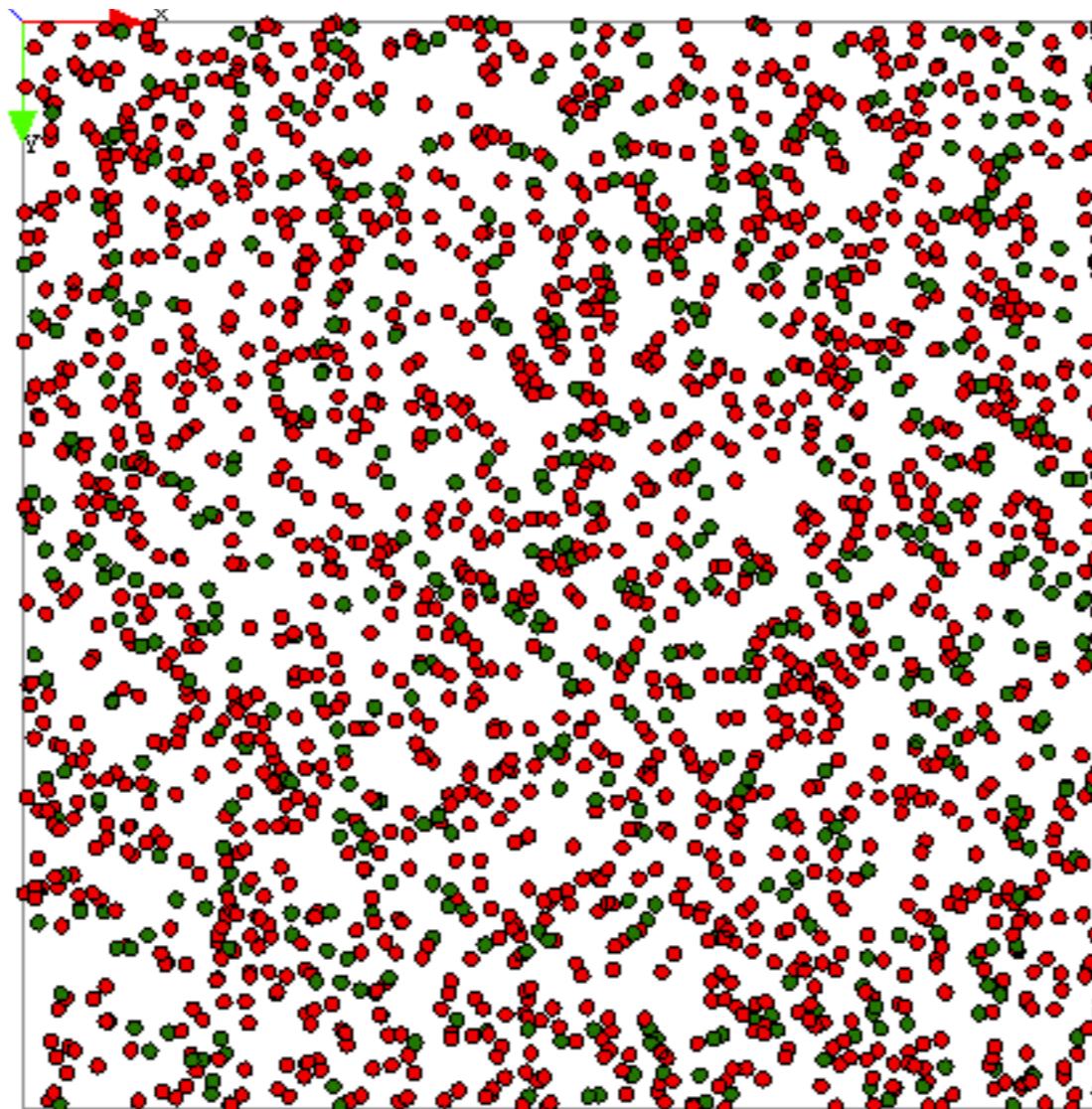
  output {
    display map type: opengl{
      species people aspect:circle;
    }
  }
}
```

Allows to use optimized  
opengl displays

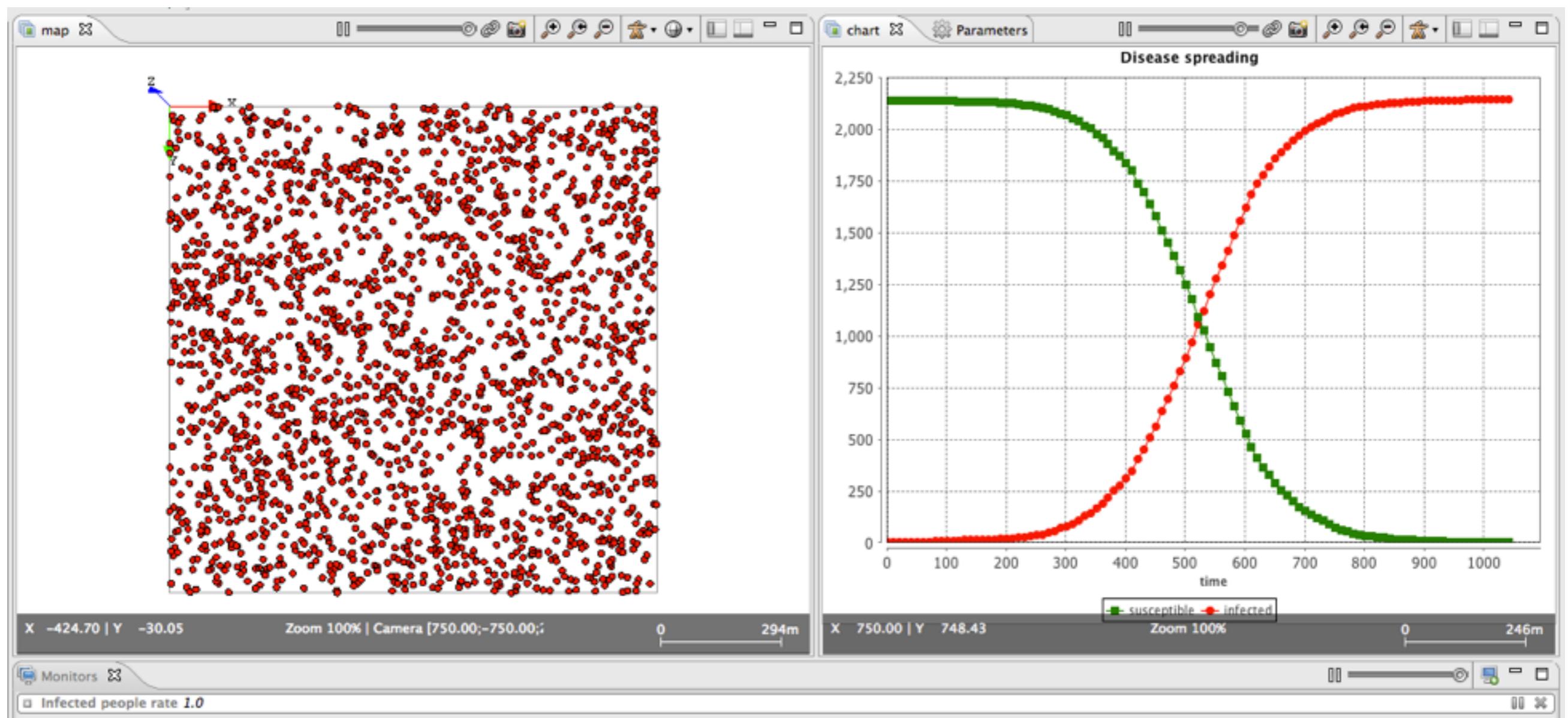
Layer displaying the people  
agents with their circle aspect

**Note:** in a *display*, the  
display order of the layer  
follows the layer definition

Model: Time to test the first version  
of the model !



## Definition of outputs to follow the system evolution and of a stoping condition



## Model :World attributes

```
model my_model
  global {
  }
  species my_species{
  }
  experiment my_model type: gui {
  }
```

```
global{
  //... other attributes
  int nb_people_infected <- nb_infected_init update: people count (each.is_infected);
  int nb_people_not_infected <- nb_people - nb_infected_init update: nb_people - nb_people_infected;
  float infected_rate update: nb_people_infected/nb_people;
  //... init
}
```

infected rate: number of infected people divided by the number of people

count the number of infected agents among the people agents

people not infected: number of people - number of infected people

**Note:** The *update* facet allows to recompute the value of the variable at each simulation step

## Model : Reflex *end\_simulation* of the *world* agent

```
global {  
    //.. variable and init definition  
  
    reflex end_simulation when: infected_rate = 1.0 {  
        do pause;  
    }  
}
```

Reflex activated when the infected rate is equal to 1.0 (i.e. all people agents are infected) and that pause the simulation

# experiment block: monitor definition

- ❖ A monitor is an output allowing to display the current value of an expression
- ❖ The data to display have to be defined inside the **output** block:  
**monitor legend value: value**

```
model my_model

global {
}

species my_species{

}

experiment my_model type: gui {
}
```

## Model : monitor display

```
experiment main_experiment type:gui{
    //...parameters
    output {
        monitor "Infected people rate" value: infected_rate;

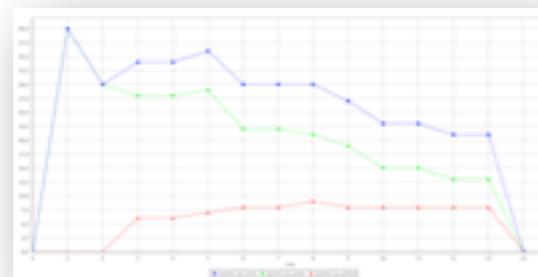
    //...display
    }
}
```



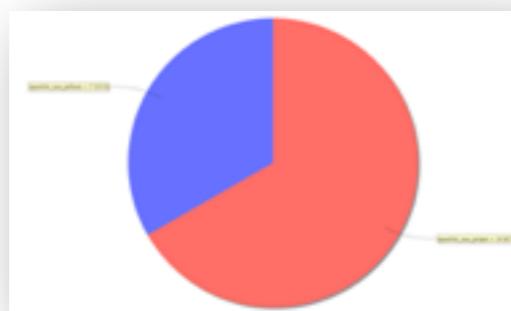
# experiment block: chart definition

- ❖ GAMA allows to display several type of charts :

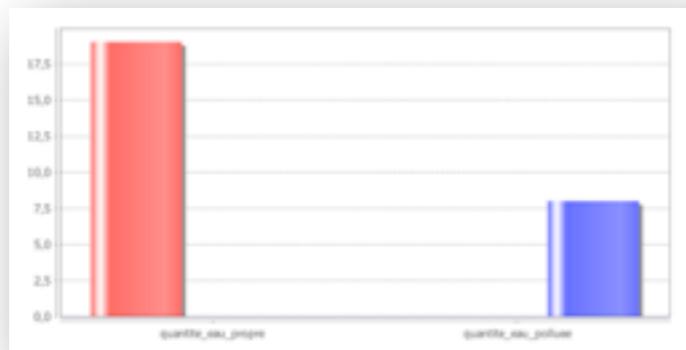
- Series



- Pie



- *Histogram*



- *XY chart*

```

model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}

```

# experiment block: chart definition

- ❖ A chart is a layer in a display: **chart legend type:**  
*chart\_type*
- ❖ The data to display have to be defined inside the **chart** block:  
**data legend value: value color: color**

```
model my_model

global {
}

species my_species{

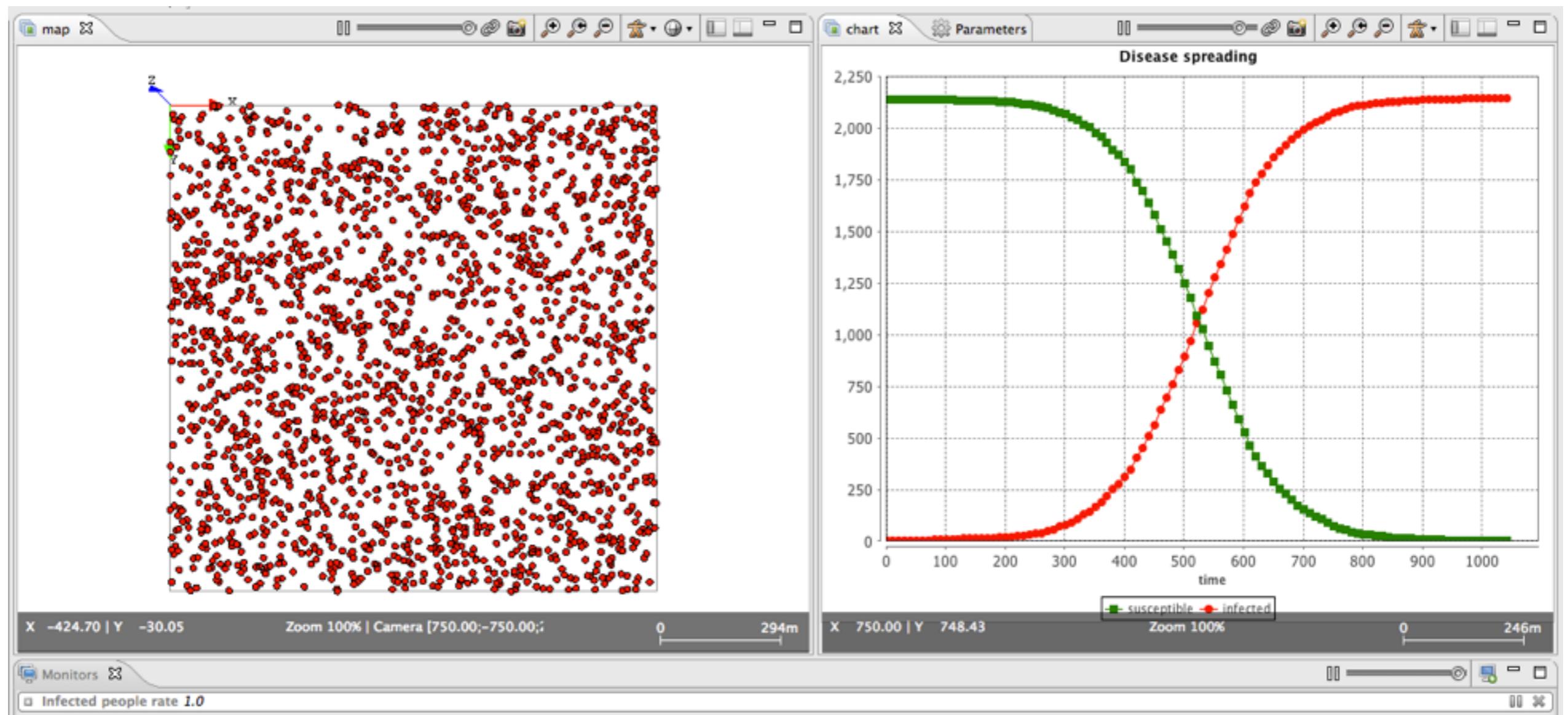
experiment my_model type: gui {
}
```

## Model : chart display

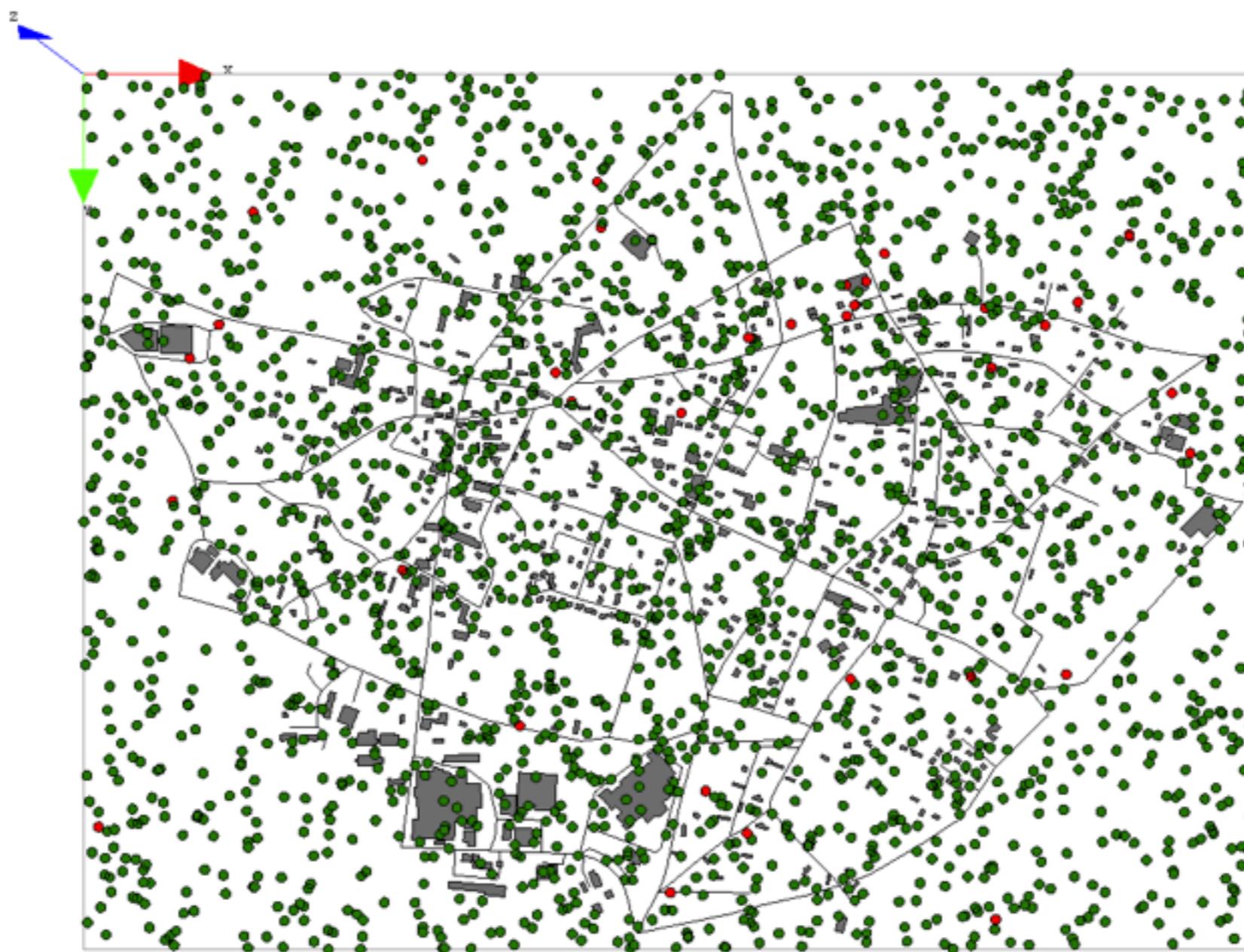
```
experiment main_experiment type:gui{
    //...parameters
    output {
        //...display and monitors
        display chart_display refresh_every: 10 {
            chart "Disease spreading" type: series {
                data "susceptible" value: nb_people_not_infected color: #green;
                data "infected" value: nb_people_infected color: #red;
            }
        }
    }
}
```

Display refreshed every  
10 simulation steps

## Model: Time to test the second version of the model !



Loading of Geographical vector data to  
create buildings and roads



## Model : road and building agents

```
model my_model

global {

species my_species{

}

experiment my_model type: gui {
```

```
species road {
    aspect geom {
        draw shape color: #black;
    }
}
```

```
species building {
    aspect geom {
        draw shape color: #gray;
    }
}
```

## Model :Add a house to people agent

```
species people skills:[moving]{  
    float speed <- (2 + rnd(3)) #km/#h;  
    bool is_infected <- false;  
    building my_house;  
}
```

It is possible to use species name  
to define new variable type

## Model :World attributes

```
global{  
  //... other attributes  
  file roads_shapefile <- file("../includes/routes.shp");  
  file buildings_shapefile <- file("../includes/batiments.shp");  
  geometry shape <- envelope(roads_shapefile);  
  //... init  
}
```

Shapefile of the roads

model my\_model

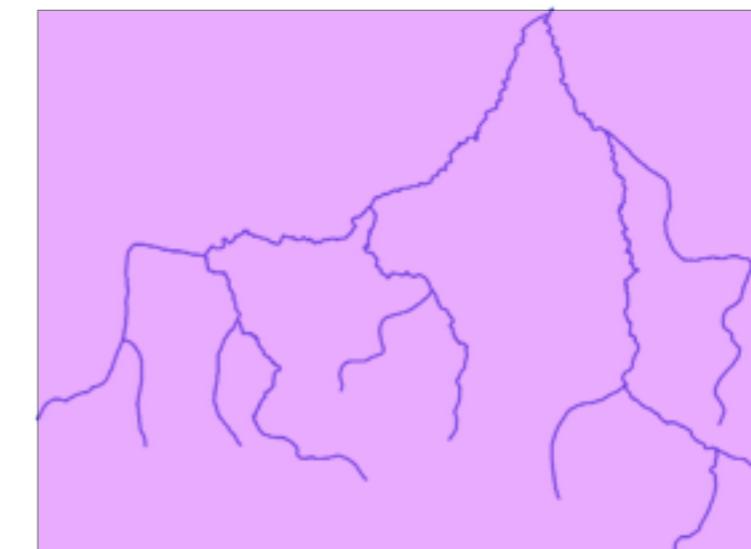
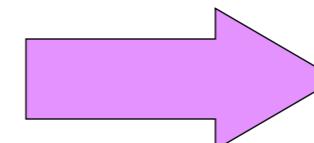
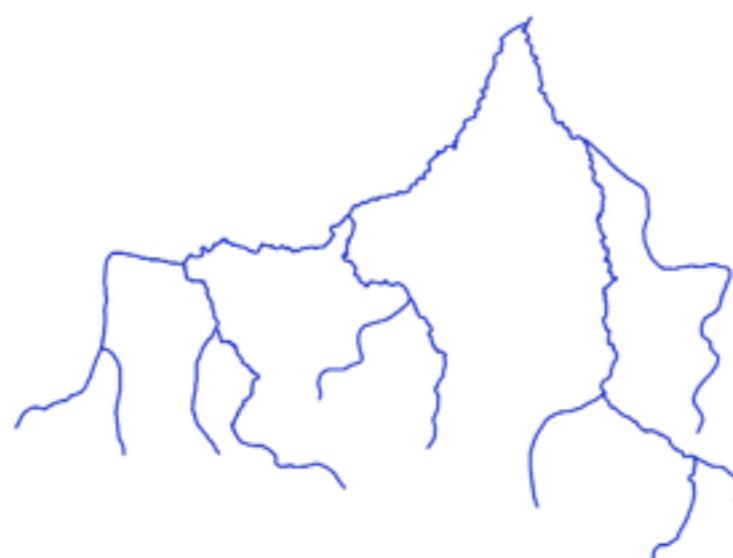
global {  
}

species my\_species{  
}

experiment my\_model type: gui {  
}

Shapefile of the buildings

Computation of the world  
geometry from the envelope  
of the road shapefile



Environment

# Block global : agent creation from GIS data

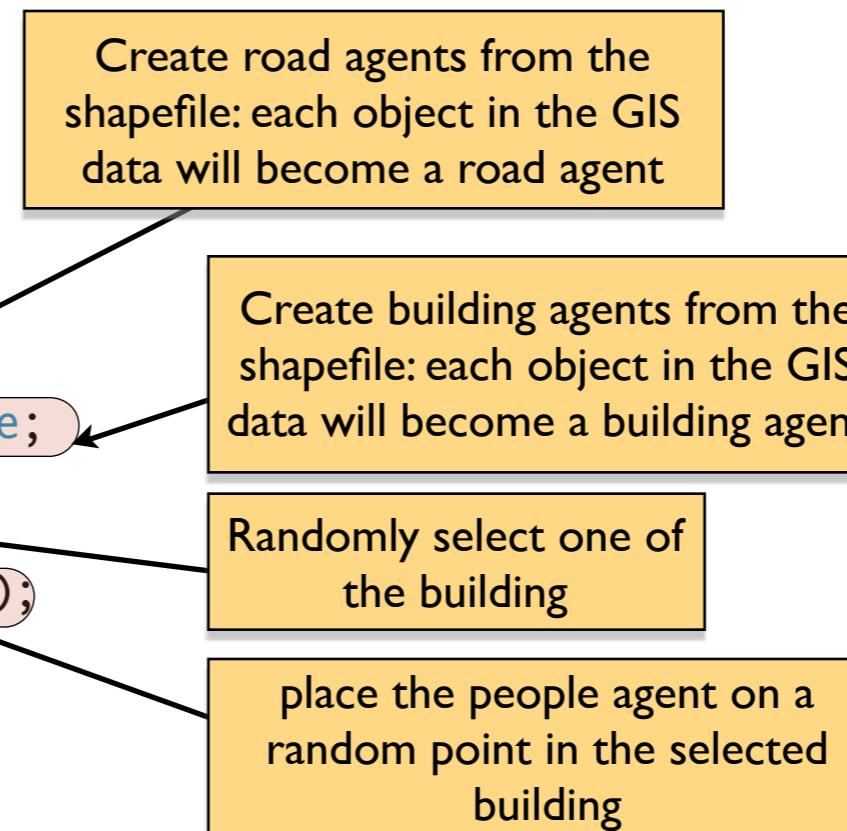
## ❖ Creation of agents : use of the statement: **create species\_name +**

- number : number of agent to create (int, by default, 1)
- from : GIS or Raster file (string ou file)
- with : allows to give initial values to the agent variables
- returns: list of created agents

## Model :World init

```
global {
    // world variable definition

    init{
        create road from: roads_shapefile;
        create building from: buildings_shapefile;
        create people number:nb_people {
            my_house <- one_of(building);
            location <- any_location_in(my_house);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

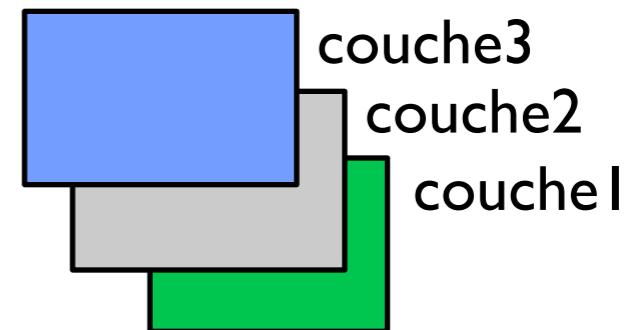


# experiment block: add the road and the building to the map

## Model : experiment

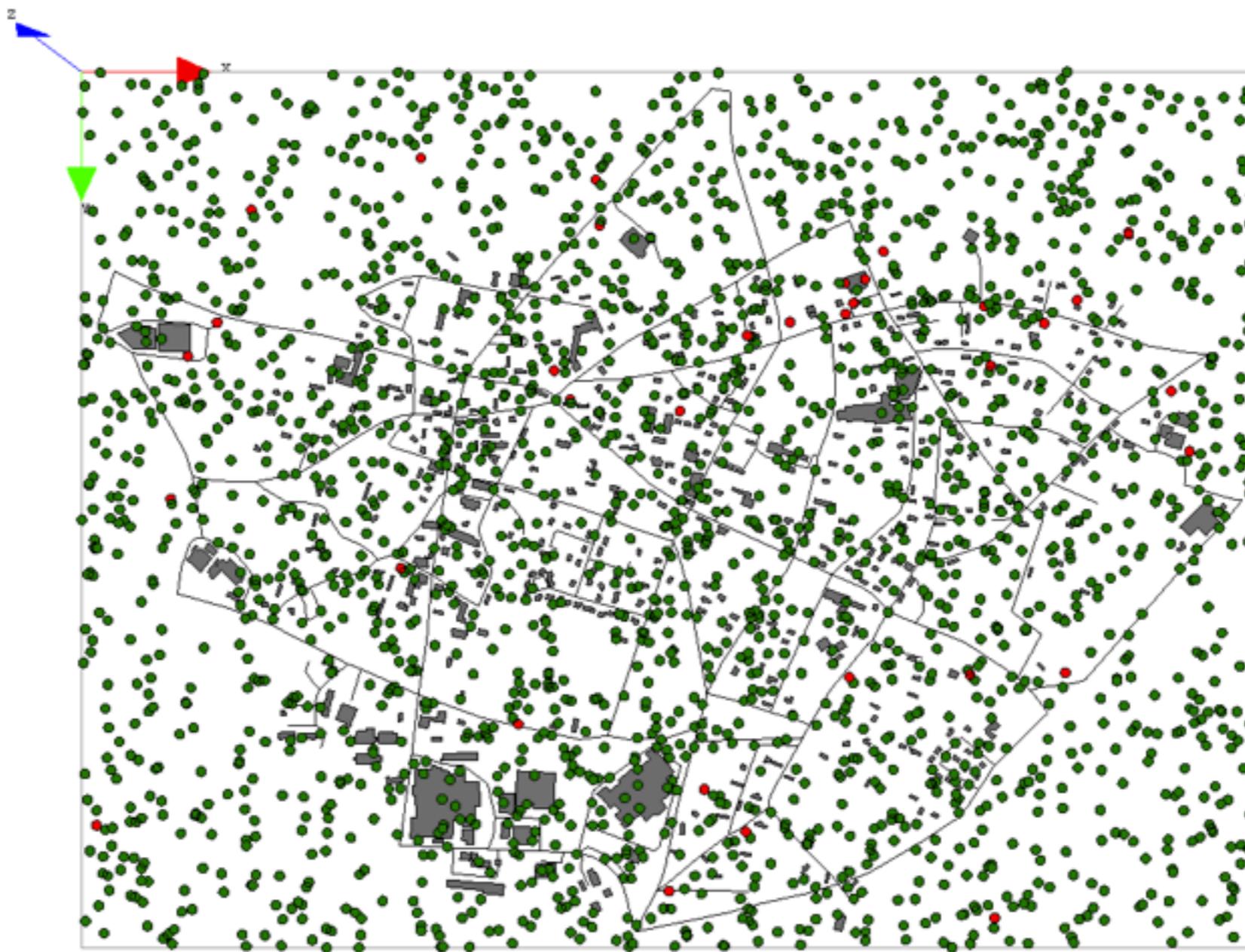
```
experiment main_experiment type: gui {  
    ... //parameter definition  
  
    output {  
        ... //monitor definition  
  
        display map type: opengl{  
            species road aspect:geom;  
            species building aspect:geom;  
            species people aspect:circle;  
        }  
        ... //chart display definition  
    }  
}
```

```
model my_model  
  
global {  
}  
  
species my_species{  
}  
  
experiment my_model type: gui {  
}
```

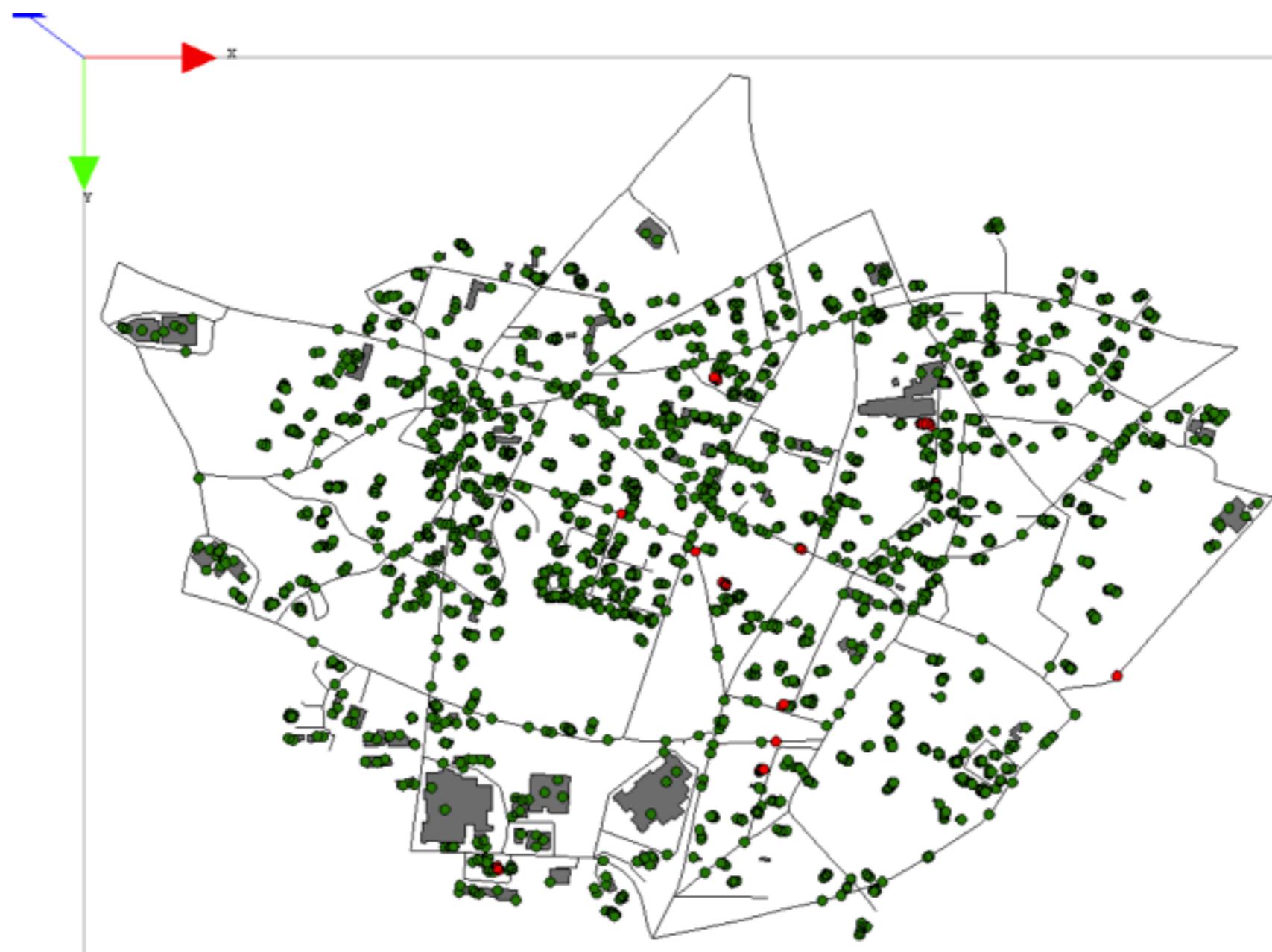


**Note:** in a *display*, the display order of the layer follows the layer definition

Model: Time to test the third version  
of the model !



People move from building to building using the road network



## Model :World attributes

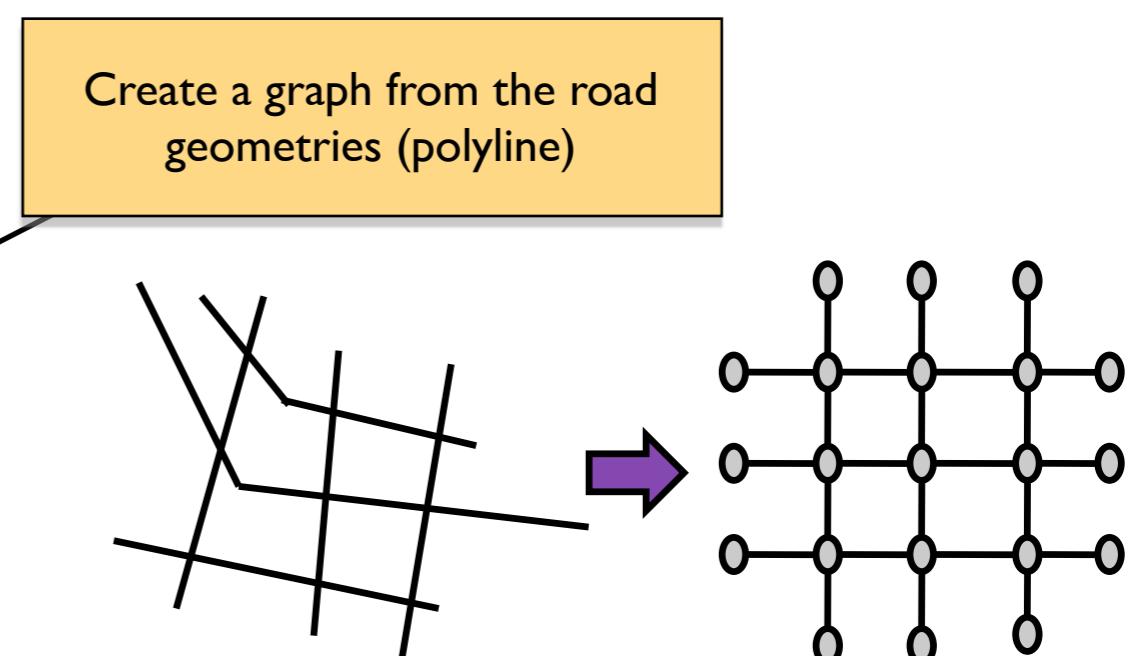
```
model my_model  
global {  
}  
species my_species{  
}  
experiment my_model type: gui {  
}
```

```
global{  
    //... other attributes  
    graph road_network;   
}  
  
//... init  
}
```

- ❖ GAMA allows to define graph variable and offers many tools to build spatial graphs

## Model :World init

```
global {  
    // world variable definition  
  
init{  
    create road from: roads_shapefile;  
    road_network <- as_edge_graph(road);  
    create building from: buildings_shapefile;  
    create people number:nb_people {  
        my_home <- one_of(building);  
        location <- any_location_in(my_home);  
    }  
    ask nb_infected_init among people {  
        is_infected <- true;  
    }  
}
```



## Model : People attributes

```
model my_model

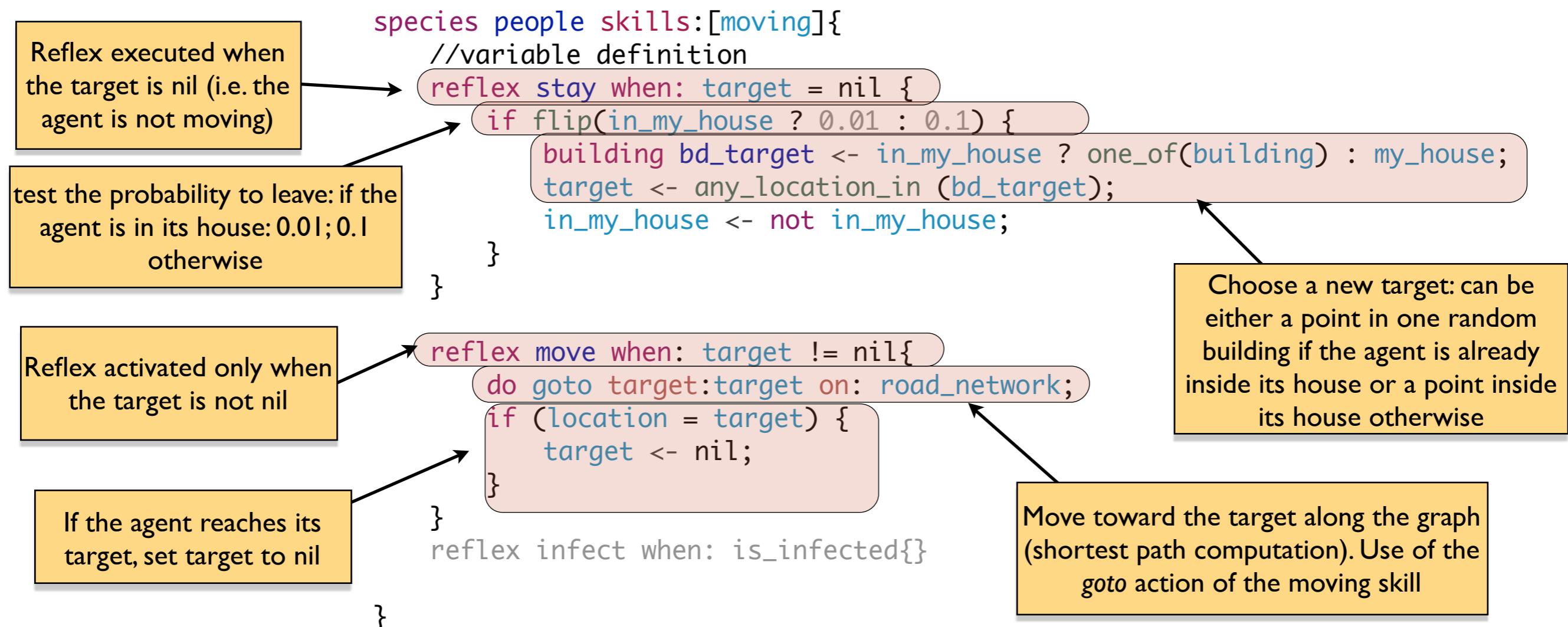
global {
}

species my_species{
}

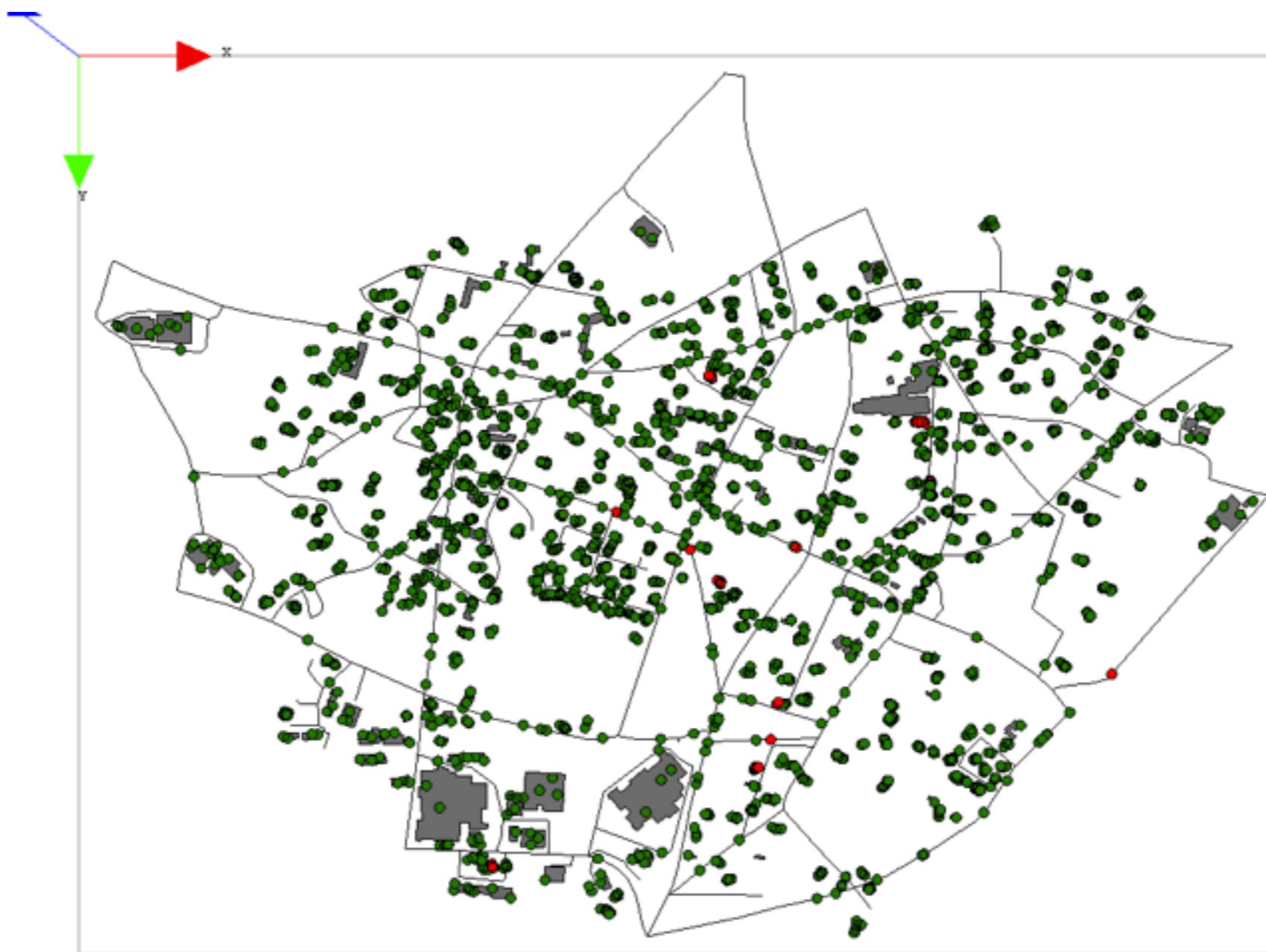
experiment my_model type: gui {
}
```

```
species people skills:[moving]{
    //...the other attributes
    point target;
    bool in_my_house <- true;
    //...
}
```

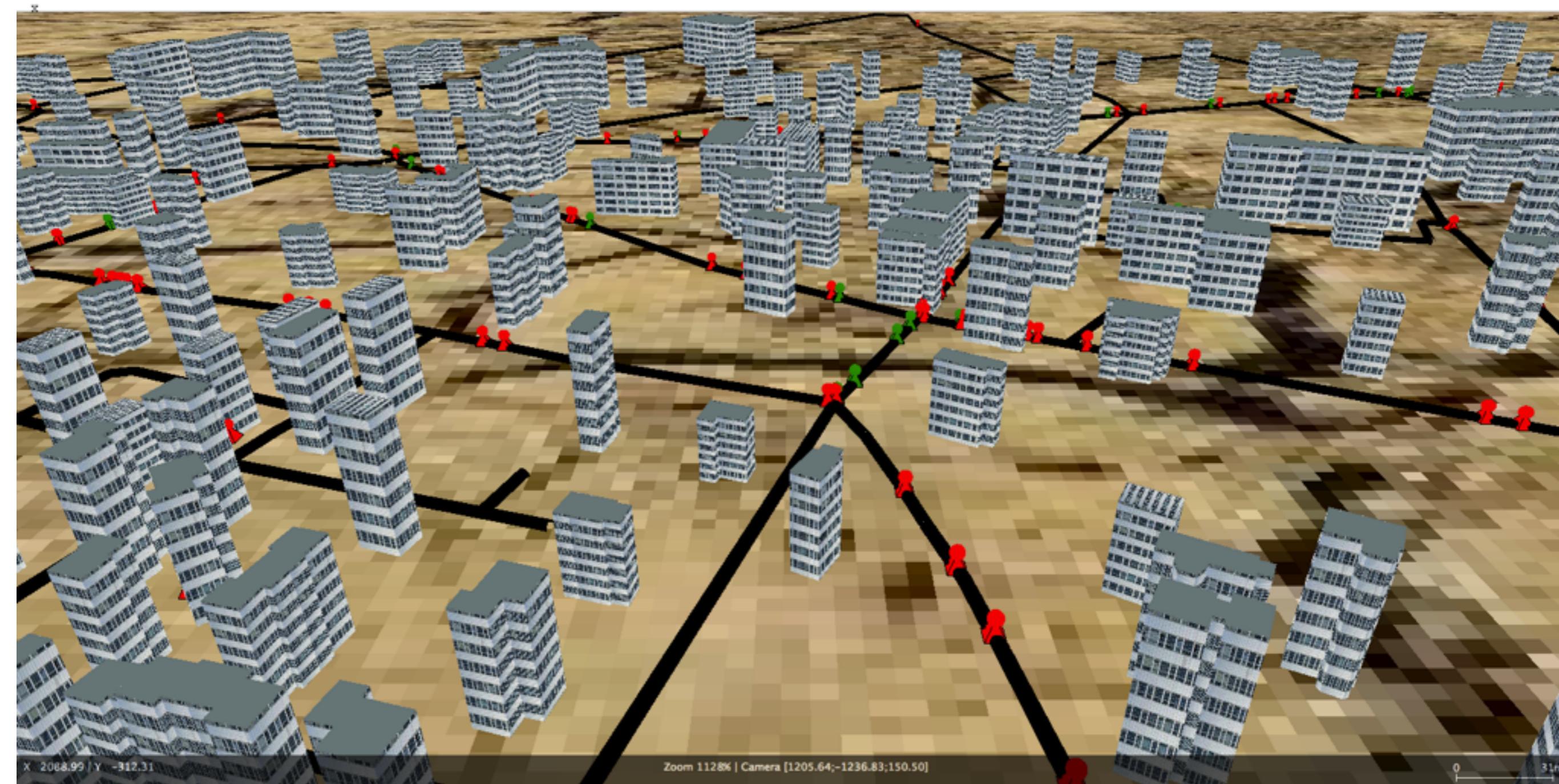
## Model : Some reflexes of people



Model: Time to test the fourth version  
of the model !



Time to prepare a new (cool) 3D display!



```
model my_model
```

```
global {  
}
```

```
species my_species{  
}
```

```
experiment my_model type: gui {  
}
```

## Model : road and building agents

Definition of new variable of type geometry that represent the geometry of the road (polyline) as tube of radius 2m

```
species road {  
    geometry display_shape <- line(shape.points, 2.0);
```

New aspect: draw the display\_shape geometry

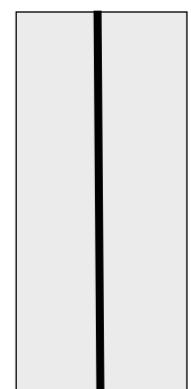
```
//....  
aspect geom3D {  
    draw display_shape color: #black;  
}
```

New variable *height*: value between 10 and 10 meters

```
species building {  
    float height <- 20#m + rnd(20) #m;
```

New aspect: draw the shape of the agent with a depth of *height* meters and texture

```
//....  
aspect geom3D {  
    draw shape depth: height texture:["../includes/texture.jpg"];  
}
```



## Model : people agents

```
species people skills:[moving]{  
  //....  
  aspect geom3D{  
    draw pyramid(5) color: is_infected ? #red : #green;  
    draw sphere(2) at: {location.x,location.y,5} color: is_infected ? #red : #green;  
  }  
}
```

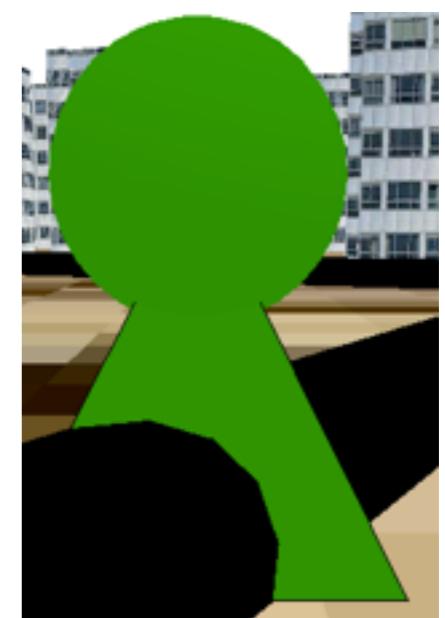
New aspect: instead of drawing a circle, draw a pyramid, with a sphere at its top

```
model my_model
```

```
global {  
}
```

```
species my_species{  
}
```

```
experiment my_model type: gui {  
}
```



# experiment block: output definition

## Model : new 3D display

```

model my_model

global {
}

species my_species{

}

experiment my_model type: gui {
}

```

```

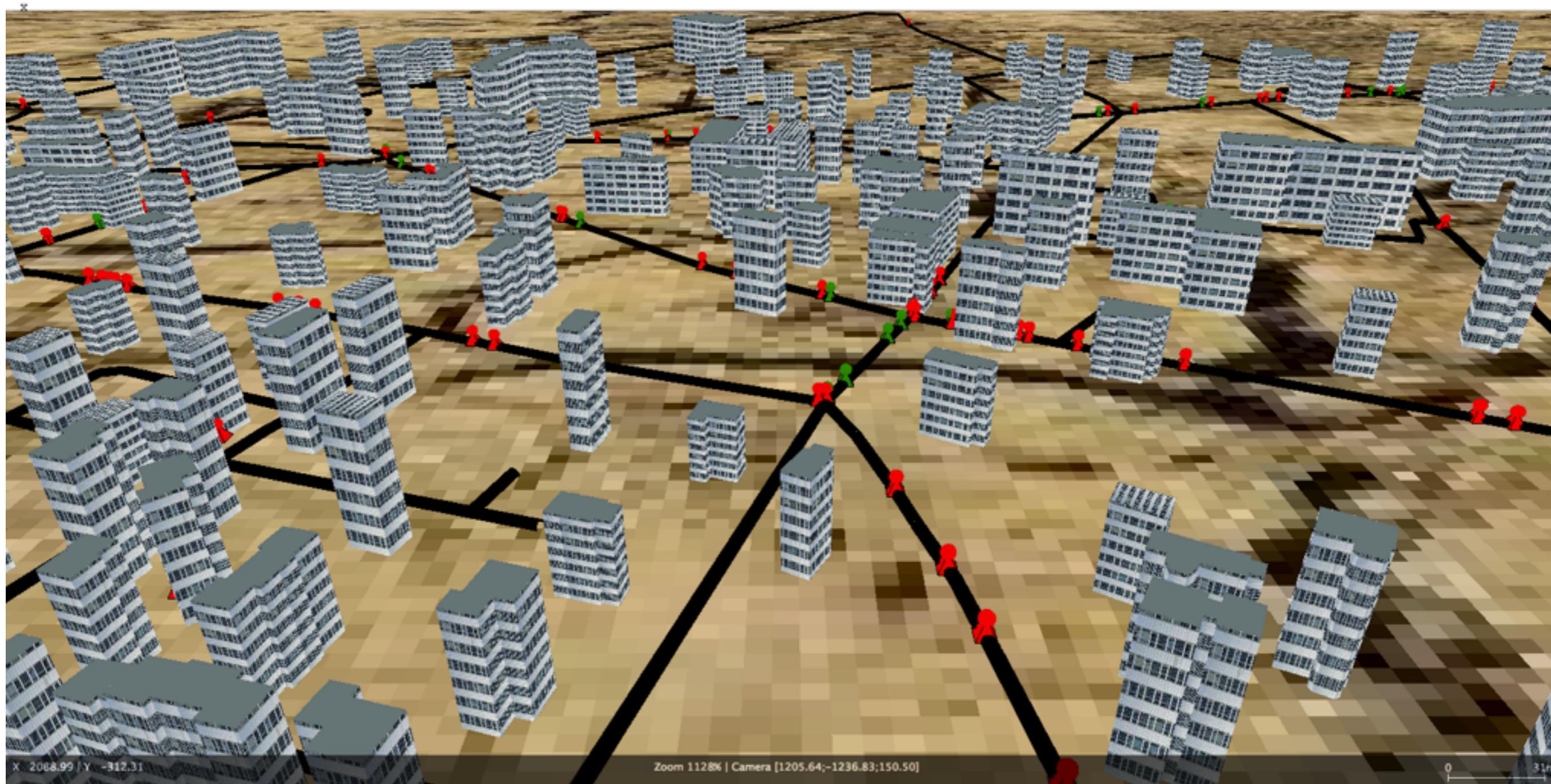
experiment main_experiment type: gui {
    output {
        // monitor and other displays
        display map_3D type: opengl ambient_light: 120 {
            image "../includes/soil.jpg" refresh: false;
            species road aspect:geom3D refresh: false;
            species building aspect:geom3D refresh: false;
            species people aspect:geom3D;
        }
    }
}

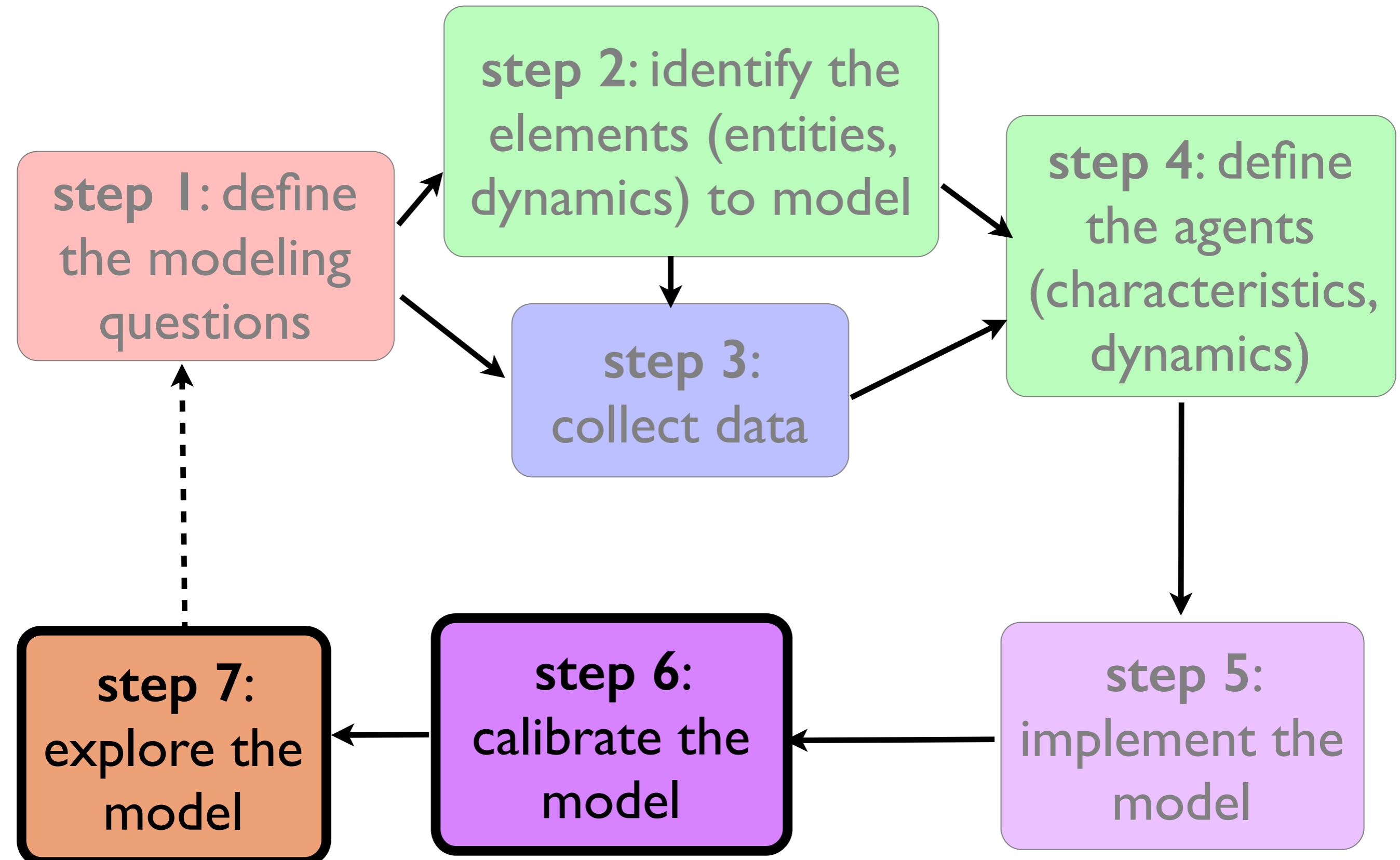
```

Add an image... just because it is cool !

the **refresh: false** facet allows to specify that a layer does not have to be refresh and allows to optimize the display

Model: Time to test the last version of the model !





- GAMA proposes several tools to explore the parameter space (exhaustive exploration, optimization)

[https://github.com/gama-platform/gama/wiki/  
G\\_BatchExperiments](https://github.com/gama-platform/gama/wiki/G_BatchExperiments)

# Conclusion

- GAMA is a generic open-source platform dedicated to the development and simulation of agent-based models
- More and more used (more than 3 000 downloads for the last versions, more than 180 research papers citing GAMA)
- In the next version of GAMA (December 2015):
  - Performance improvement
  - Bug fixes
  - Improvement of the graphic user interface
  - Co-modeling
  - Possibility to read more file formats (in particular for 3D data)
  - Better integration of BDI agent architectures