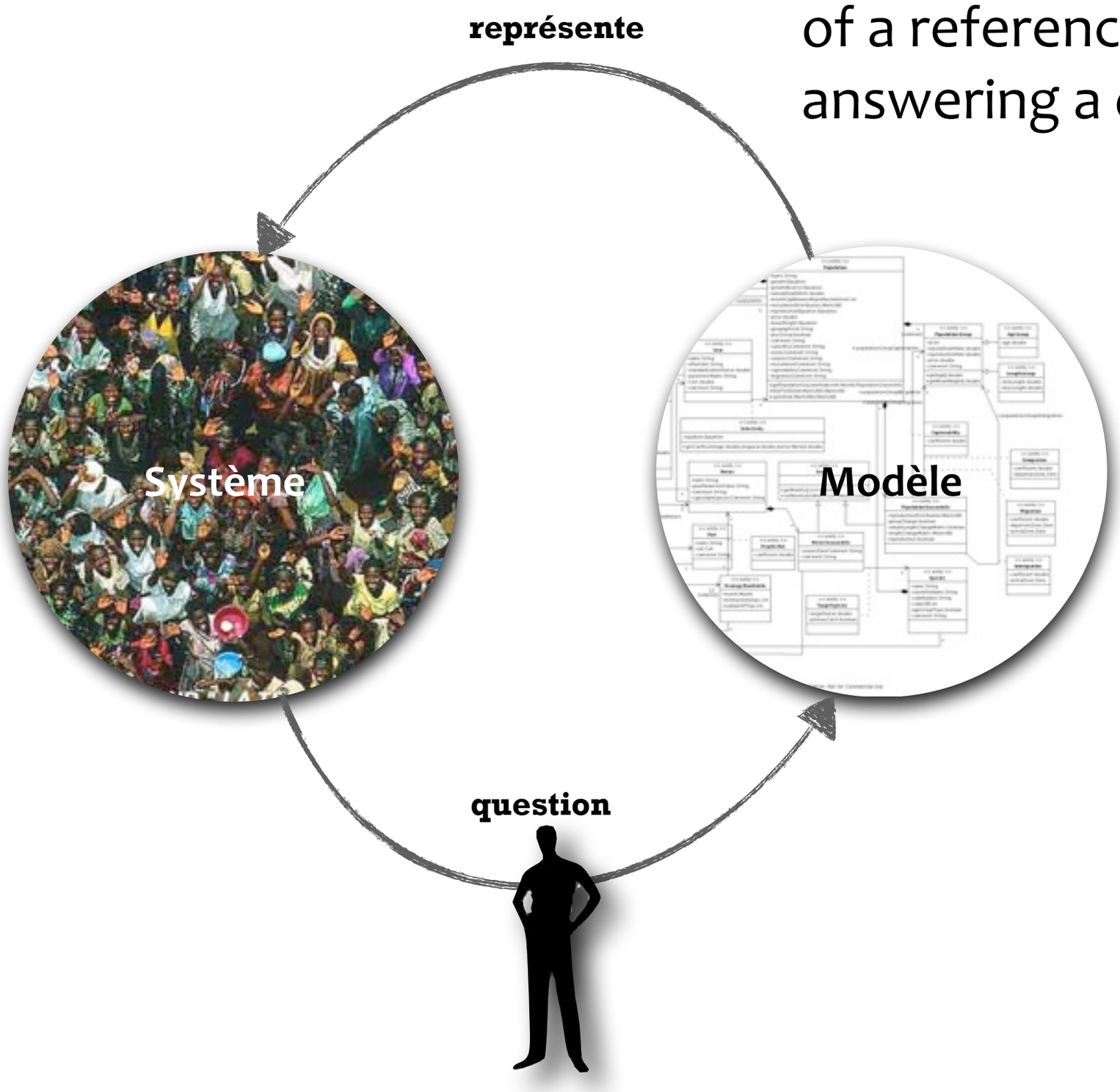
Ecole ResCom - 13 mai 2014

Patrick Taillandier

UMR 6266 IDEES, Université de Rouen / CNRS

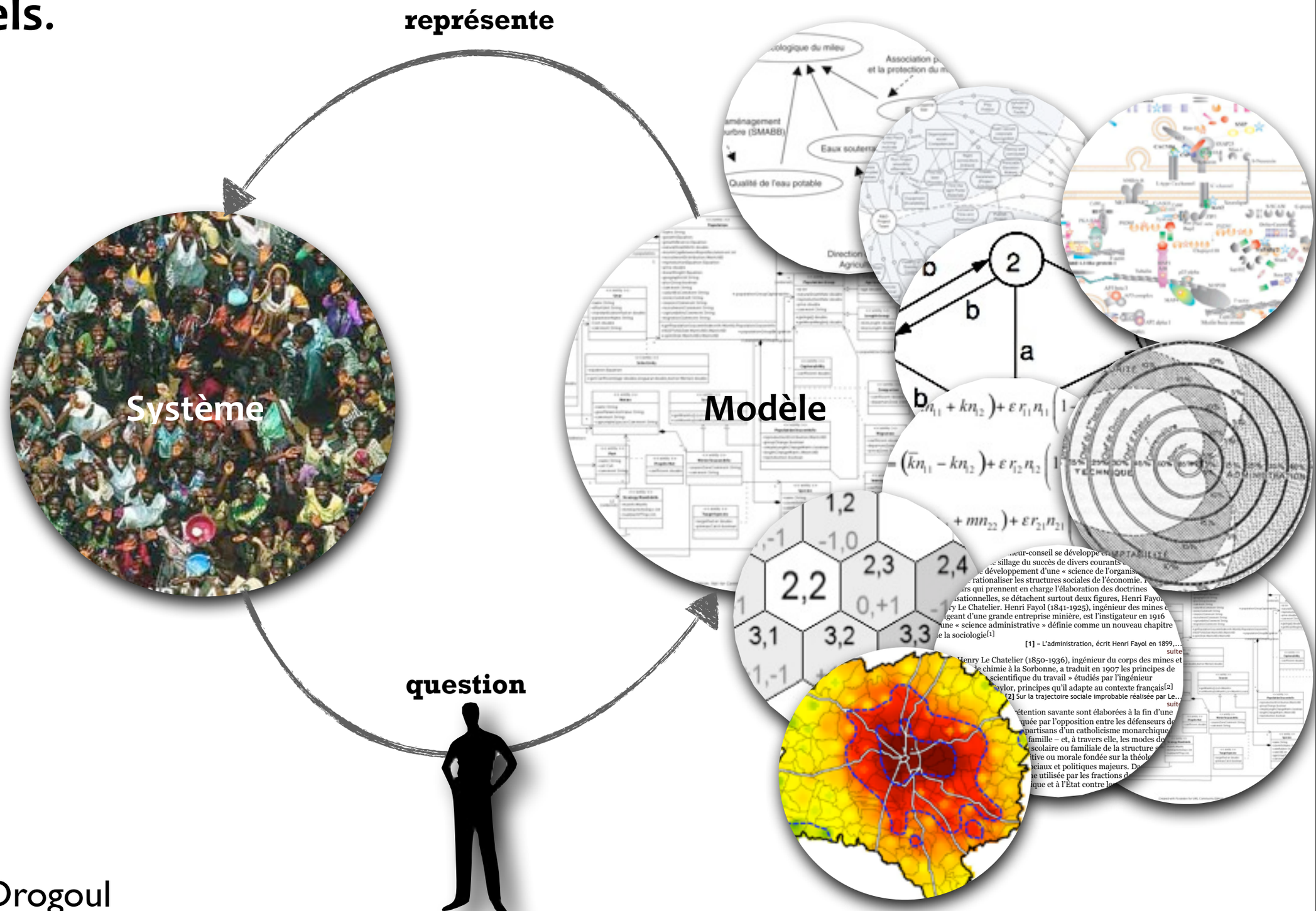
A Practical Introduction to the GAMA platform



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

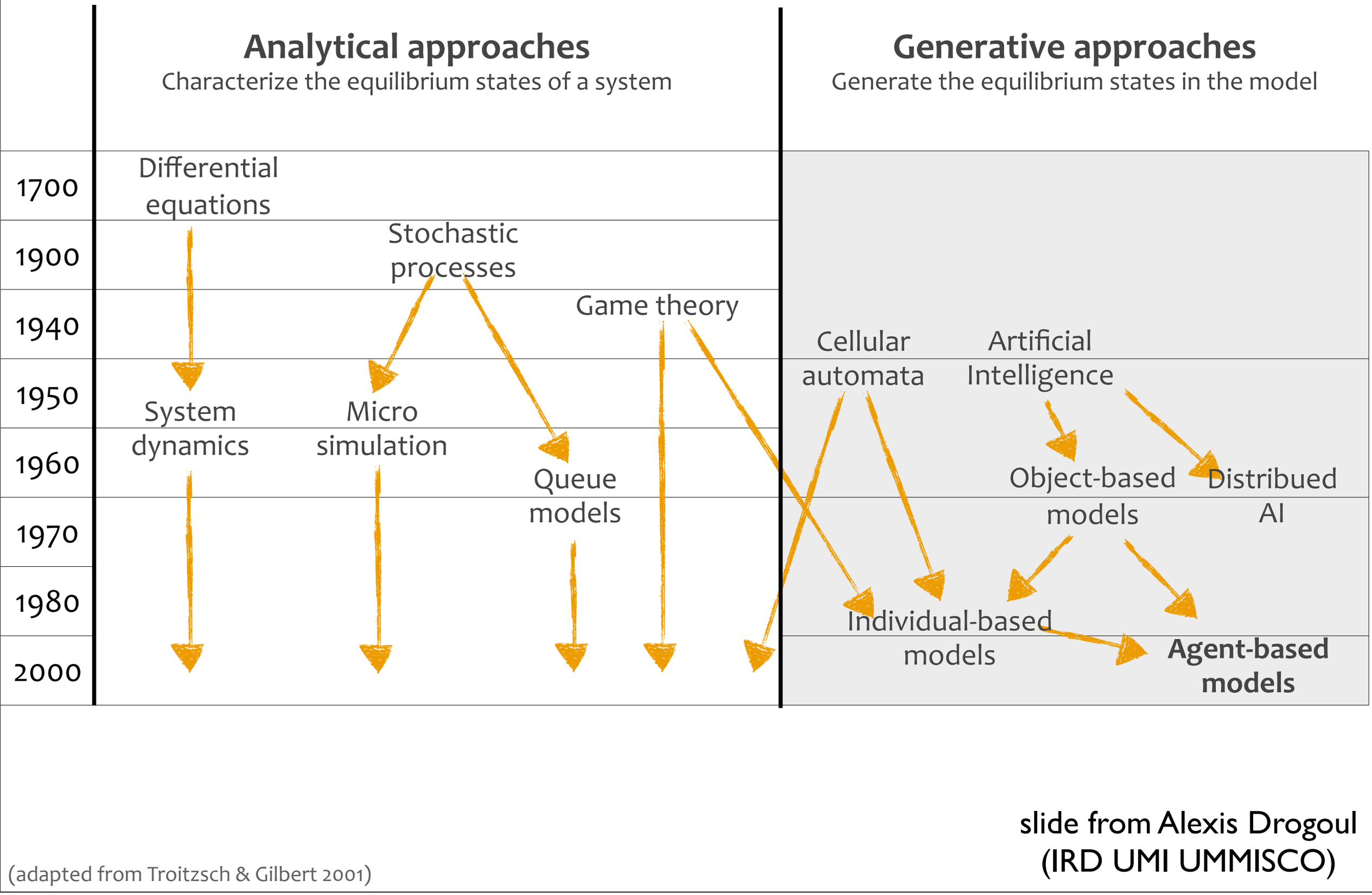
slide from Alexis Drogoul
(IRD UMI UMMISCO)

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



slide from Alexis Drogoul (IRD UMI UMMISCO)

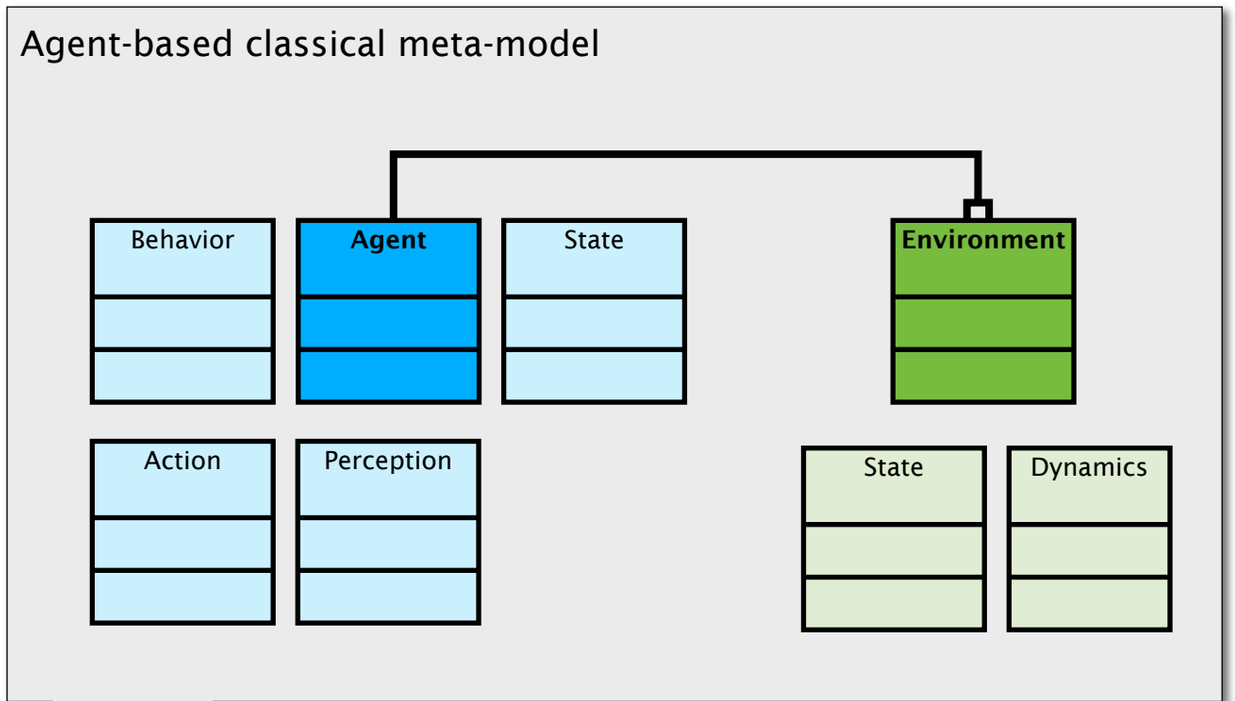
Different modeling approaches



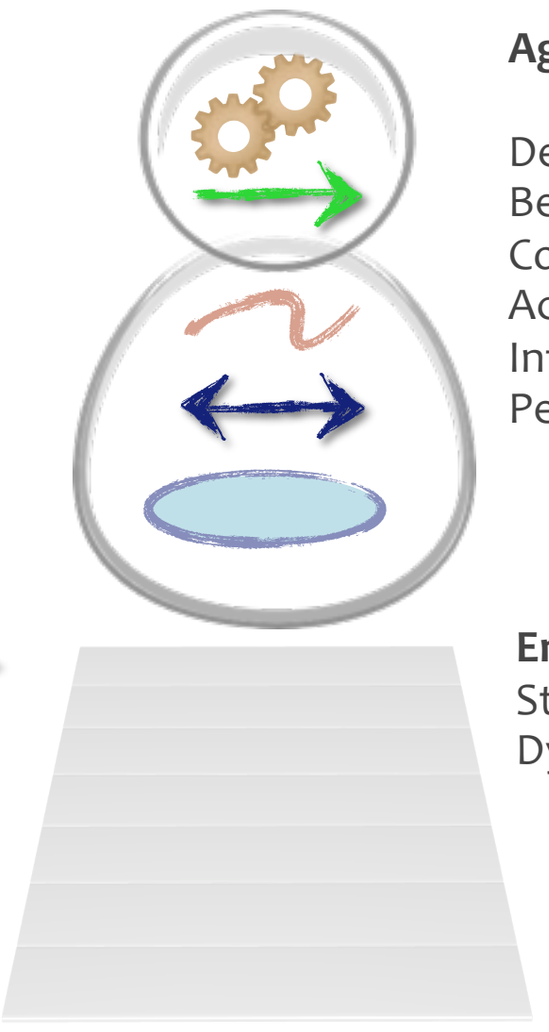
(adapted from Troitzsch & Gilbert 2001)

slide from Alexis Drogoul
(IRD UMI UMMISCO)

Structure of agent-based models




```
ironment 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_env
     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 my leader <- (range of my leader) -
```



Agents

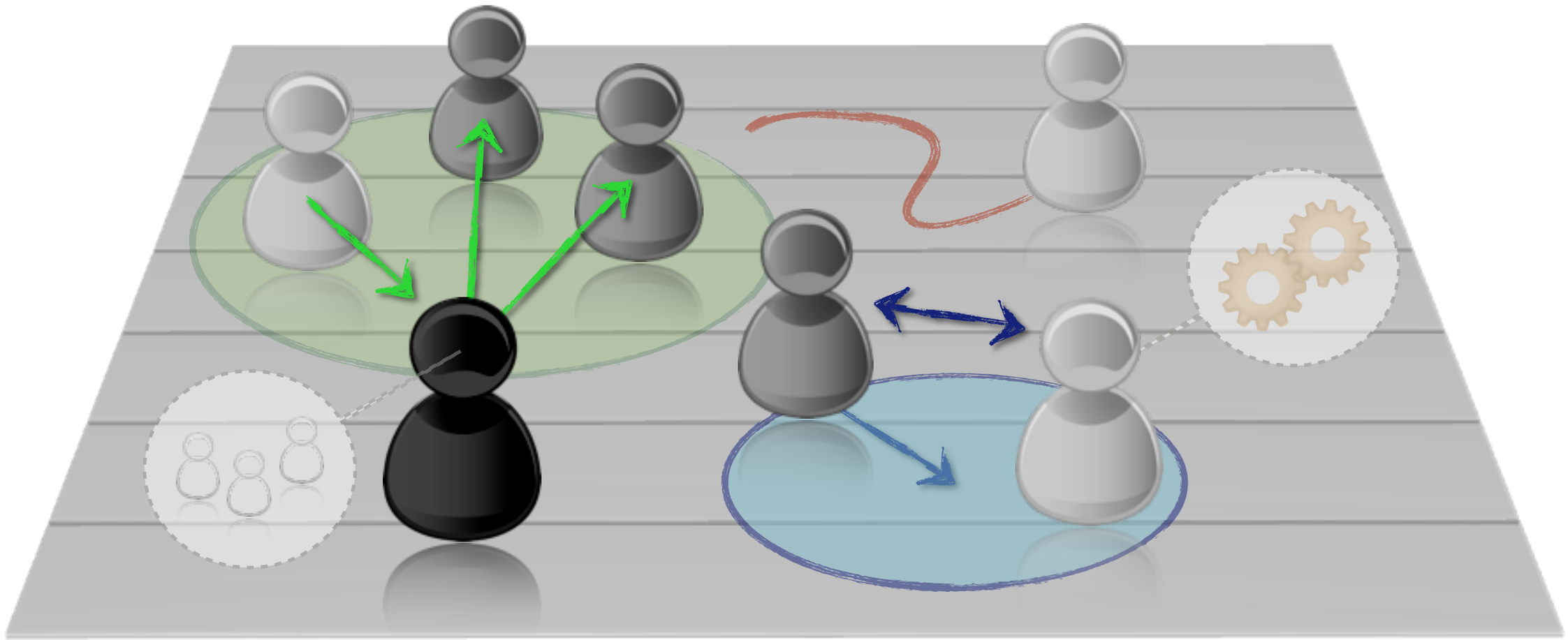
Decision-making
Behavior
Communication
Actions
Interactions
Perception

Environment
State(s)
Dynamics

slide from Alexis Drogoul
(IRD UMI UMMISCO)

Agent-based models

The agents and their environment constitute a virtual «**micro-world**» that can be **experimented** like a real system (with more freedom)



slide from Alexis Drogoul (IRD UMI UMMISCO)

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

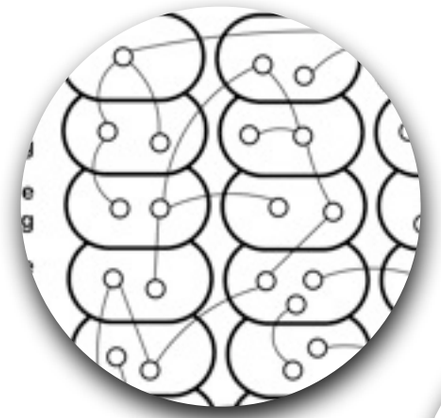
SimPop model
http://www.simpop.parisgeo.cnrs.fr/

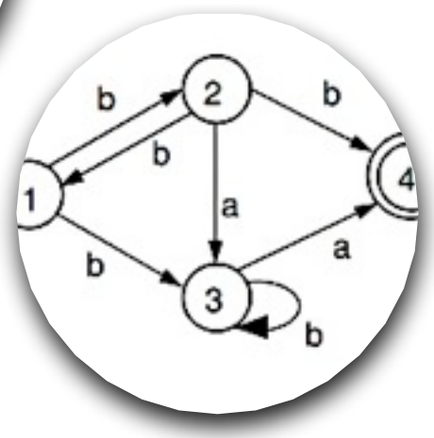
slide from Alexis Drogoul
(IRD UMI UMMISCO)

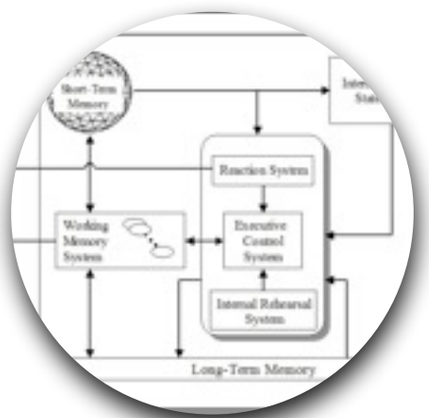
Agent behaviors

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



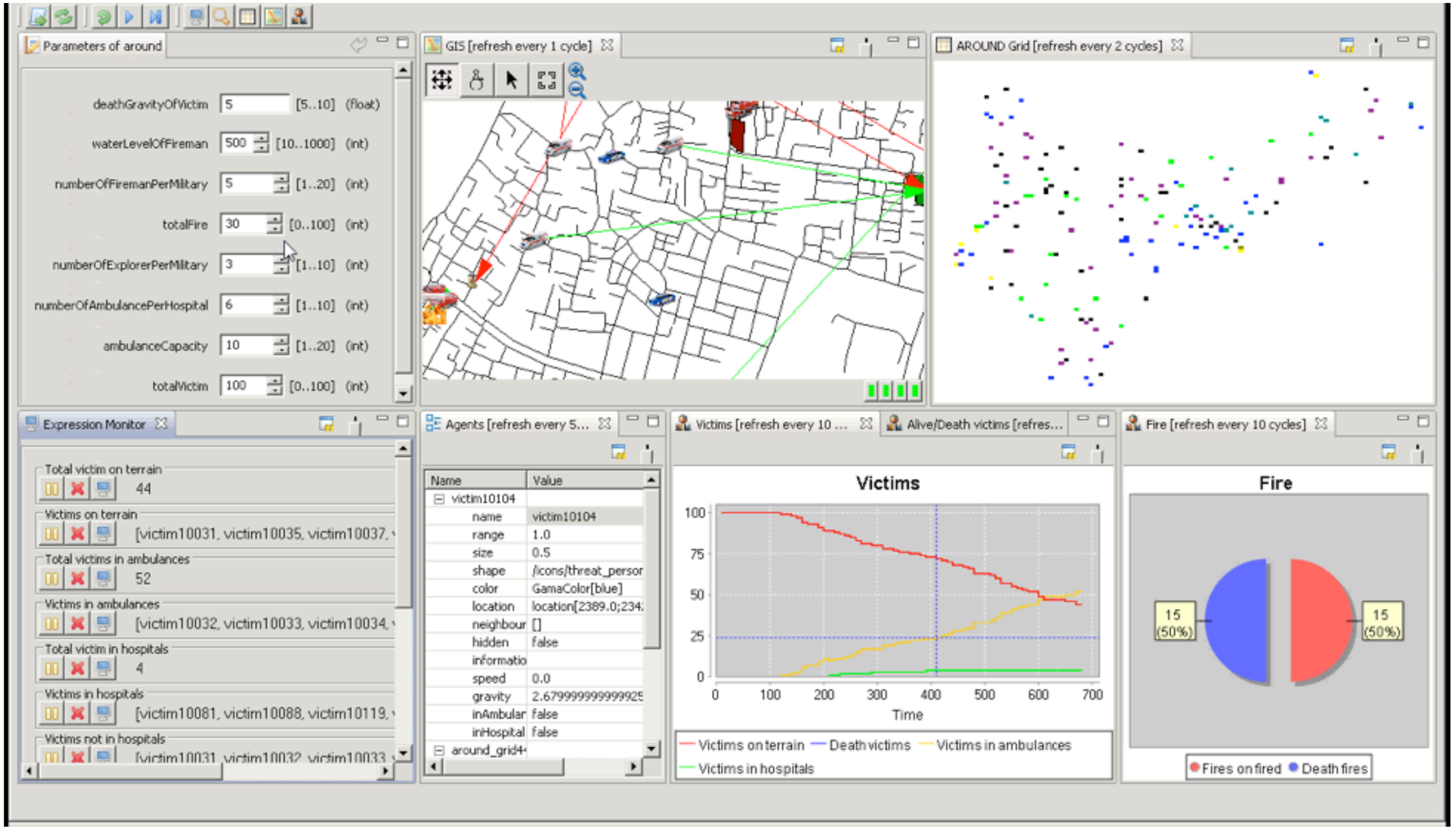




slide from Alexis Drogoul
(IRD UMI UMMISCO)

Miniature laboratories

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



Agent-based models are well adapted to situations where...

1. it is difficult to test hypotheses solely based on observations of the reference system
2. the actors of a reference system are heterogeneous
3. it is possible to identify intermediary levels/organizations that influence the dynamics of the reference system
4. the level of analysis/observation is not fixed
5. changes at the macro-level should be outcomes, and not inputs, of the model

slide from Alexis Drogoul
(IRD UMI UMMISCO)

References: Web

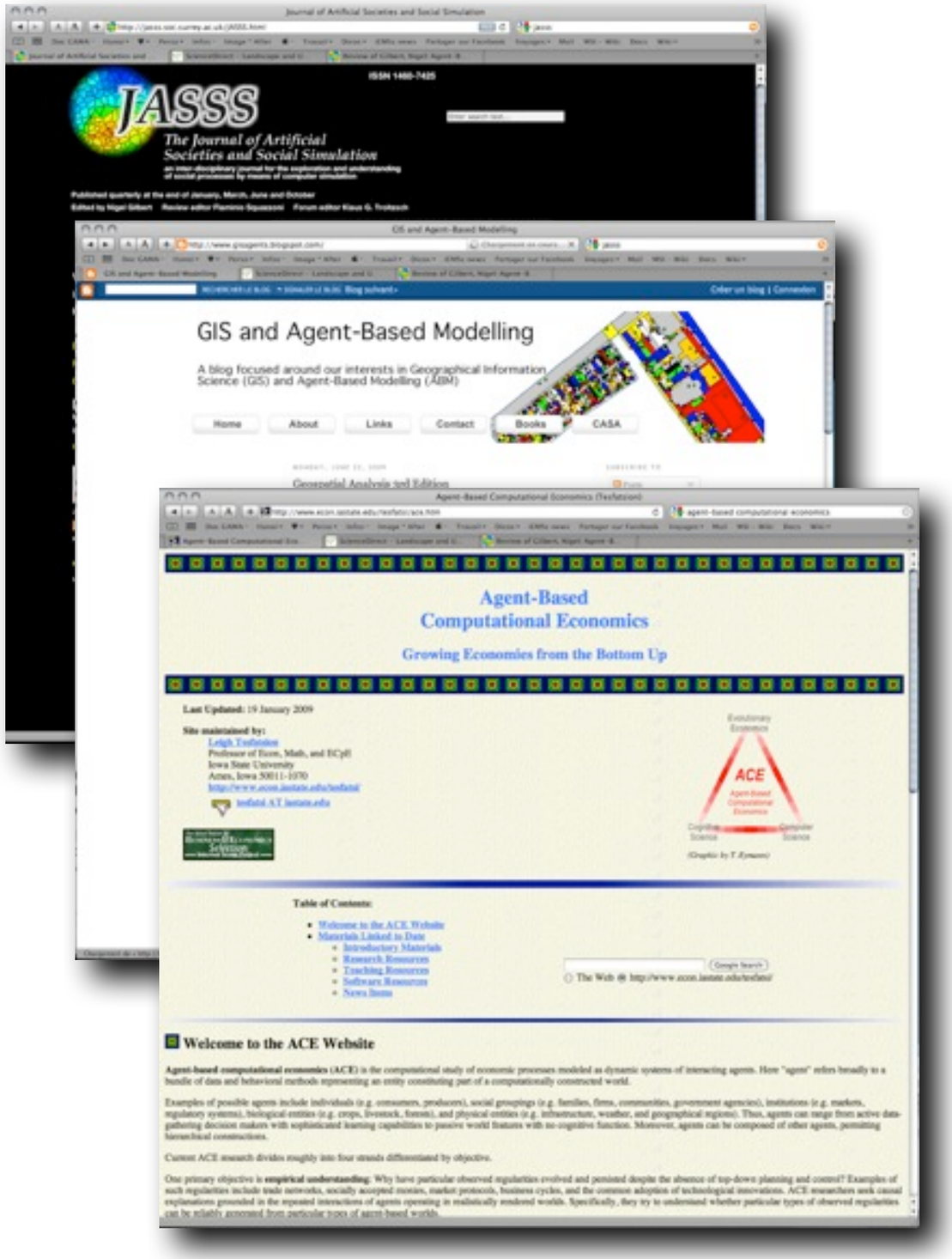
A number of web references are available to go further

JASSS : Journal of Artificial Societies and Social Simulation
http://jasss.soc.surrey.ac.uk/

ACE : Agent-based Computational Economics
http://www.econ.iastate.edu/tesfatsi/ace.htm

GisAgents : GIS and Agent-Based Modelling
http://www.gisagents.blogspot.com/

OpenABM: Open Agent-Based Modeling Consortium
http://www.openabm.org/



slide from Alexis Drogoul
(IRD UMI UMMISCO)

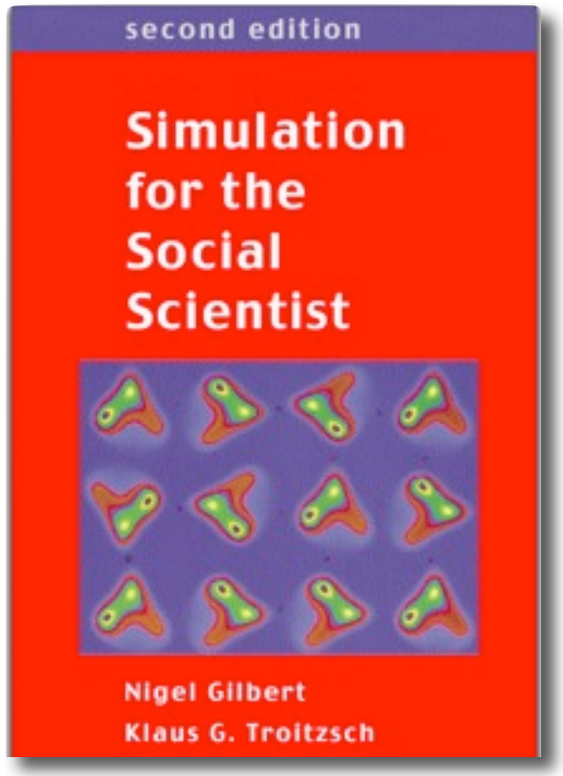
Some books as well (two in English, two in French)

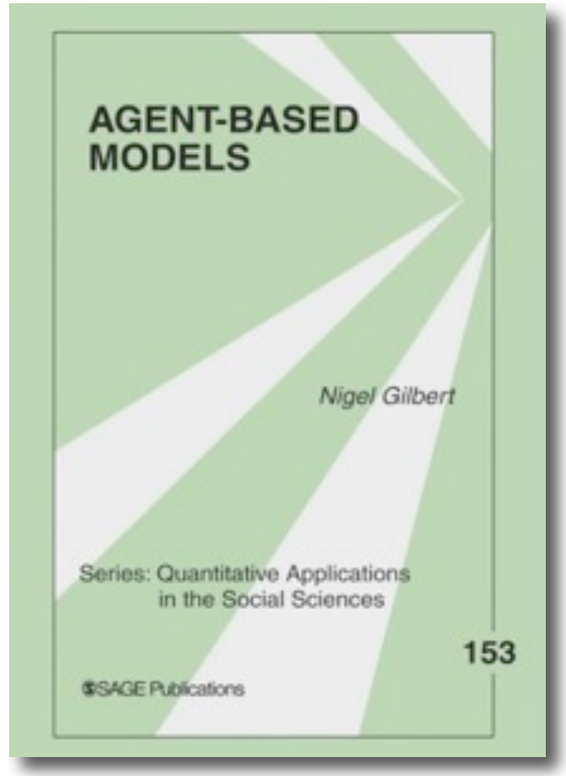
GILBERT N and TROITZSCH K. G. (2005) *Simulation for the Social Scientist.* Milton Keynes: Open University Press, Second Edition.

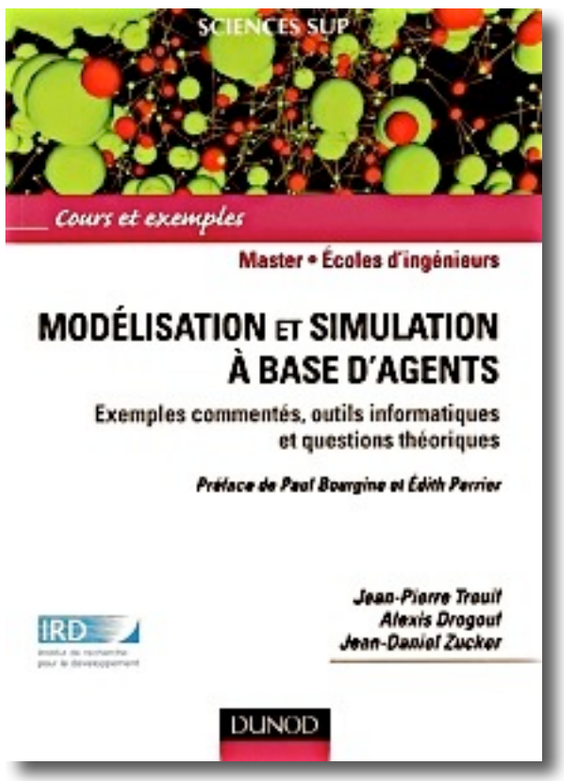
GILBERT N (2006) *Agent-based Models*. Series: Quantitative applications in the Social Sciences. Sage Publications.

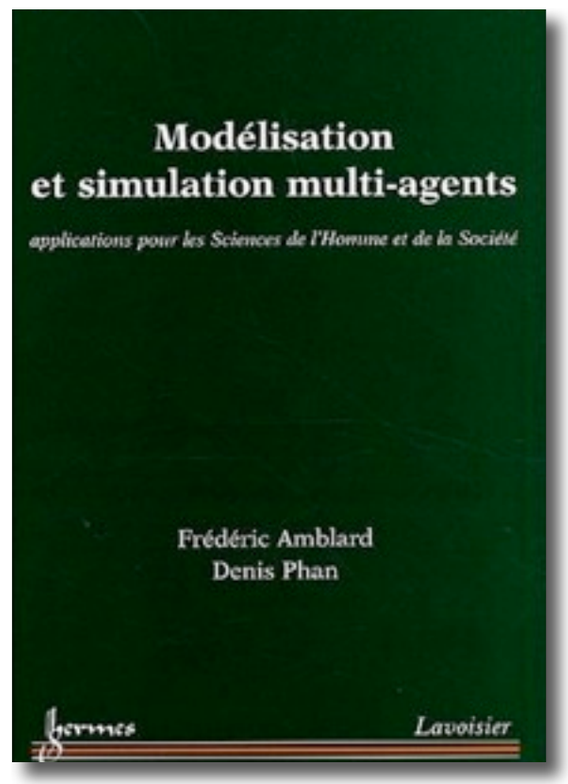
AMBLARD F et PHAN D (2007) *Modélisation et Simulation multi-agents: applications pour les Sciences de l'Homme et de la Société*. Hermès Editions.

TREUIL JP, DROGOUL A et ZUCKER JD (2008) *Modélisation et Simulation à base d'agents.* Dunod & IRD Editions.









slide from Alexis Drogoul (IRD UMI UMMISCO)

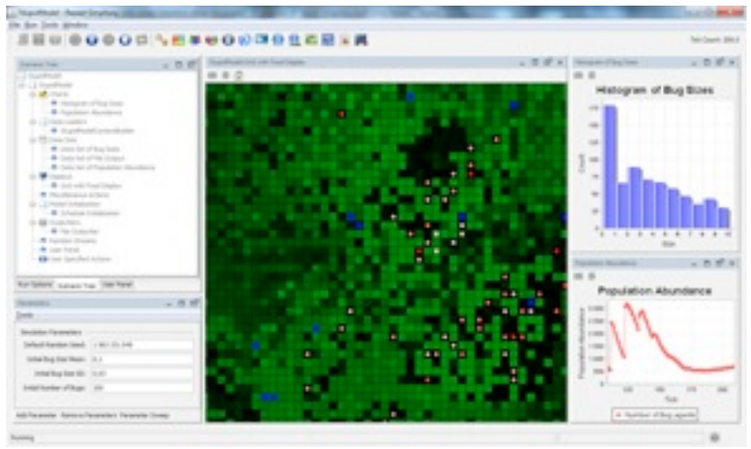
References: tools

Three open-source free platforms for building and experimenting agent-based models

NetLogo

http://ccl.northwestern.edu/netlogo/

Pedagogical tool with a specific modeling language. Numerous examples in various domains. Ideal for beginners, but somehow limited for large models.



Repast Simphony

http://repast.sourceforge.net/

Toolbox for agent-based modeling in Java, numerous libraries of connected technologies (statistical analysis, GIS, 3D models, etc.). Requires very good programming skills.

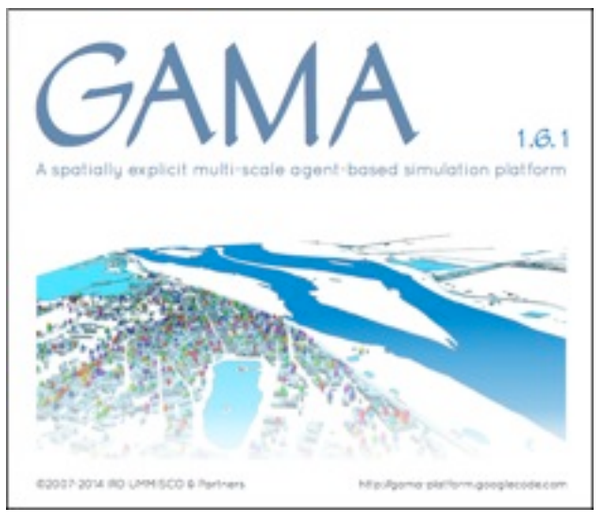
GAMA

http://gama-platform.googlecode.com

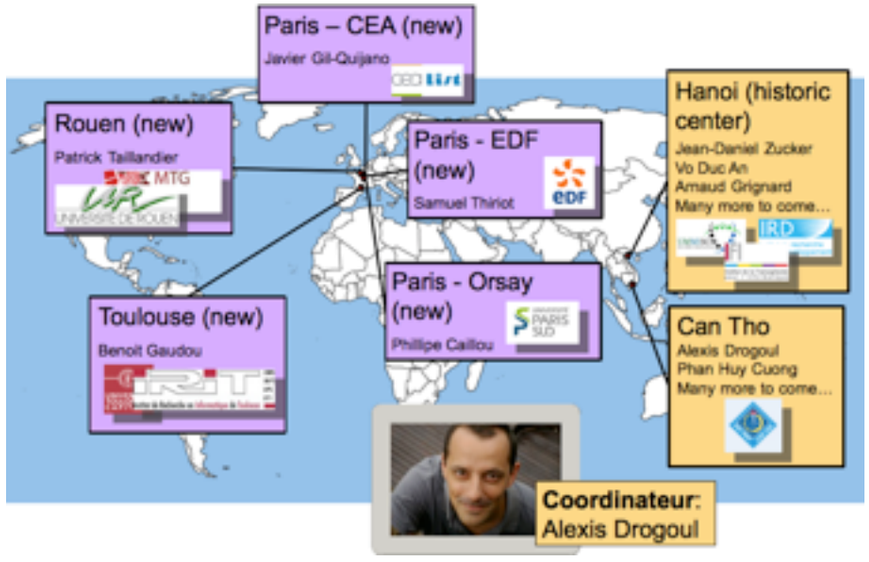
A compromise between the two platforms above. Using a dedicated modeling language, this platform pays a great deal of attention to the modeling of the environment and the use of GIS data, while staying open to new technologies through plug-ins.

slide from Alexis Drogoul (IRD UMI UMMISCO)

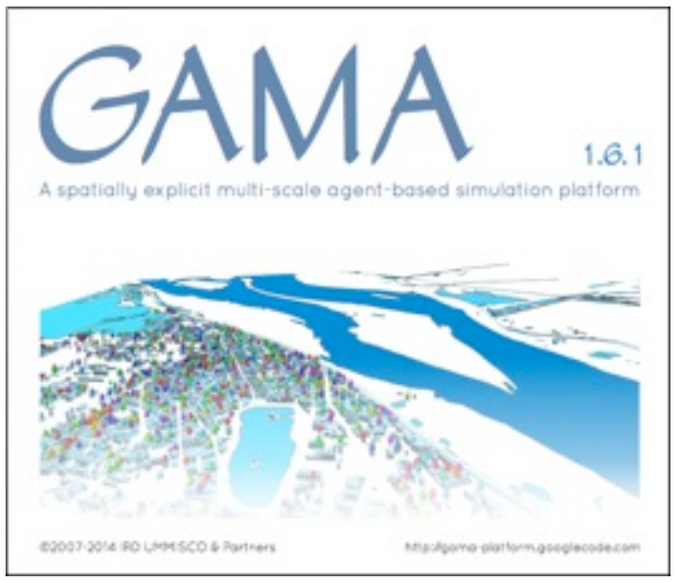
- Software platform dedicated to building spatially explicit agent-based simulations
  - Developed by a consortium of research team : UMMISCO (IRD - coordinator), DREAMS (Can Tho), IFI (Hanoi), IDEES (Rouen), IRIT (Toulouse), LRI (Orsay), ...
  - Generic : can be used for a wide range of applications
  - Developed under GPL/LGPL 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



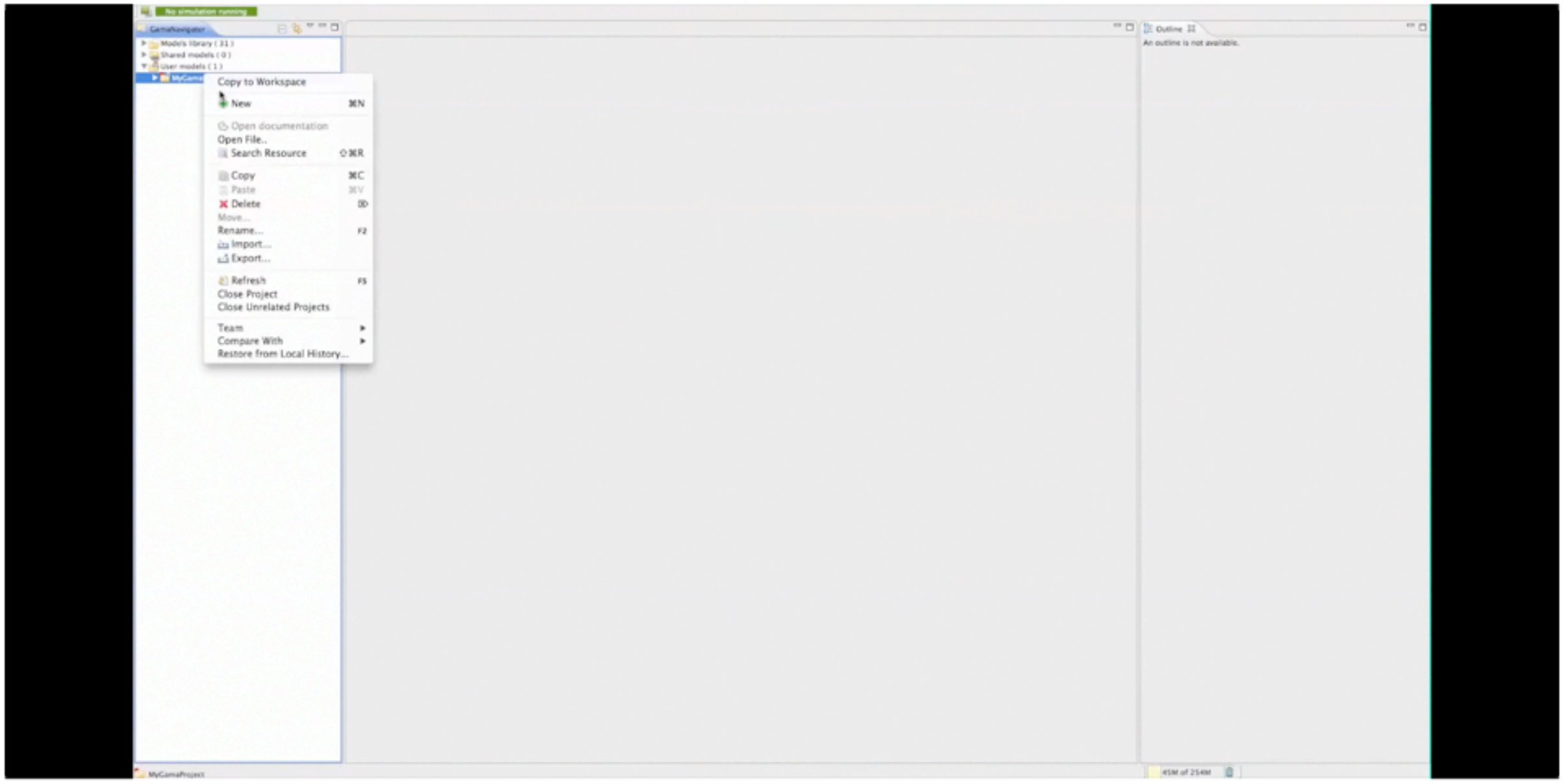
Last version: 1.6.1



- 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

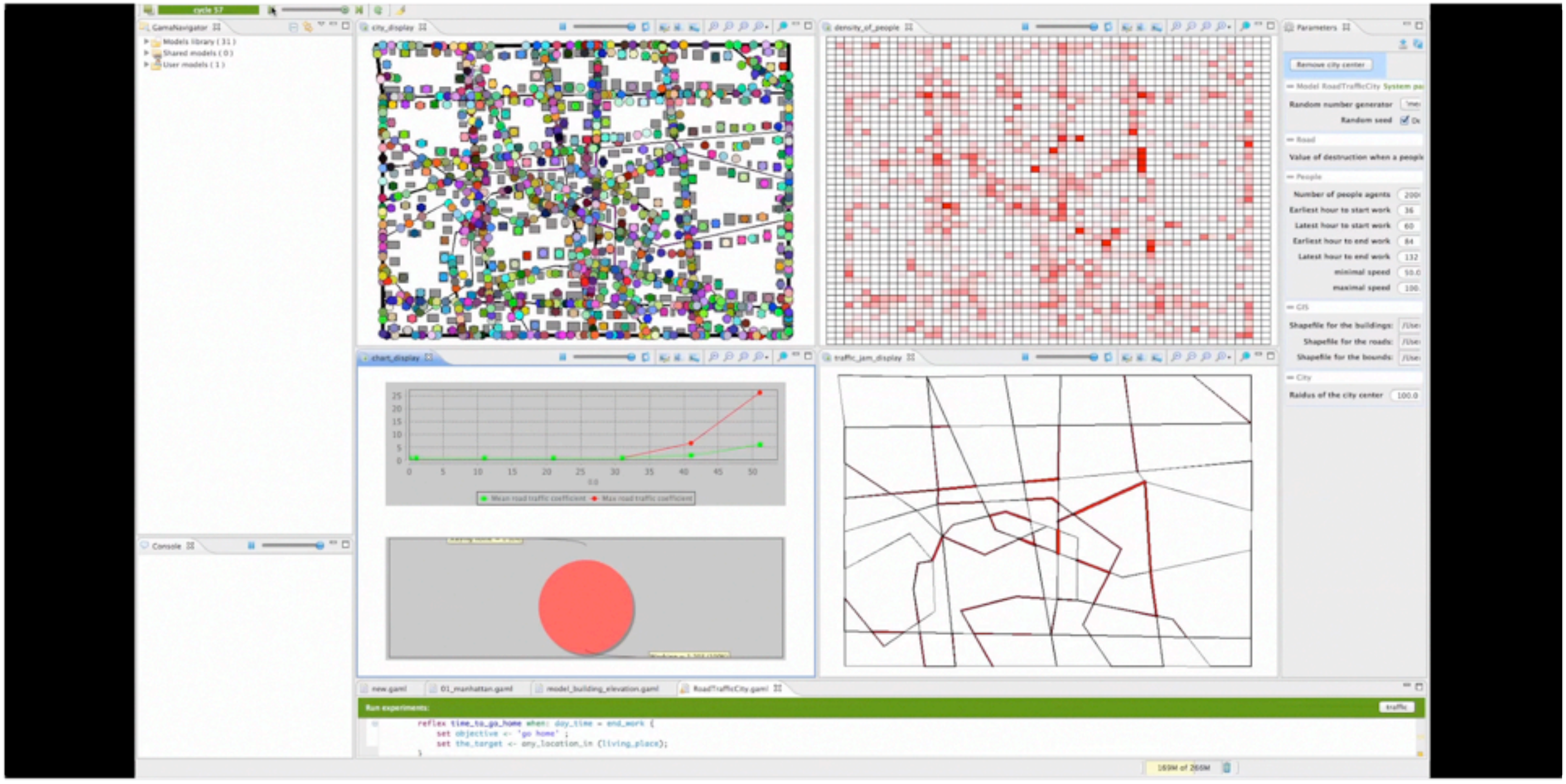


- GAMA provides a complete Integrated Development environment (IDE) to build models



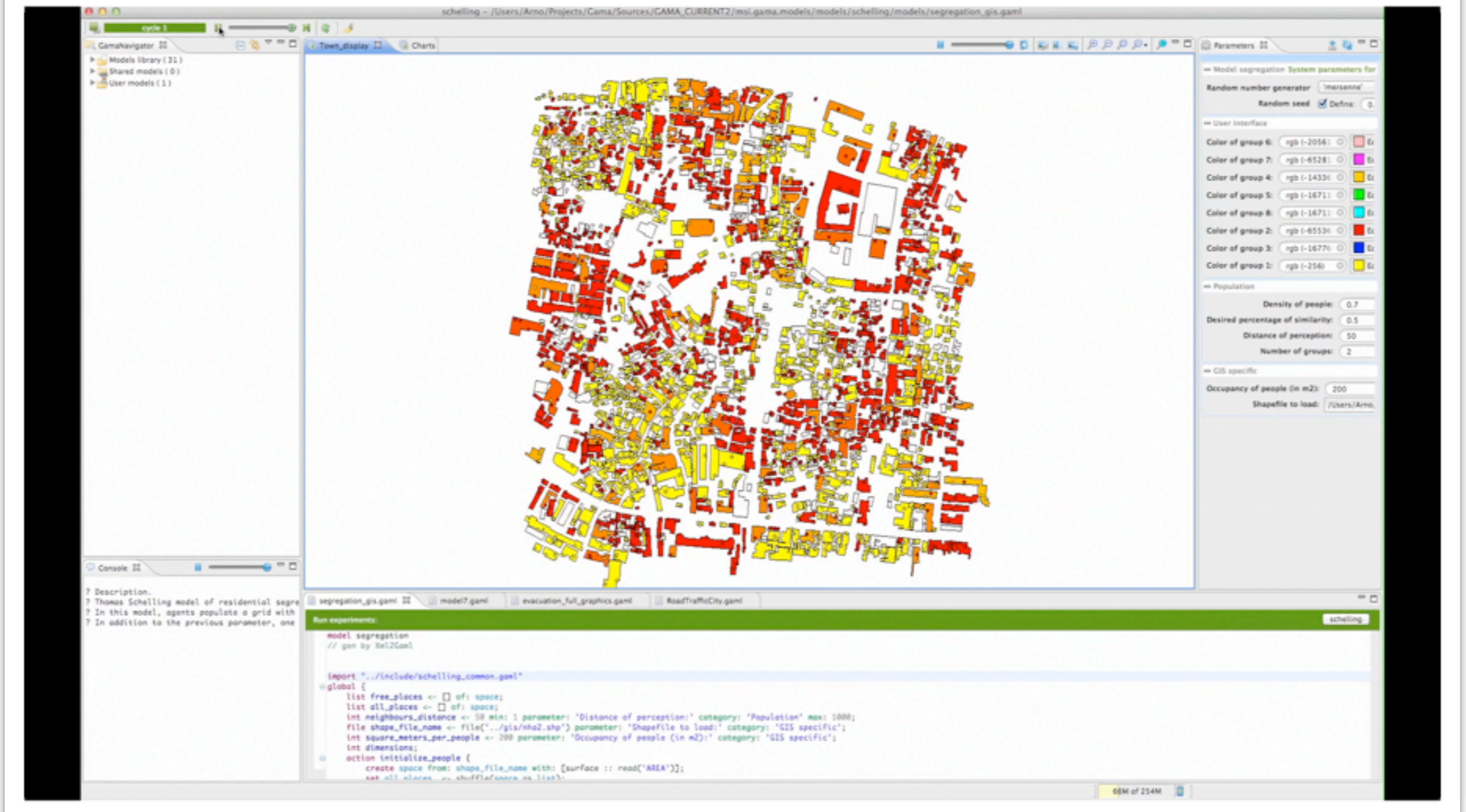
Dedicated modeling Language (GAML), easy to learn and to extend

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



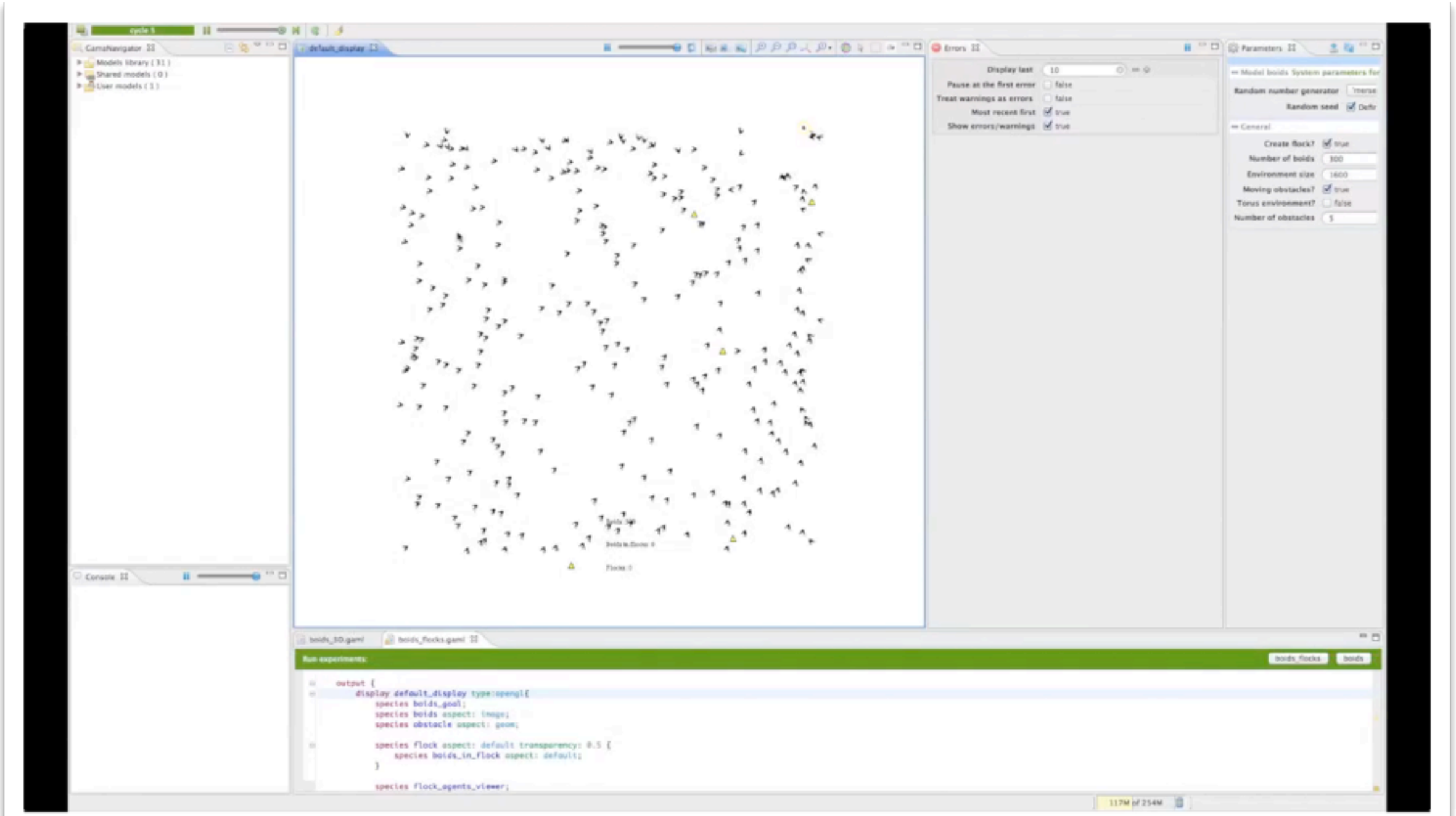
2D Grid (rectangle, hexagon)
Continuous environment, torus environment, graphs

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



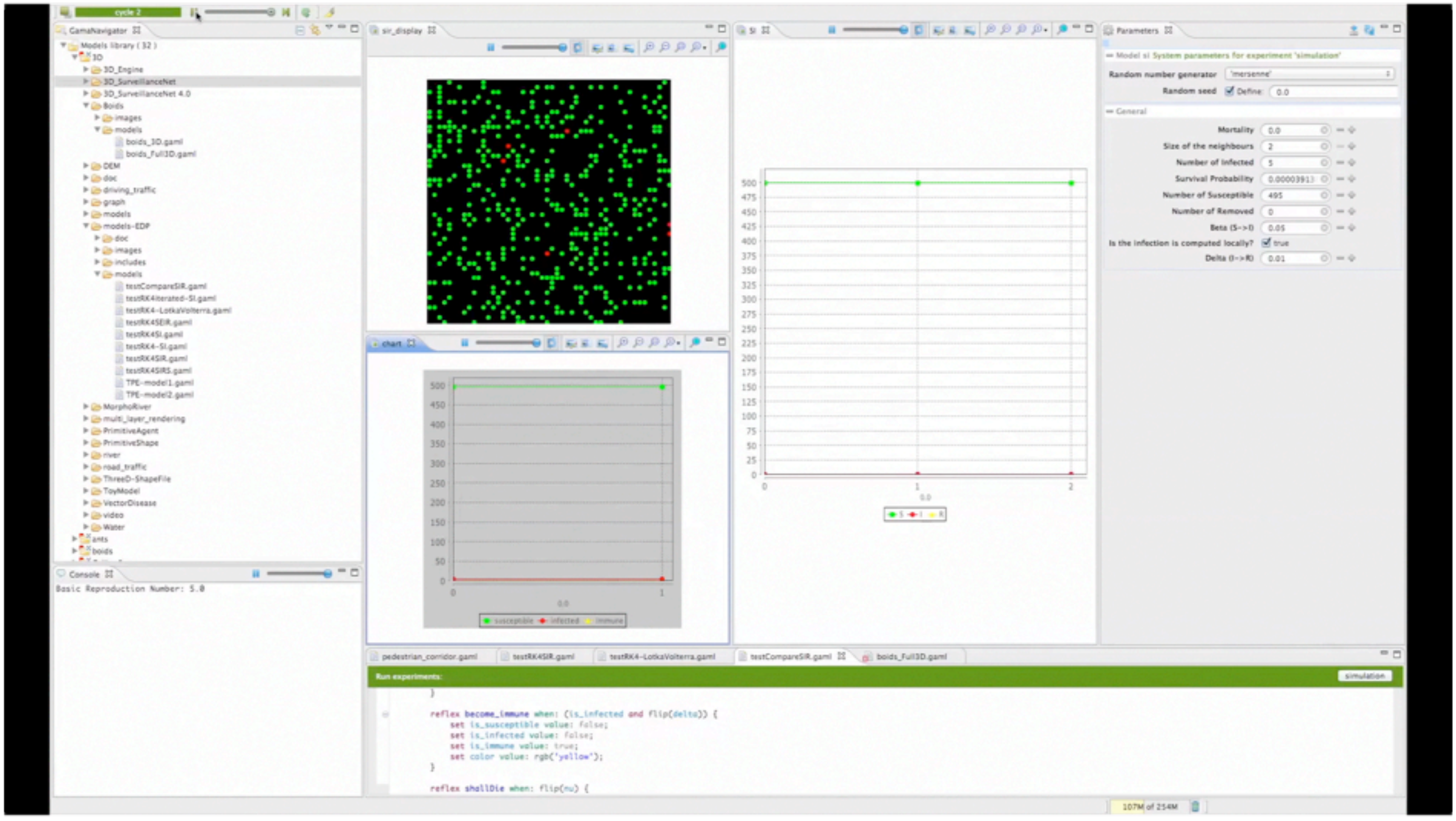
Transparent agentification of 2D/3D GIS data
powerful geometrical operations (union, difference, spatial queries....)

- Allows to define multi-level models



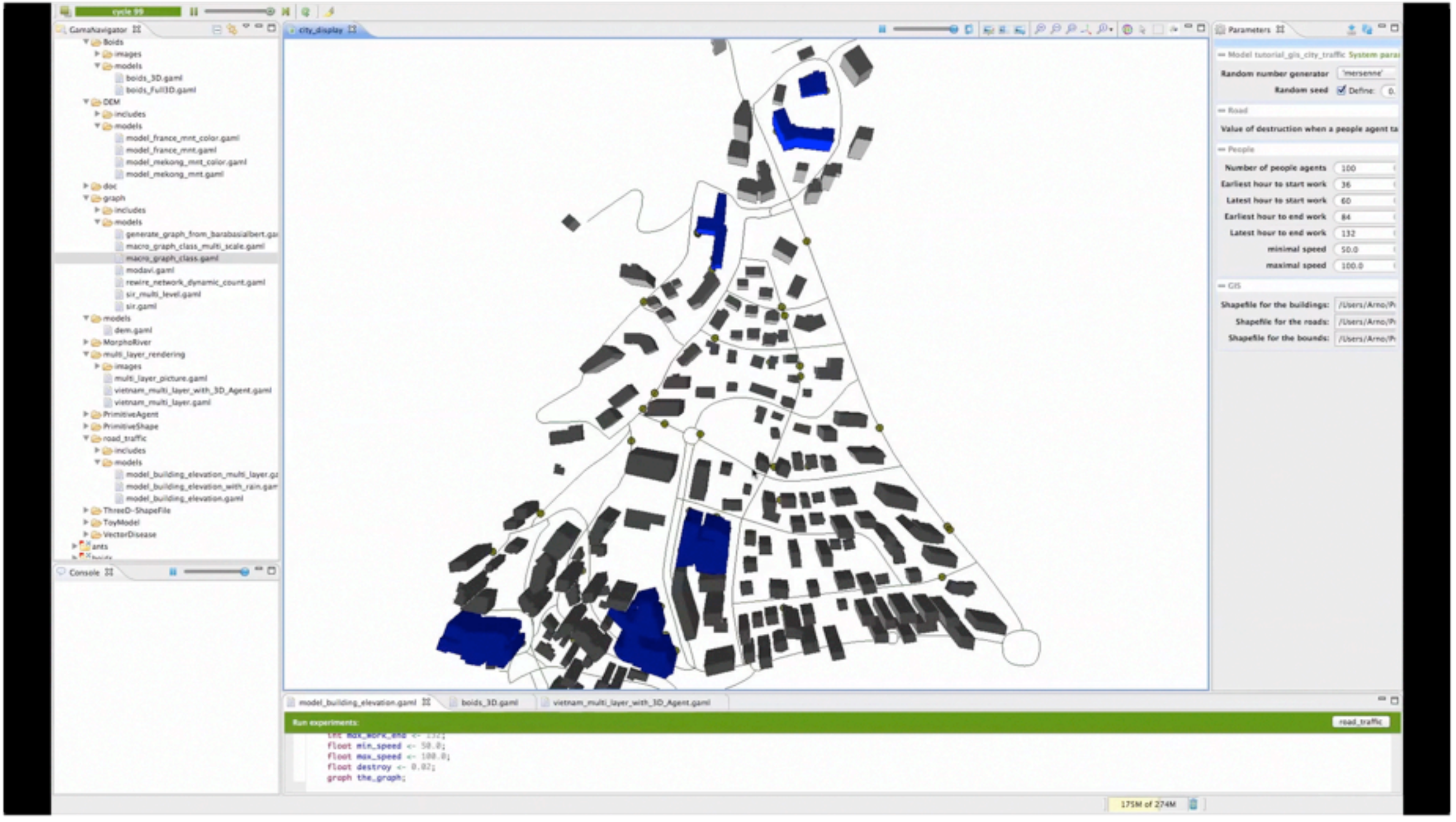
Different level of abstraction, aggregation operators...

- Allows to use different formalisms to define agent behaviors



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

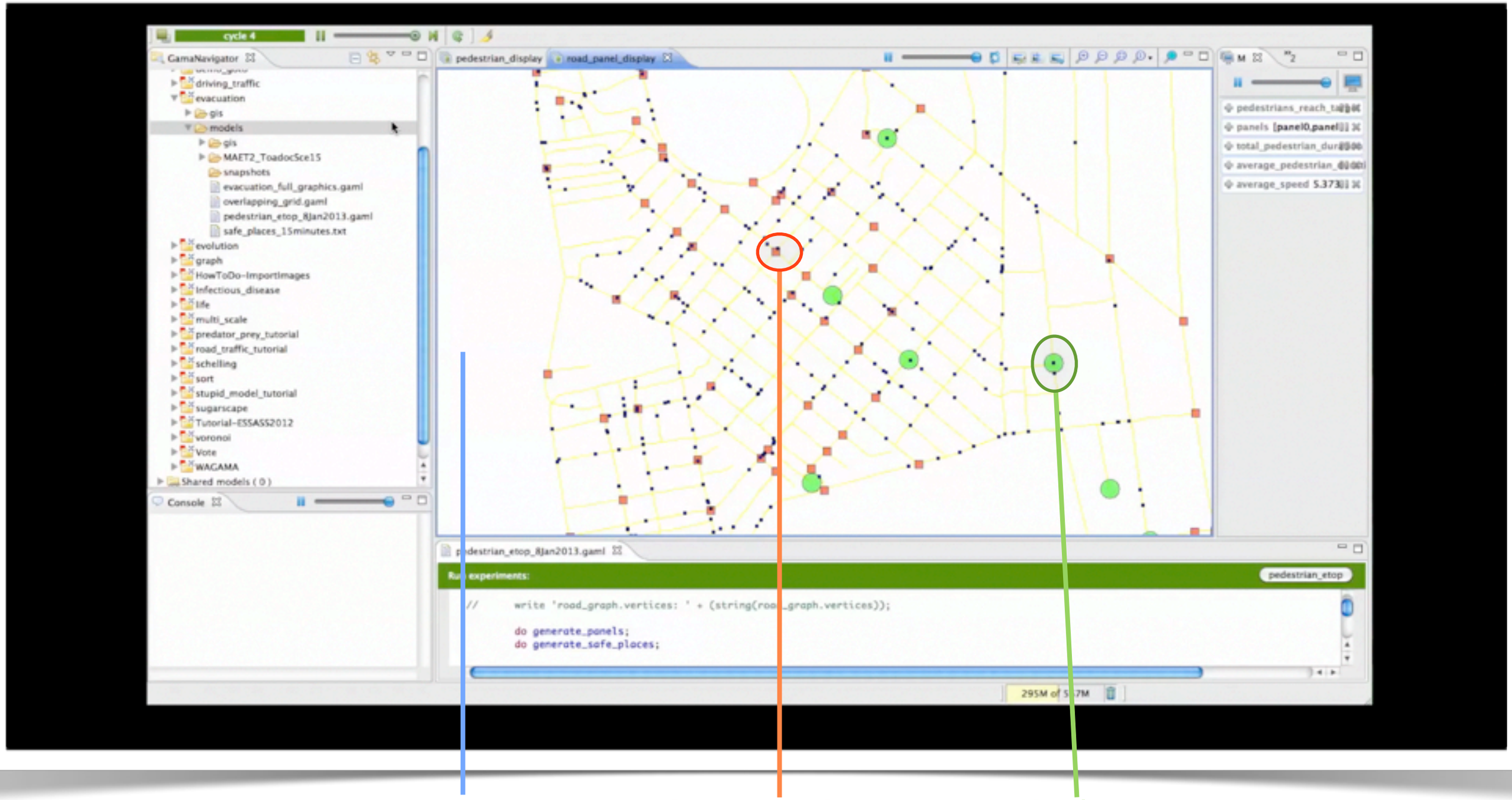
- Powerful visualization tools allowing to define as many displays as necessary



3D display, use of layers, DEM display....

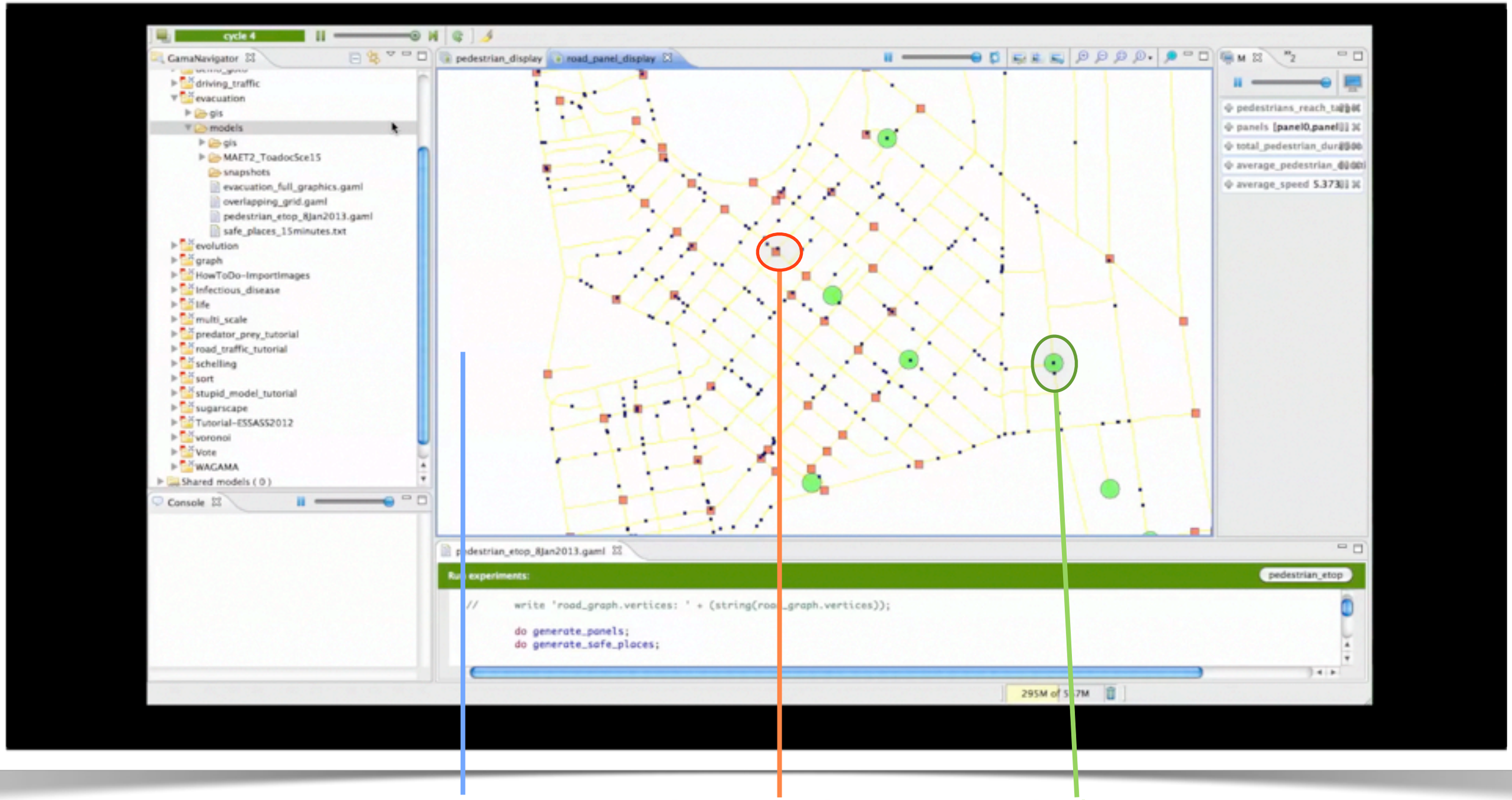
Example of real models: people evacuation during Tsunami

- **Nha Trang city, Vietnam**: where to place crossroad evacuation signs in order to maximize the number of survivors in an evacuation following a Tsunami alert?



Example of real models: people evacuation during Tsunami

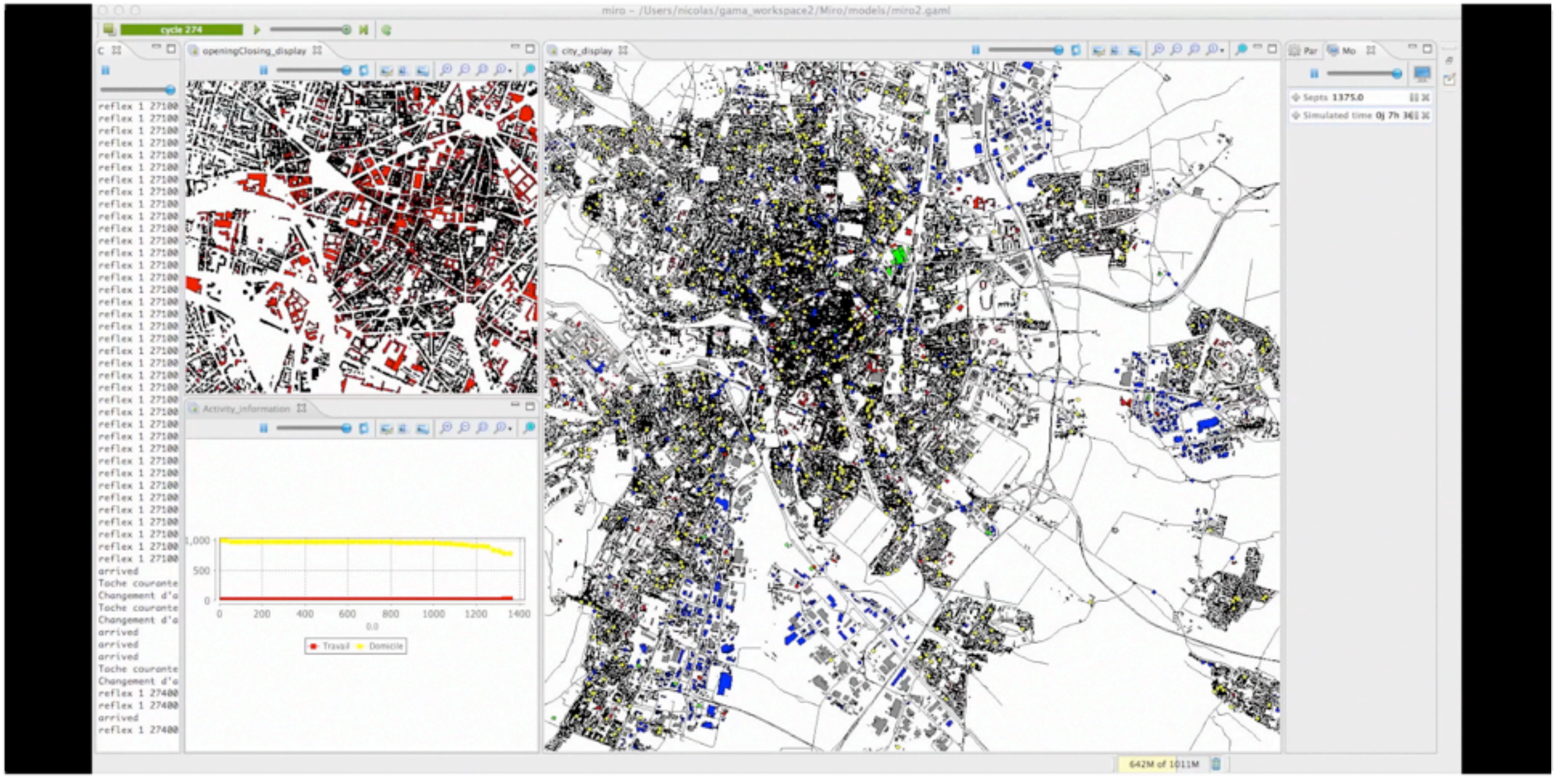
- **Nha Trang city, Vietnam**: where to place crossroad evacuation signs in order to maximize the number of survivors in an evacuation following a Tsunami alert?



Anh T.-N. Nguyen, J-D Zucker, H. M. Nguyen , A Drogoul, Phuong H. N.,"Simulation of Emergency Evacuation of Pedestrians along the Road Networks in Nhatrang City" , Computing and Communication Technologies, Research, Innovation, and Vision for the Future, 2012 IEEE RIVF

Example of real models: MIRO 2

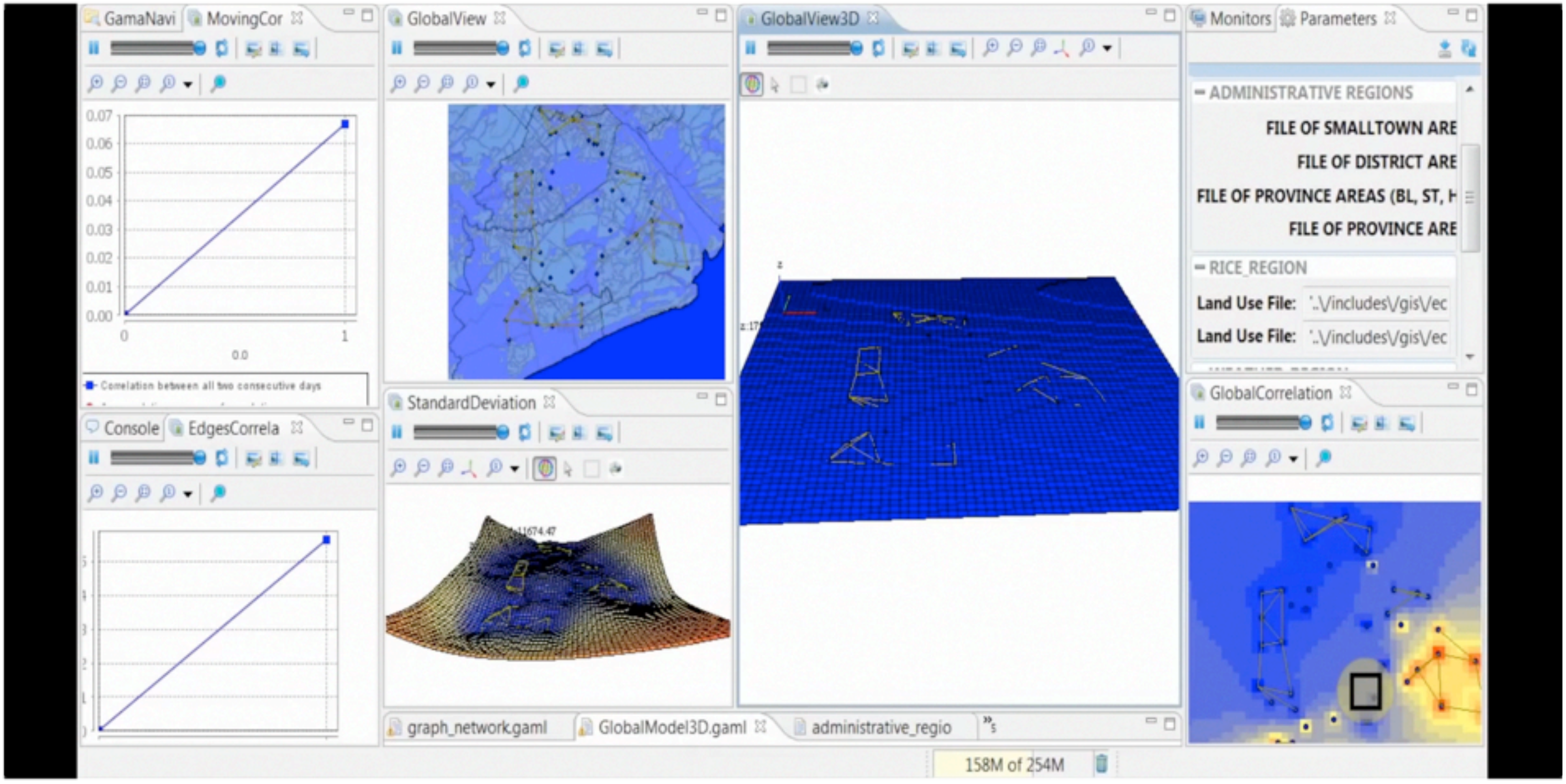
- **Dijon city, France**: how to improve the individual accessibility to the city in order to better manage urban mobility ?



A. Banos, N. Marilleau, and MIRO Team, “Improving individual accessibility to the city: an Agent Based Modelling approach,” ECCS 2012 – European Conference of Complex Systems, Bruxelles 2012

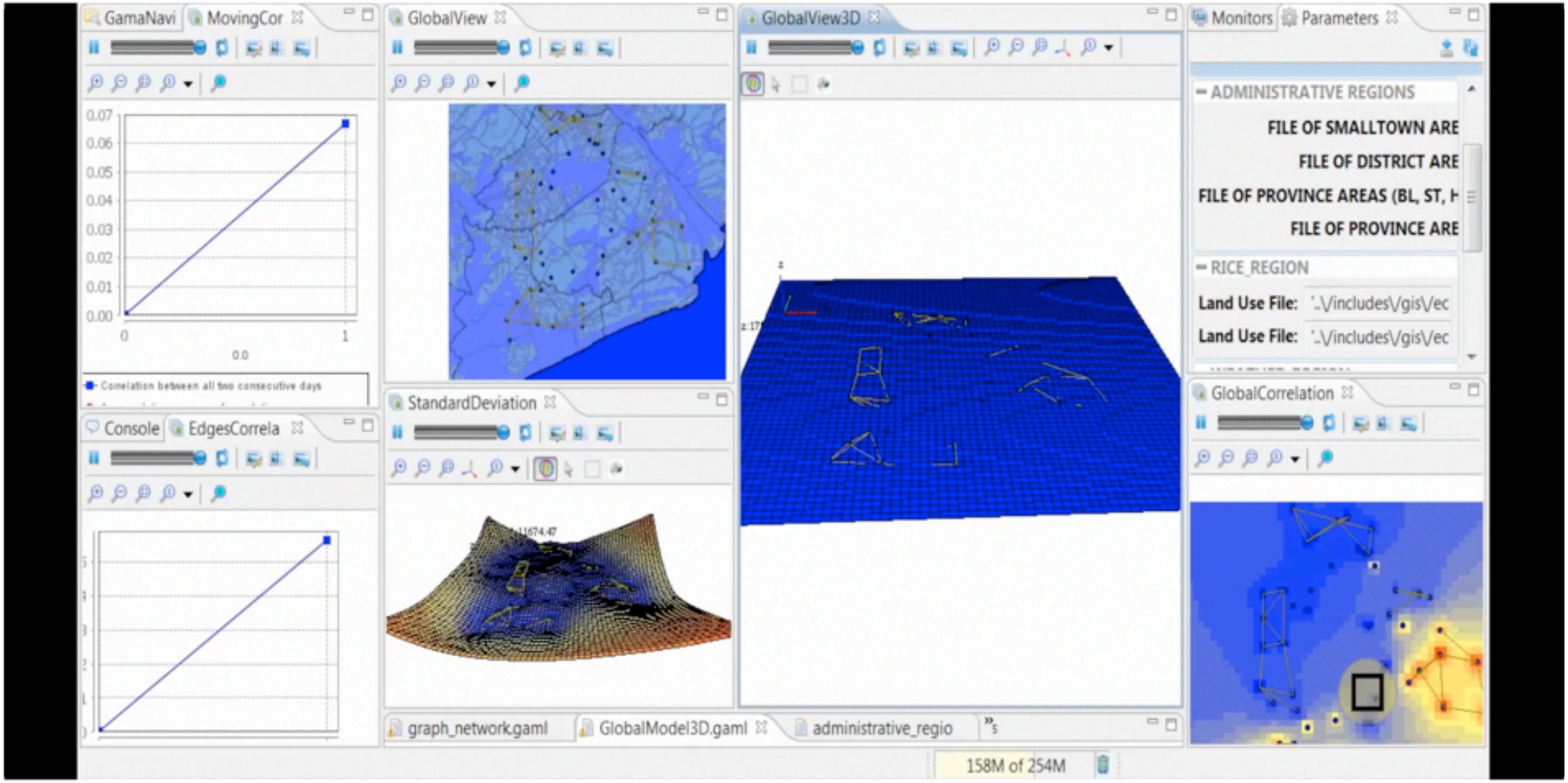
Example of real models: Brown hopper invasion

- **Mekong Delta, Vietnam:** how to optimize the spatial arrangement of surveillance networks in different scenarios of brown hoppers invasions ?



Example of real models: Brown hopper invasion

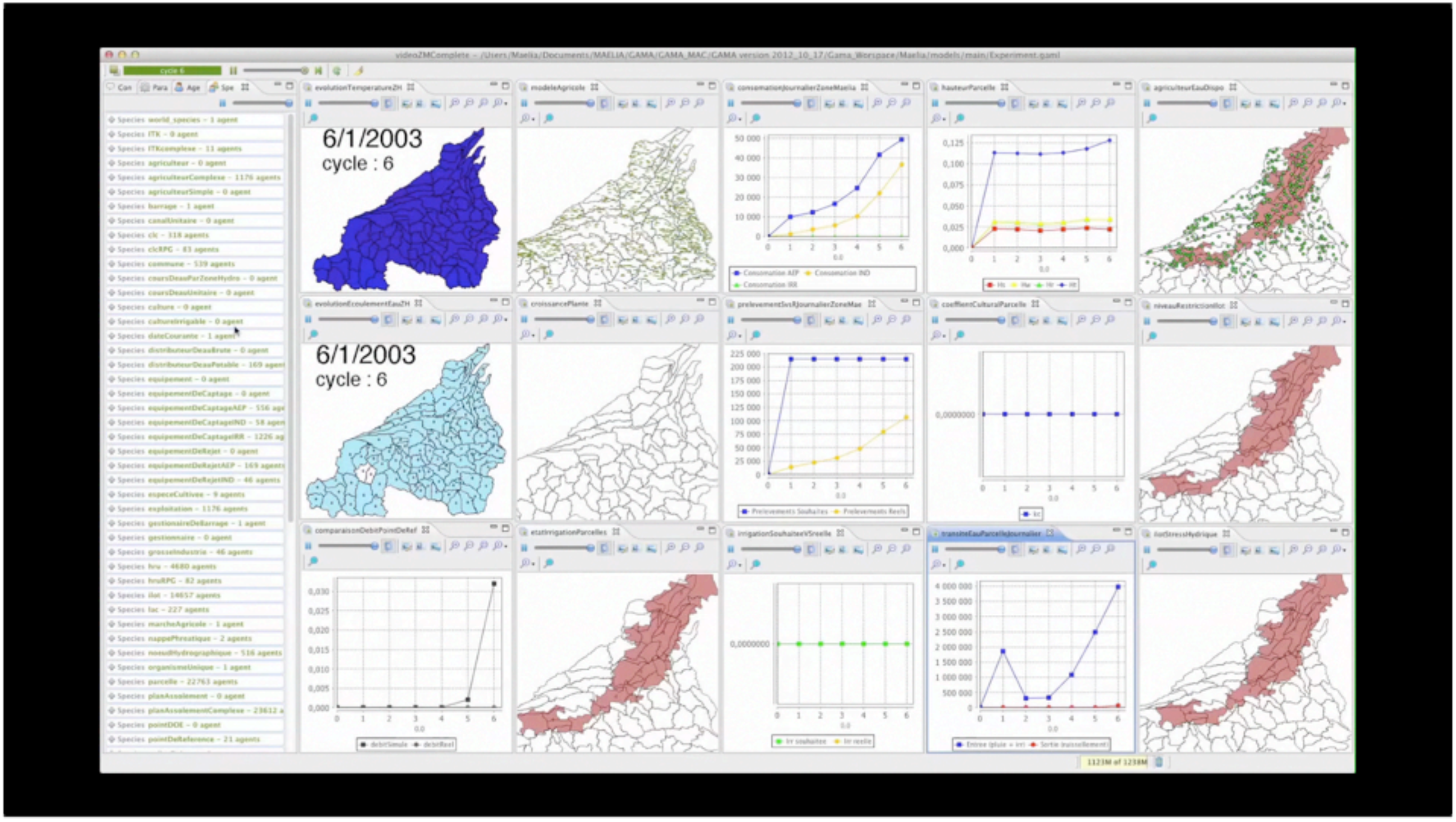
- **Mekong Delta, Vietnam:** how to optimize the spatial arrangement of surveillance networks in different scenarios of brown hoppers invasions ?



Truong, X.V., Huynh, X.H., Le, N.M., Drogoul, A. 2012. Modeling a Surveillance Network based on Unit Disk Graph technique – Application for monitoring the invasion of insects in Mekong Delta Region. 15th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2012), September 5-7 2012, Kuching, Malaysia. LNAI 7455, pp. 228-242, 2012.

Example of real models : MAELIA

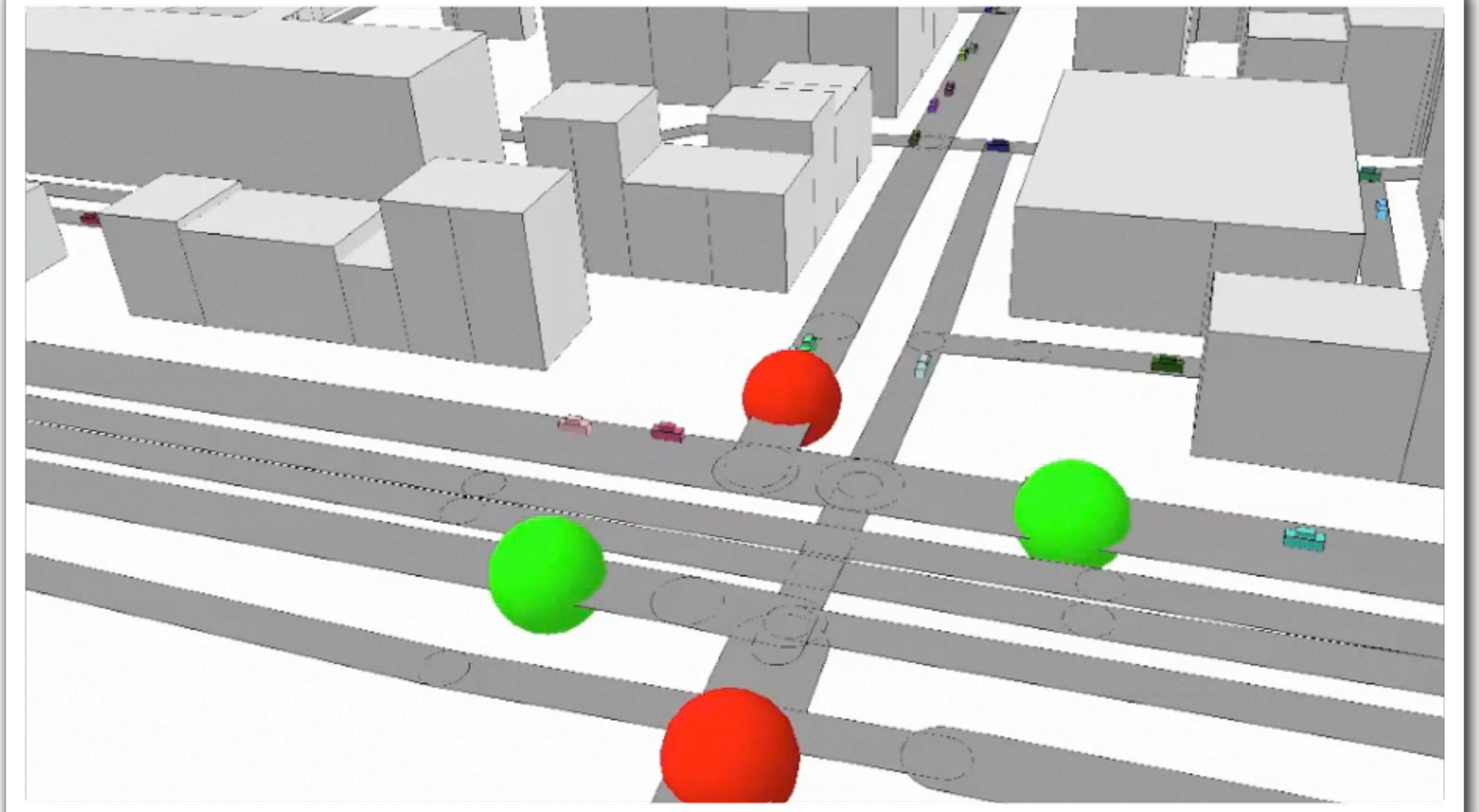
- **Adour-Garonne basin, France**: what is the socio-environmental impact of water management norms on water resources?



Taillandier, P., Therond, O., Gaudou, B, "A new BDI agent architecture based on the belief theory. Application to the modelling of cropping plan decision-making", Environmental Modelling and Software Society (iEMSs), Leipzig, Germany. 2012

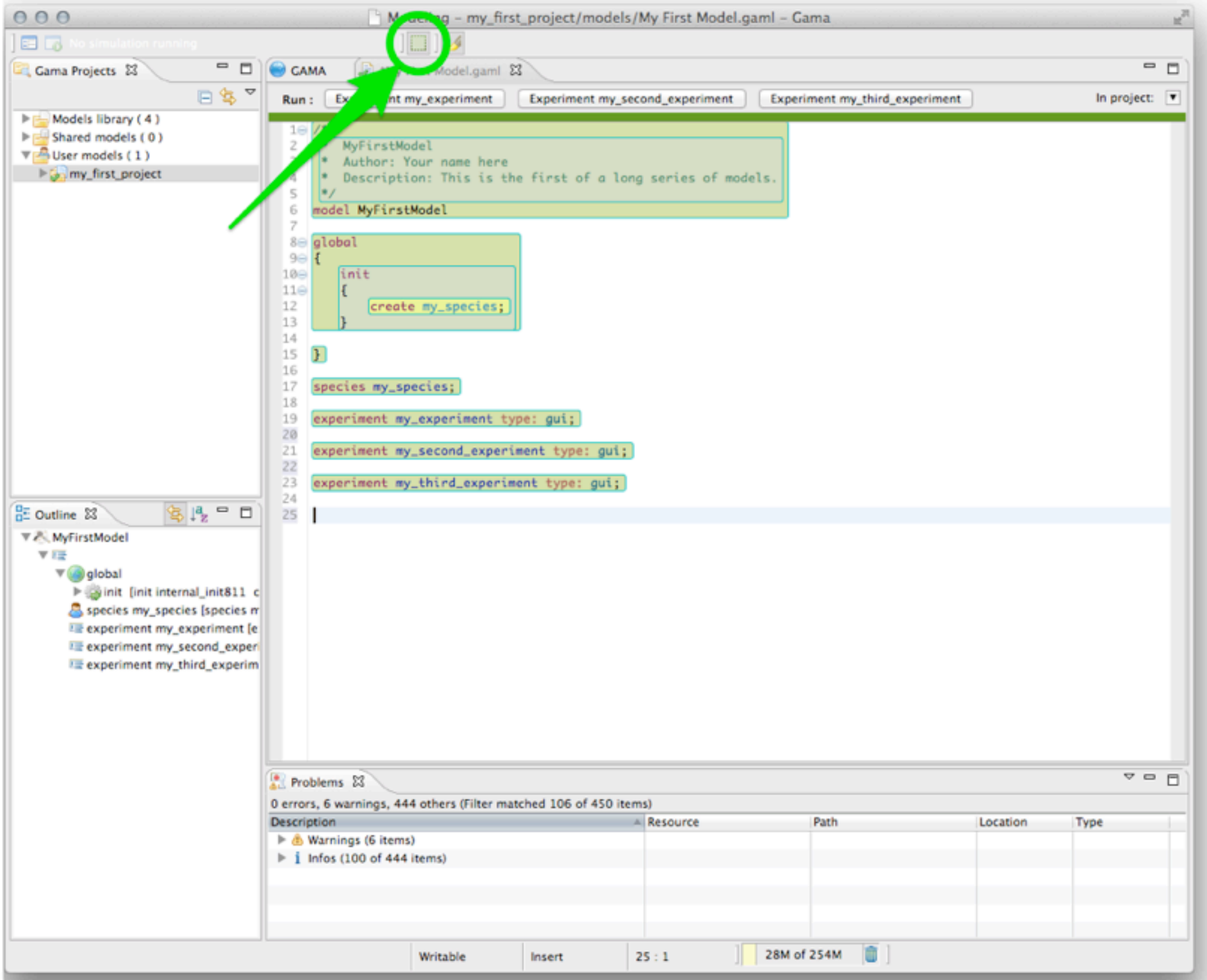
Example of real models : MOSAIIC

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



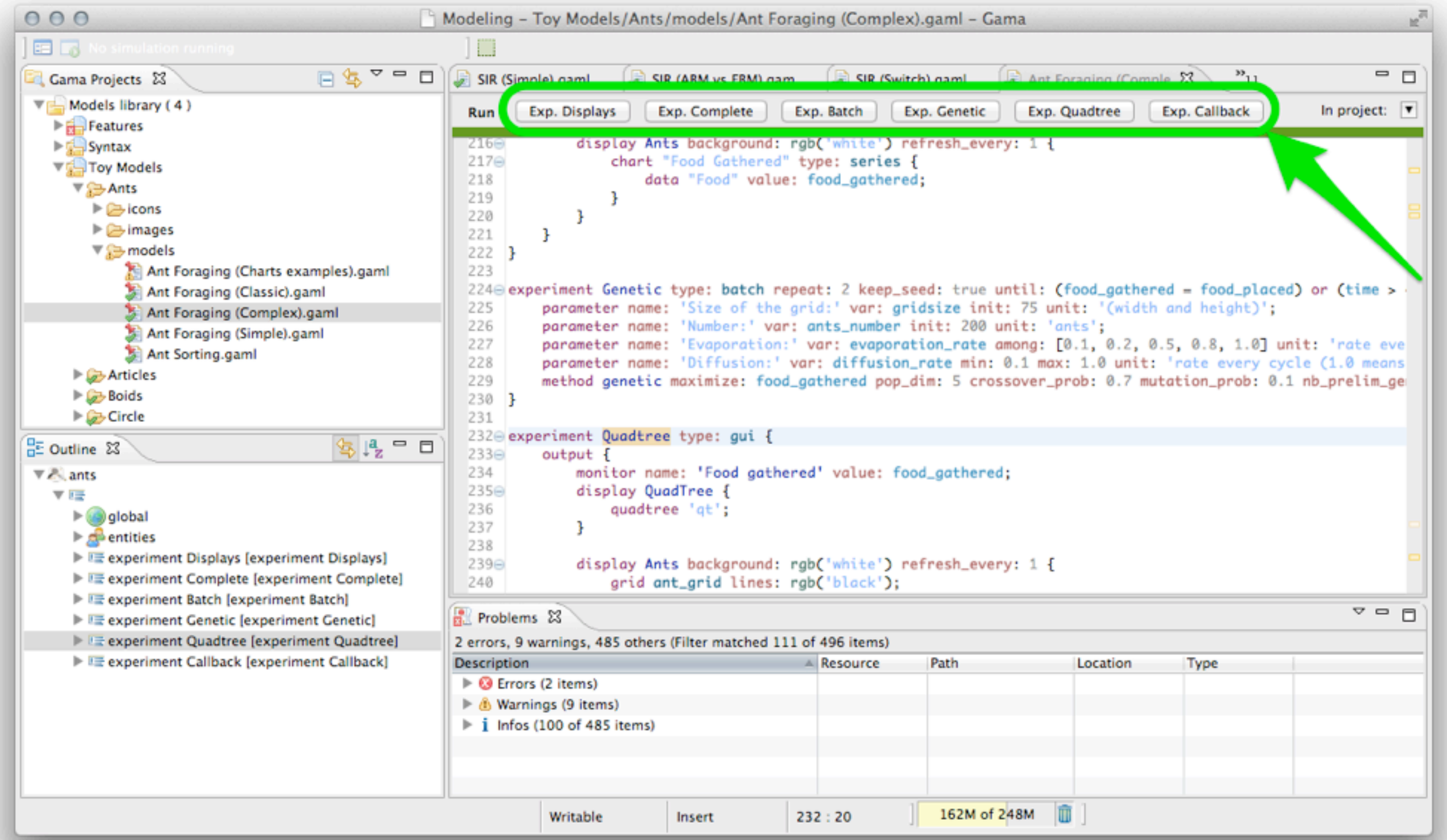
DAUDE É., TRANOUEZ P., LANGLOIS P., (2009), A multiagent urban traffic simulation. Part II : dealing with the extraordinary, Conference Proceedings ICCSA'09, The 3rd International Conference on Complex Systems and Applications, pp. 116-121.

Modeling interface: structural highlighting



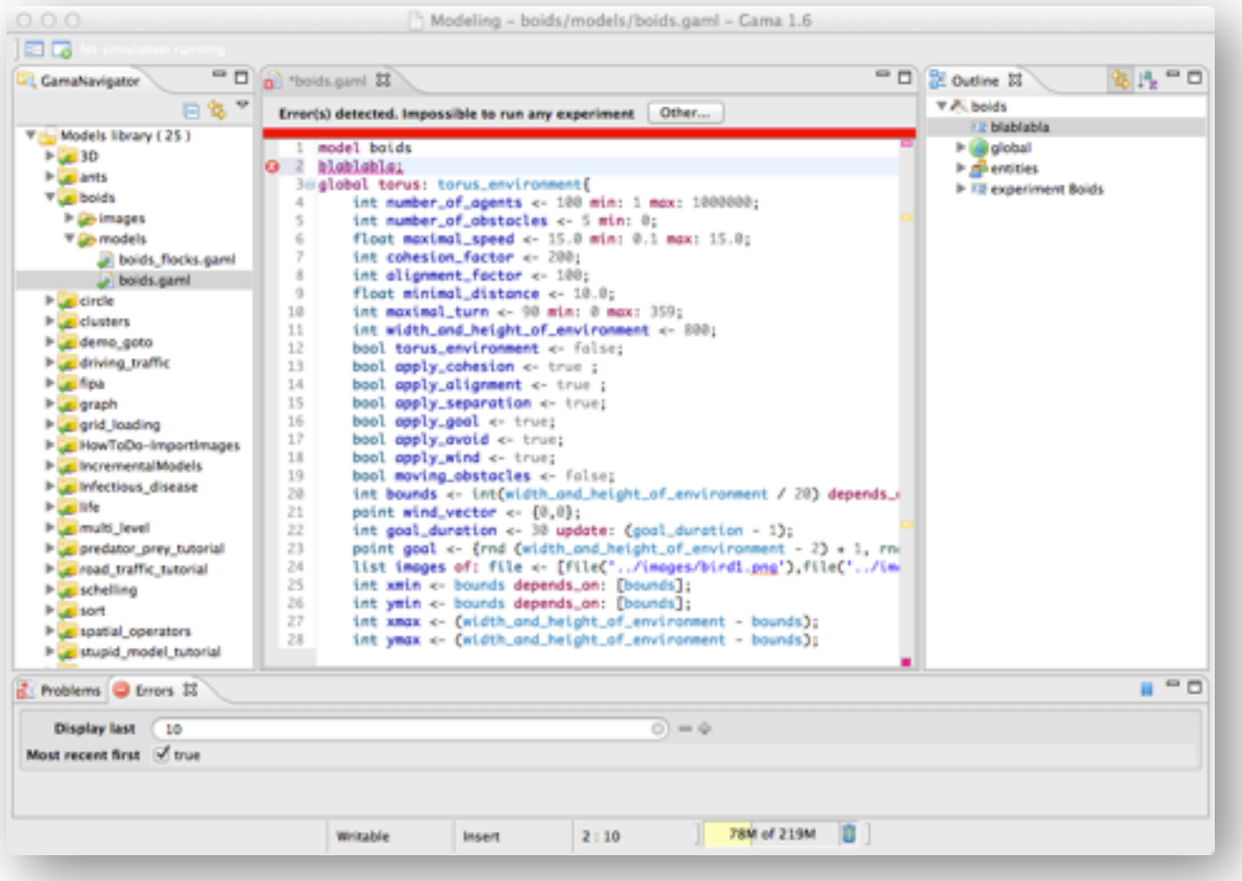
Loading an experiment

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

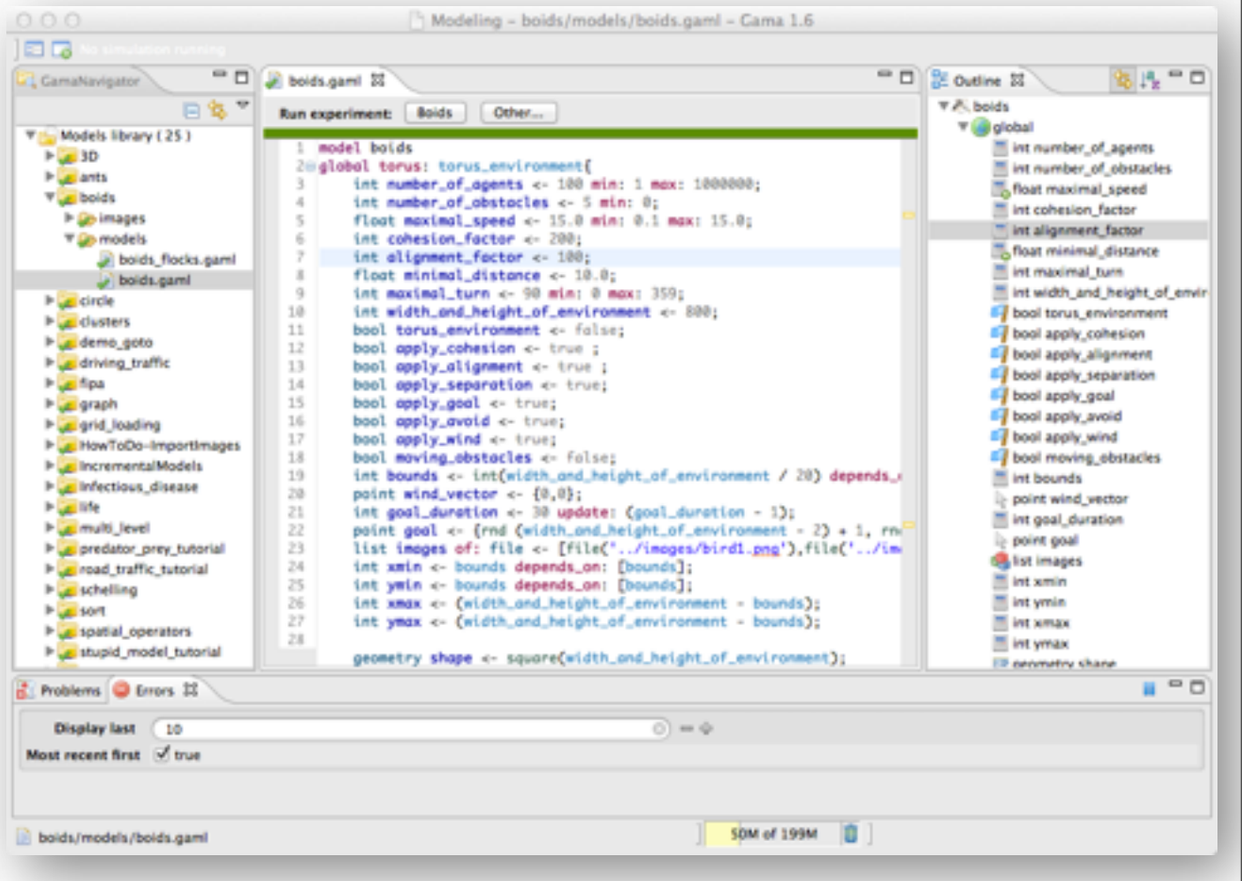


Loading an experiment: model errors

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

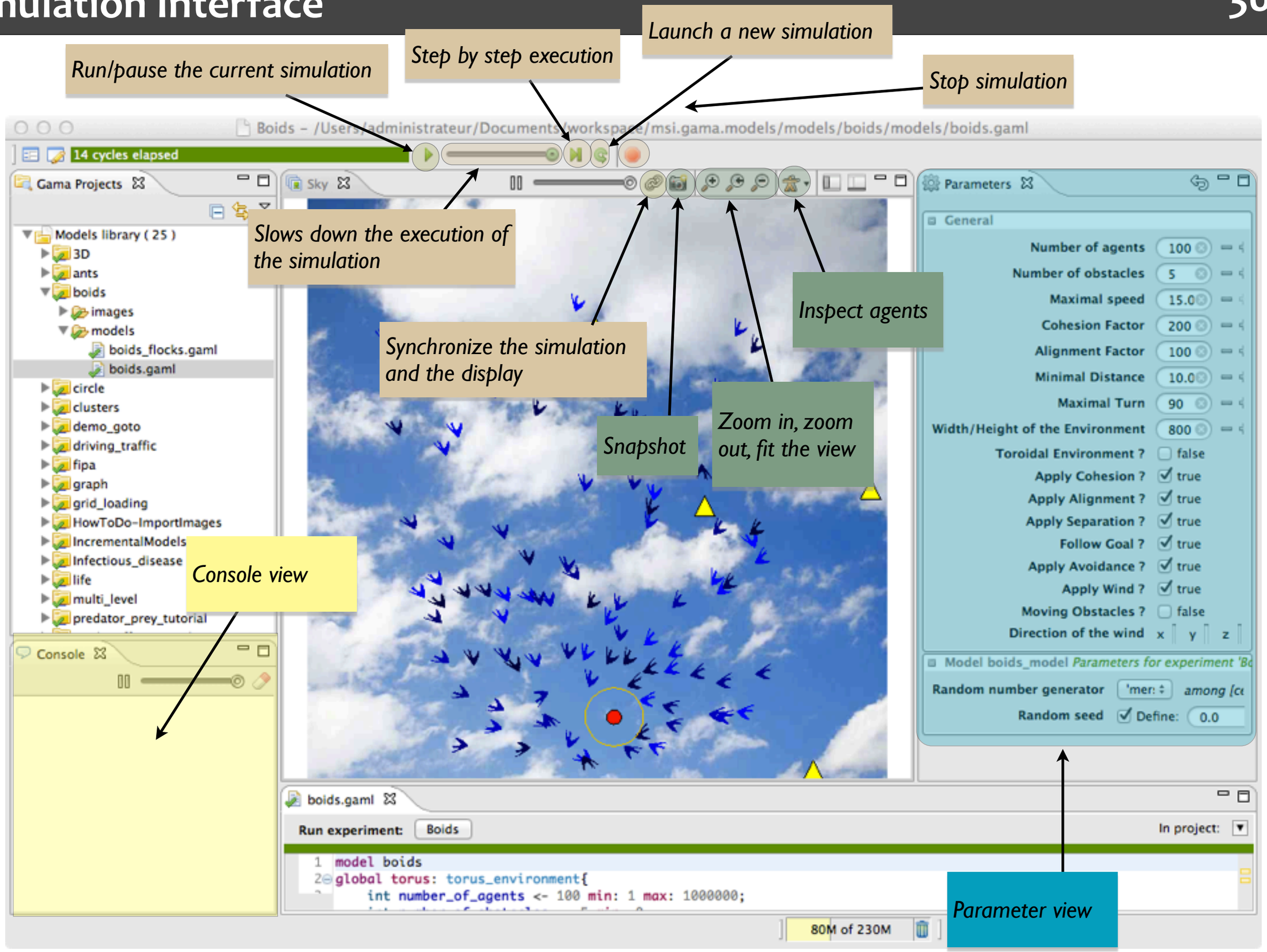


Red: Model with error(s)

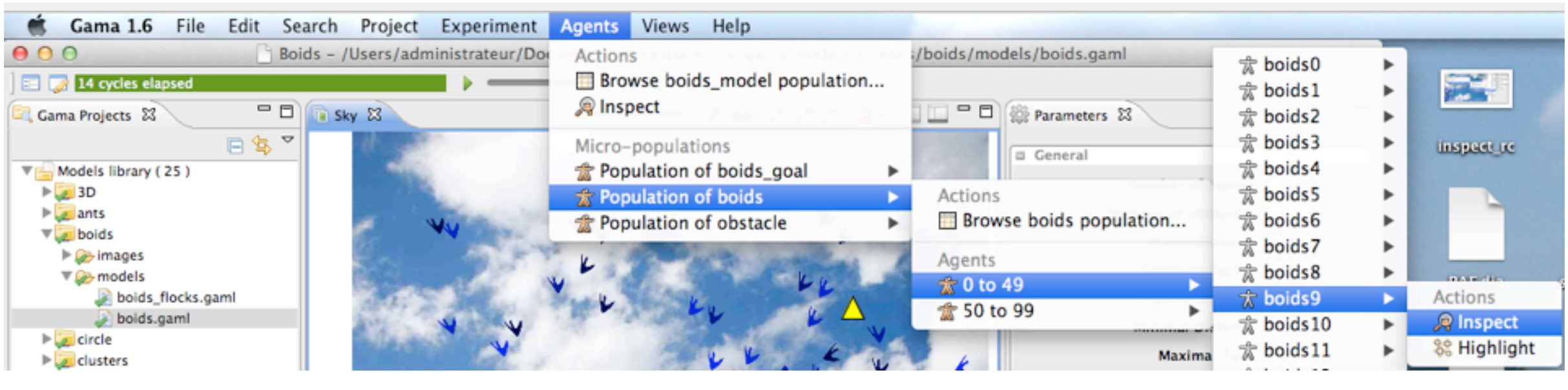


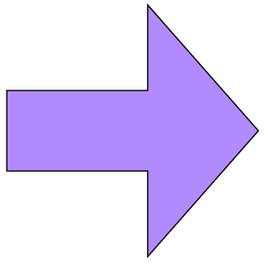
Green: Model without error(s) : ready to load !

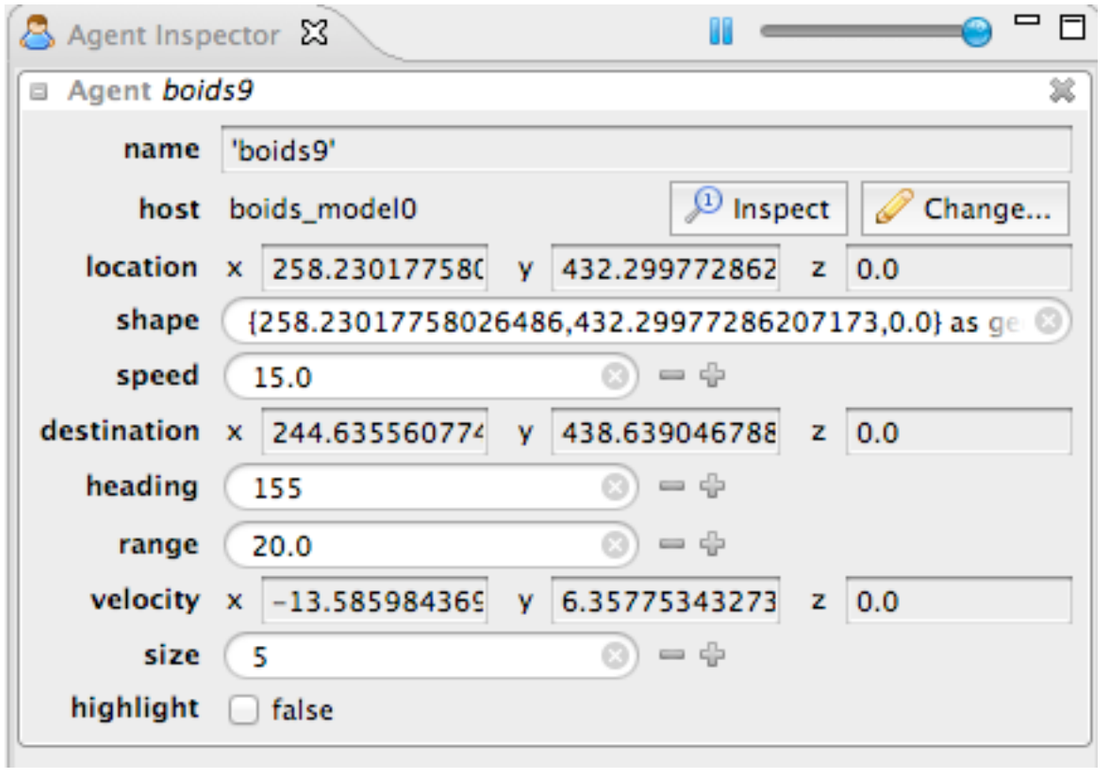
Simulation Interface



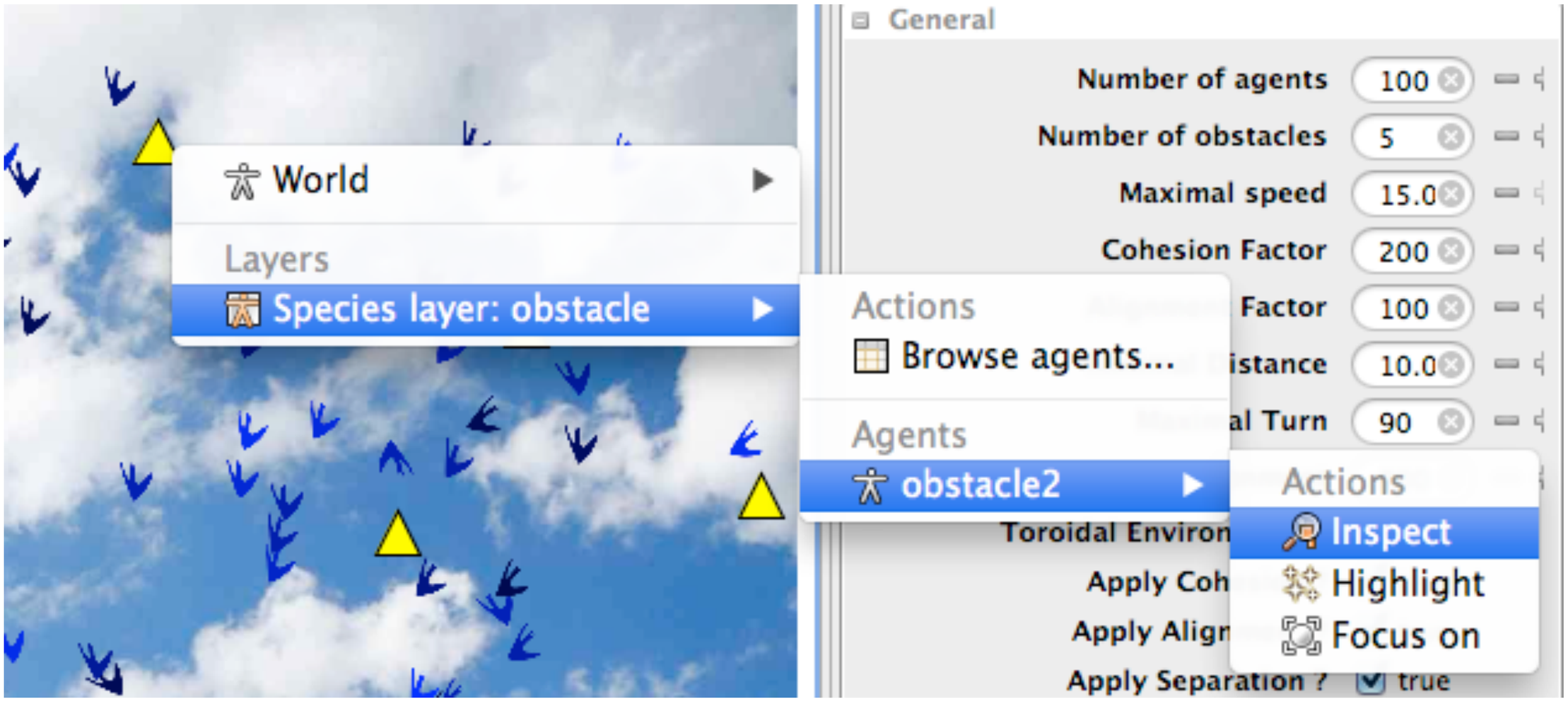
- **Inspector**: allows to obtain information on agents



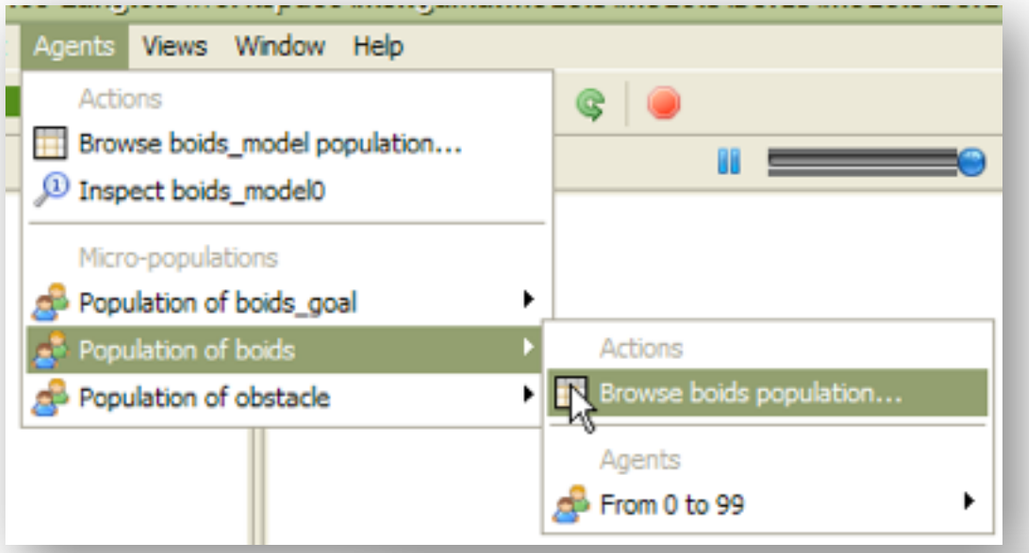


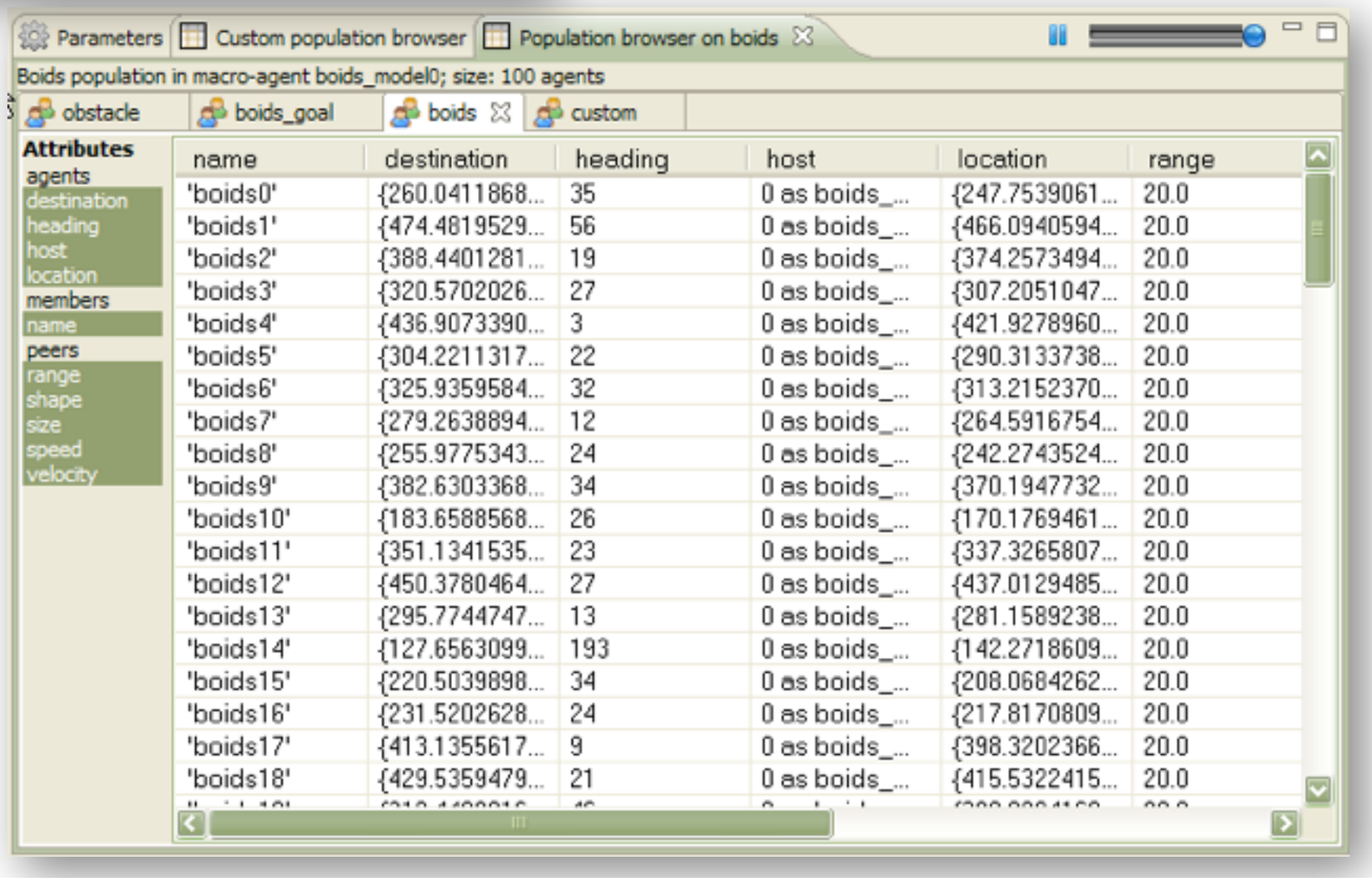


- **Inspector**: allows to obtain information on agents
  - It is possible to inspect an agent by right clicking on it (in a display)



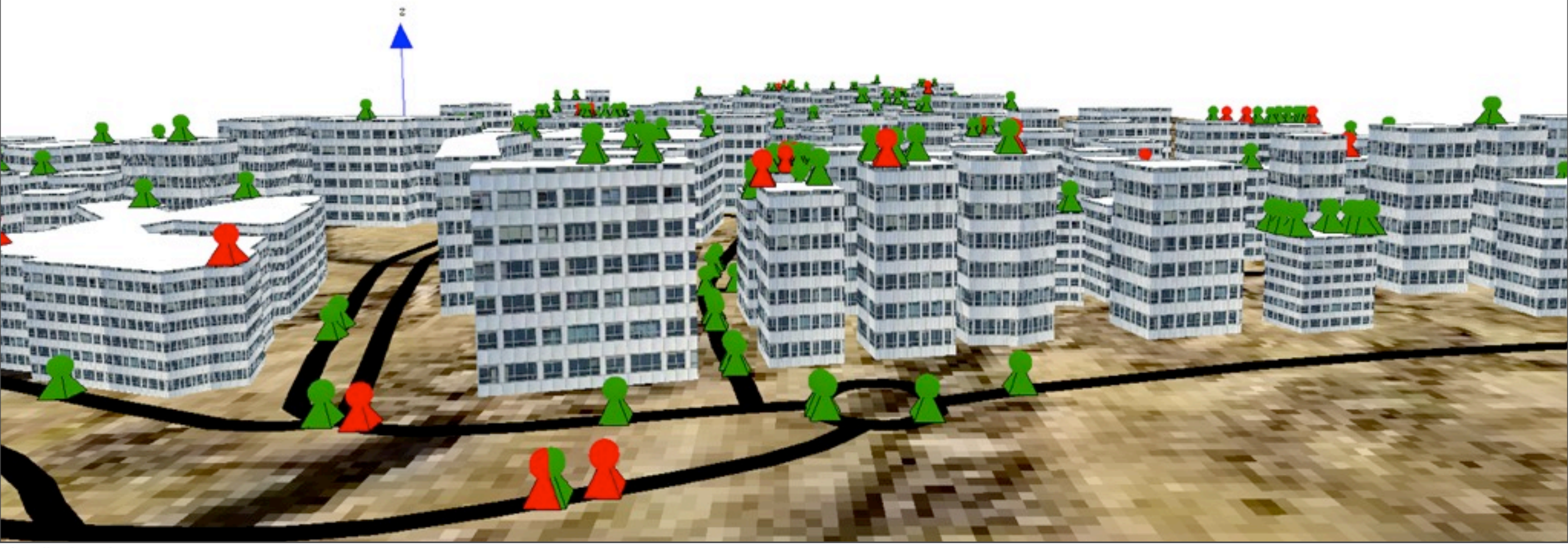
- ***Agent browser***: gives information on all agents of a specific species





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

Every year, the abms flu is spreading in the small city of Olppletto in Corsica !



step 1: define the modeling questions

step 2: identify the elements (entities, dynamics) to model

step 3: collect data

step 4: define the agents (characteristics, dynamics)

step 5: implement the model

step 6: calibrate the model

step 7: explore the model

Step 1: modeling questions

Questions to answer

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

step 1: define the modeling questions

step 2: identify the elements (entities, dynamics) to model

step 3: collect data

step 4: define the agents (characteristics, dynamics)

step 5: implement the model

step 6: calibrate the model

step 7: explore the model

Step 2: identification of the elements to model

- Space scale
  - City of Olppletto, area = close to 500m x 500m
- Time scale
  - Several days
- 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 1: define the modeling questions

step 2: identify the elements (entities, dynamics) to model

step 3: collect data

step 4: define the agents (characteristics, dynamics)

step 5: implement the model

step 6: calibrate the model

step 7: explore the model

Step 3: data collection

Data available:

- Shapefile of the roads

- Shapefile of the buildings

- Number of inhabitants of the Olppletto: 300
- Each inhabitant of the city has at a least one friend (type of social network: scale free)
- Mean speed of the inhabitants (while moving on the road) : 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: 2 meters
- Infection probability (when two people are at infection distance) : 0.05/minute

step 1: define the modeling questions

step 2: identify the elements (entities, dynamics) to model

step 3: collect data

step 4: define the agents (characteristics, dynamics)

step 5: implement the model

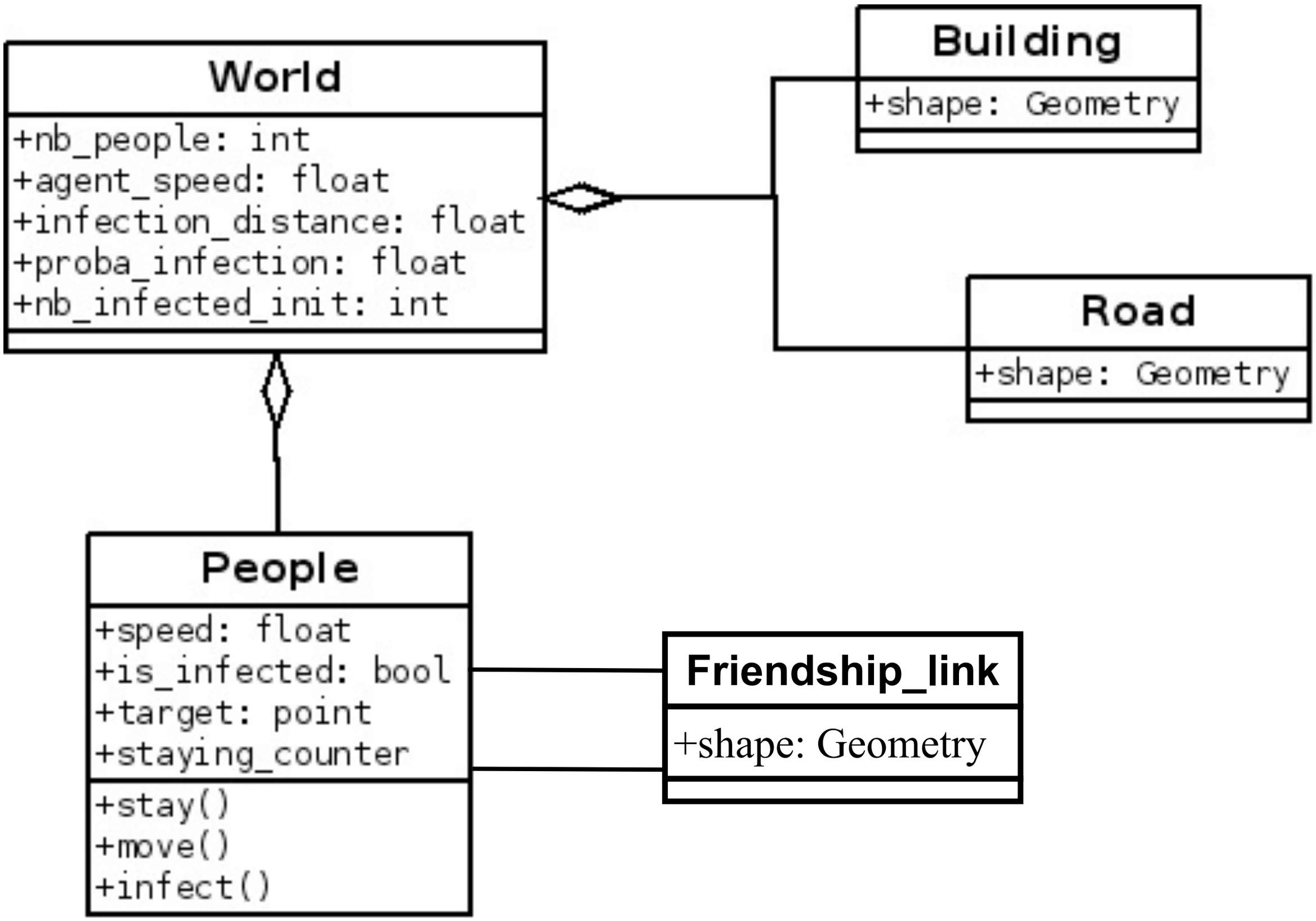
step 6: calibrate the model

step 7: explore the model

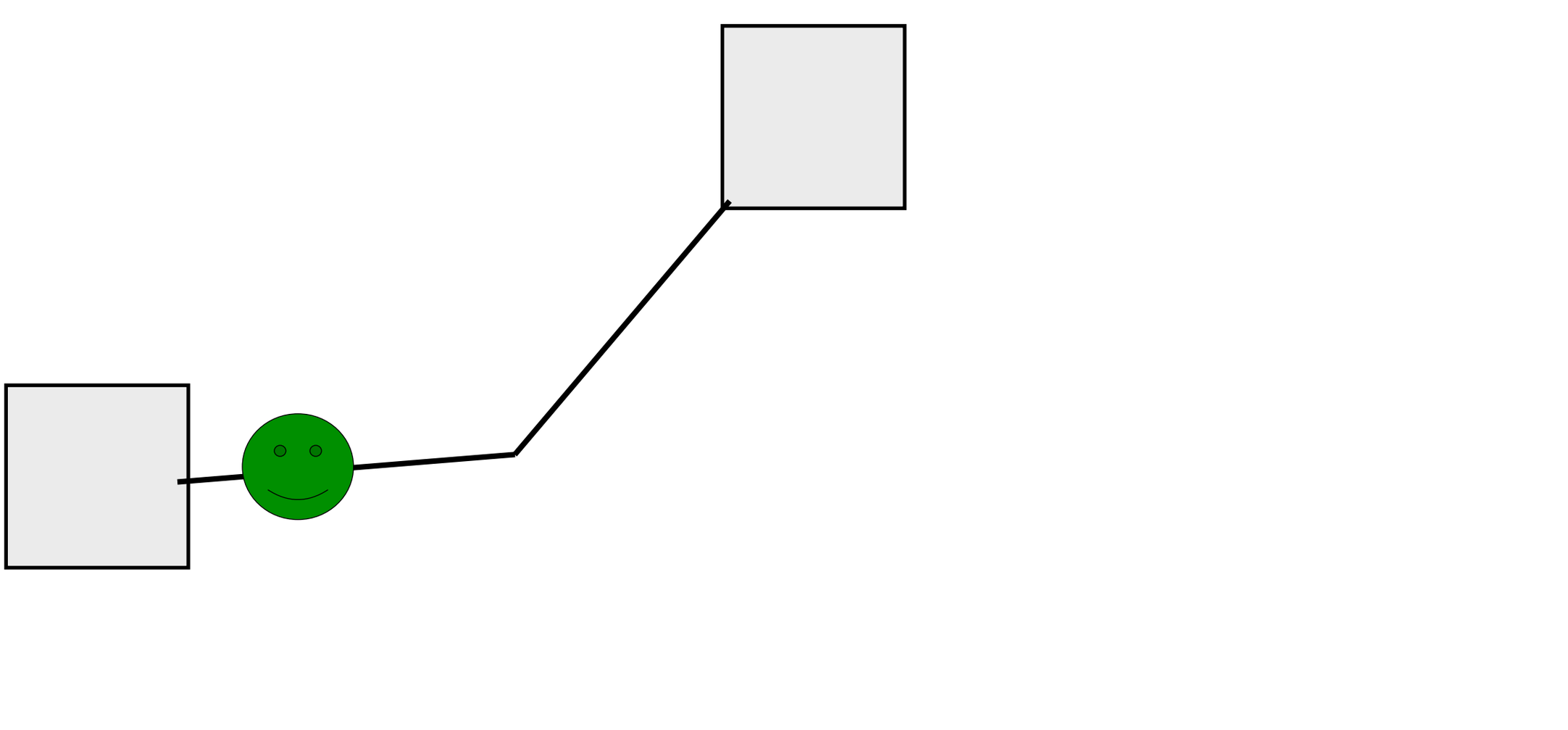
Step 4: Modeling choice

Modeling choices:

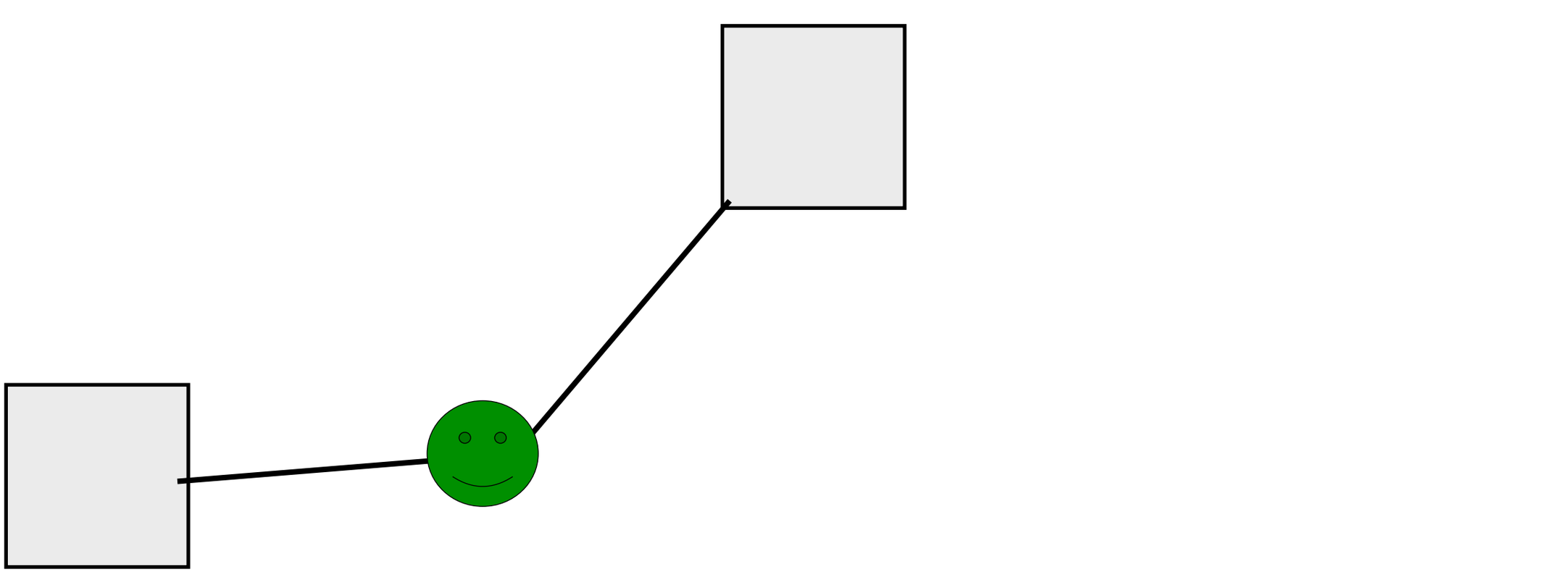
- Simulation step: 1 minute
- People are moving on the roads from building to building,
- Most of time people are moving to meet their friend (i.e. moving toward the building that is the closest to their friend), but not always
- People use the shortest path to move between buildings
- All people have the same speed and move at constant speed
- Each time, people arrived at a building they are staying a certain time
- The staying time depends on the current hour (lower at 9h - go to work - at 12h go to lunch - at 18h - go back home)
- Infected people are never cured


World
+nb_people: int
+agent_speed: float
+infection_distance: float
+proba_infection: float
+nb_infected_init: int
Building
+shape: Geometry
Road
+shape: Geometry
People
+speed: float
+is_infected: bool
+target: point
+staying_counter
+stay()
+move()
+infect()
Friendship_link
+shape: Geometry

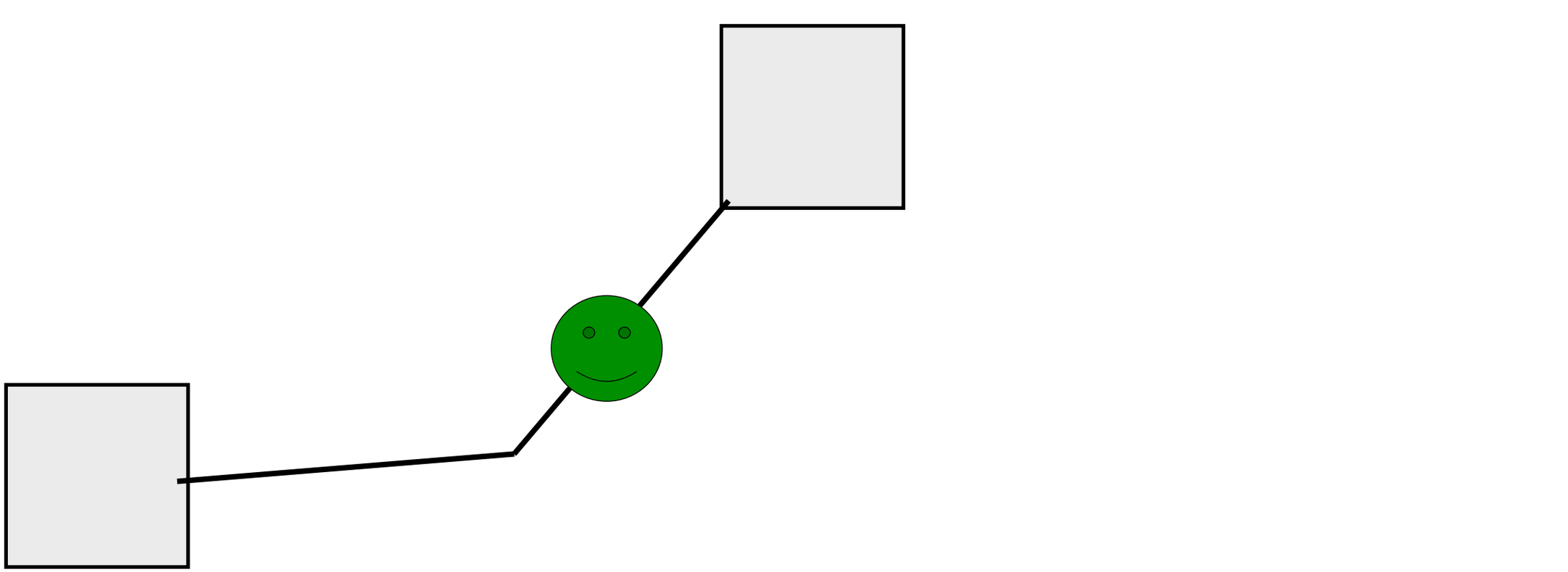
Moving dynamic



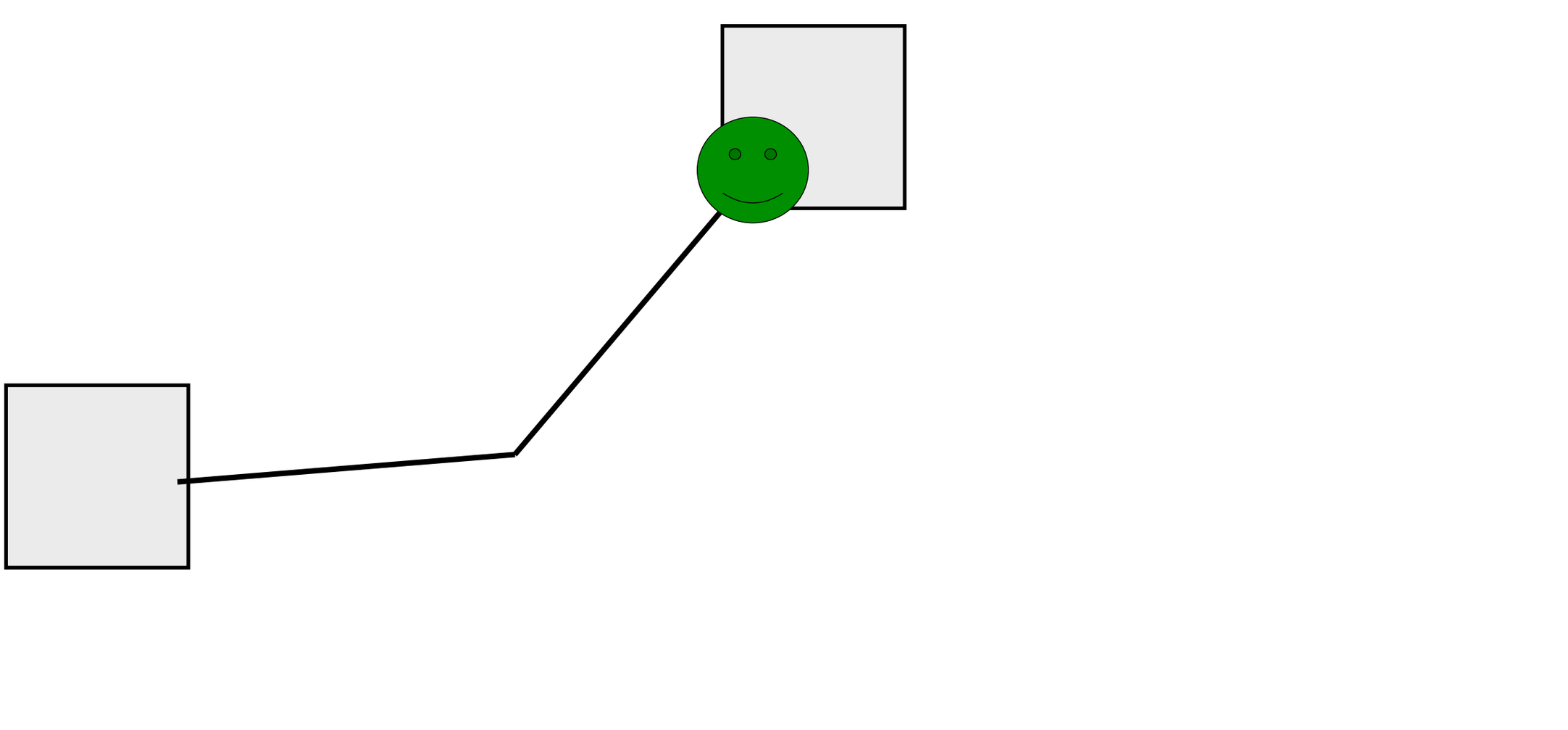
Moving dynamic



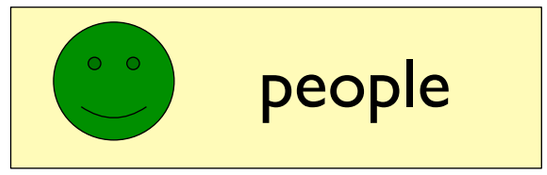
Moving dynamic



Moving dynamic

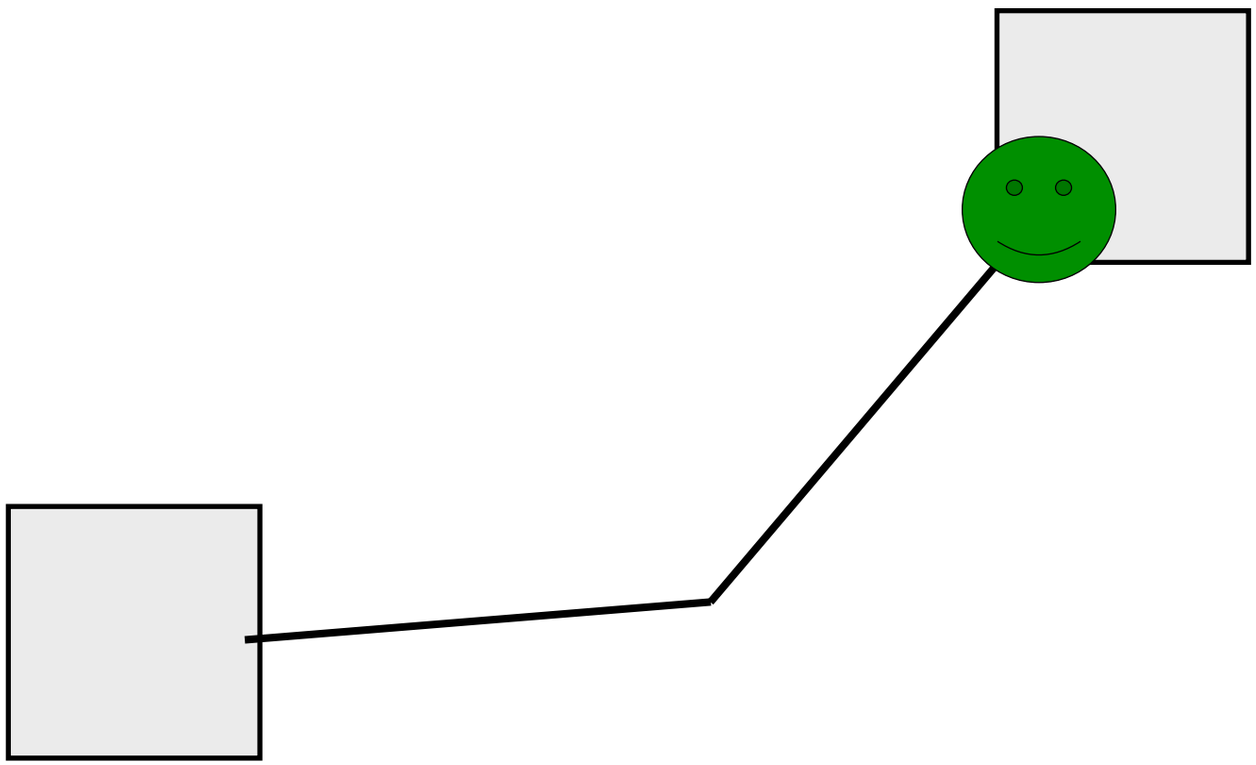


Moving dynamic

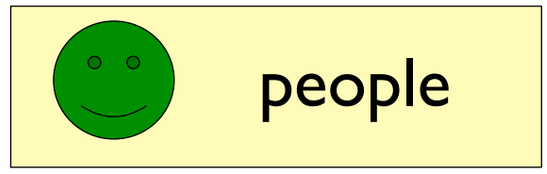


Stay for a certain time: at each simulation step, probability to leave:
staying_counter / *staying_coeff*

with *staying_coeff* depending on the current hour of the day

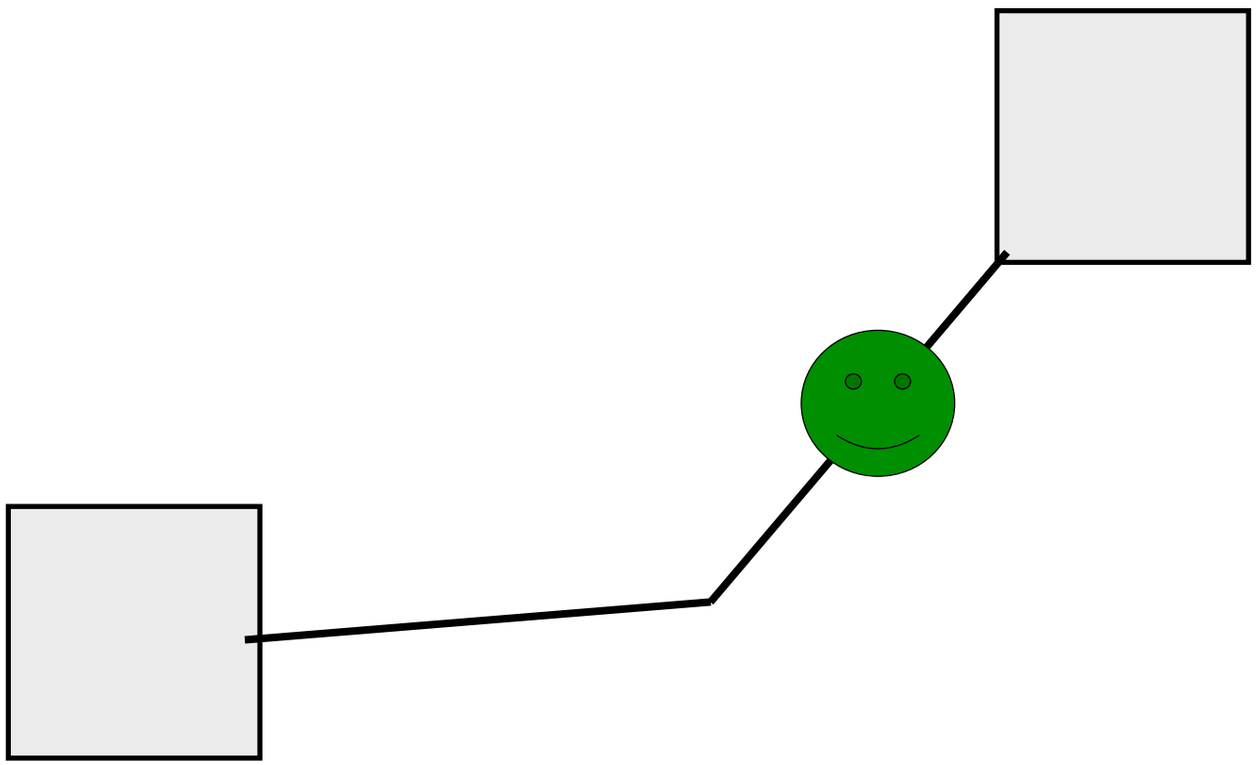


Moving dynamic

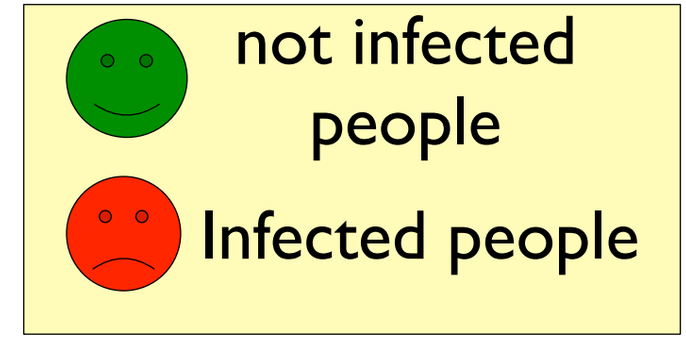


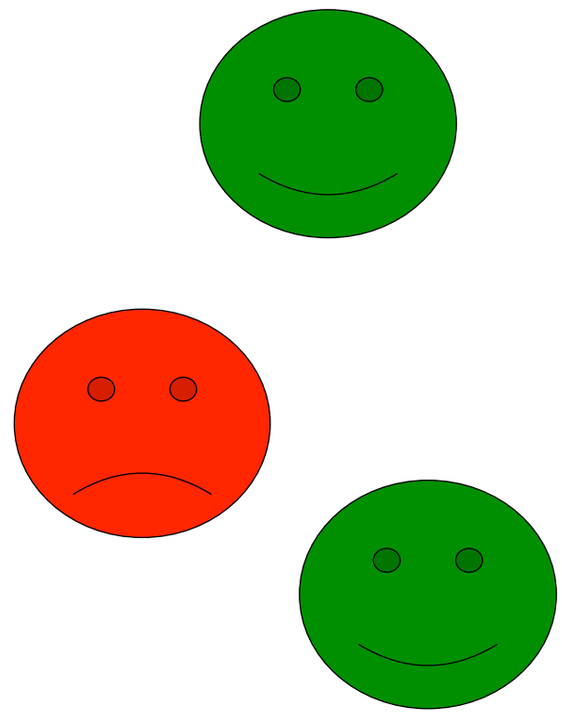
Stay for a certain time: at each simulation step, probability to leave:
staying_counter / *staying_coeff*

with *staying_coeff* depending on the current hour of the day

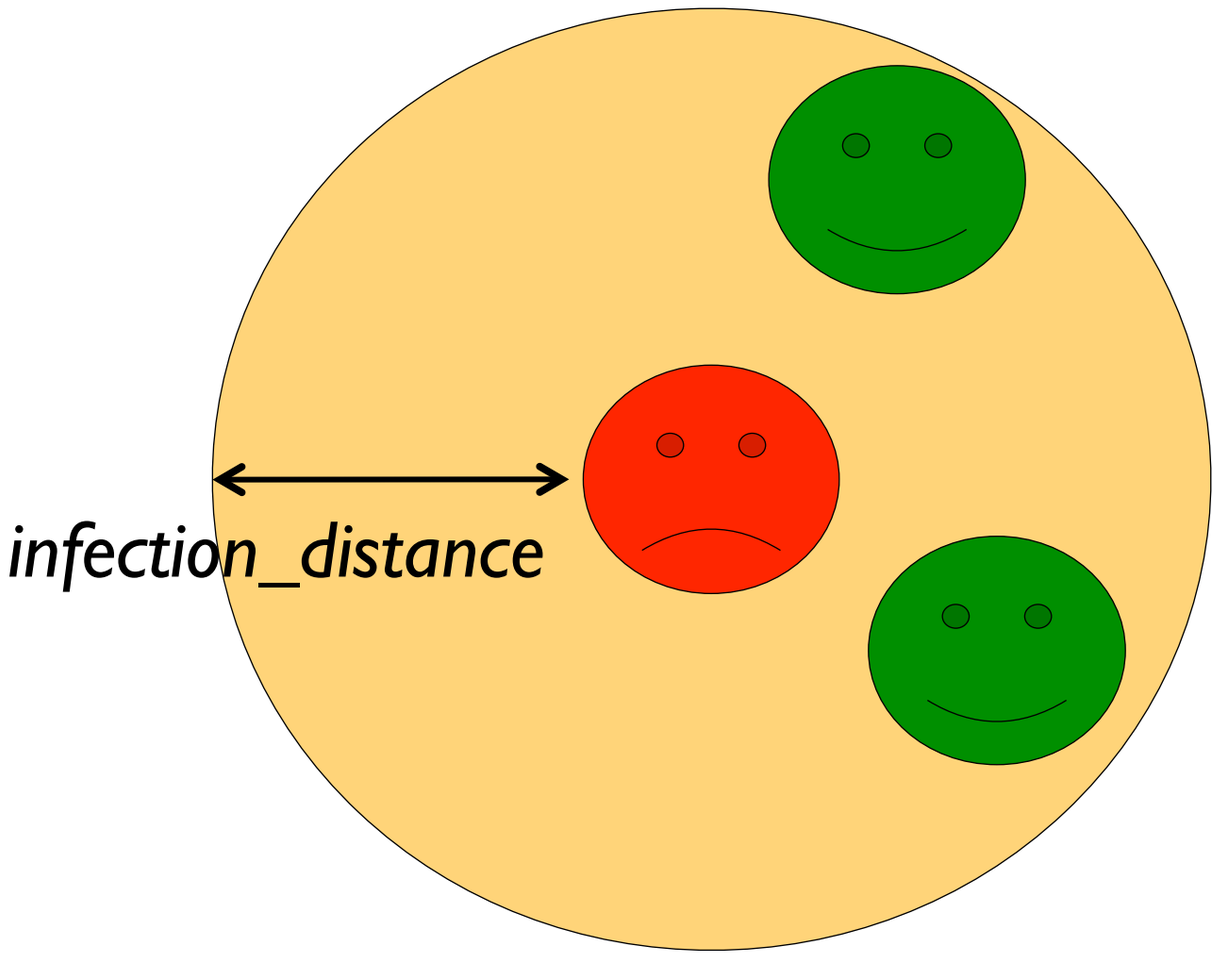


Infection dynamic

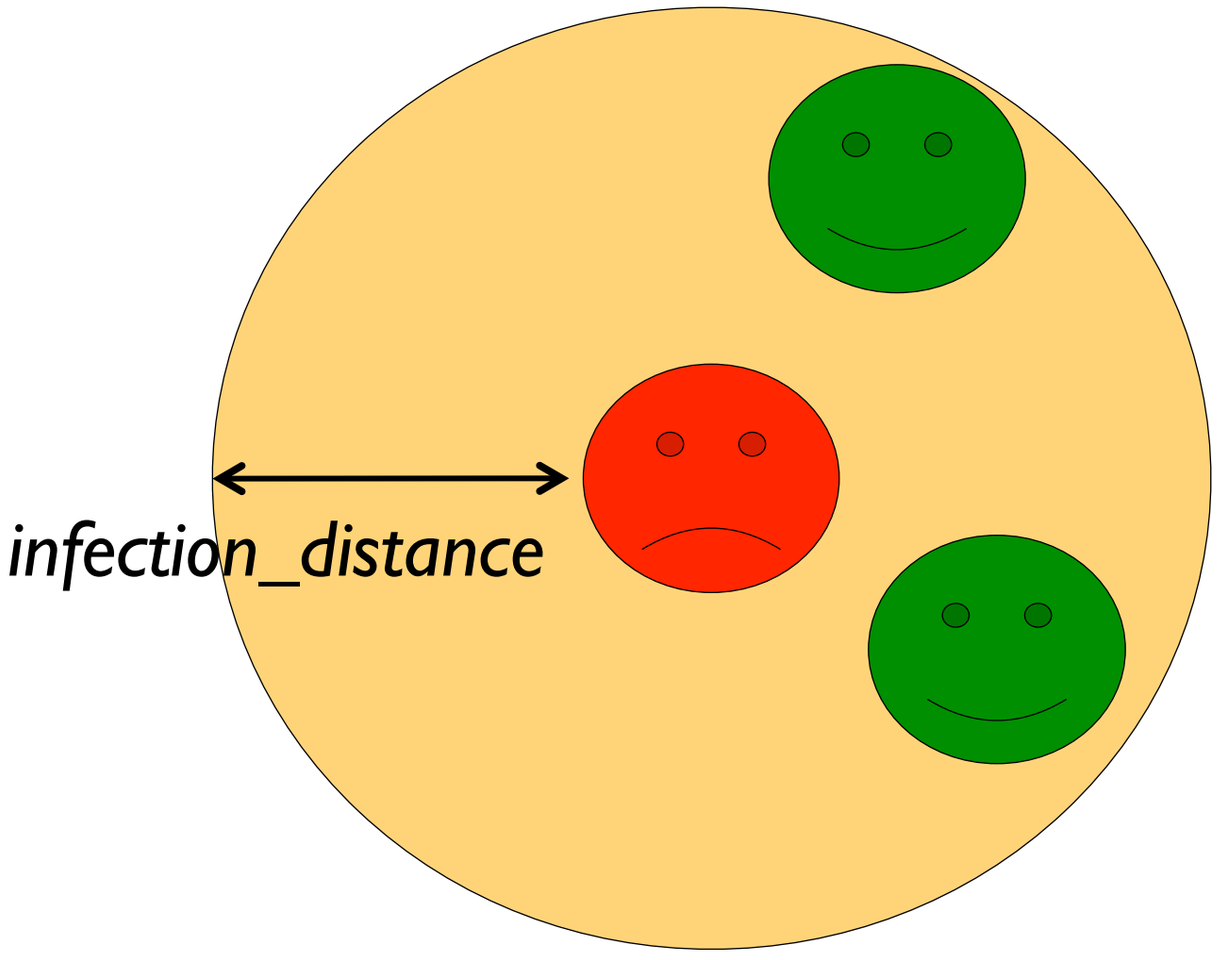




Infection dynamic



Infection dynamic

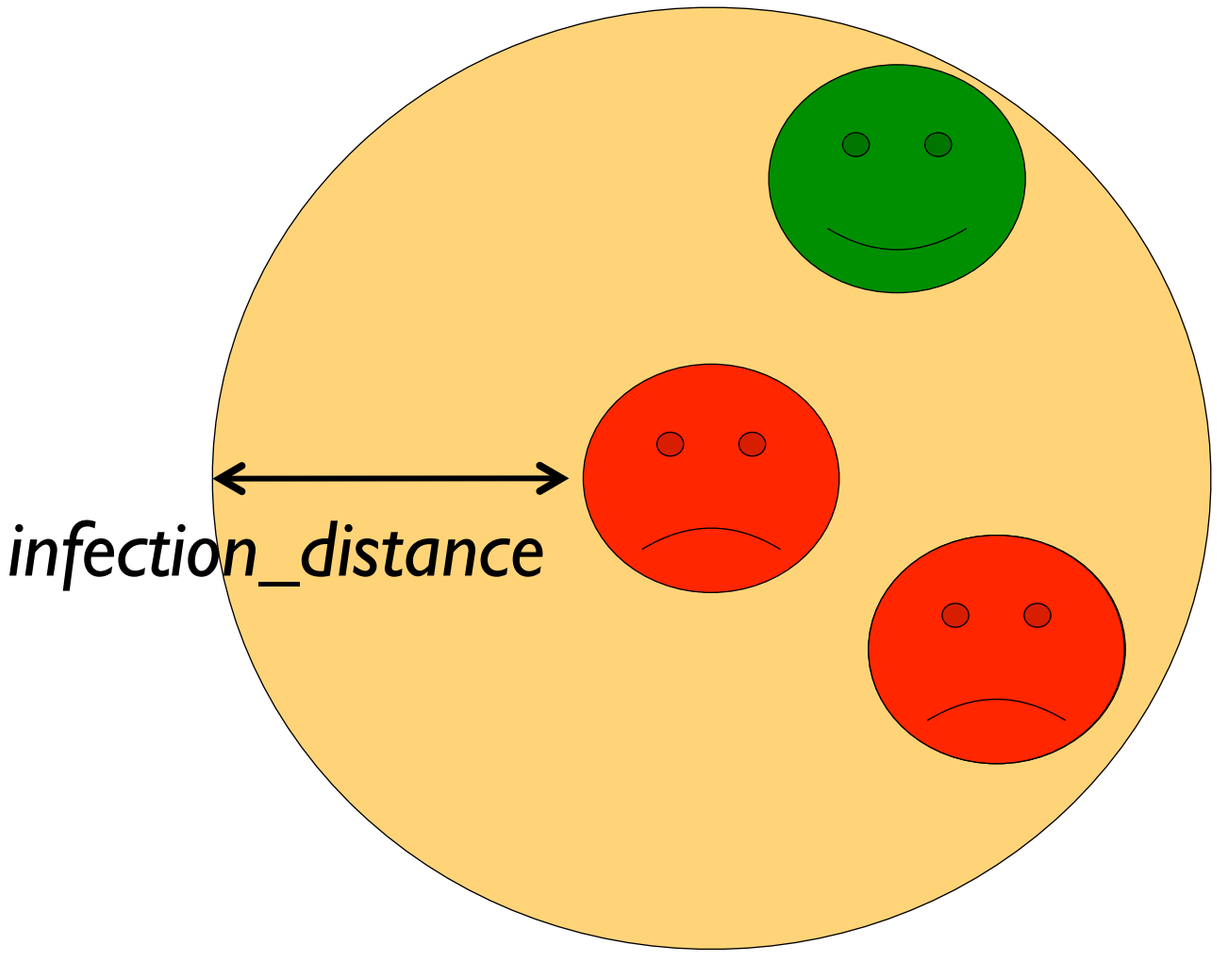


not infected people

Infected people

For each potential victims, probability to be infected: *proba_infection*

Infection dynamic



For each potential victims, probability to be infected: *proba_infection*

step 1: define the modeling questions

step 2: identify the elements (entities, dynamics) to model

step 3: collect data

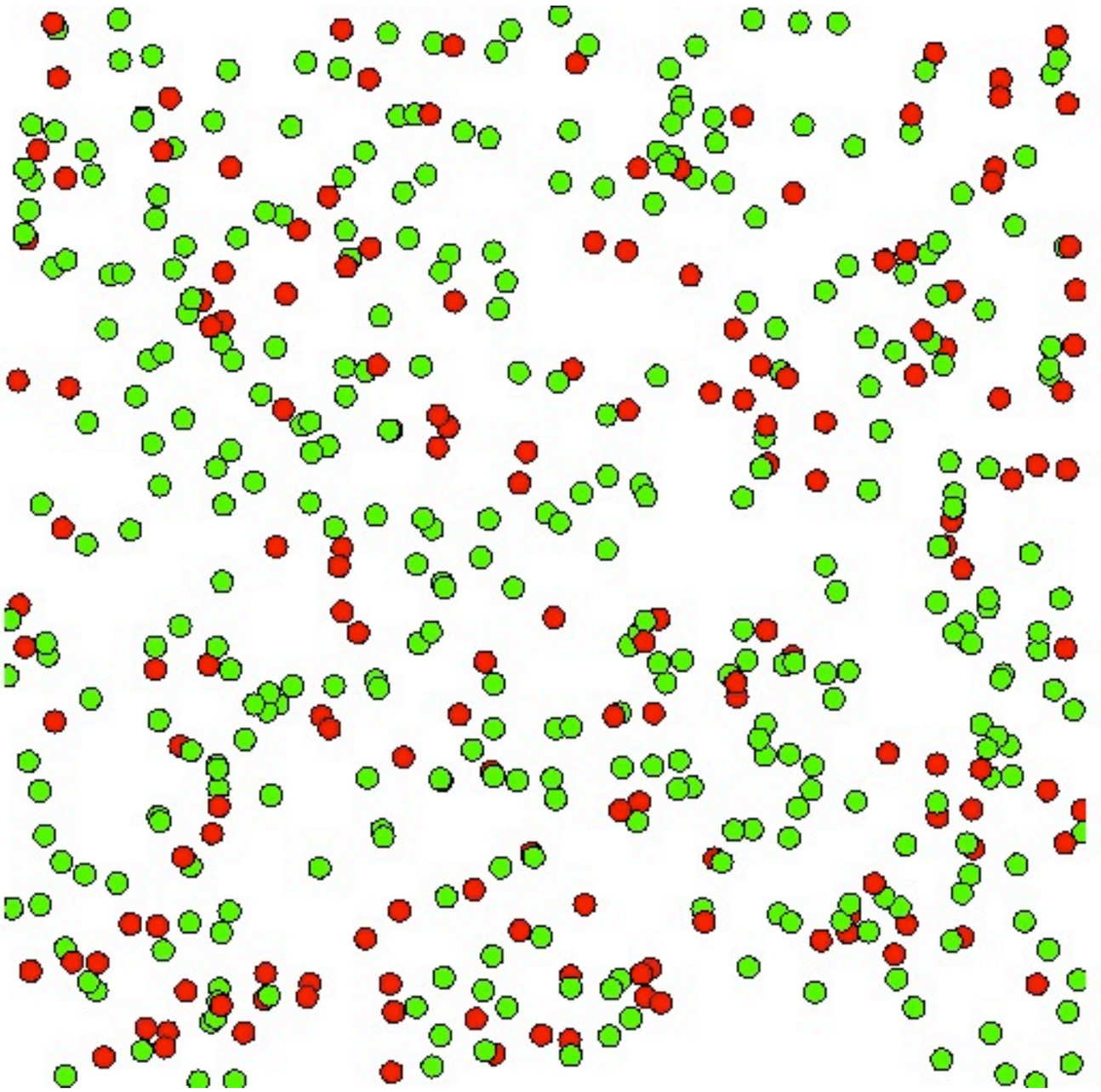
step 4: define the agents (characteristics, dynamics)

step 5: implement the model

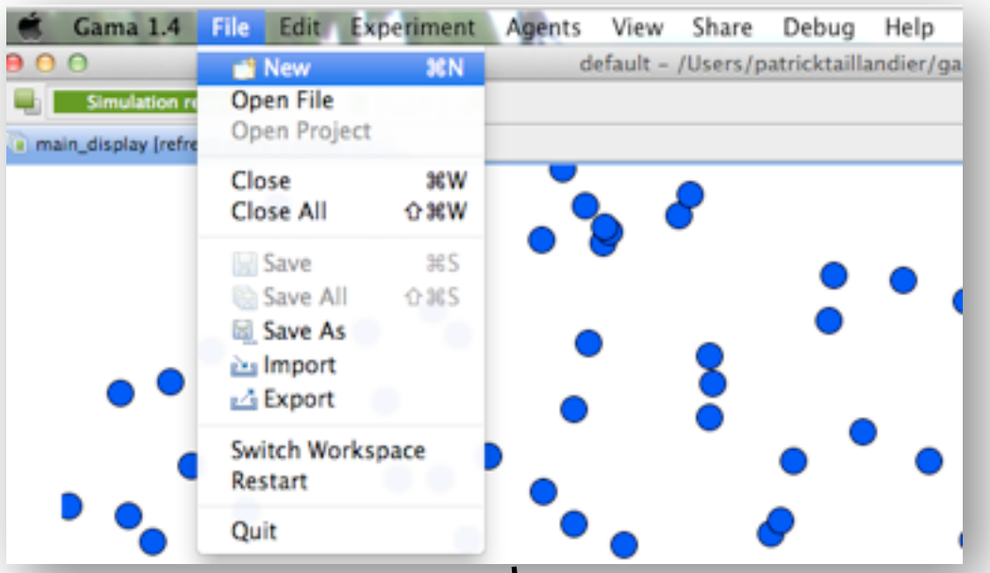
step 6: calibrate the model

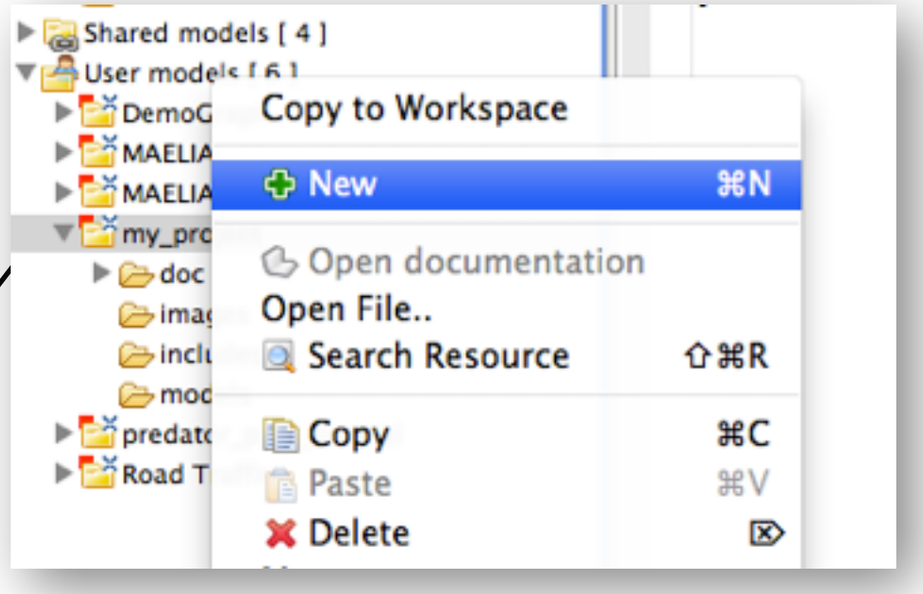
step 7: explore the model

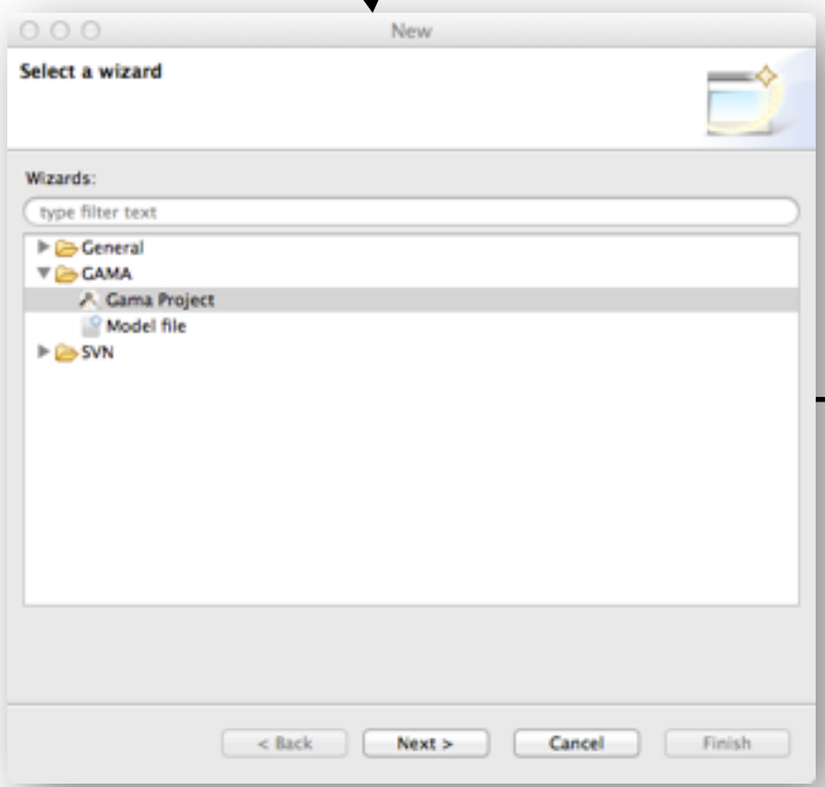
Definition of the structure of the model and of the basic behaviors of the people agents

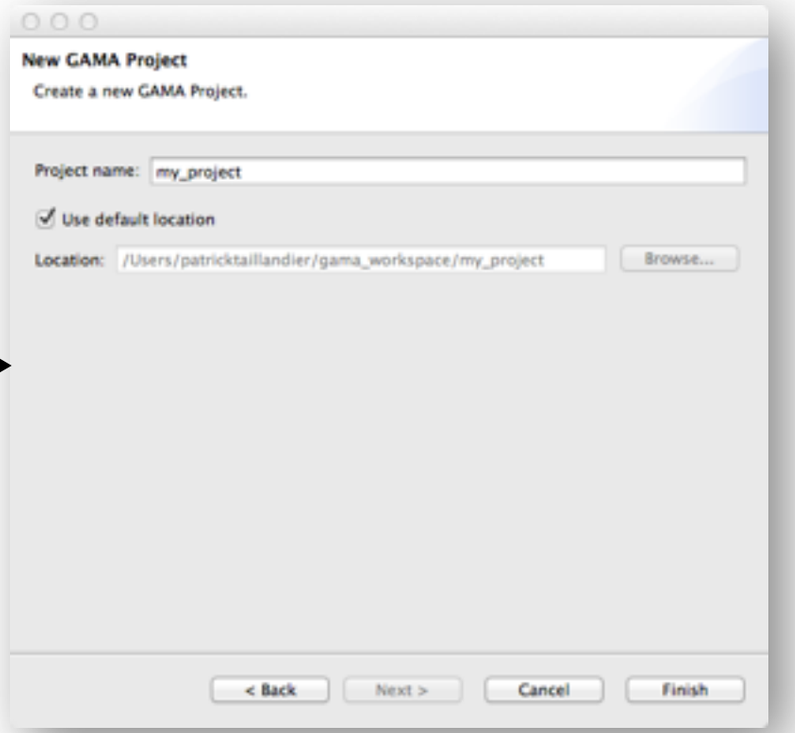


Creation of a new project

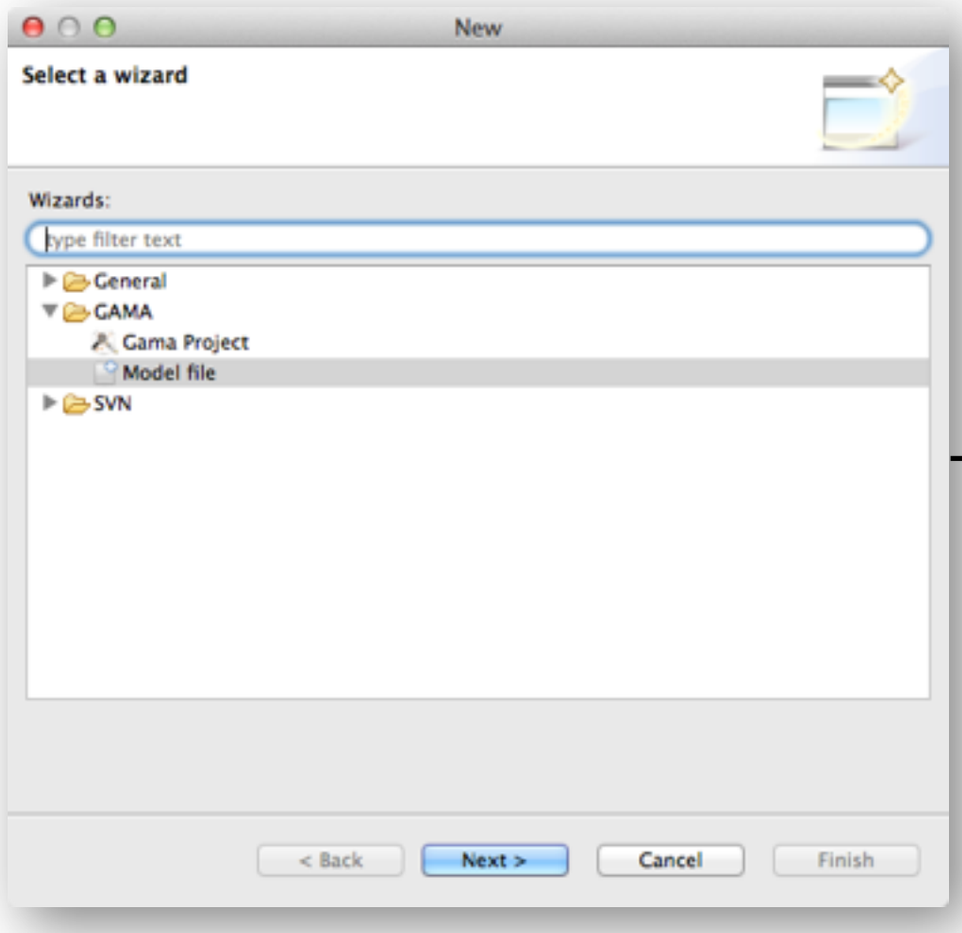


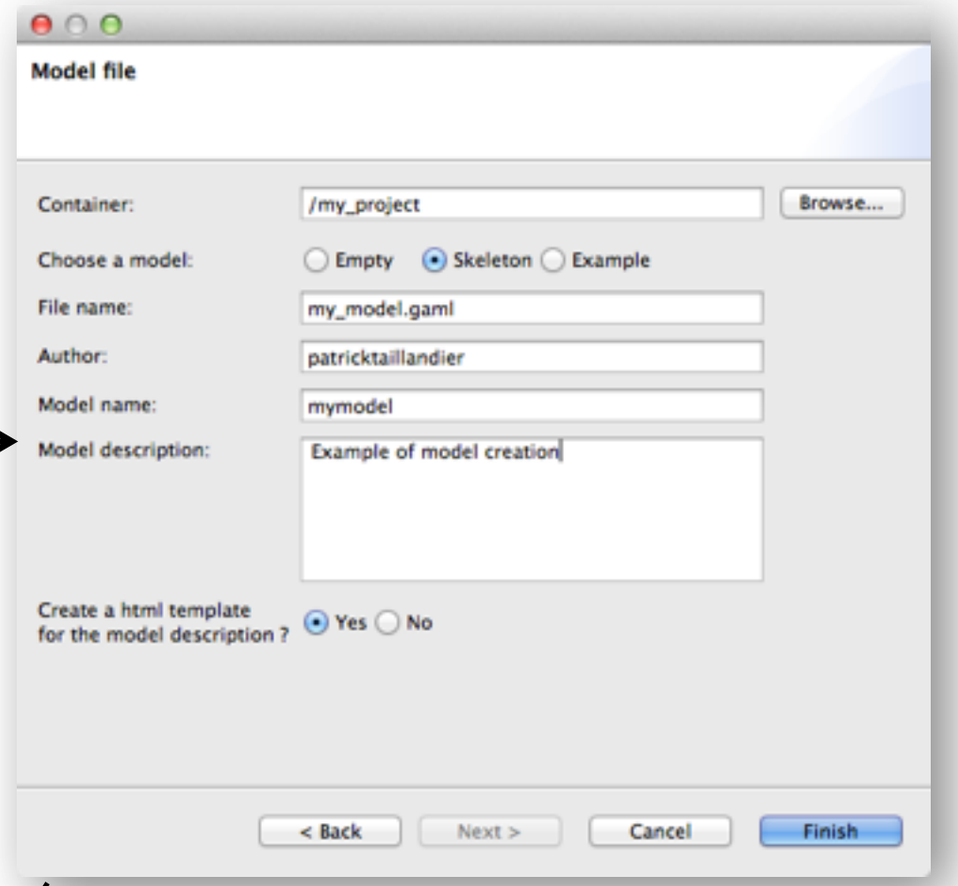






Creation of a new model






```
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 {
	}
}
```

- **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
     */
}
```

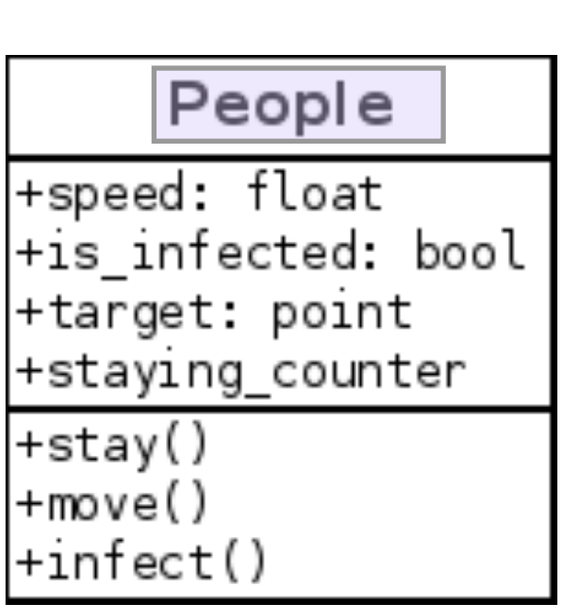
2 ways to write commentaries (texts that are not just part of the model but here for information purpose):

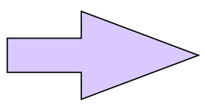
- //… : for one line. Example : //this is a commentary
- /* … */ : can be used for several lines. Example : /* this is as well a commentay */

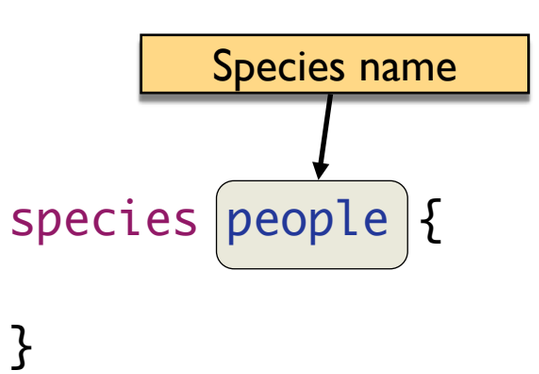
- **To define a species, 3 elements have to be defined :**
  - the internal state of the agents (attributes - variables)
  - their behaviors (methods)
  - their display (aspects)

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : People species







- An agent can have *skills*
- Un *skilll* 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*)

species block: Species definition - variables 1/3

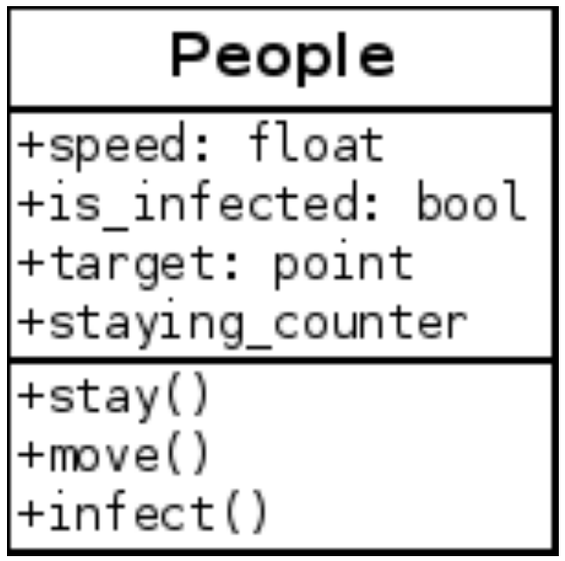
- Variable definition : ***variable type*** + *nom*
  - ***type***: int, float, string, bool (boolean, can be either *true* or *false*), point, list, pair, map, file, matrix, agent species, rgb (Red, Green, Blue - color), graph, path…
  - Optional attributes:
    - **<-** : initial value,
    - ***update*** : 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 delete 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 :

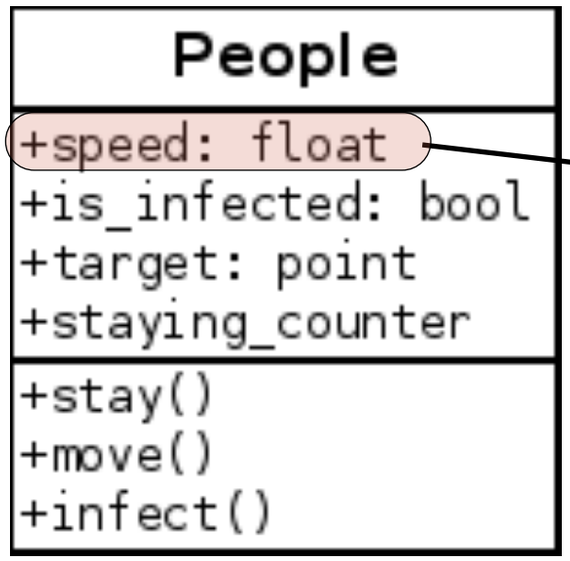
- ***name*** (*string*)
- ***shape*** (*geometry*) 
- ***location*** (*point*) : centroid of its shape

Model : People attributes



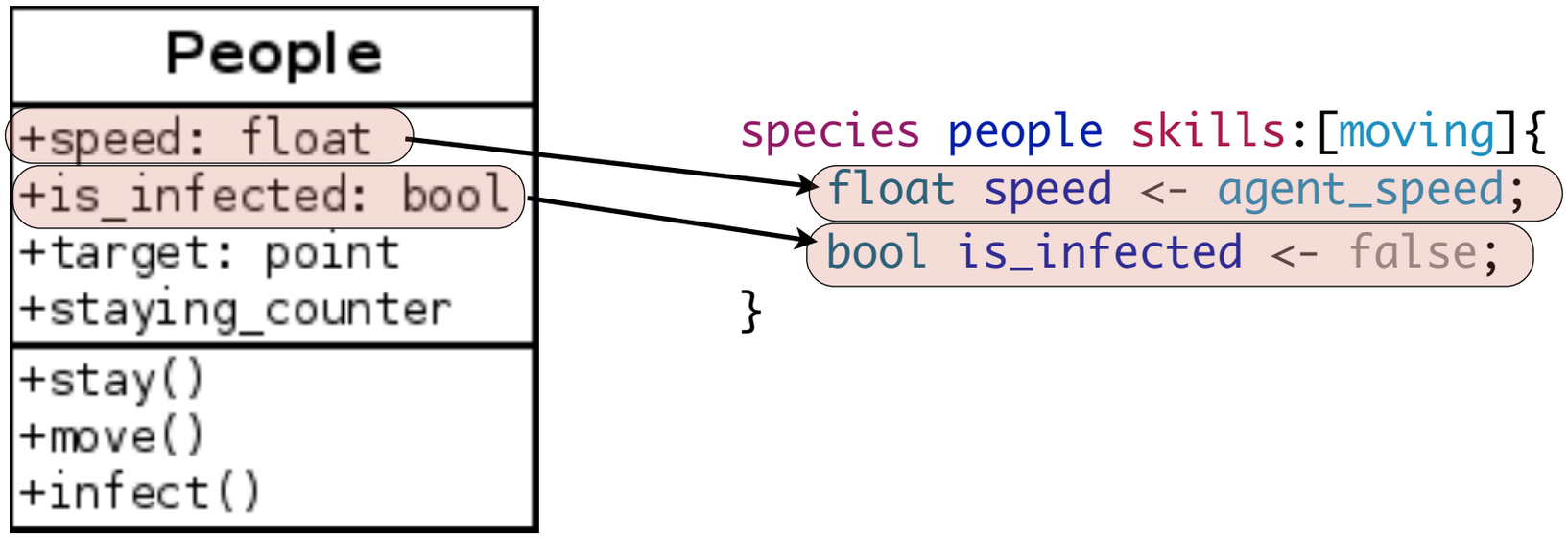

```
species people skills:[moving]{
    float speed <- agent_speed;
    bool is_infected <- false;
}
```

Model : People attributes



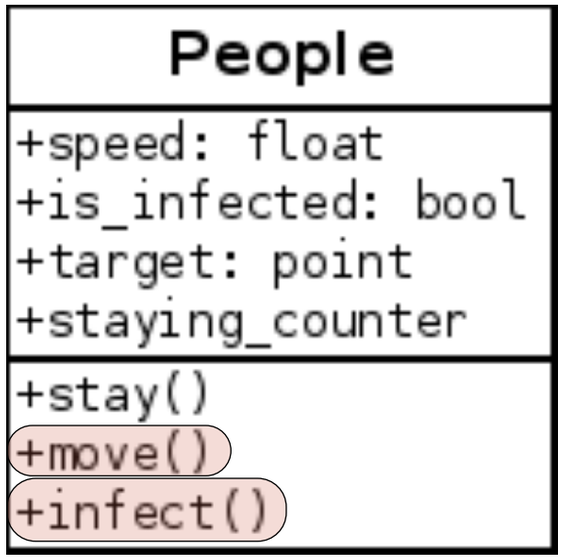

```
species people skills:[moving]{
    float speed <- agent_speed;
    bool is_infected <- false;
}
```

Model : People attributes



- 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



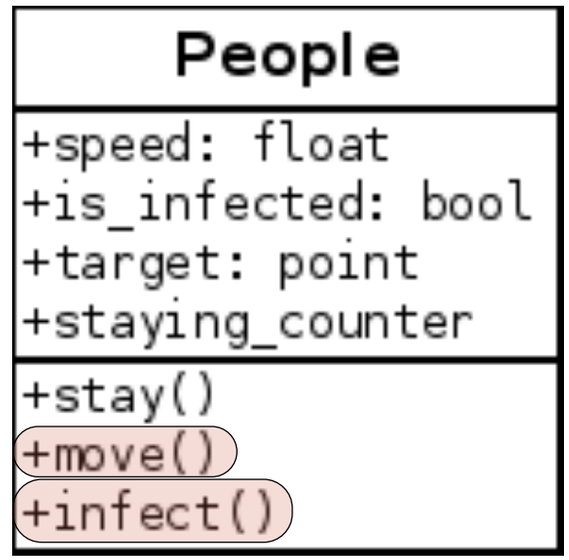

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance infection_distance {
            if flip(proba_infection) {
                is_infected <- true;
            }
        }
    }
}
```

- 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

Reflex executed at each simulation step

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance infection_distance {
            if flip(proba_infection) {
                is_infected <- true;
            }
        }
    }
}
```



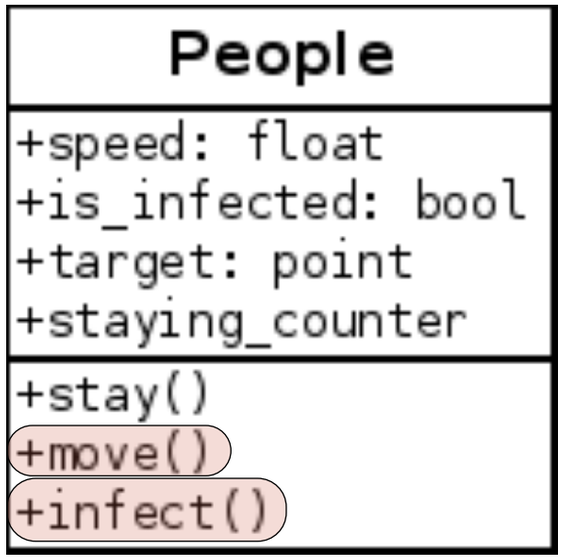
- A reflex is a statement block that will be executed if its attached condition is checked
- **reflex** *reflex_name* **when:** *execution_condition* {…}
- The **when** facet is optional: if this facet is not defined, the reflex is executed at each simulation step.

Model : People Reflex

Reflex executed at each simulation step

Application of the *wander* action of the moving *skills*

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance infection_distance {
            if flip(proba_infection) {
                is_infected <- true;
            }
        }
    }
}
```



- 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.

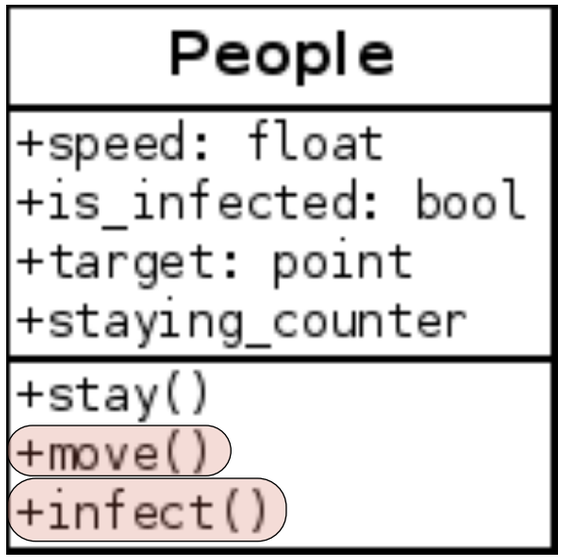
Model : People Reflex

Reflex executed at each simulation step

Application of the *wander* action of the moving *skills*

Reflex activated only when the agent is infected

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance infection_distance {
            if flip(proba_infection) {
                is_infected <- true;
            }
        }
    }
}
```



- 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

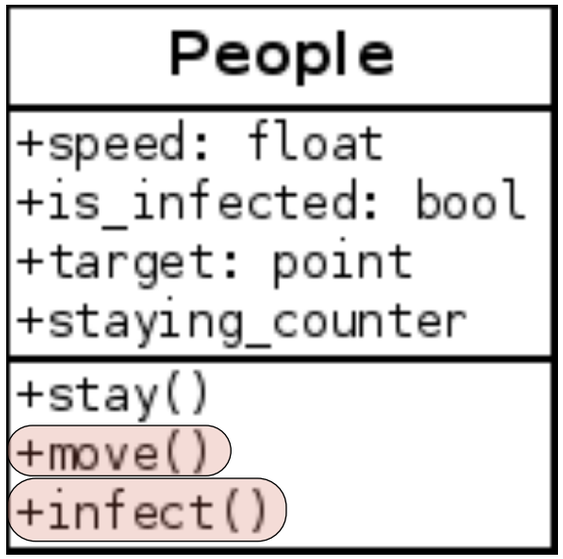
Reflex executed at each simulation step

Application of the *wander* action of the moving *skills*

Reflex activated only when the agent is infected

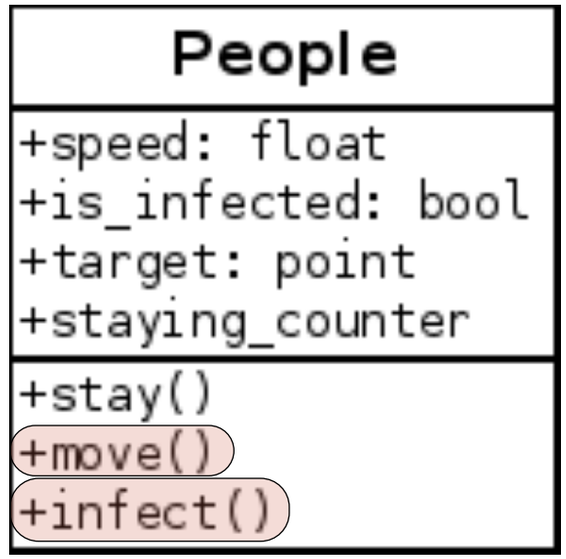
ask all the people agents located at a (euclidean) distance inferior or equal to *infection_distance*

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance infection_distance {
            if flip(proba_infection) {
                is_infected <- true;
            }
        }
    }
}
```



- 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



Reflex executed at each simulation step

Application of the *wander* action of the moving *skills*

Reflex activated only when the agent is infected

ask all the people agents located at a (euclidean) distance inferior or equal to *infection_distance*

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance infection_distance {
            if flip(proba_infection) {
                is_infected <- true;
            }
        }
    }
}
```

to draw a probability *proba_infected* to be infected

- 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(5) color:is_infected ? #red : #green;
    }
}
```

The symbol # allows to define a color

- 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(5) color:is_infected ? #red : #green;
    }
}
```

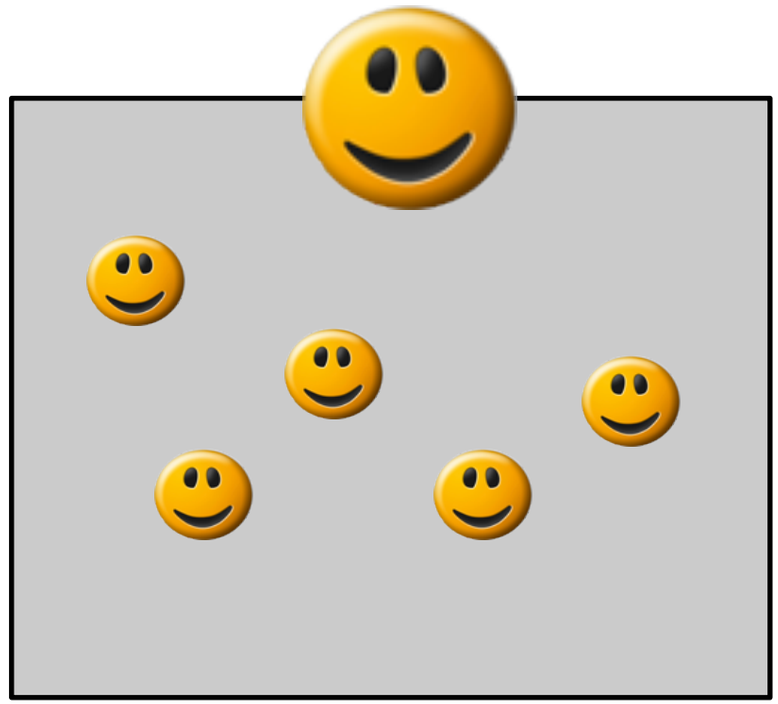
This aspect allow to display each people agent as a red or green (depending on *is_infected*) circle of radius 5m

The symbol # allows to define a color

global block

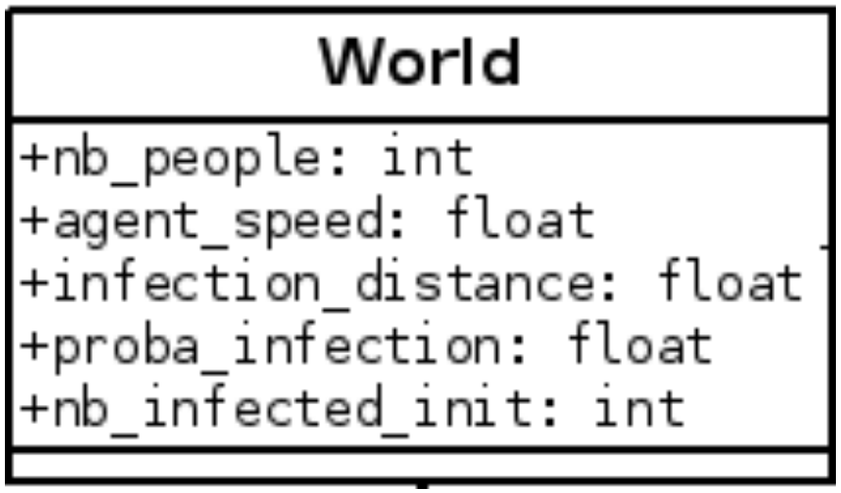
- Global block
  - Define a specific agent (called *world*)
  - Represent all that is global to the model: variables, actions, reflexes....
  - Initialize the simulation (*init* block): when the experiment button is pushed, the world agent is created and then activates its *init* block
  - The geometry 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 {
}
```

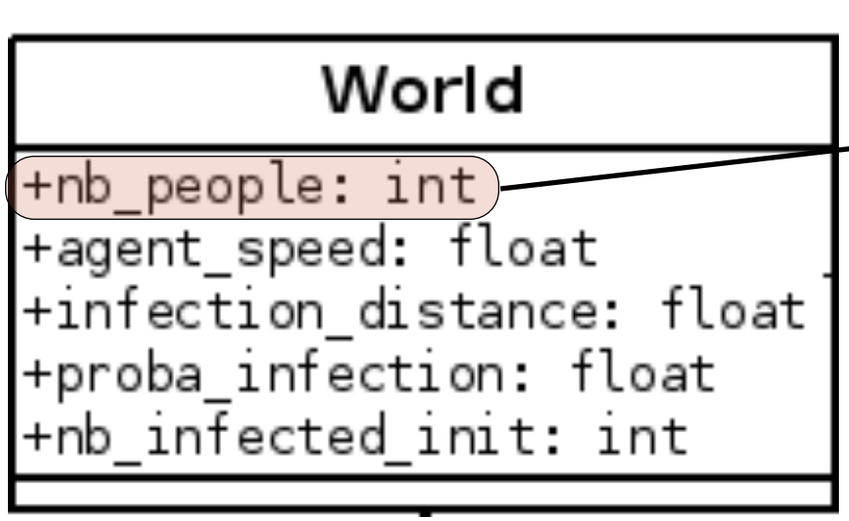



```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

The symbol # allows also to precise the unity of a value

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes

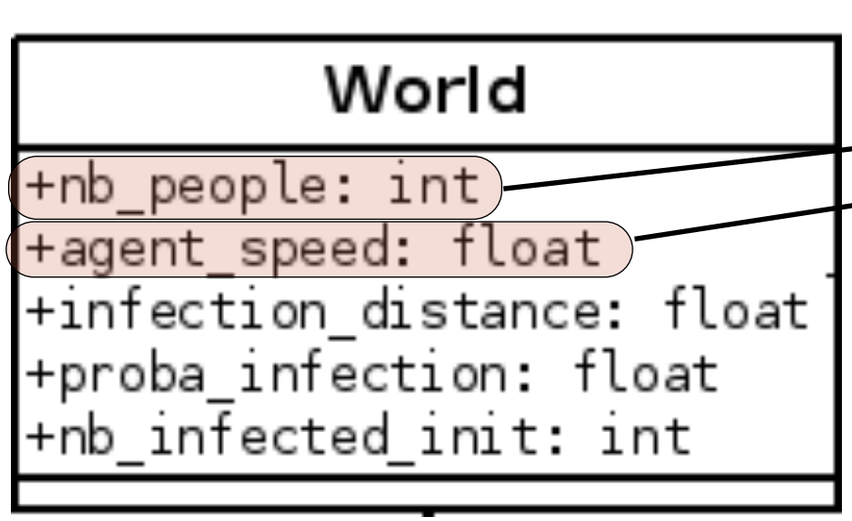



```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

The symbol # allows also to precise the unity of a value

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes

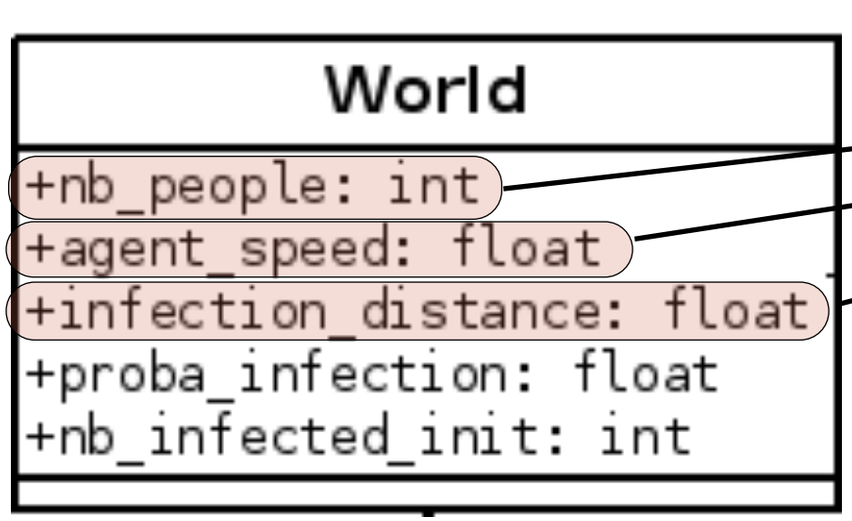



```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

The symbol # allows also to precise the unity of a value

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes

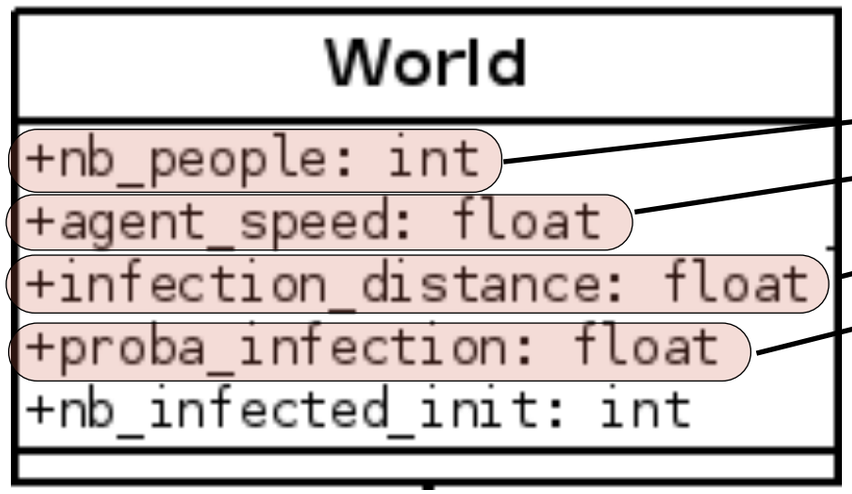



```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

The symbol # allows also to precise the unity of a value

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes

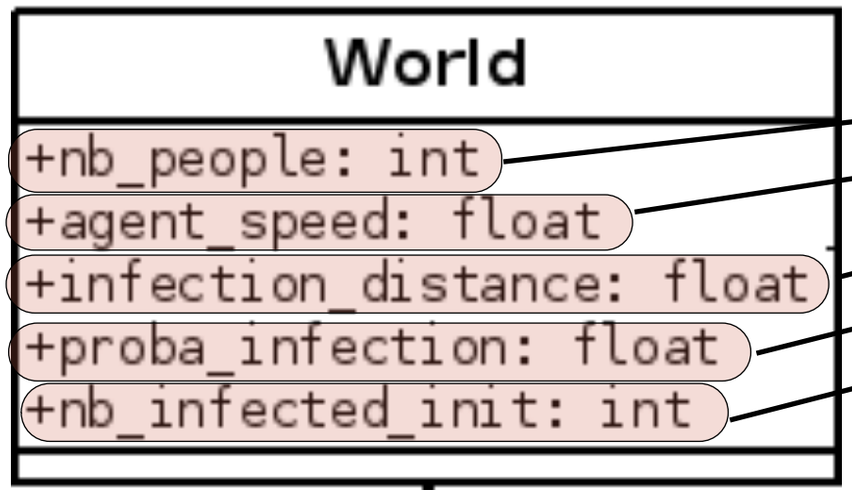



```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

The symbol # allows also to precise the unity of a value

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes

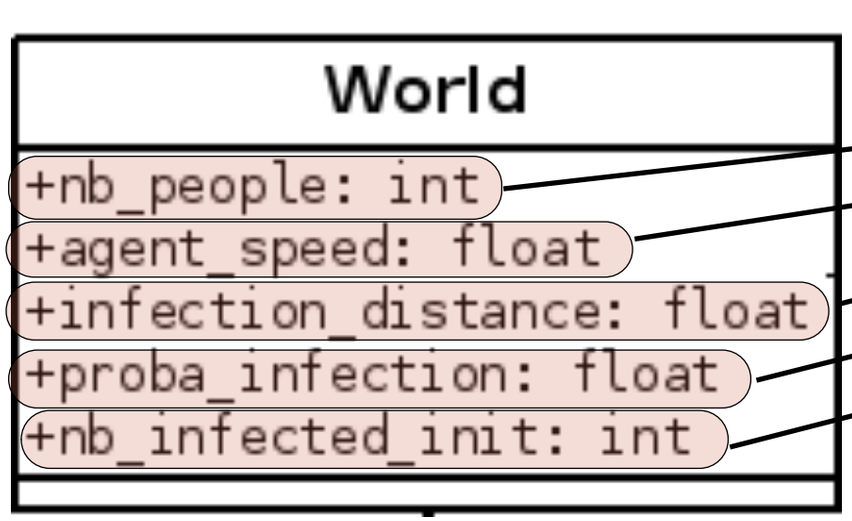



```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

The symbol # allows also to precise the unity of a value

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes



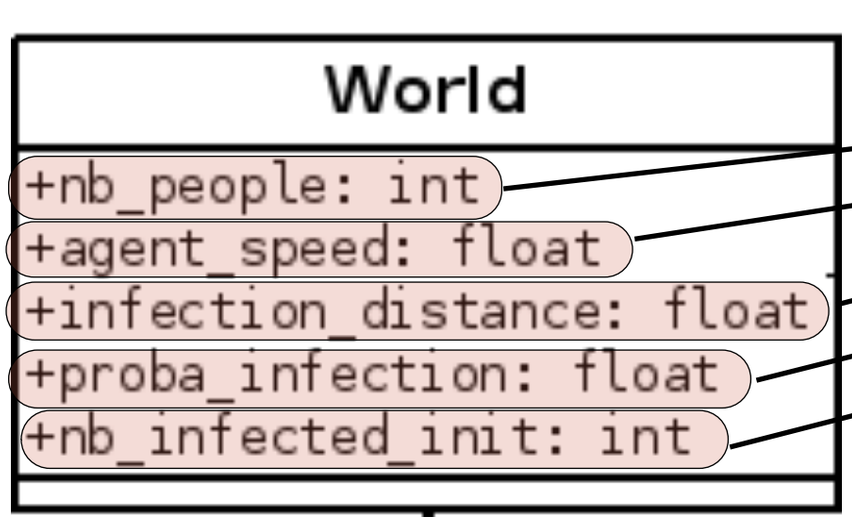

```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

Built-in variable: define the duration of simulation step

The symbol # allows also to precise the unity of a value

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes




```
global {
    int nb_people <- 300;
    float agent_speed <- 5.0 #km/#h;
    float infection_distance <- 2.0 #m;
    float proba_infection <- 0.05;
    int nb_infected_init <- 5;
    float step <- 1 #minutes;
    geometry shape<-square(500 #m);
}
```

Built-in variable: define the duration of simulation step

Define the geometry of the world: a square of 500m side size

The symbol # allows also to precise the unity of a value

Block *global* : agent creation

- 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

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

Model :World init

```
global {
    // world variable definition

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

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

- 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

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

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

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

- 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

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

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: 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 a 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 "Infection distance" var: infection_distance;
    parameter "Proba infection" var: proba_infection min: 0.0 max: 1.0;
    parameter "Nb people infected at init" var: nb_infected_init ;
}
```

- 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

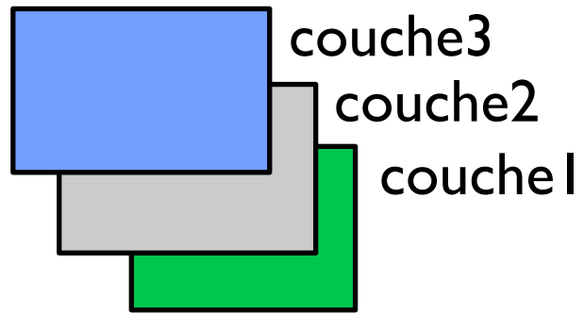
```
experiment main_experiment type:gui{
    parameter "Infection distance" var: infection_distance;
    parameter "Proba infection" var: proba_infection min: 0.0 max: 1.0;
    parameter "Nb people infected at init" var: nb_infected_init ;
}
```

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 {
}
```



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

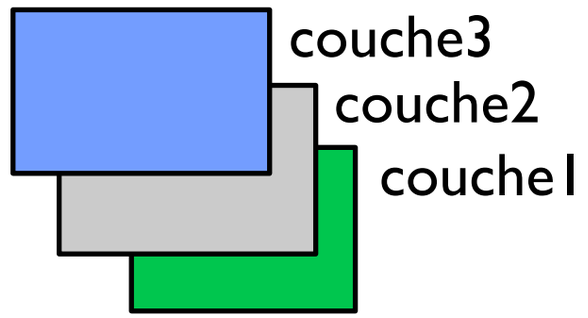
Model : experiment

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

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

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 {
}
```



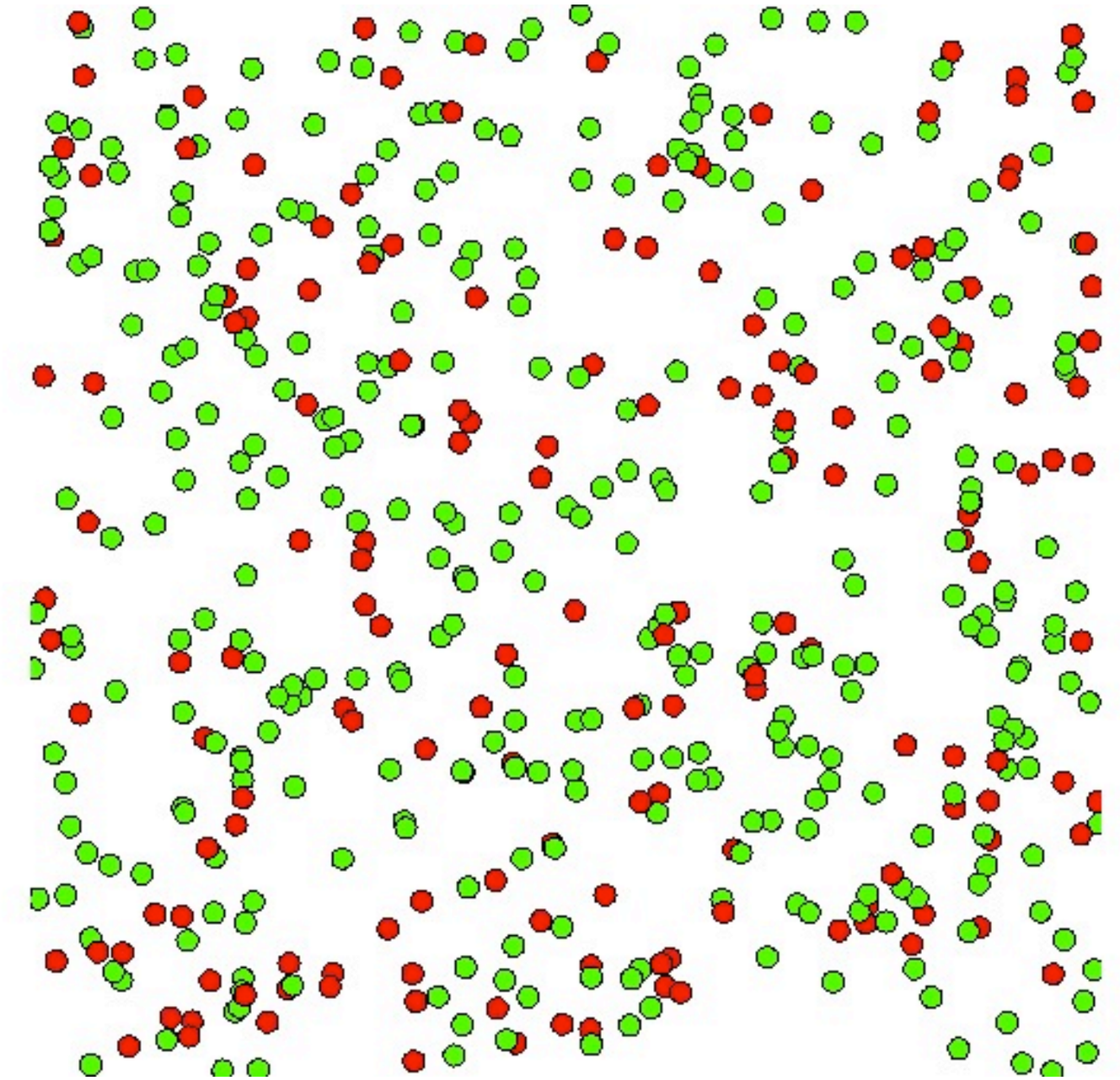
Note: in a *display*, the display order of the layer follows the layer definition

Model : experiment

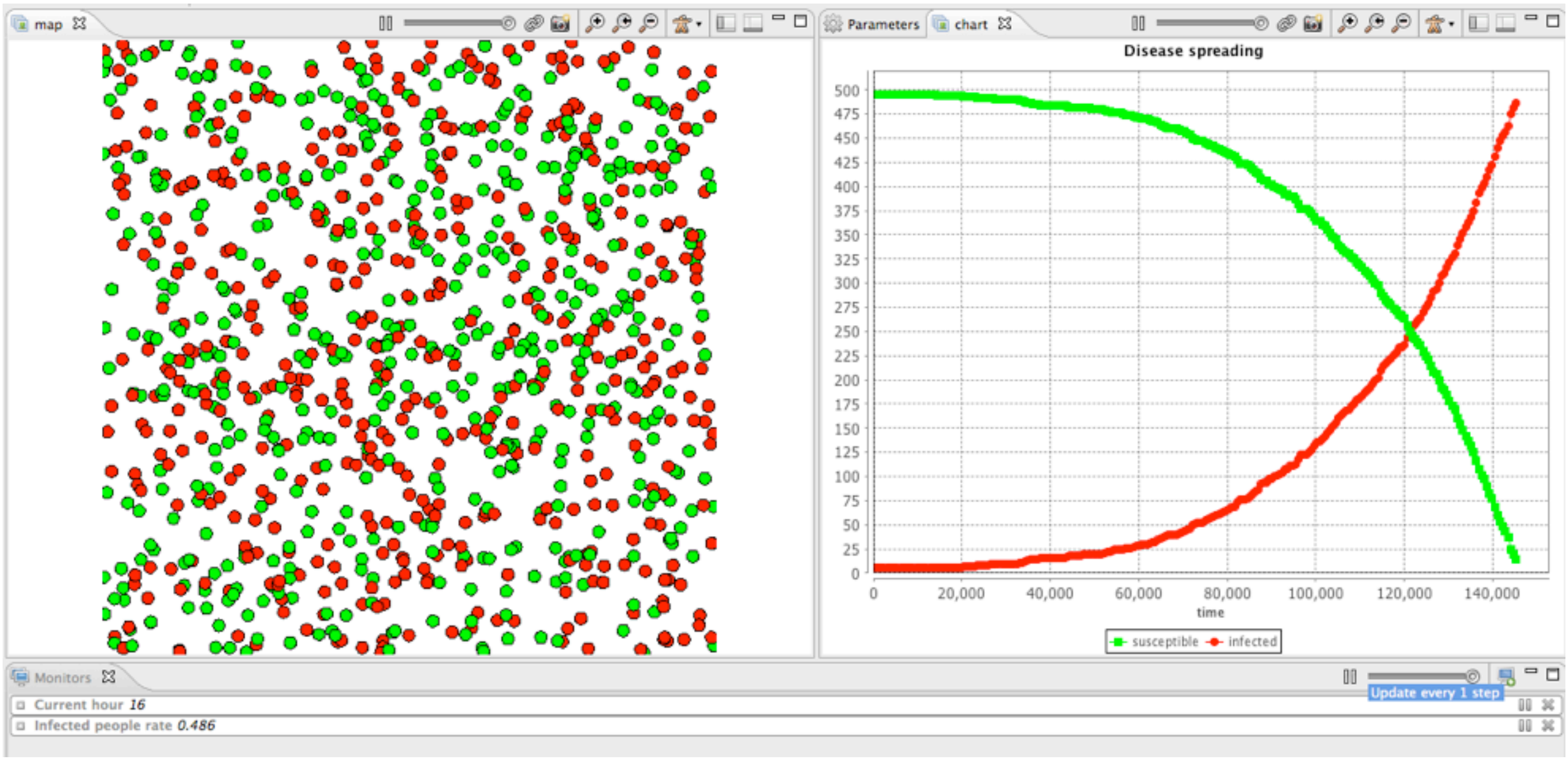
```
experiment main_experiment type: gui {
    ... //parameter definition

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

Layer displaying the people agents with their *circle* aspect

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 current_hour update: (cycle / 60) mod 24;
    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
}
```

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

Model : World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

In our model 1 cycle = 1 minute (step definition); we compute the current hour of the day by dividing the cycle by 60 (to have the number of hours) and computing its modulo to 24 (to limit its value to 24 hours)

```
global{
    //... other attributes
    int current_hour update: (cycle / 60) mod 24;
    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
}
```

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

Model : World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

In our model 1 cycle = 1 minute (step definition); we compute the current hour of the day by dividing the cycle by 60 (to have the number of hours) and computing its modulo to 24 (to limit its value to 24 hours)

count the number of infected agents among the people agents

```
global{
    //... other attributes
    int current_hour update: (cycle / 60) mod 24;
    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
}
```

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

Model : World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

In our model 1 cycle = 1 minute (step definition); we compute the current hour of the day by dividing the cycle by 60 (to have the number of hours) and computing its modulo to 24 (to limit its value to 24 hours)

count the number of infected agents among the people agents

```
global{
    //... other attributes
    int current_hour update: (cycle / 60) mod 24;
    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
}
```

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 : World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

In our model 1 cycle = 1 minute (step definition); we compute the current hour of the day by dividing the cycle by 60 (to have the number of hours) and computing its modulo to 24 (to limit its value to 24 hours)

count the number of infected agents among the people agents

```
global{
    //... other attributes
    int current_hour update: (cycle / 60) mod 24;
    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

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 halt;
    }
}
```

Model : Reflex *end_simulation* of the *world* agent

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

```
global {
    //.. variable and init definition

    reflex end_simulation when: infected_rate = 1.0 {
        do halt;
    }
}
```

- 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 {
}
```

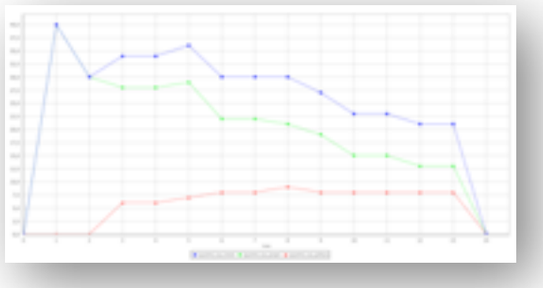
Model : monitor display

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

        //...display
    }
}
```

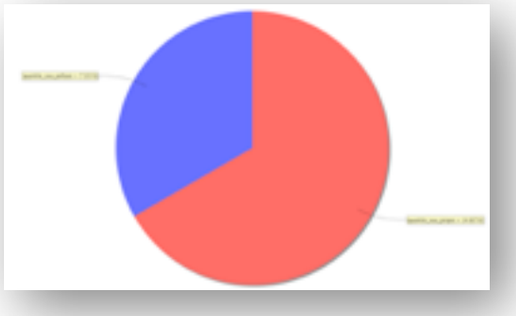
experiment block: chart definition

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

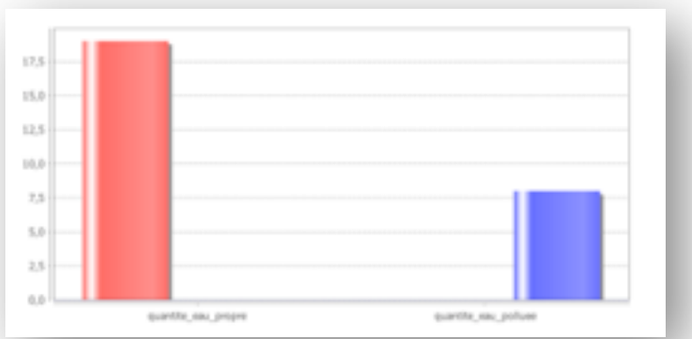
- GAMA allows to display several type of charts :
  - Series



  - Pie



  - H*istogram*



  - *XY chart*

- 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;
            }
        }
    }
}
```

- 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*

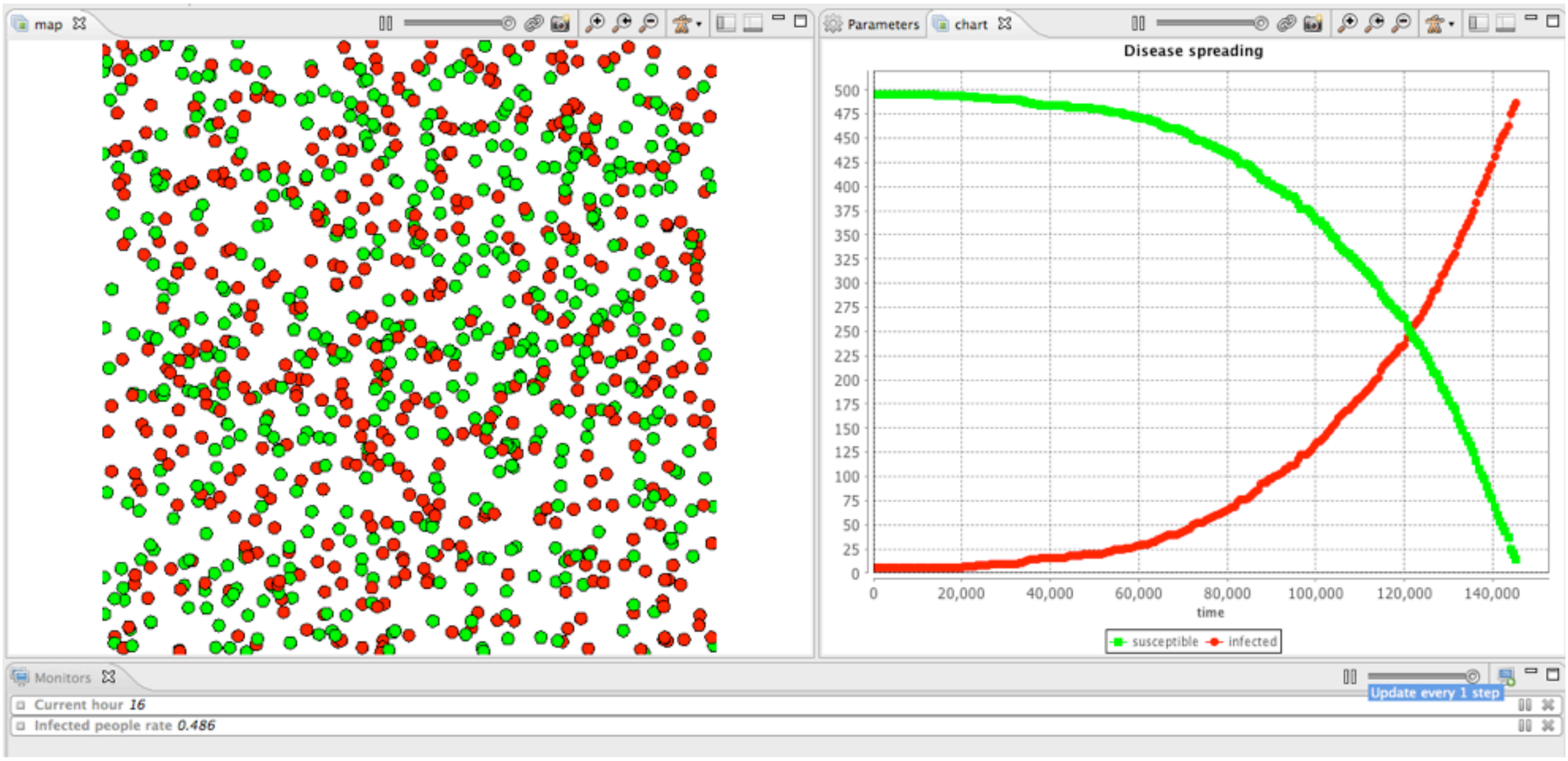
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : chart display

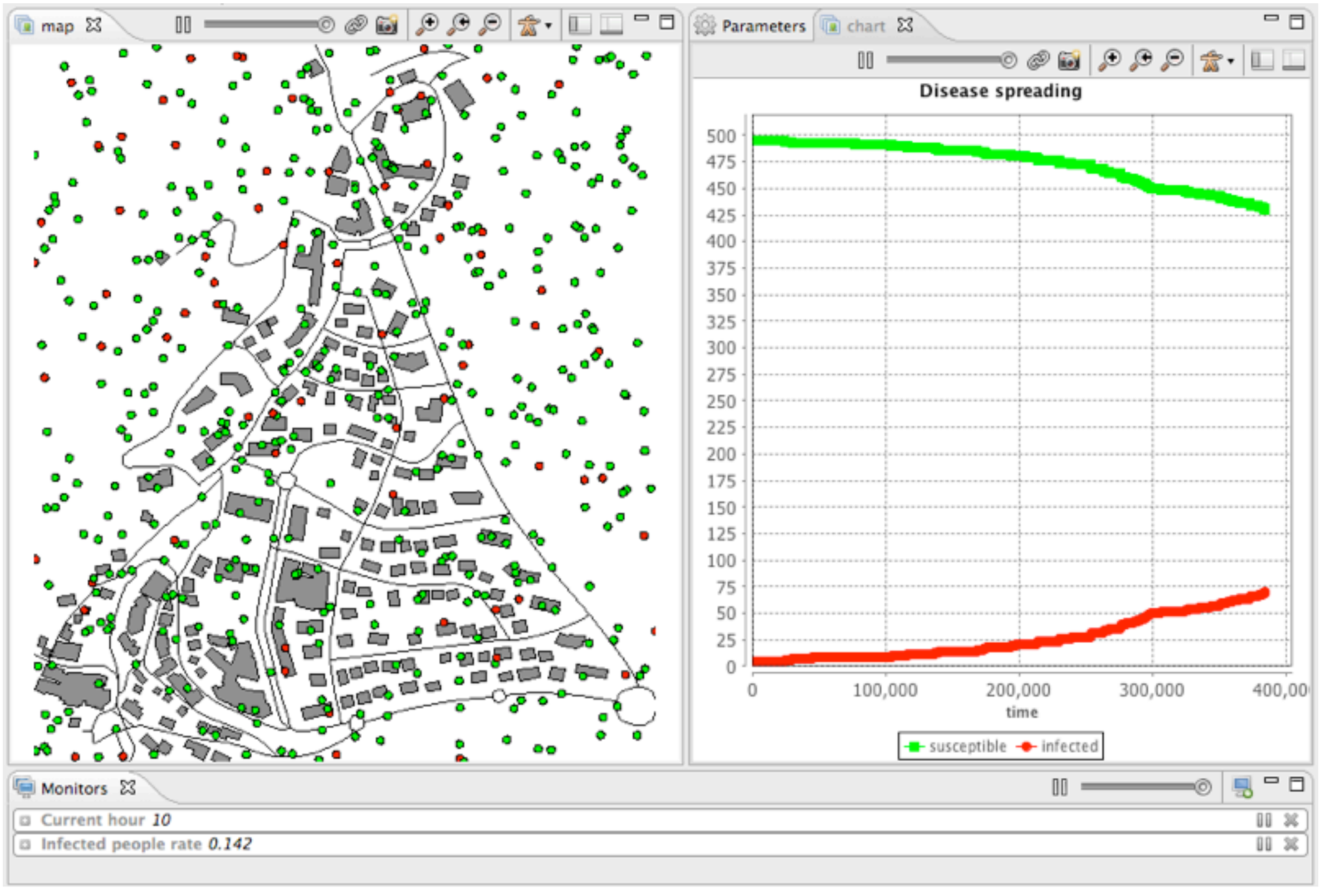
```
experiment main_experiment type:gui{
    //...parameters
    output {
        //...display and monitors

        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 !



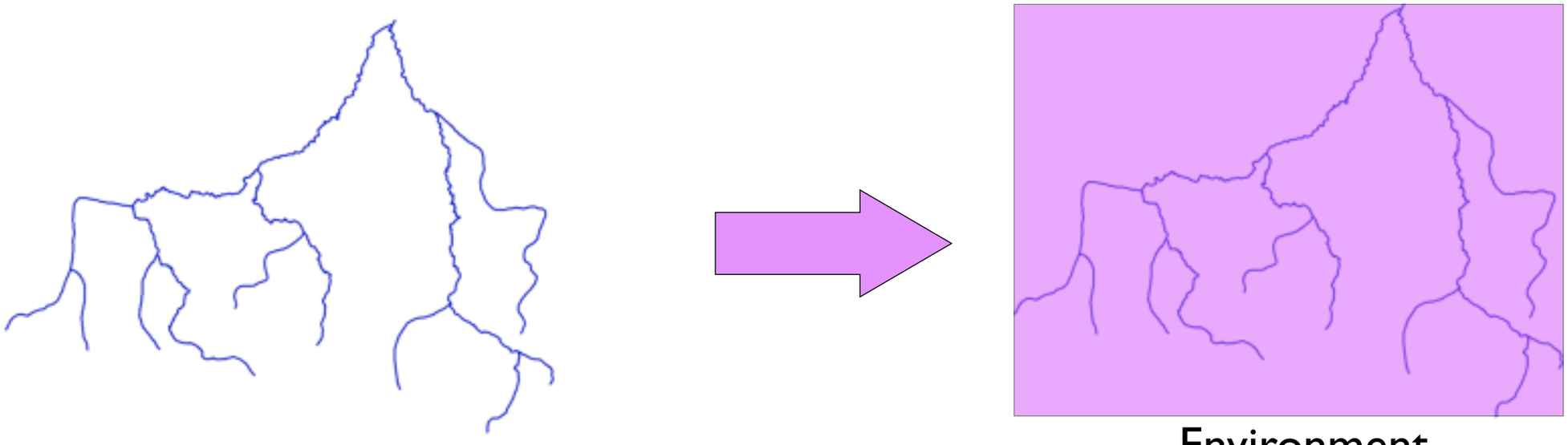
Loading of Geographical vector data to create buildings and roads



Model : World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

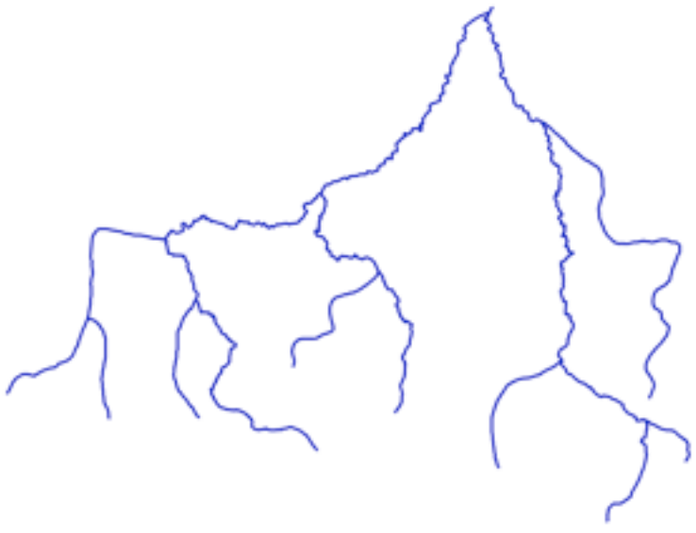


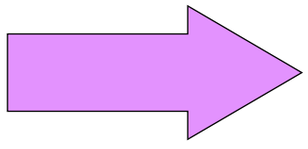
Model : World attributes

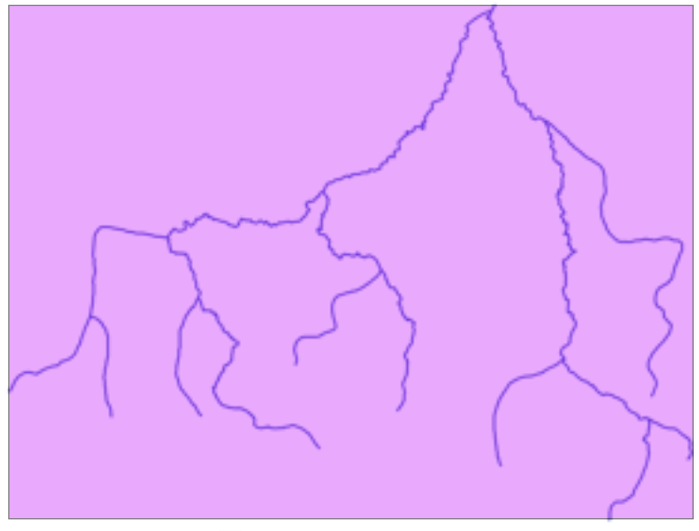
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Shapefile of the roads

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







Environment

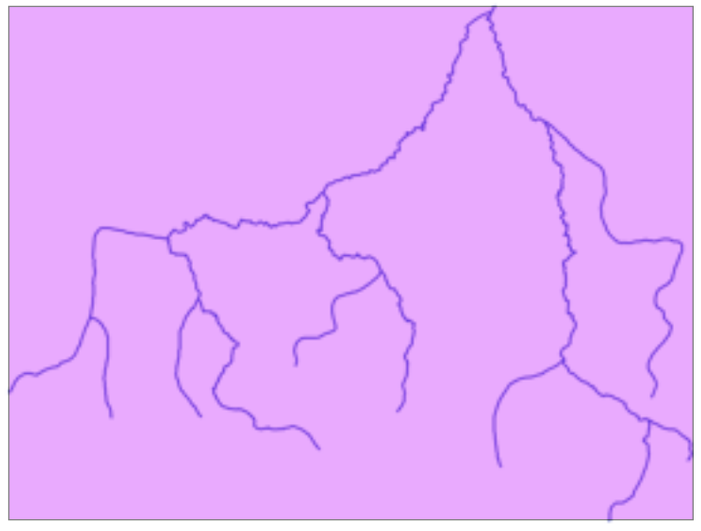
Model : World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Shapefile of the roads

Shapefile of the buildings

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



Environment

Model :World attributes

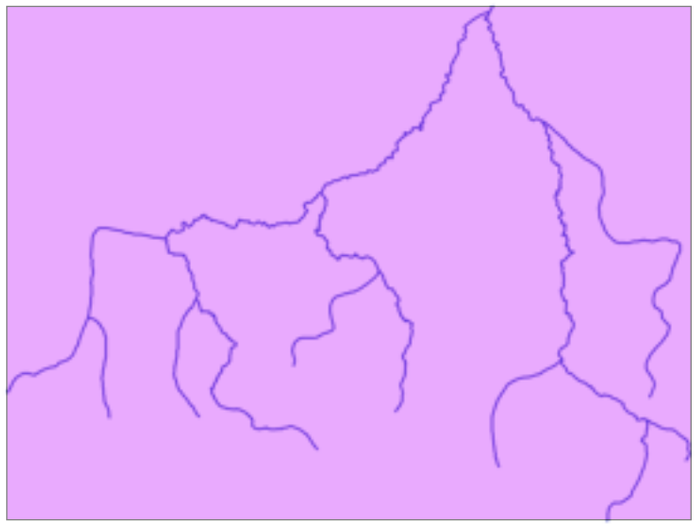
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Shapefile of the roads

Shapefile of the buildings

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

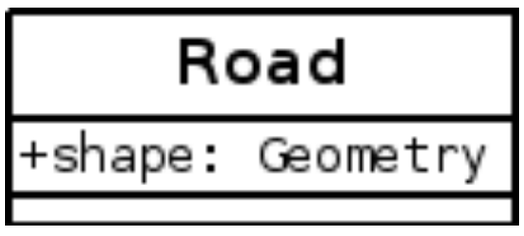
Computation of the world geometry from the envelope of the road shapefile



Environment

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 : 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 my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : road and building agents


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


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

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 {
            building bd <- one_of(building);
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

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

Create road agents from the shapefile: each object in the GIS data will become a road agent

```
global {
    // world variable definition

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

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 {
            building bd <- one_of(building);
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Create road agents from the shapefile: each object in the GIS data will become a road agent

Create building agents from the shapefile: each object in the GIS data will become a building agent

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 {
            building bd <- one_of(building);
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Create road agents from the shapefile: each object in the GIS data will become a road agent

Create building agents from the shapefile: each object in the GIS data will become a building agent

Randomly select one of the building

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 {
            building bd <- one_of(building);
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Create road agents from the shapefile: each object in the GIS data will become a road agent

Create building agents from the shapefile: each object in the GIS data will become a building agent

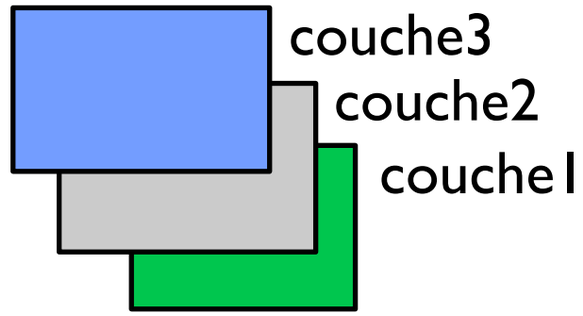
Randomly select one of the building

place the people agent on a random point in the *bd* building

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

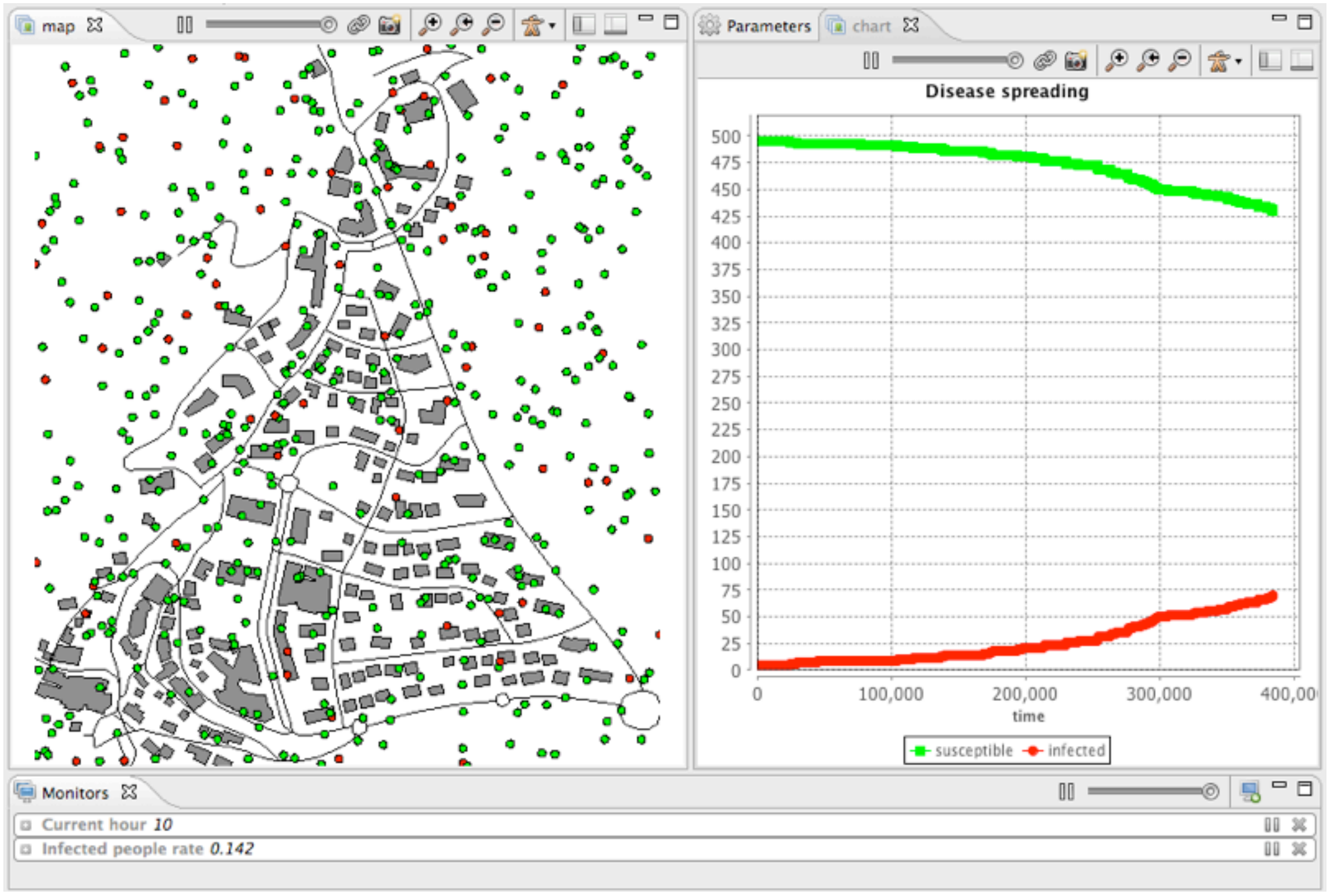
Model : experiment

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

    output {
        ... //monitor definition

        display map {
            species road aspect:geom;
            species building aspect:geom;
            species people aspect:circle;
        }
        ... //chart display definition
    }
}
```

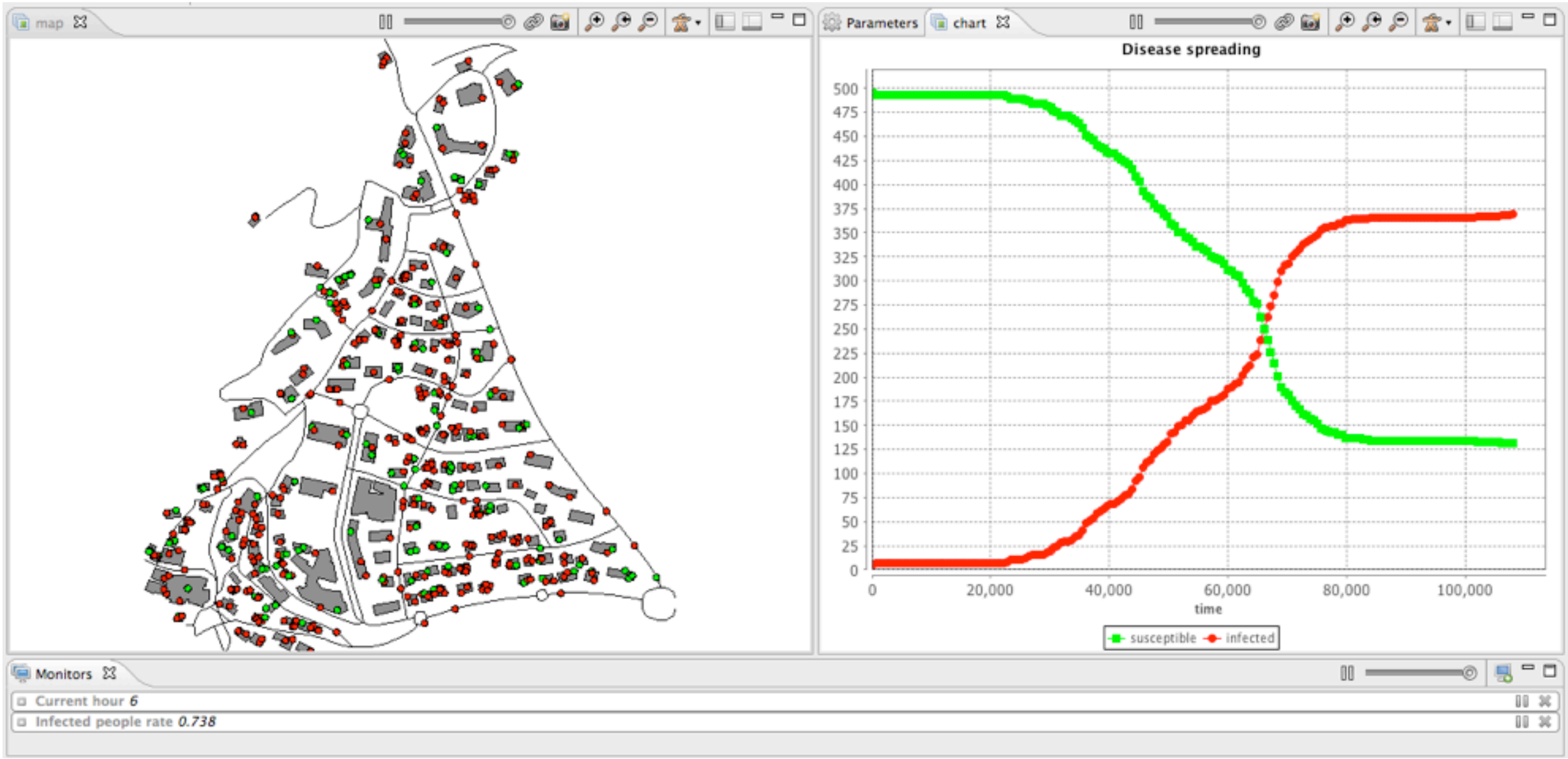


Note: in a *display*, the display order of the layer follows the layer definition

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;
    float staying_coeff update: 10.0 ^ (1 + min([abs(current_hour - 9), abs(current_hour - 12),
        abs(current_hour - 18)]));
        //... init
}
```

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    graph road_network;
    float staying_coeff update: 10.0 ^ (1 + min([abs(current_hour - 9), abs(current_hour - 12),
        abs(current_hour - 18)]));
        //... init
}
```

Graph variable

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    graph road_network;
    float staying_coeff update: 10.0 ^ (1 + min([abs(current_hour - 9), abs(current_hour - 12),
        abs(current_hour - 18)]));
    //... init
}
```

Graph variable

value updated at each simulation step: higher when the hour is far from 9h, 12h and 18h

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

Model :World init

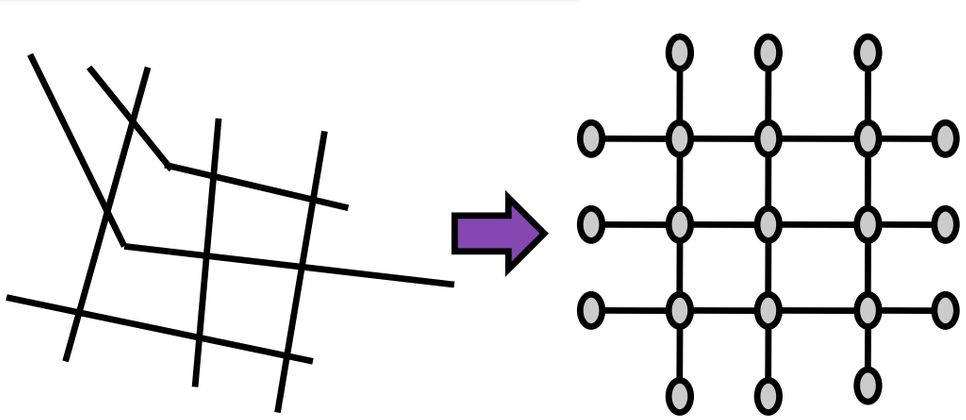
```
global {
    // world variable definition

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

        }
    }
}
```

- 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 {
            building bd <- one_of(building);
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;

        }
    }
}
```

Create a graph from the road geometries (polyline)



Model : People attributes

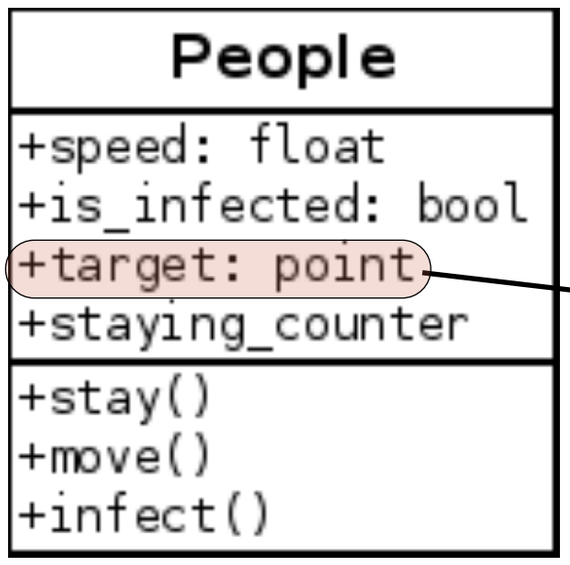
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

```
species people skills:[moving]{
    //...the other attributes
    point target;
    int staying_counter;
    //....
}
```

Model : People attributes

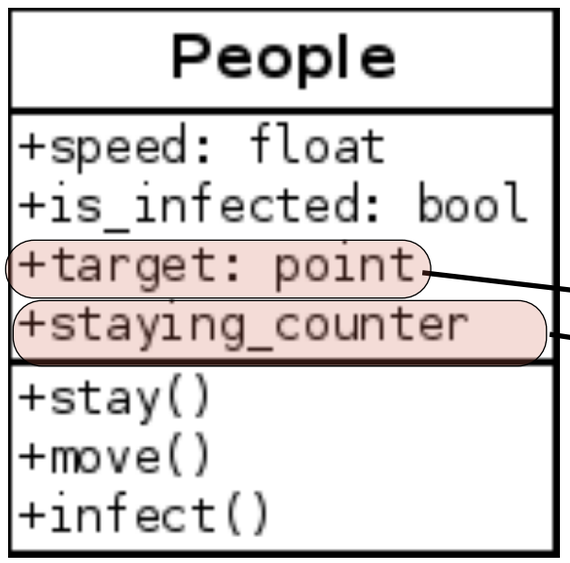
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```




```
species people skills:[moving]{
    //...the other attributes
    point target;
    int staying_counter;
    //....
}
```

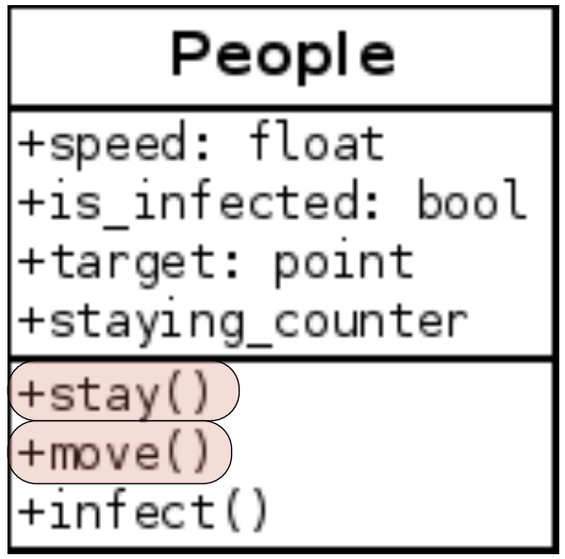
Model : People attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```




```
species people skills:[moving]{
    //...the other attributes
    point target;
    int staying_counter;
    //....
}
```

Model : Some reflexes of people




```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            target <- any_location_in (one_of(building));
        }
    }

    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            target <- nil;
            staying_counter <- 0;
        }
    }
}
```

Model : Some reflexes of people

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

Reflex executed when the target is nil (i.e. the agent is not moving)

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            target <- any_location_in (one_of(building));
        }
    }

    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            target <- nil;
            staying_counter <- 0;
        }
    }
}
```

Model : Some reflexes of people

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

Reflex executed when the target is nil (i.e. the agent is not moving)

Incrementation of the staying counter

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            target <- any_location_in (one_of(building));
        }
    }

    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            target <- nil;
            staying_counter <- 0;
        }
    }
}
```

Model : Some reflexes of people

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

Reflex executed when the target is nil (i.e. the agent is not moving)

Incrementation of the staying counter

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            target <- any_location_in (one_of(building));
        }
    }

    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            target <- nil;
            staying_counter <- 0;
        }
    }
}
```

Probability *staying_counter/ staying_coeff* to choose a new target (ready to leave)

Model : Some reflexes of people

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

Reflex executed when the target is nil (i.e. the agent is not moving)

Incrementation of the staying counter

Reflex activated only when the target is not nil

Probability *staying_counter/ staying_coeff* to choose a new target (ready to leave)

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            target <- any_location_in (one_of(building));
        }
    }
    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            target <- nil;
            staying_counter <- 0;
        }
    }
}
```

Model : Some reflexes of people

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

Reflex executed when the target is nil (i.e. the agent is not moving)

Incrementation of the staying counter

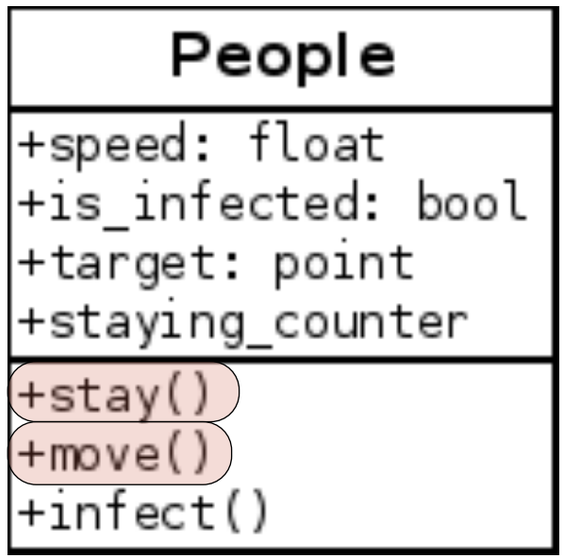
```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            target <- any_location_in (one_of(building));
        }
    }

    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            target <- nil;
            staying_counter <- 0;
        }
    }
}
```

Probability *staying_counter/ staying_coeff* to choose a new target (ready to leave)

Reflex activated only when the target is not nil

Move toward the target along the graph (shortest path computation). Use of the *goto* action of the moving skill

Model : Some reflexes of people



Reflex executed when the target is nil (i.e. the agent is not moving)

Incrementation of the staying counter

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            target <- any_location_in (one_of(building));
        }
    }
    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            target <- nil;
            staying_counter <- 0;
        }
    }
}
```

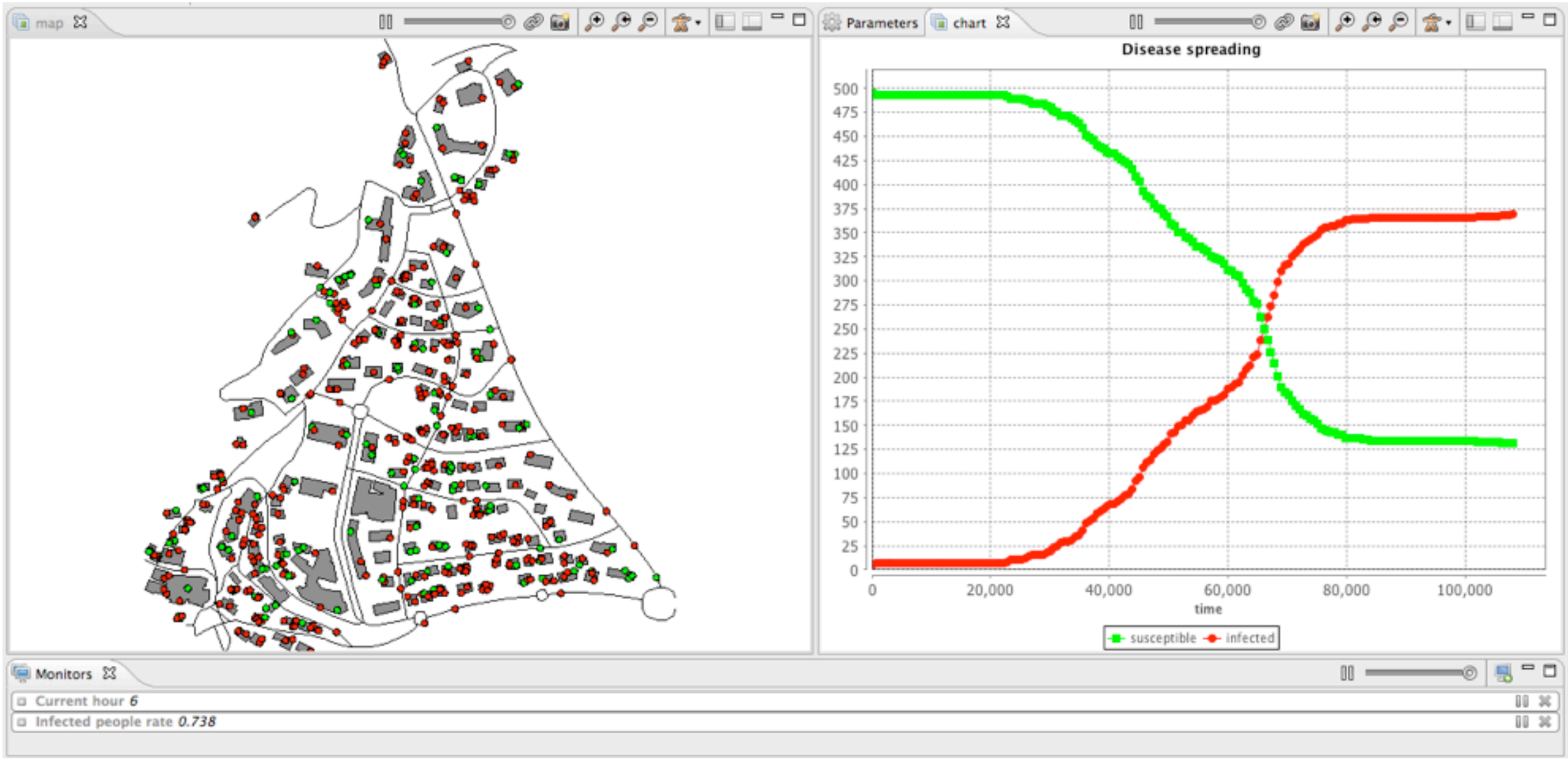
Probability *staying_counter/ staying_coeff* to choose a new target (ready to leave)

Reflex activated only when the target is not nil

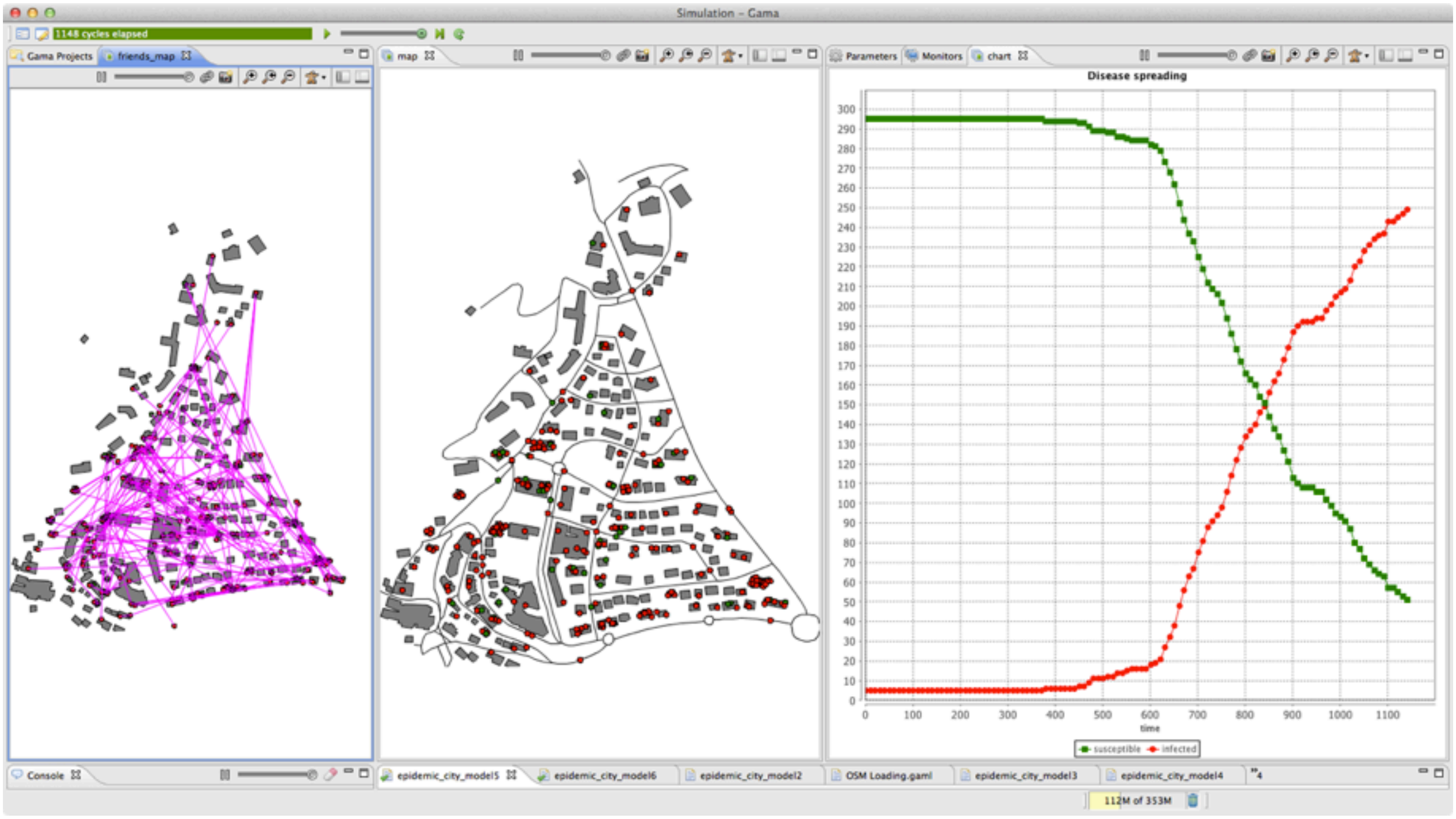
Move toward the target along the graph (shortest path computation). Use of the *goto* action of the moving skill

If the agent reachs its target, set target to nil and *staying_counter* to 0

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

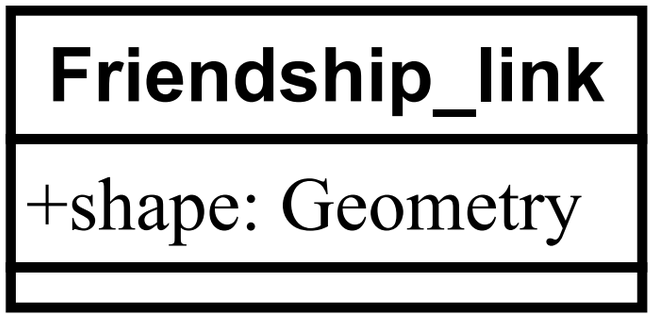


People move (most of time) toward their friend (house)



Model : *friendship_link* species

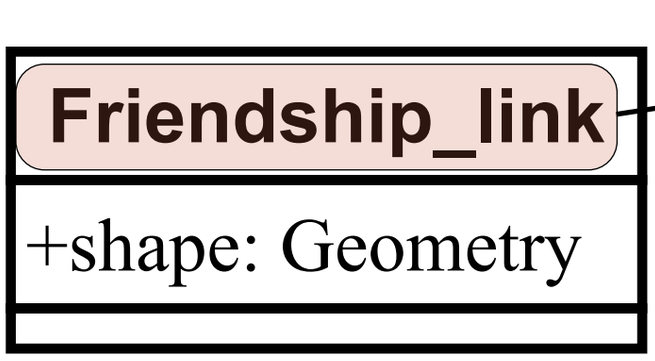
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```




```
species friendship_link {
    aspect arrow {
        draw shape end_arrow: 5 color: #magenta;
    }
}
```

Model : *friendship_link* species

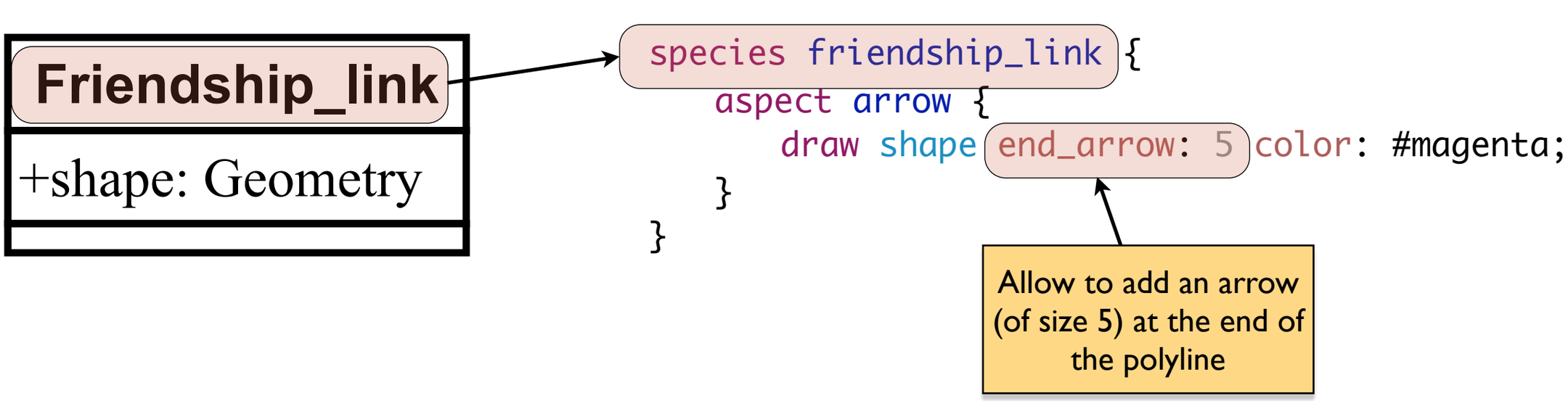
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```




```
species friendship_link {
    aspect arrow {
        draw shape end_arrow: 5 color: #magenta;
    }
}
```

Model : *friendship_link* species

```
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{
    //... other attributes
    graph friendship_graph;
    int nb_edges_nodes <- 1;
    float proba_see_friend <- 0.9;
          //... init
}
```

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    graph friendship_graph;
    int nb_edges_nodes <- 1;
    float proba_see_friend <- 0.9;
          //... init
}
```

Graph variable

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    graph friendship_graph;
    int nb_edges_nodes <- 1;
    float proba_see_friend <- 0.9;
            //... init
}
```

Graph variable

variable that defines the number of edges from a node in the friendship network

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    graph friendship_graph;
    int nb_edges_nodes <- 1;
    float proba_see_friend <- 0.9;
        //... init
}
```

Graph variable

variable that defines the number of edges from a node in the friendship network

variable that defines the probability for a people agent to go to see a friend

Block *global* : scale-free graph generation

- GAMA provides modelers with classic graph generation tools

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World init

```
global {
    // world variable definition

    init {
        create road from: roads_shapefile;
        road_network <- as_edge_graph(road);
        create building from: buildings_shapefile;
        friendship_graph<-generate_barabasi_albert(people,friendship_link,nb_people,nb_edges_nodes,false);
        ask people {
            building bd <- building closest_to self;
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Note: there are four ways to built graphs in GAMA :
- by hand (by adding edges/nodes)
- geometric operator (e.g. *as_edge_graph*)
- generator (e.g. *generate_barabasi_albert*)
- loading classic graph files

Block *global* : scale-free graph generation

- GAMA provides modelers with classic graph generation tools

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : World init

```
global {
    // world variable definition

    init {
        create road from: roads_shapefile;
        road_network <- as_edge_graph(road);
        create building from: buildings_shapefile;
        friendship_graph<-generate_barabasi_albert(people,friendship_link,nb_people,nb_edges_nodes,false);
        ask people {
            building bd <- building closest_to self;
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Create a scale-free graph (using Barabasi Albert model), *nb_people* people agents will be created and the corresponding number of *friendship_link* agents

Note: there are four ways to built graphs in GAMA :
- by hand (by adding edges/nodes)
- geometric operator (e.g. *as_edge_graph*)
- generator (e.g. *generate_barabasi_albert*)
- loading classic graph files

Block *global* : scale-free graph generation

- GAMA provides modelers with classic graph generation tools

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : World init

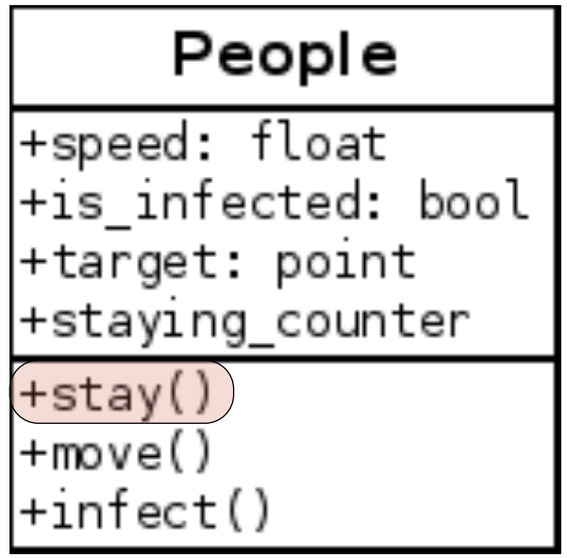
```
global {
    // world variable definition

    init {
        create road from: roads_shapefile;
        road_network <- as_edge_graph(road);
        create building from: buildings_shapefile;
        friendship_graph<-generate_barabasi_albert(people,friendship_link,nb_people,nb_edges_nodes,false);
        ask people {
            building bd <- building closest_to self;
            location <- any_location_in(bd);
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Create a scale-free graph (using Barabasi Albert model), *nb_people* people agents will be created and the corresponding number of *friendship_link* agents

As the *people* agents are already created (by the graph generation operator), we just place the *people* agent in a building (the closest building of their current location)

Note: there are four ways to built graphs in GAMA :
- by hand (by adding edges/nodes)
- geometric operator (e.g. *as_edge_graph*)
- generator (e.g. *generate_barabasi_albert*)
- loading classic graph files

Model : Some reflexes of people




```
species people skills:[moving]{
    //variable definition
    reflex staying when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            if (flip(proba_see_friend)) {
                people my_friend <- people(one_of(friendship_graph neighbours_of self));
                target <- any_location_in (building closest_to my_friend);
            } else {
                target <- any_location_in (one_of(building));
            }
        }
    }
    ...
}
```

Model : Some reflexes of people

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

```
species people skills:[moving]{
    //variable definition
    reflex staying when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            if (flip(proba_see_friend)) {
                people my_friend <- people(one_of(friendship_graph neighbours_of self));
                target <- any_location_in (building closest_to my_friend);
            } else {
                target <- any_location_in (one_of(building));
            }
        }
    }
    ...
}
```

Probability *proba_see_friend* to go to see a friend

Model : Some reflexes of people

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

```
species people skills:[moving]{
    //variable definition
    reflex staying when: target = nil {
        staying_counter <- staying_counter + 1;
        if flip(staying_counter / staying_coeff) {
            if (flip(proba_see_friend)) {
                people my_friend <- people(one_of(friendship_graph neighbours_of self));
                target <- any_location_in (building closest_to my_friend);
            } else {
                target <- any_location_in (one_of(building));
            }
        }
    }
    ...
}
```

Probability *proba_see_friend* to go to see a friend

If the agent chooses to see a friend, it chooses one of its friends (from its neighbourhood according to the *friendship_graph*) then it defined its target as a location inside the building that is the closest to its friend

Model : experiment

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
experiment main_experiment type:gui{
    parameter "Infection distance" var: infection_distance;
    parameter "Proba infection" var: proba_infection min: 0.0 max: 1.0;
    parameter "Nb people infected at init" var: nb_infected_init ;
    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3  ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0  ;

    output {
        // monitor and other displays
        display friends_map {
            species building aspect:geom;
            species people aspect:circle;
            species friendship_link aspect: arrow;
        }
    }
}
```

Model : experiment

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
experiment main_experiment type:gui{
    parameter "Infection distance" var: infection_distance;
    parameter "Proba infection" var: proba_infection min: 0.0 max: 1.0;
    parameter "Nb people infected at init" var: nb_infected_init ;
    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3  ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0  ;

    output {
        // monitor and other displays
        display friends_map {
            species building aspect:geom;
            species people aspect:circle;
            species friendship_link aspect: arrow;
        }
    }
}
```

Definition of 2 new parameters

Model : experiment

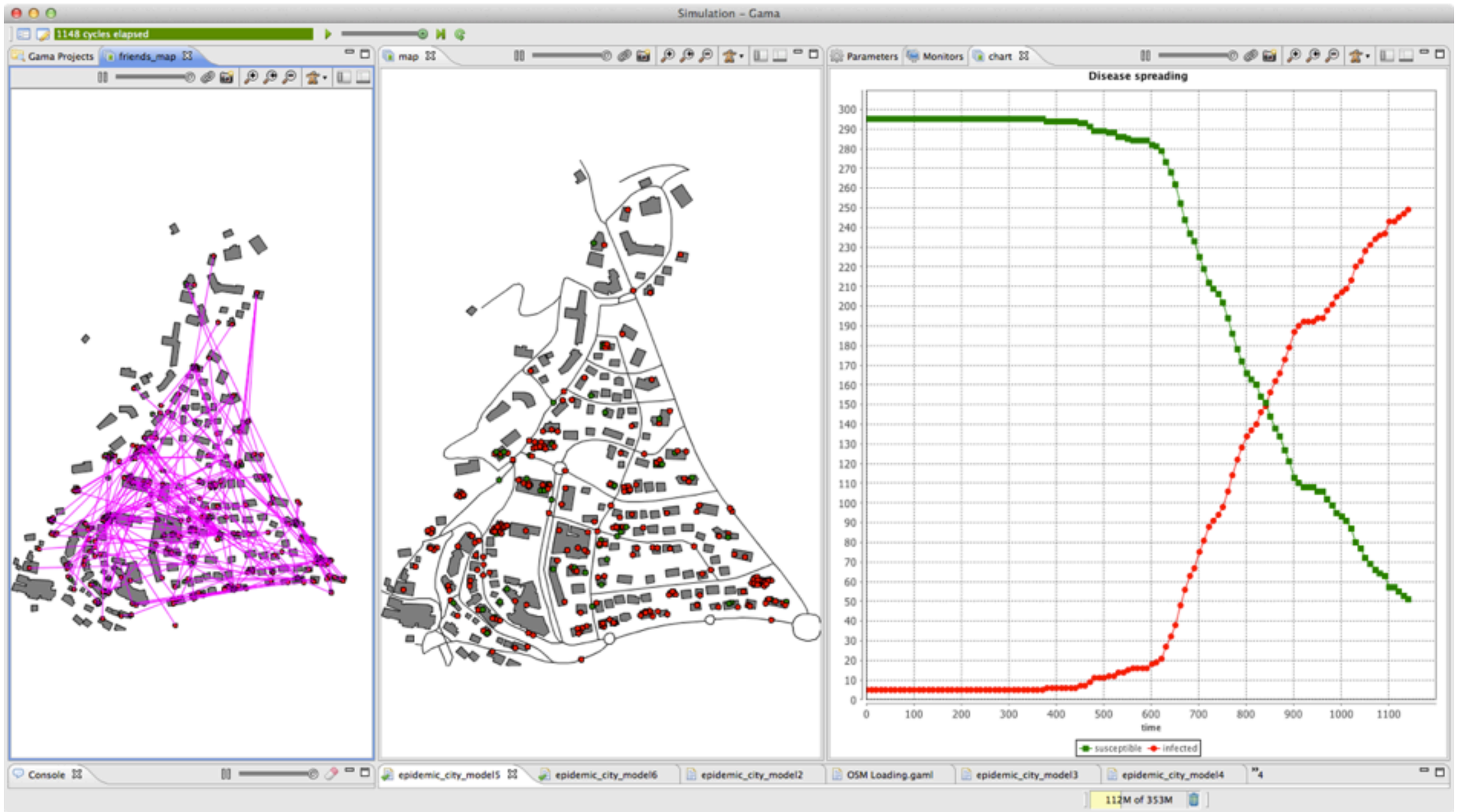
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
experiment main_experiment type:gui{
    parameter "Infection distance" var: infection_distance;
    parameter "Proba infection" var: proba_infection min: 0.0 max: 1.0;
    parameter "Nb people infected at init" var: nb_infected_init ;
    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3  ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0  ;

    output {
        // monitor and other displays
        display friends_map {
            species building aspect:geom;
            species people aspect:circle;
            species friendship_link aspect: arrow;
        }
    }
}
```

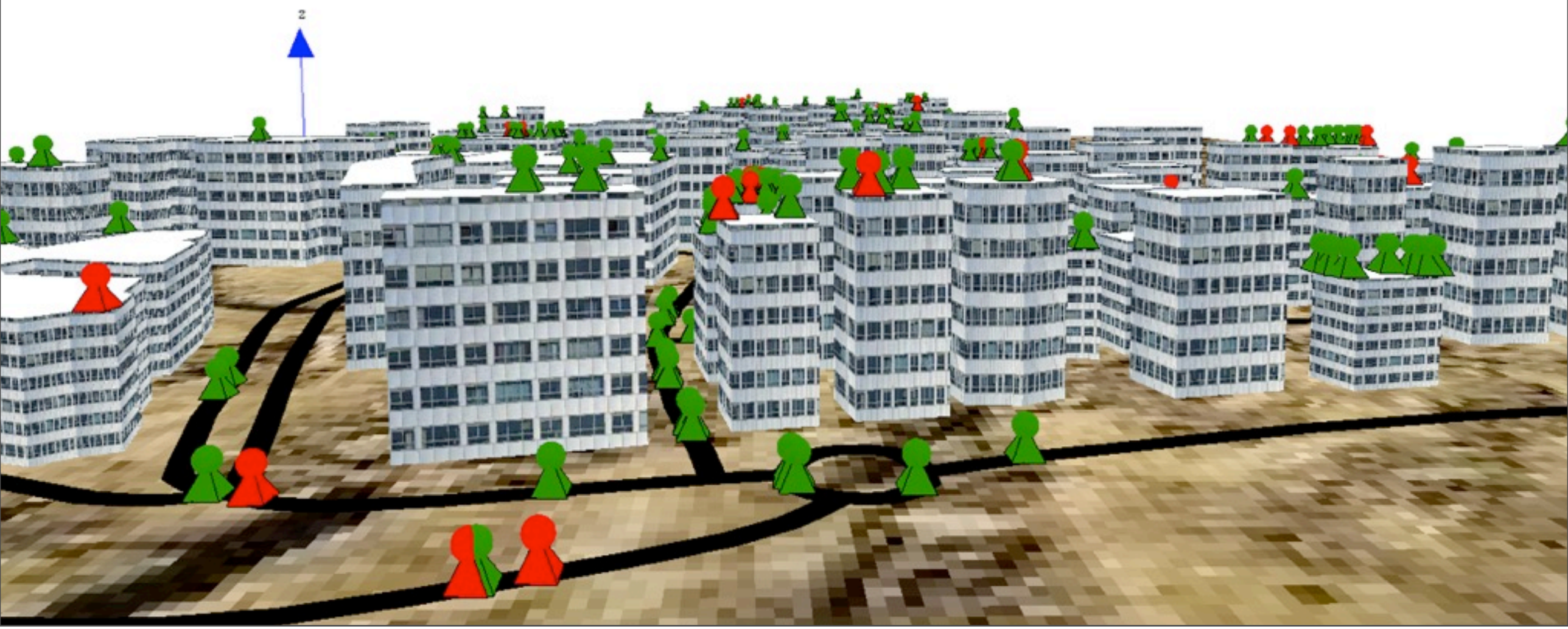
Definition of 2 new parameters

Definition of a new display with building, people and friendship_link

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



Time to prepare a new experiment for demonstrations !



Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    bool is_night <- true update: current_hour < 7 or current_hour > 20;

    init {
        create road from: roads_shapefile;
        road_network <- as_edge_graph(road);
        create building from: buildings_shapefile;
        friendship_graph <-generate_barabasi_albert(people,friendship_link, nb_people,
        nb_edges_nodes, false);
        ask people {
            building bd <- building closest_to self;
            location <- any_location_in(bd);
            location <- {location.x,location.y,bd.height};
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    bool is_night <- true update: current_hour < 7 or current_hour > 20;

    init {
        create road from: roads_shapefile;
        road_network <- as_edge_graph(road);
        create building from: buildings_shapefile;
        friendship_graph <-generate_barabasi_albert(people,friendship_link, nb_people,
        nb_edges_nodes, false);
        ask people {
            building bd <- building closest_to self;
            location <- any_location_in(bd);
            location <- {location.x,location.y,bd.height};
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

New *is_night* variable: if hour < 7 and > 20, then *true*

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
global{
    //... other attributes
    bool is_night <- true update: current_hour < 7 or current_hour > 20;

    init {
        create road from: roads_shapefile;
        road_network <- as_edge_graph(road);
        create building from: buildings_shapefile;
        friendship_graph <-generate_barabasi_albert(people,friendship_link, nb_people,
        nb_edges_nodes, false);
        ask people {
            building bd <- building closest_to self;
            location <- any_location_in(bd);
            location <- {location.x,location.y,bd.height};
        }
        ask nb_infected_init among people {
            is_infected <- true;
        }
    }
}
```

New *is_night* variable: if hour < 7 and > 20, then *true*

Add a z to the location to put the people agent on the top of the building

Model : road and building agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

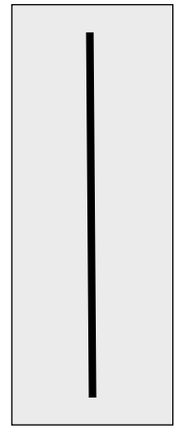
```
species road {
    geometry display_shape <- shape + 2.0;
    //....
    aspect geom3D {
        draw display_shape color: #black depth: 0.1;
    }
}

species building {
    float height <- 20#m + rnd(20) #m;
    //....
    aspect geom3D {
        draw shape color: #gray depth: height texture:["../includes/
roof_top.png","../includes/texture1.jpg"];
    }
}
```

Model : road and building agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Definition of new variable of type geometry that represent the geometry of the road (polyline) with a buffer of 2m

```
species road {
    geometry display_shape <- shape + 2.0;
    //....
    aspect geom3D {
        draw display_shape color: #black depth: 0.1;
    }
}

species building {
    float height <- 20#m + rnd(20) #m;
    //....
    aspect geom3D {
        draw shape color: #gray depth: height texture:["../includes/
roof_top.png","../includes/texture1.jpg"];
    }
}
```



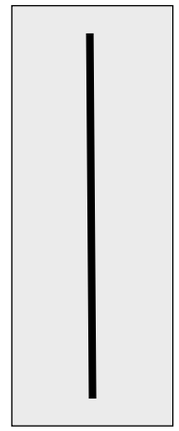
```
model my_model

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) with a buffer of 2m

New aspect: draw the display_shape geometry with a depth of 3m

```
species road {
    geometry display_shape <- shape + 2.0;
    //....
    aspect geom3D {
        draw display_shape color: #black depth: 0.1;
    }
}

species building {
    float height <- 20#m + rnd(20) #m;
    //....
    aspect geom3D {
        draw shape color: #gray depth: height texture:["../includes/
roof_top.png","../includes/texture1.jpg"];
    }
}
```



Model : road and building agents

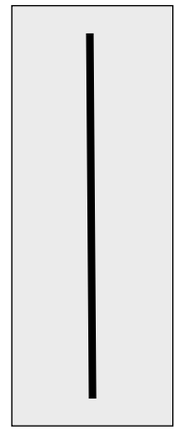
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Definition of new variable of type geometry that represent the geometry of the road (polyline) with a buffer of 2m

New aspect: draw the display_shape geometry with a depth of 3m

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

```
species road {
    geometry display_shape <- shape + 2.0;
    //....
    aspect geom3D {
        draw display_shape color: #black depth: 0.1;
    }
}

species building {
    float height <- 20#m + rnd(20) #m;
    //....
    aspect geom3D {
        draw shape color: #gray depth: height texture:["../includes/
roof_top.png","../includes/texture1.jpg"];
    }
}
```



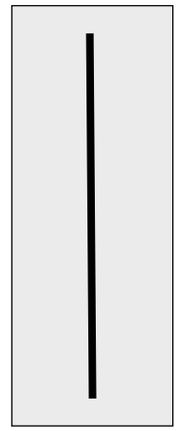
Model : road and building agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Definition of new variable of type geometry that represent the geometry of the road (polyline) with a buffer of 2m

New aspect: draw the display_shape geometry with a depth of 3m

```
species road {
    geometry display_shape <- shape + 2.0;
    //....
    aspect geom3D {
        draw display_shape color: #black depth: 0.1;
    }
}
```



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

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

```
species building {
    float height <- 20#m + rnd(20) #m;
    //....
    aspect geom3D {
        draw shape color: #gray depth: height texture:["../includes/roof_top.png","../includes/texture1.jpg"];
    }
}
```

Model : people agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
species people skills:[moving]{
    //....
    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            location <- {location.x,location.y,(building closest_to self).height};
            target <- nil;
            staying_counter <- 0;
        }
    }
    aspect geom3D{
        draw pyramid(5) color: is_infected ? #red : #green;
        draw sphere(2) at: {location.x,location.y,location.z + 5} color: is_infected ? #red : #green;
    }
}
```

Model : people agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Add a z to the location to put the people agent on the top of the closest building

```
species people skills:[moving]{
    //....
    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            location <- {location.x,location.y,(building closest_to self).height};
            target <- nil;
            staying_counter <- 0;
        }
    }
    aspect geom3D{
        draw pyramid(5) color: is_infected ? #red : #green;
        draw sphere(2) at: {location.x,location.y,location.z + 5} color: is_infected ? #red : #green;
    }
}
```

Model : people agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
species people skills:[moving]{
    //....
    reflex move when: target != nil{
        do goto target:target on: road_network;
        if (location = target) {
            location <- {location.x,location.y,(building closest_to self).height};
            target <- nil;
            staying_counter <- 0;
        }
    }
    aspect geom3D{
        draw pyramid(5) color: is_infected ? #red : #green;
        draw sphere(2) at: {location.x,location.y,location.z + 5} color: is_infected ? #red : #green;
    }
}
```

Add a z to the location to put the people agent on the top of the closest building

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

Model : *experiment3D* experiment

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
experiment experiment3D type: gui {
    output {
    // monitor and other displays
        display map_3D type: opengl ambient_light: is_night ? 30 : 120{
            image "../includes/soil.jpg";
            species road aspect:geom3D;
            species building aspect:geom3D;
            species people aspect:geom3D;
        }
    }
}
```

Model : *experiment3D* experiment

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
experiment experiment3D type: gui {
    output {
    // monitor and other displays
        display map_3D type: opengl ambient_light: is_night ? 30 : 120{
            image "../includes/soil.jpg";
            species road aspect:geom3D;
            species building aspect:geom3D;
            species people aspect:geom3D;
        }
    }
}
```

define a display of type opengl (3D) with an ambient light (darker during night)

Model : *experiment3D* experiment

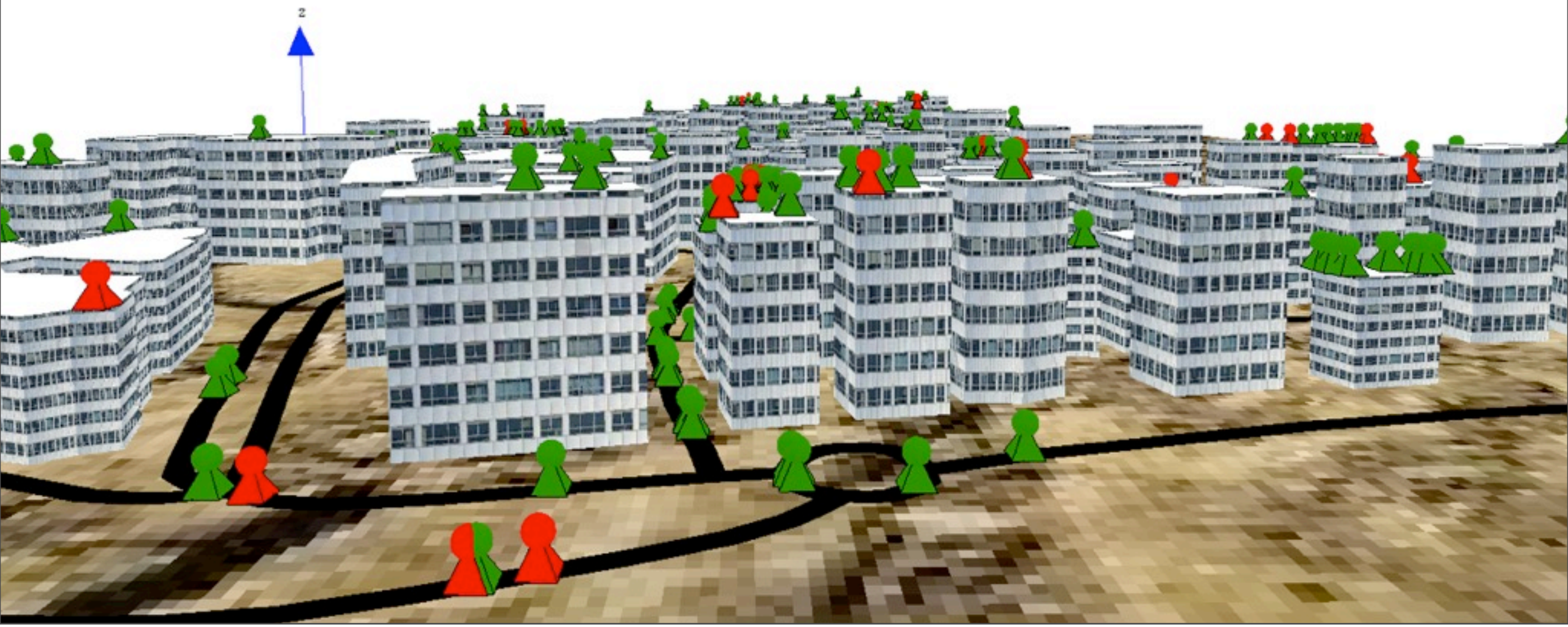
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

```
experiment experiment3D type: gui {
    output {
    // monitor and other displays
        display map_3D type: opengl ambient_light: is_night ? 30 : 120{
            image "../includes/soil.jpg";
            species road aspect:geom3D;
            species building aspect:geom3D;
            species people aspect:geom3D;
        }
    }
}
```

define a display of type opengl (3D) with an ambient light (darker during night)

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

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



step 1: define the modeling questions

step 2: identify the elements (entities, dynamics) to model

step 3: collect data

step 4: define the agents (characteristics, dynamics)

step 5: implement the model

step 6: calibrate the model

step 7: explore the model

Not this time... but a short demonstration of parameter space exploration features

```
experiment Optimization type: batch keep_seed: true repeat: 2 until: ( time > 24 #h ) {

    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3 step: 1 ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0 step: 0.1 ;



    method hill_climbing minimize: infected_rate;



    reflex save_result {
        save [nb_edges_nodes, proba_see_friend, infected_rate] to: "result.csv" type: "csv";
    }
}
```

Not this time... but a short demonstration of parameter space exploration features

Batch experiment: each simulation will be stopped after 24h, and each parameter combination will be tested twice (2 simulations per parameter combination)

```
experiment Optimization type: batch keep_seed: true repeat: 2 until: ( time > 24 #h ) {

    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3 step: 1 ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0 step: 0.1 ;

    method hill_climbing minimize: infected_rate;

    reflex save_result {
        save [nb_edges_nodes, proba_see_friend, infected_rate] to: "result.csv" type: "csv";
    }
}
```

Not this time... but a short demonstration of parameter space exploration features

Batch experiment: each simulation will be stopped after 24h, and each parameter combination will be tested twice (2 simulations per parameter combination)

```
experiment Optimization type: batch keep_seed: true repeat: 2 until: ( time > 24 #h ) {

    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3 step: 1 ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0 step: 0.1 ;

    method hill_climbing minimize: infected_rate;



    reflex save_result {
        save [nb_edges_nodes, proba_see_friend, infected_rate] to: "result.csv" type: "csv";
    }
}
```

The two parameters to explore and their possible values

Not this time... but a short demonstration of parameter space exploration features

Batch experiment: each simulation will be stopped after 24h, and each parameter combination will be tested twice (2 simulations per parameter combination)

```
experiment Optimization type: batch keep_seed: true repeat: 2 until: ( time > 24 #h ) {

    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3 step: 1 ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0 step: 0.1 ;

    method hill_climbing minimize: infected_rate;

    reflex save_result {
        save [nb_edges_nodes, proba_see_friend, infected_rate] to: "result.csv" type: "csv";
    }
}
```

Use hill-climbing to explore the parameter space. Fitness: infected_rate

The two parameters to explore and their possible values

Not this time... but a short demonstration of parameter space exploration features

Batch experiment: each simulation will be stopped after 24h, and each parameter combination will be tested twice (2 simulations per parameter combination)

```
experiment Optimization type: batch keep_seed: true repeat: 2 until: ( time > 24 #h ) {
    parameter "Nb of edges per new nodes" var: nb_edges_nodes min: 1 max: 3 step: 1 ;
    parameter "Probability to go to see a friend" var: proba_see_friend min: 0.0 max: 1.0 step: 0.1 ;

    method hill_climbing minimize: infected_rate;

    reflex save_result {
        save [nb_edges_nodes, proba_see_friend, infected_rate] to: "result.csv" type: "csv";
    }
}
```

Use hill-climbing to explore the parameter space. Fitness: infected_rate

The two parameters to explore and their possible values

Reflex activated at the end of each simulation that saves the value of the variable *nb_edges_nodes*, *proba_see_friend* and *infected_rate* into a CSV file (a line per simulation)