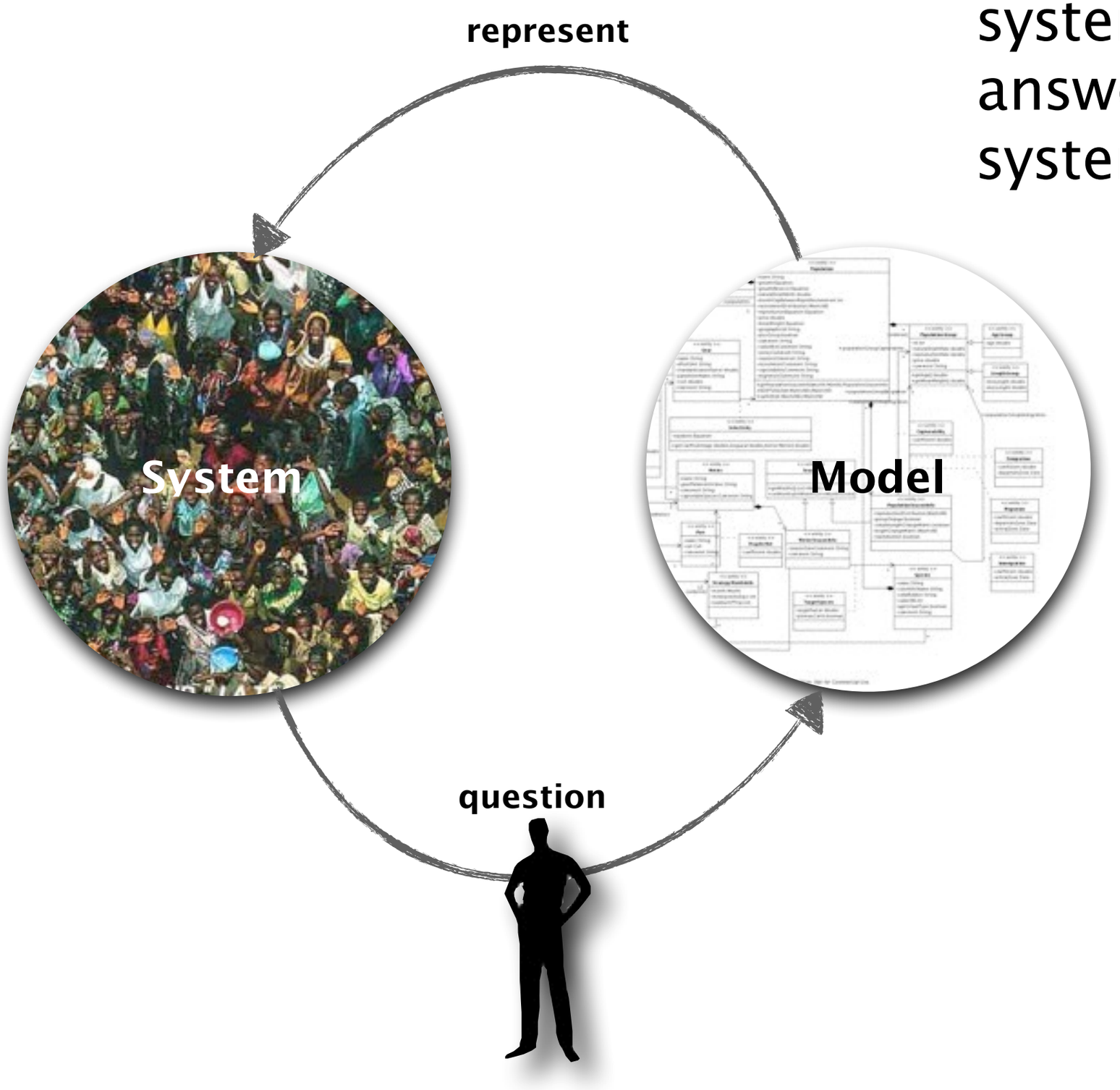
Patrick Taillandier (a), Benoit Gaudou (b), An Duc Vo (c),
Arnaud Grignard (a), Alexis Drogoul (d)

A Practical Introduction to the GAMA platform

a)UMR IDEES, University of Rouen, b)UMR IRIT, University of Toulouse 1,
c) University of Danang, d)UMI UMMISCO, IRD

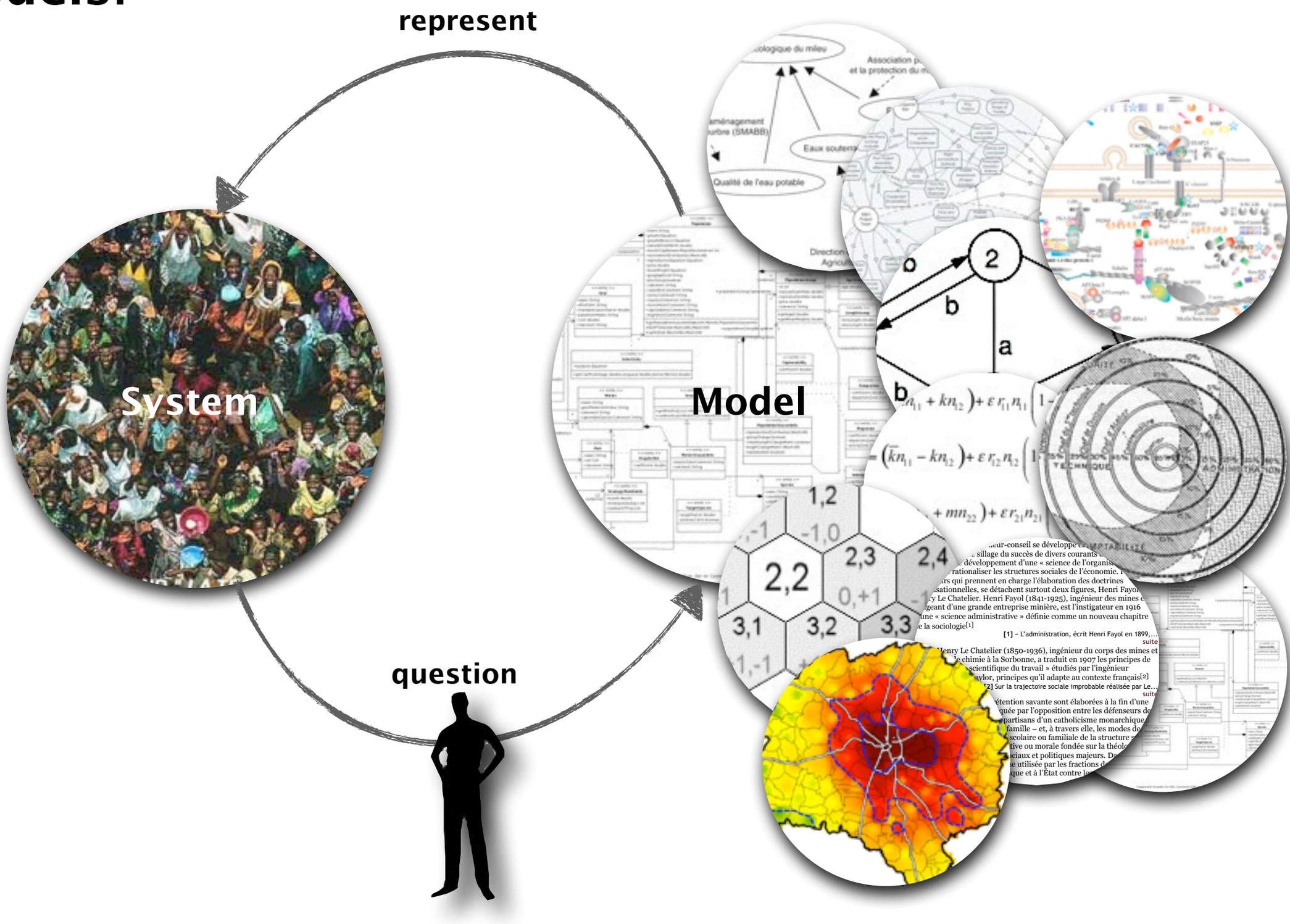
Model



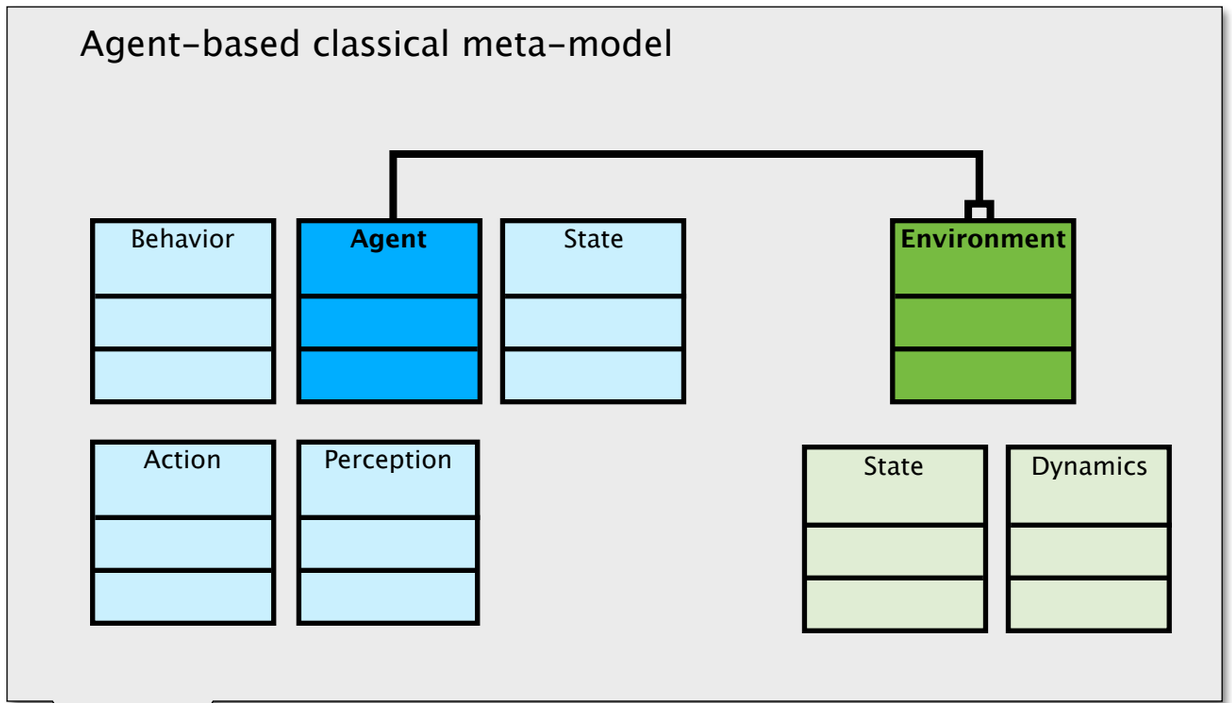
A model is a simplified representation of a reference system, designed to help answering a **question** on this system

Representation(s)

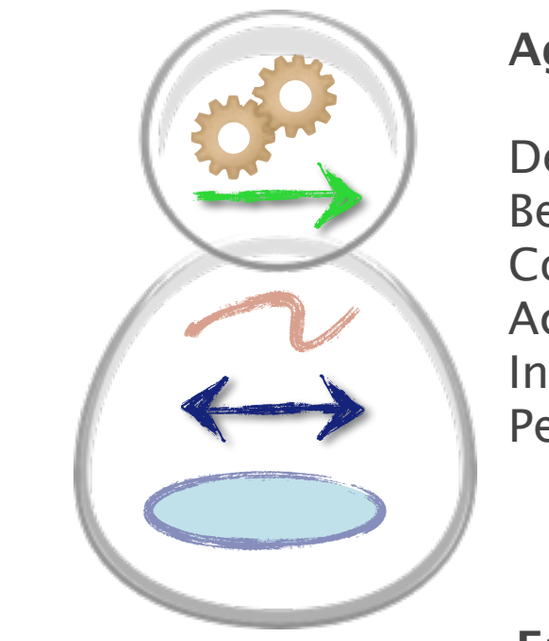
Representation can use multiple supports and languages, depending on the question, traditions... They are specified by **meta-models**.



Structure of agent-based models




```
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_en
     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

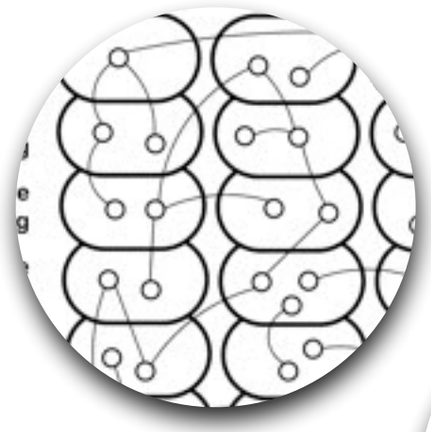
Environment
State(s)
Dynamics

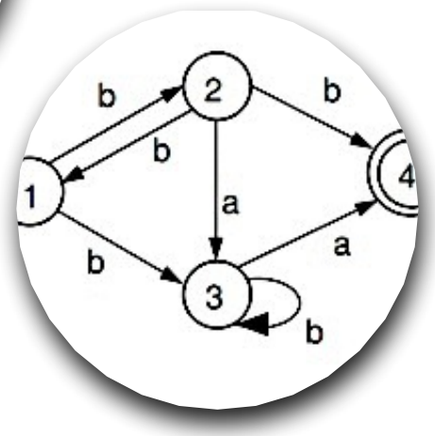
Agent-based models are **versatile** and **heterogeneous**: agents can represent any object or aggregation of objects of the reference system.

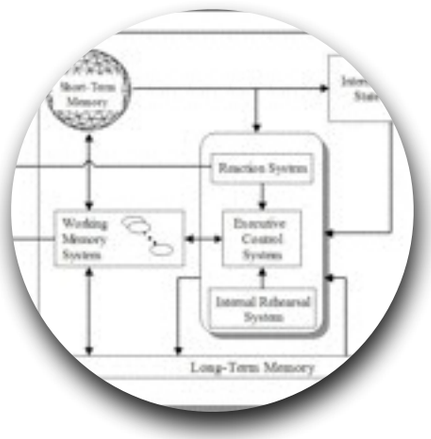
SimPop model
http://
www.simpop.parisgeo.cnrs.fr/

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

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

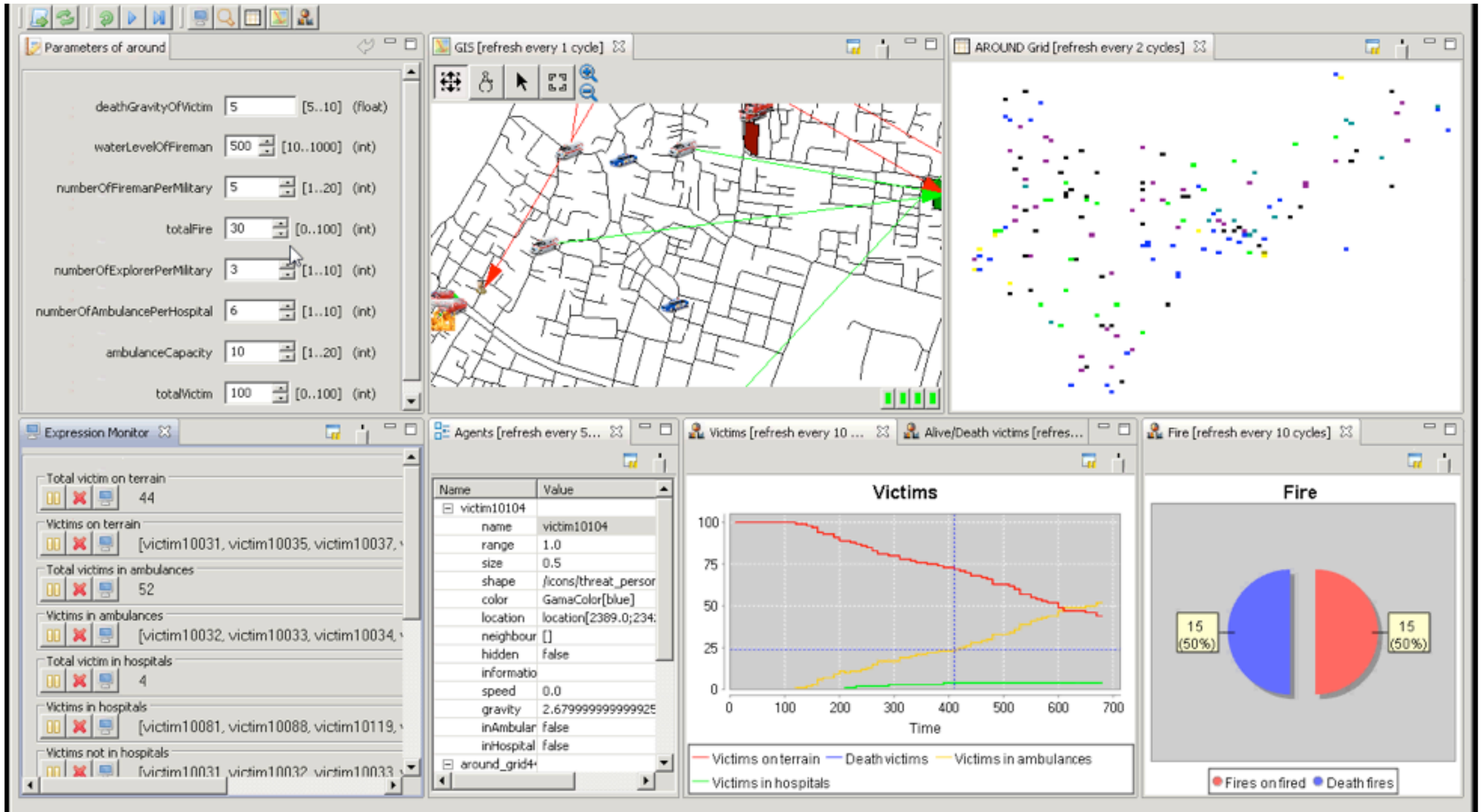






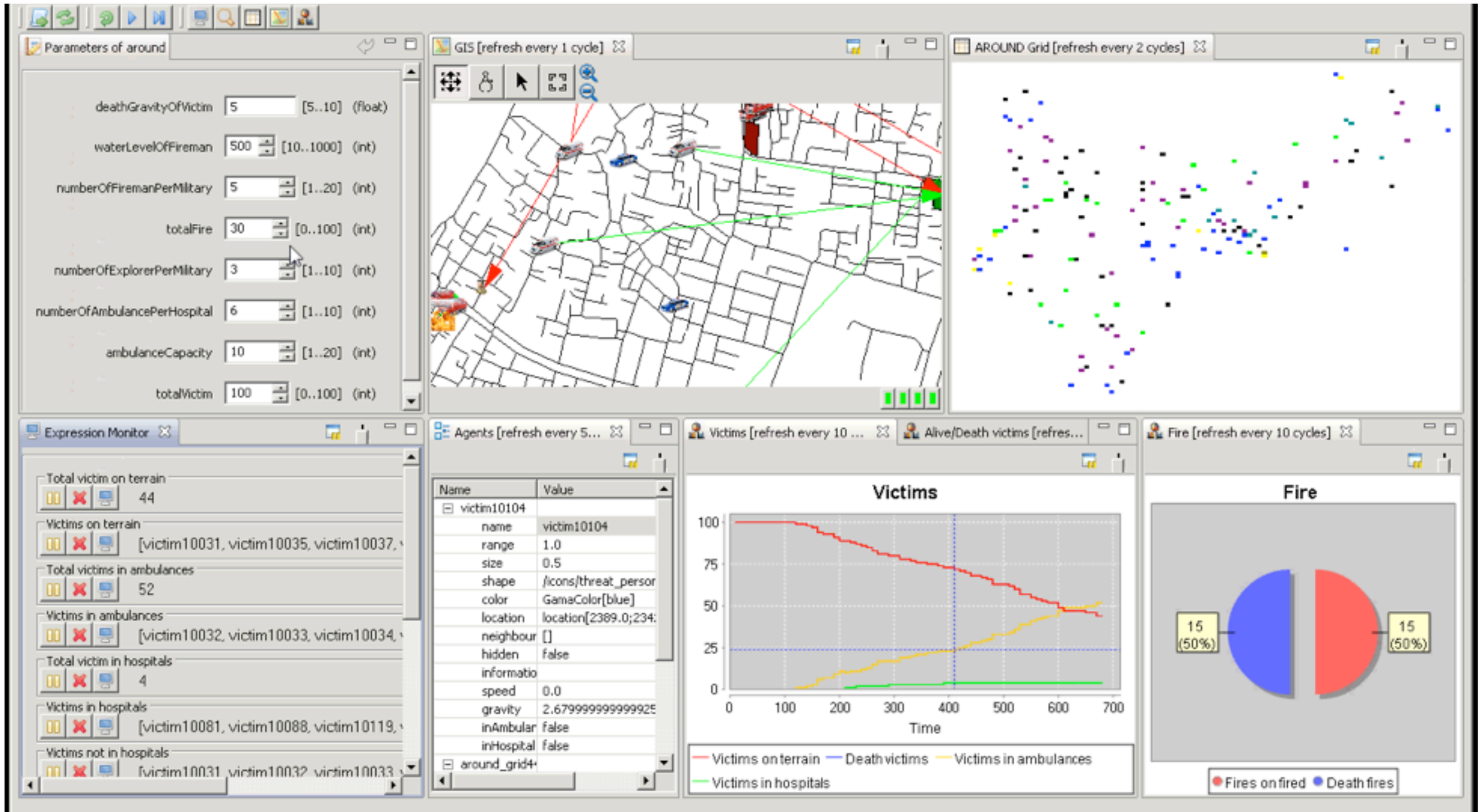
Miniature laboratories

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



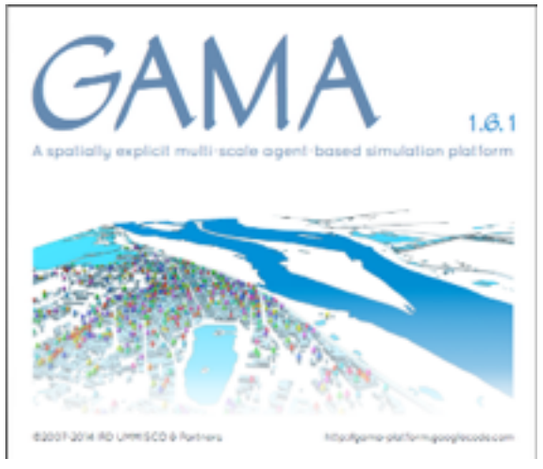
Miniature laboratories

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



Generalities on the GAMA platform

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



Current version: 1.6.1

http://www.gama-platform.org

GAMA platform Website

Download GAMA

Documentation of GAMA

- Agent-based, spatially explicit, modeling and simulation platform - Google Project Hosting

patrick.taillandier@gmail.com ▼ | My favorites ▼ | Profile | Sign out

gama-platform

Agent-based, spatially explicit, modeling and simulation p

Search projects

Project Home | Downloads | Wiki | Issues | Source | Administer | Export to GitHub

READ-ONLY: This project has been archived. For more information see this post.

Summary People

Project Information

Project feeds

Code license
GNU GPL v3

Labels
Simulation, Platform, GIS, ABM, Modeling

Members
alexis.drogoul, patrick.taillandier, benoit.gaudou
1 contributor

Your role
Owner

Links

External links
Youtube
Facebook

Groups
Users group
Developers group

The GAMA website and repository are currently being moved to GitHub. We apologize for any trouble you may experience while we achieve this transition.

Starting June, 30th 2015:

- The SVN repository on GoogleCode goes read-only (i.e. no more commits allowed). The source code is now accessible through GitHub.
- The documentation wiki on GoogleCode will remain unchanged. A new documentation wiki is available on GitHub.
- Issues on GoogleCode will remain unchanged. Please post new ones on GitHub.
- Downloads of the GAMA 1.6.1 release will remain on GoogleCode for now on. The release of GAMA 1.7, expected in July, will use another infrastructure.
- The gama-dev and gama-platform mailing lists remain unchanged.

G A M A is a modeling and simulation development environment for building spatially explicit agent-based simulations. Its latest version, 1.6.1, can be freely downloaded or built from source, and comes pre-loaded with several models, tutorials and a complete on-line documentation.

G A M A allows to:

- Design, prototype and write models in the **GAML agent-oriented language** and its optional **graphical modeling tool**.
- Instantiate agents from any kind of dataset, including **GIS data**, and execute **large-scale simulations** (up to millions of agents).

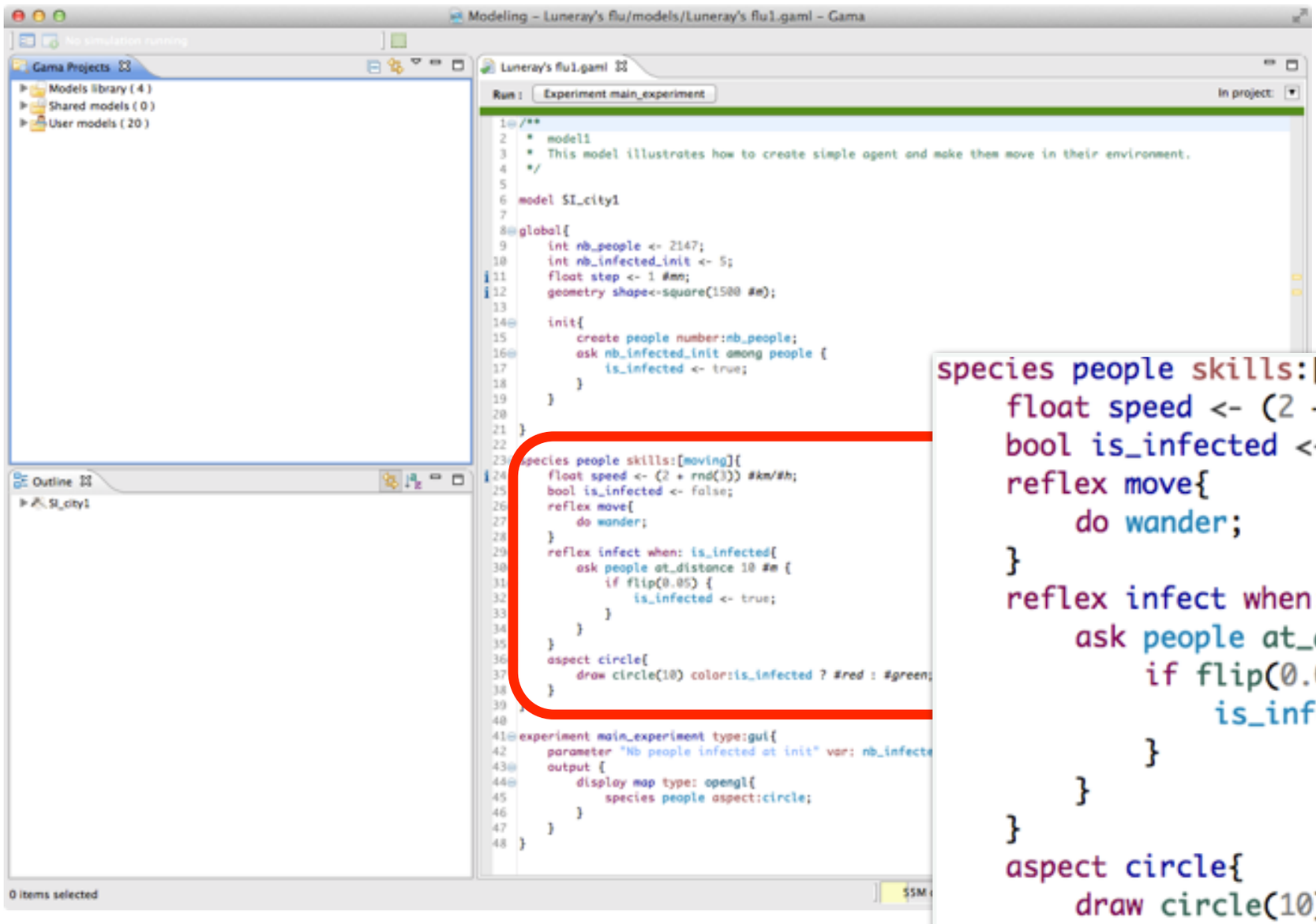
http://www.gama-platform.org

- 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

https://www.youtube.com/watch?v=ycbeYxV2B7M

Generalities concerning GAMA

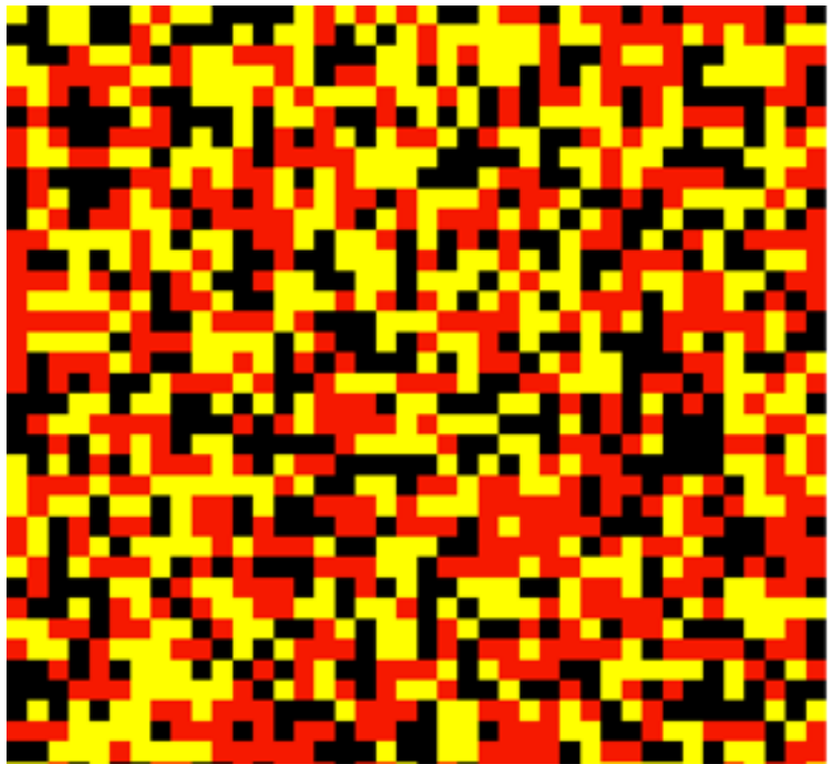
- GAMA provides a complete Integrated Development environment (IDE) to build models

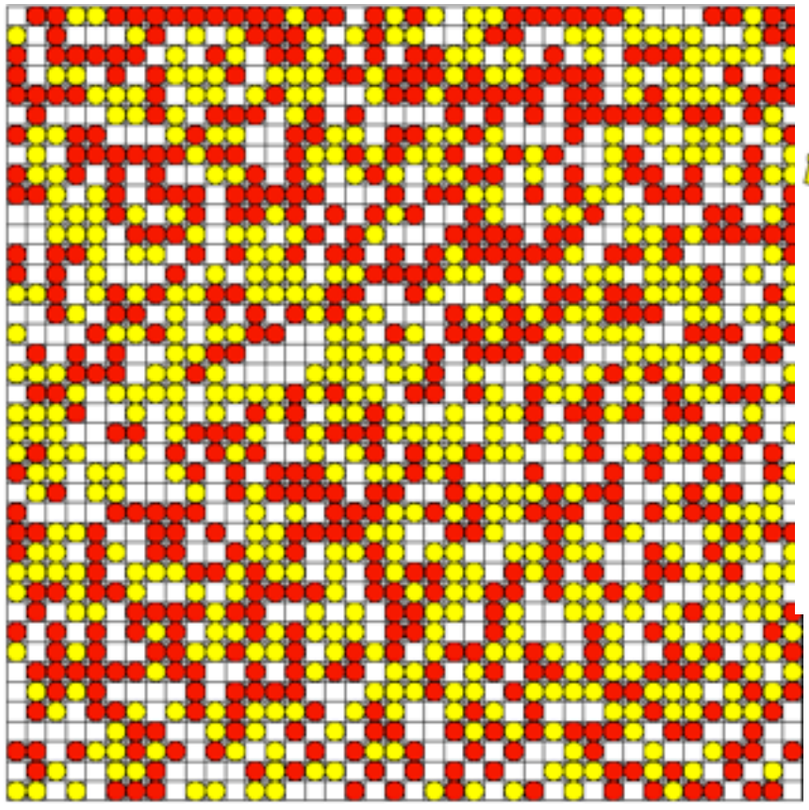


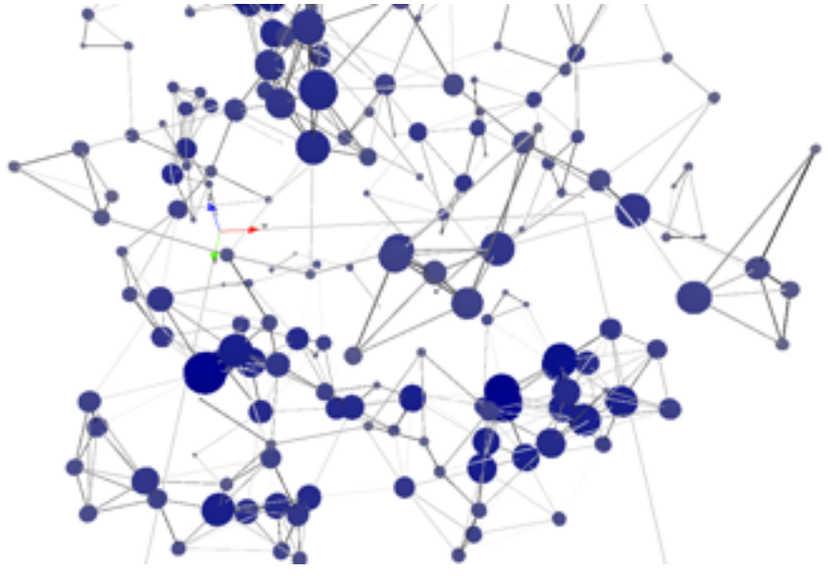

```
species people skills:[moving]{
    float speed <- (2 + rnd(3)) #km/#h;
    bool is_infected <- false;
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance 10 #m {
            if flip(0.05) {
                is_infected <- true;
            }
        }
    }
    aspect circle{
        draw circle(10) color:is_infected ? #red : #green;
    }
}
```

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)

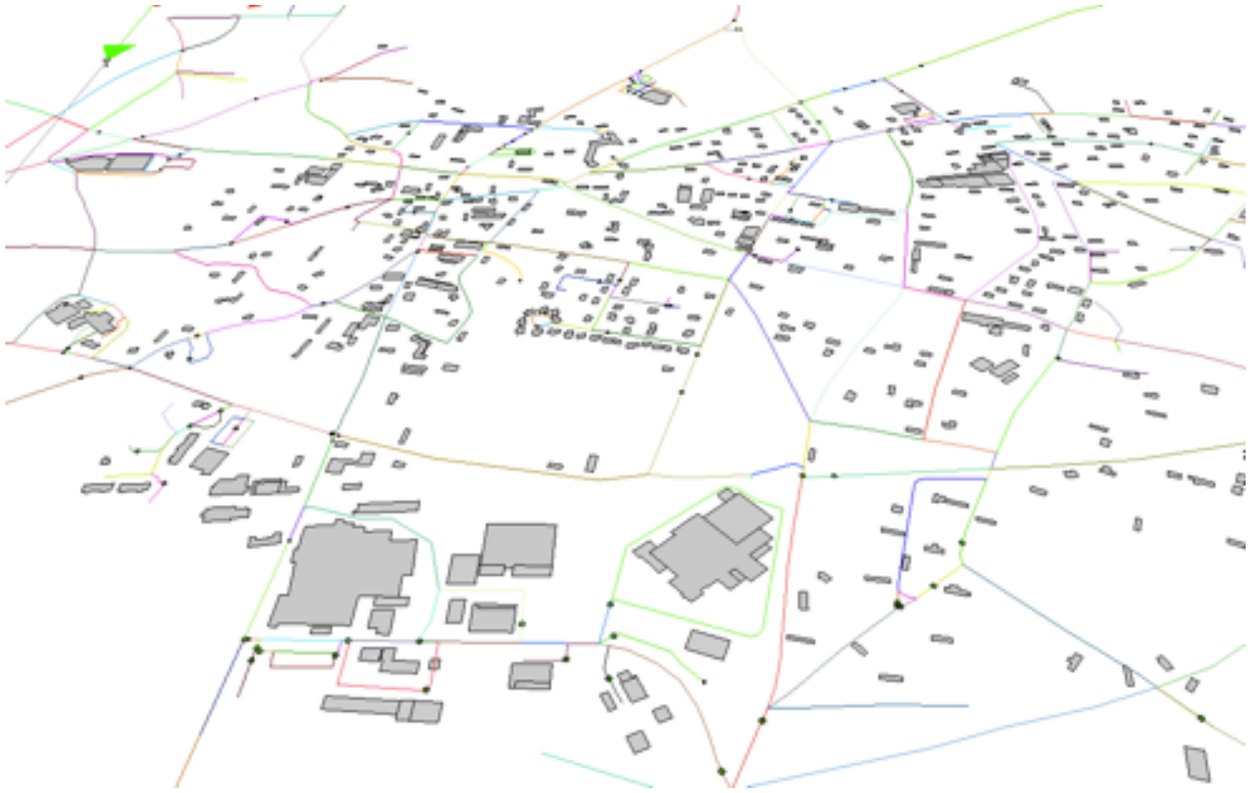


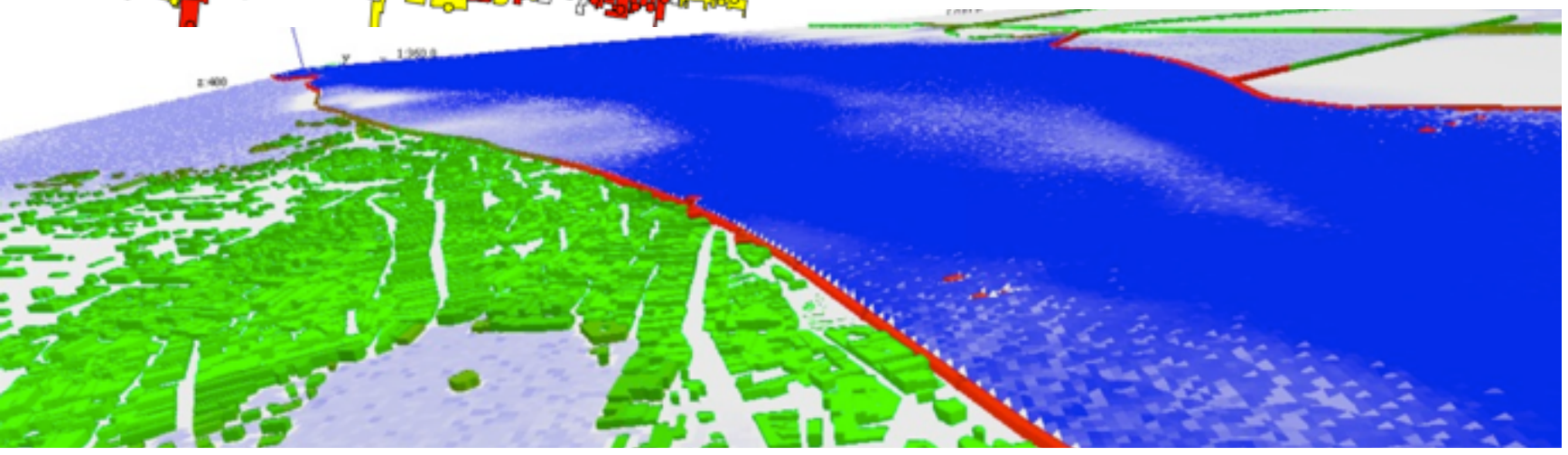


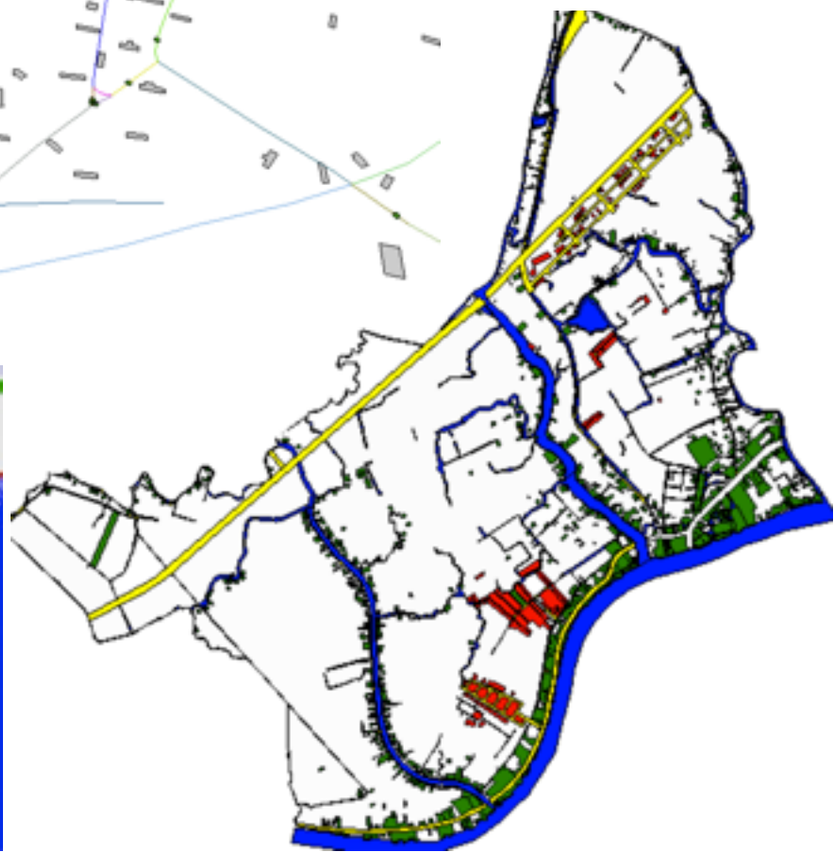


2D Grid (rectangle, hexagon)
Continuous environment, torus environment, graphs

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

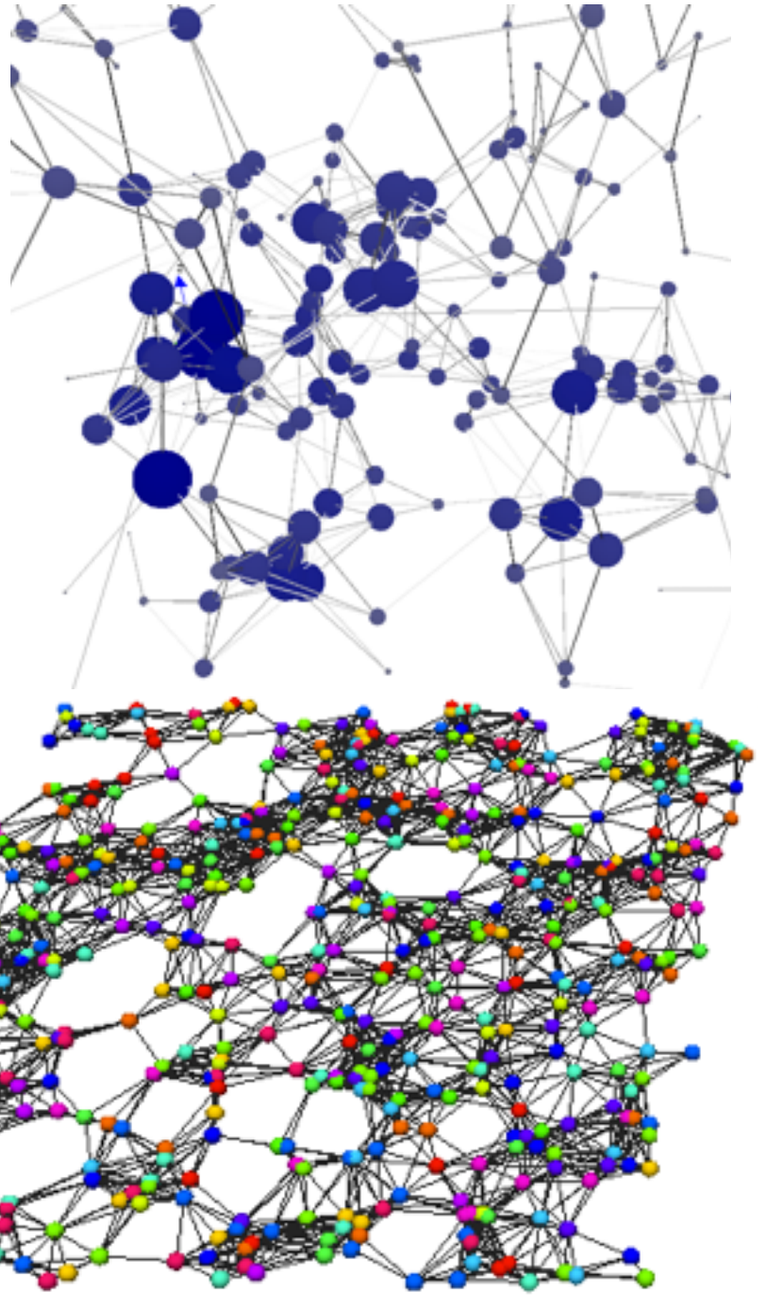


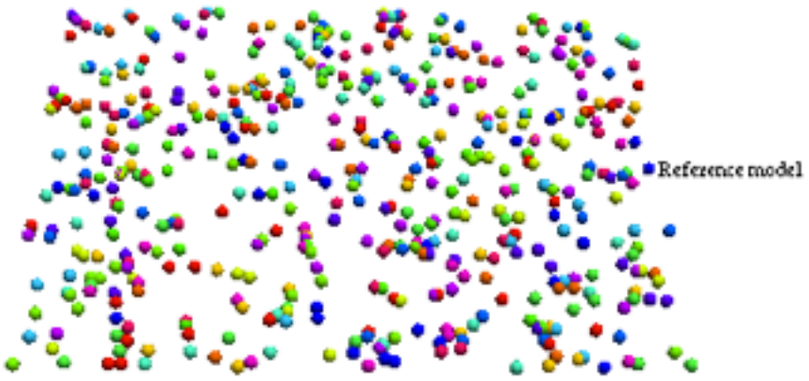


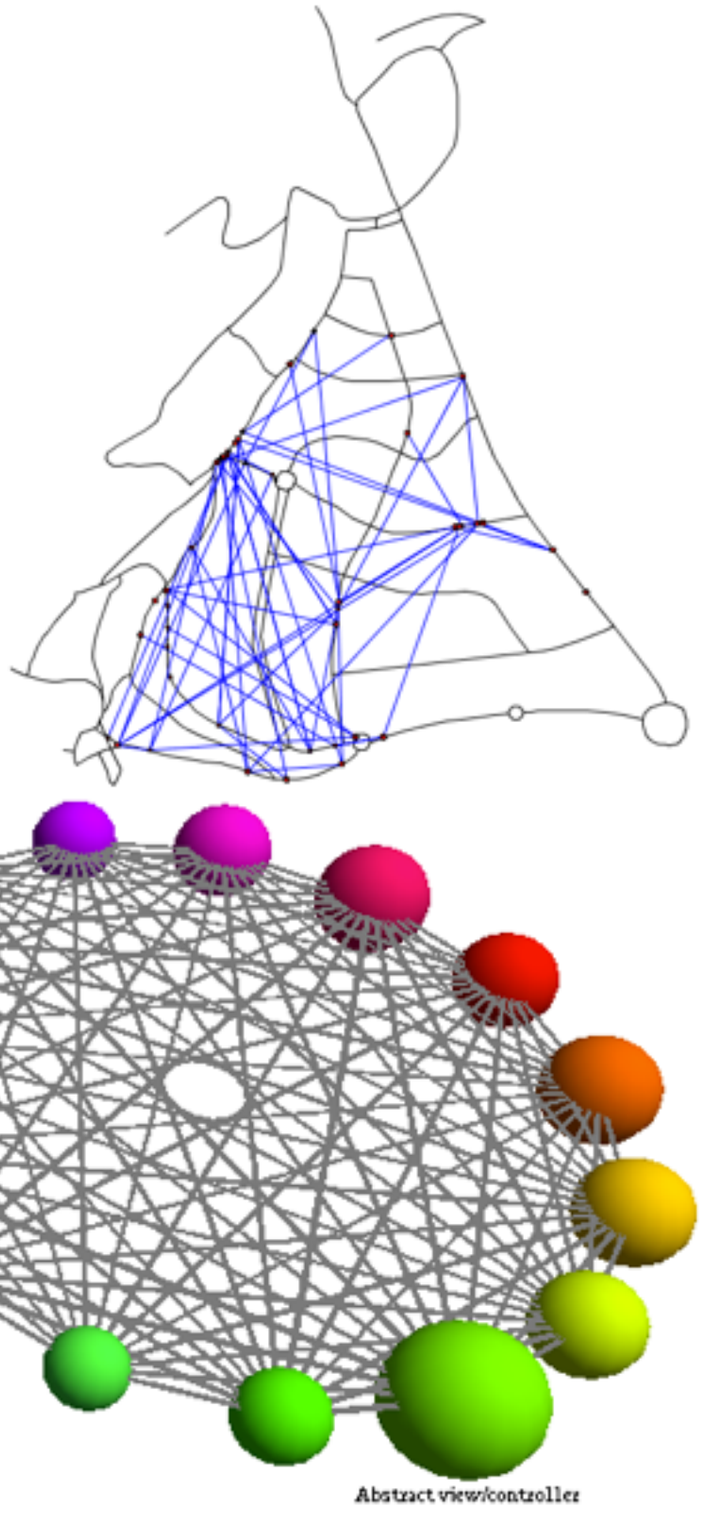


Transparent agentification of 2D/3D GIS data
powerful geometrical operations (union, difference, spatial queries....)

- Many graph operators





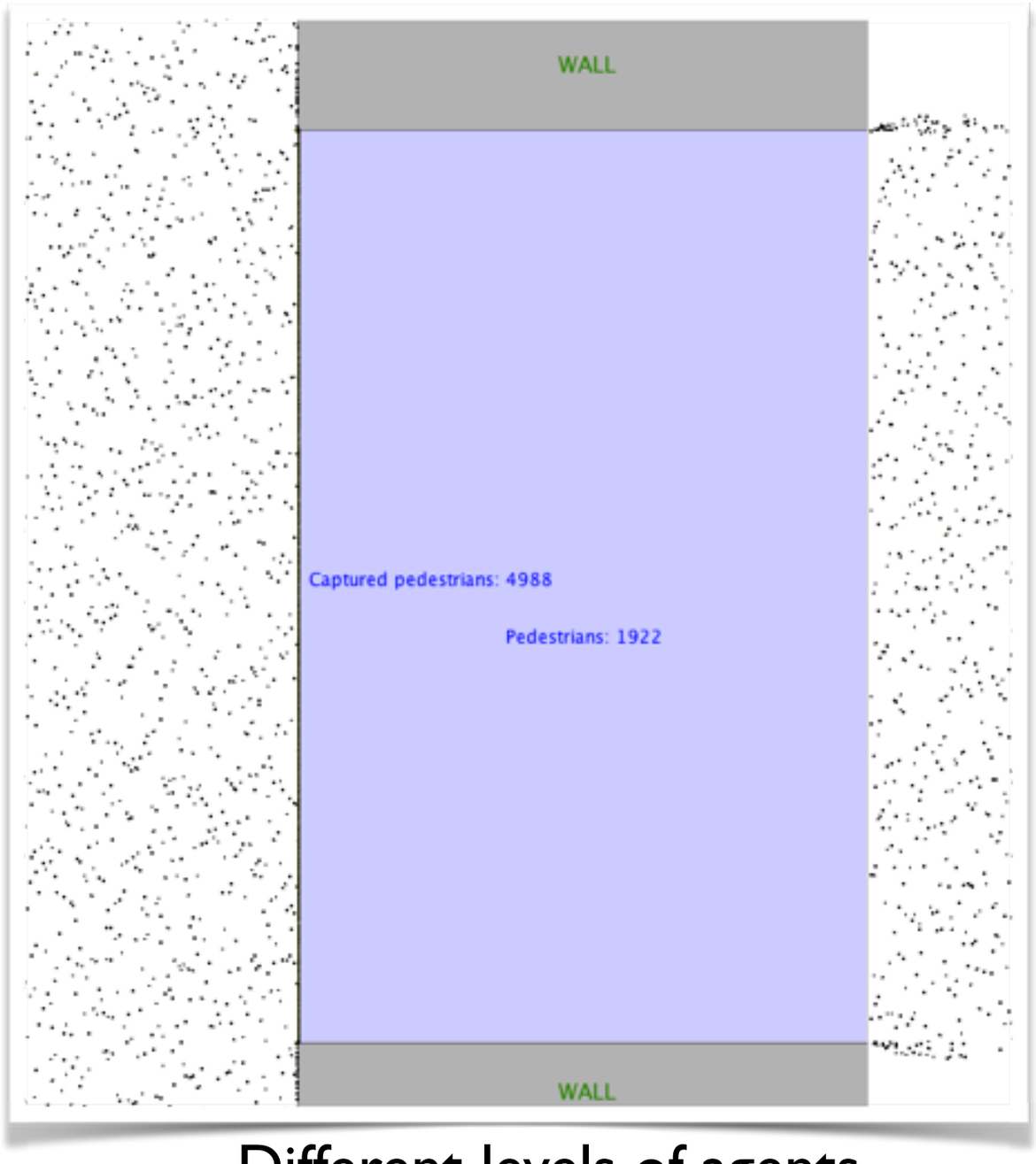


Advanced view

k-shortest paths, centrality betweenness, graph generation…

- Allows to define multi-level models

Emergence of macro-agents



Different levels of agents

Different level of abstraction, aggregation operators...

- Allows to use different formalisms to define agent behaviors

```
reflex infect when: is_infected{
    ask people at_distance 10 #m {
        if flip(0.05) {
            is_infected <- true;
        }
    }
}
```

Reflexes

```
equation eqSI {
    diff(S,t) = -beta * S * I / N ;
    diff(I,t) = beta * S * I / N ;
}
```

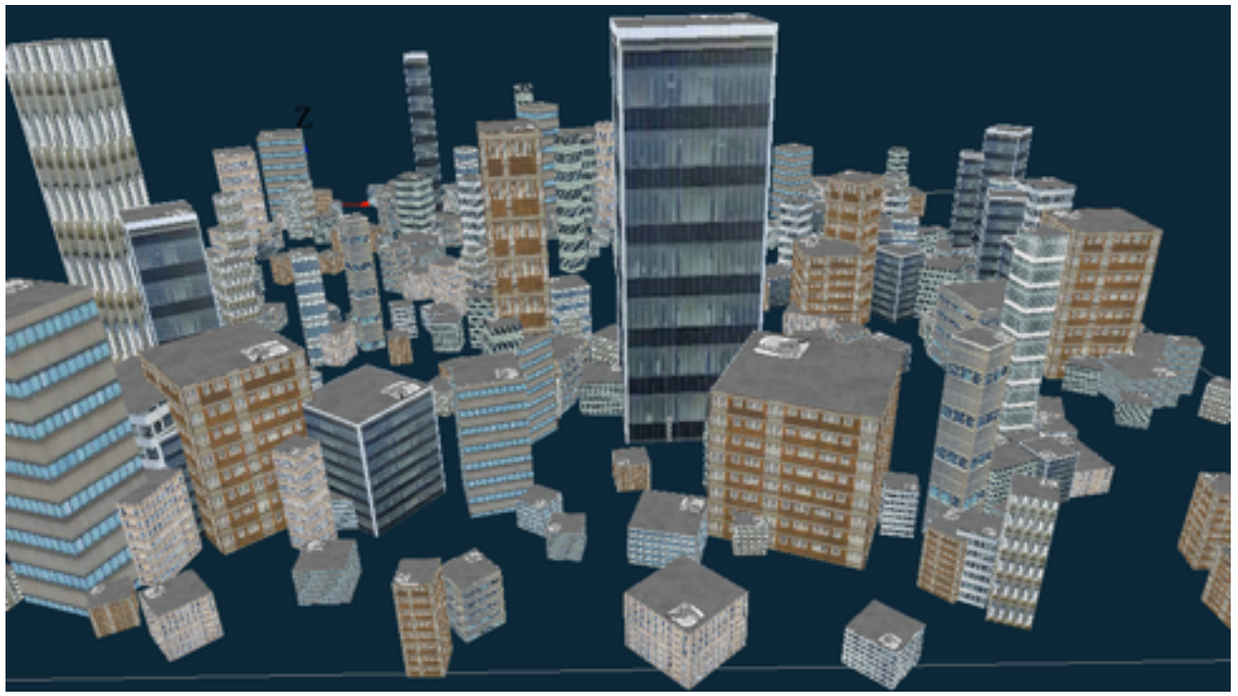
Differential Equations

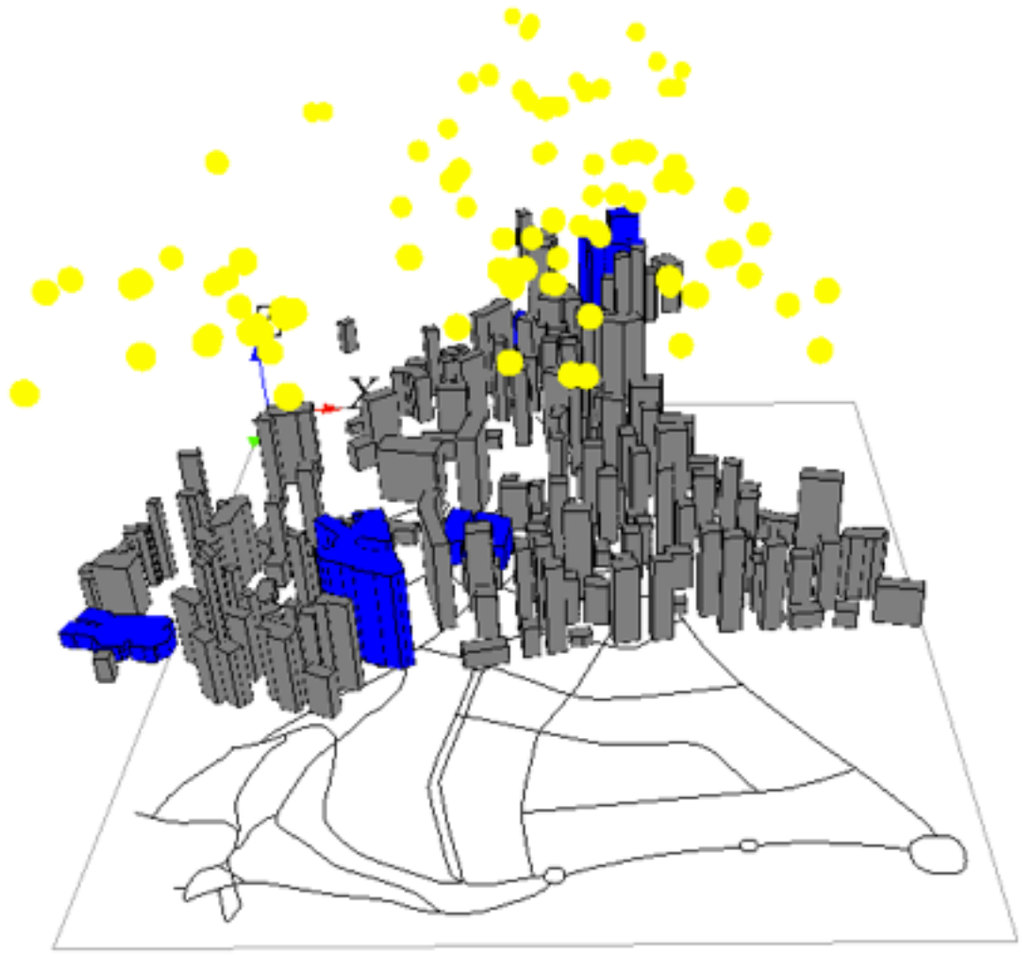
```
state followingRoad {
    location <- (self choose_best_place()) as point ;
    transition to: carryingFood when: place.food > 0 {
        do pick ;
    }
    transition to: wandering when: (place.road < 0.05) ;
}
```

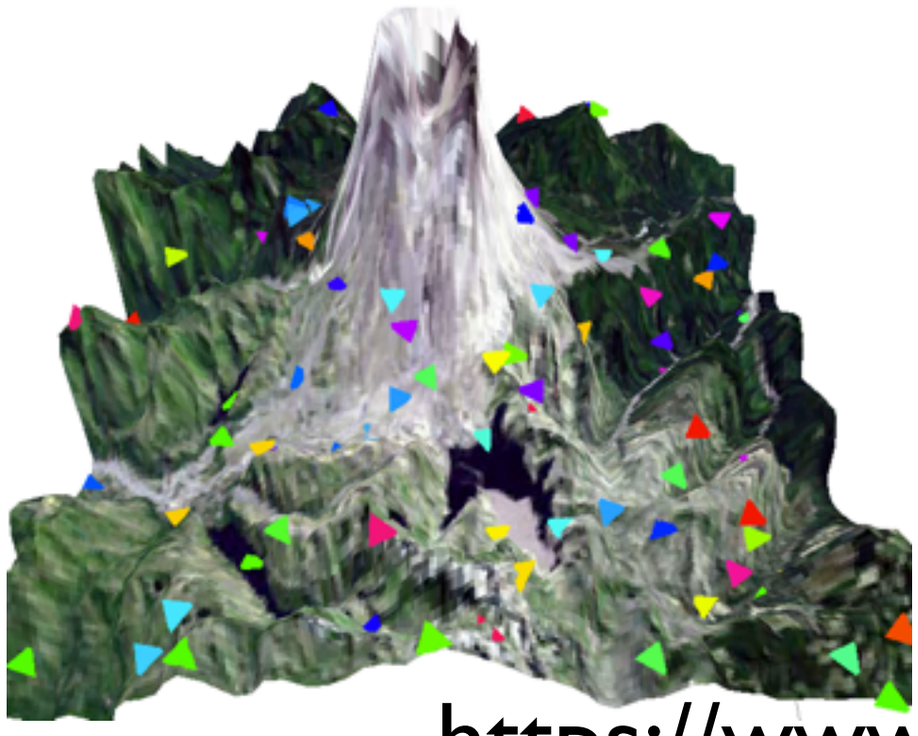
Finite state machine

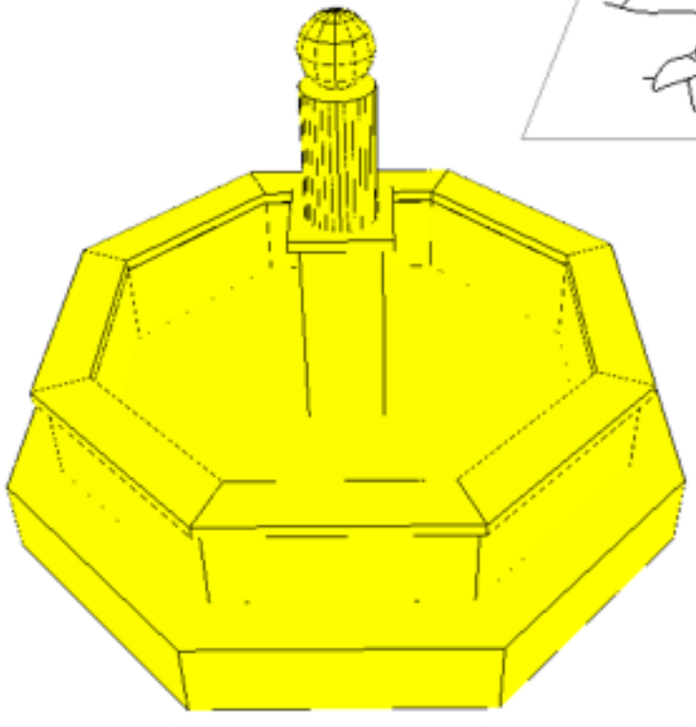
Differential equations, Finite state machine, Reflexes, ...

- 3D modeling and visualization





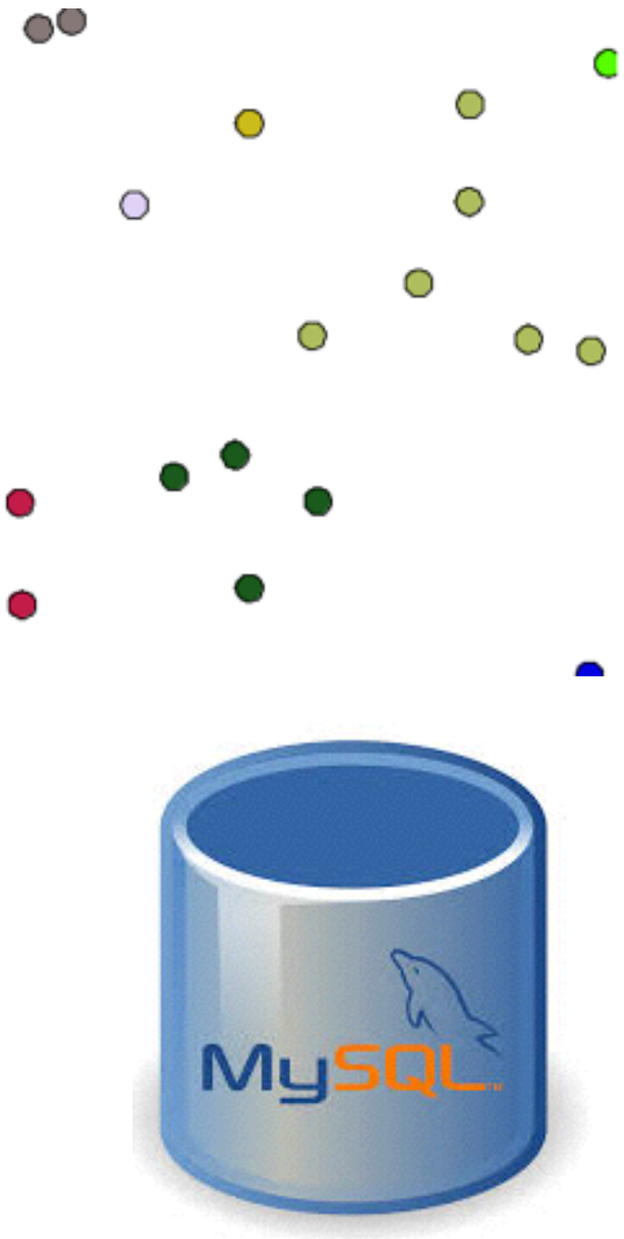


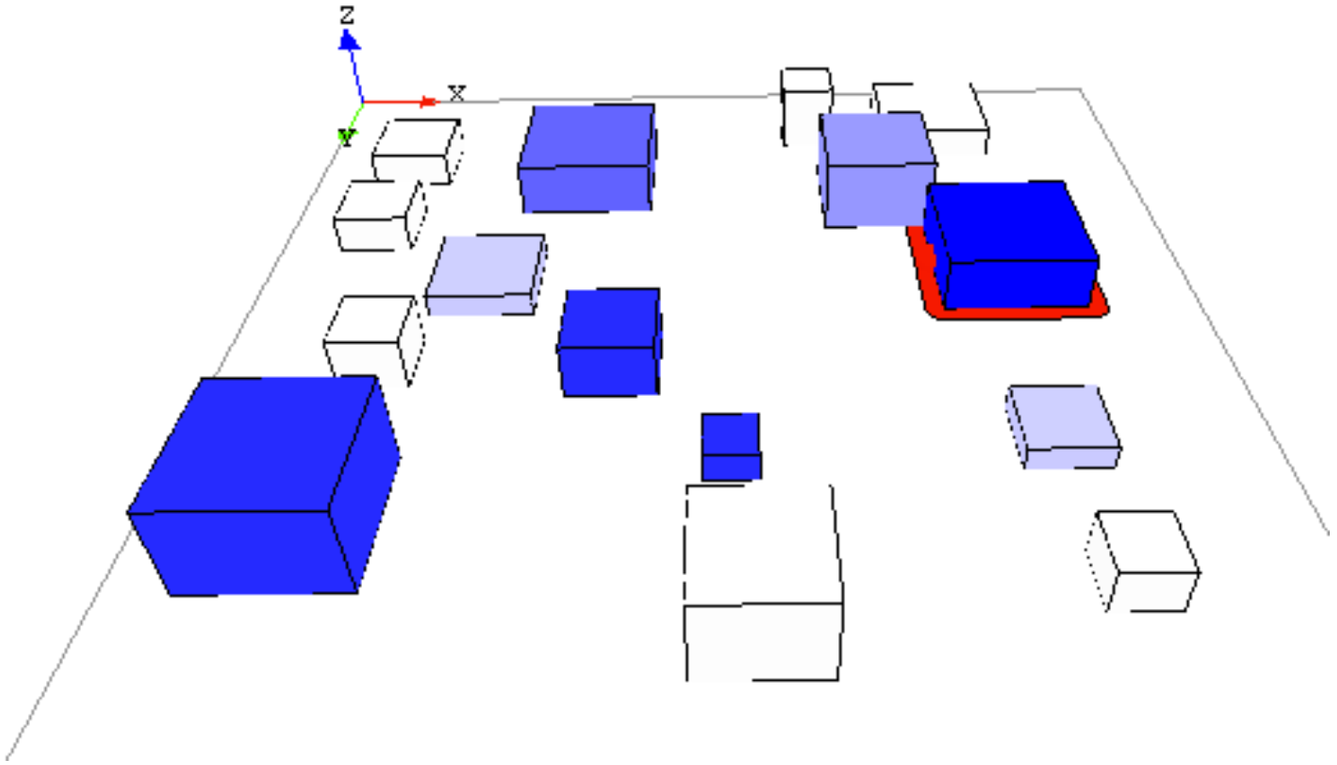


https://www.youtube.com/watch?v=t4FcZp5FbCg
https://www.youtube.com/watch?v=9C2PTLjN7x4

Generalities concerning GAMA

- Many operators and features included

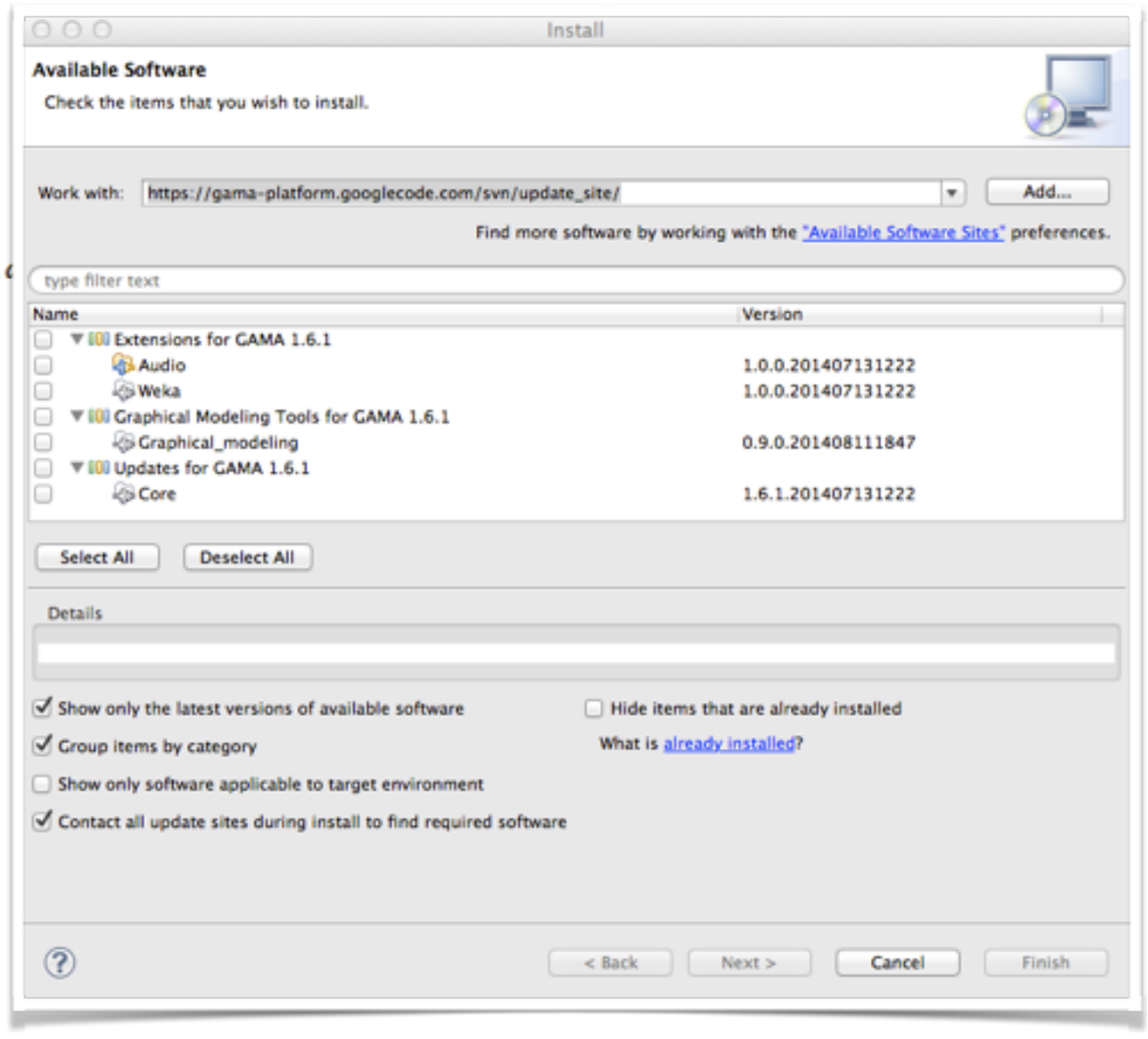




Link to database, use of R script, clustering, multi-criteria decision making, spatial discretization, spatial interpolation…

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

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

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



- Parameter space exploration tools and compatible with OpenMole

Parameters

Model si_model *Parameters for experiment 'Genetic'*

Random number generator 'mersenne' *among [cellular, xor, java, mersenne]*

Random seed Define: 0.0

Exploration method

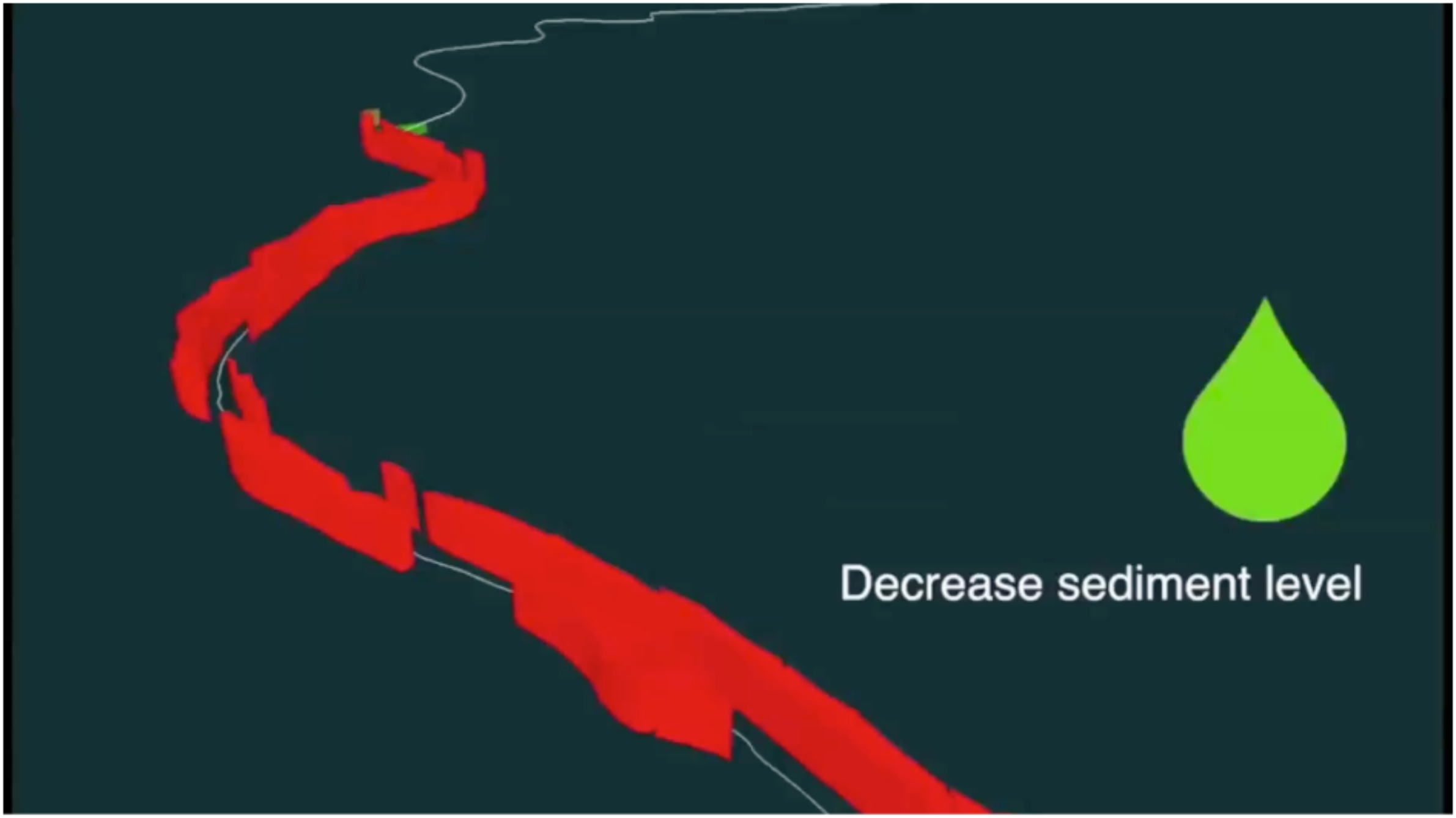
Stop condition	time > 1000	
Best fitness	1.7976931348623157E308	
Last fitness	299.0	*with {infection_rate=0.8, speed_people=5.0}*
Parameter space	infection_rate (6) * speed_people (10) = 60	
Exploration method	Method genetic | fitness = minimize nb_infected | compute the min of 3 simulations for each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

Parameters to explore

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

Example of real models: sediment transport

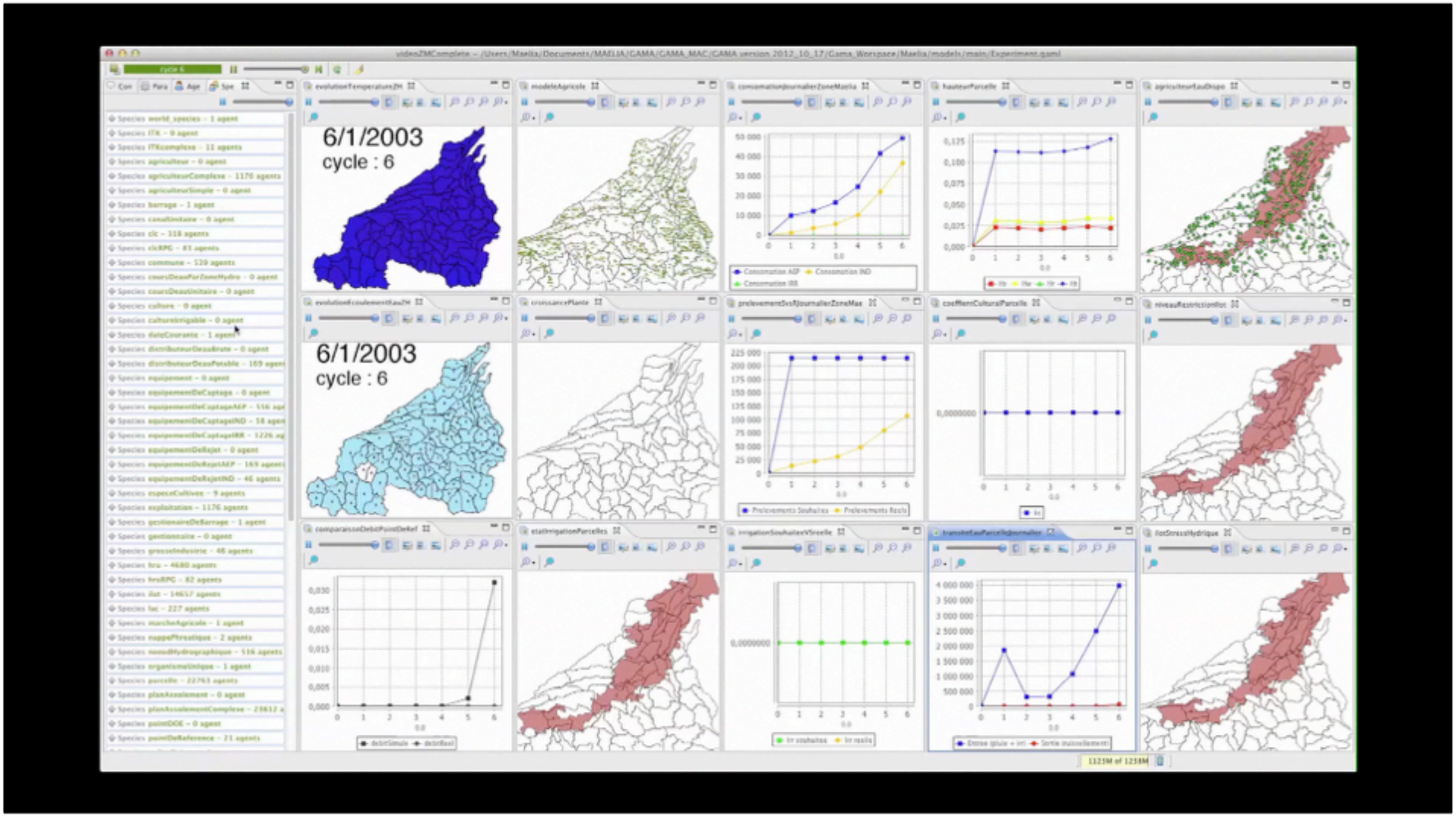
- **Rhône-Alpe**, **France**: better understand the transport of sediments



https://www.youtube.com/watch?v=HWctj1ni5Qk

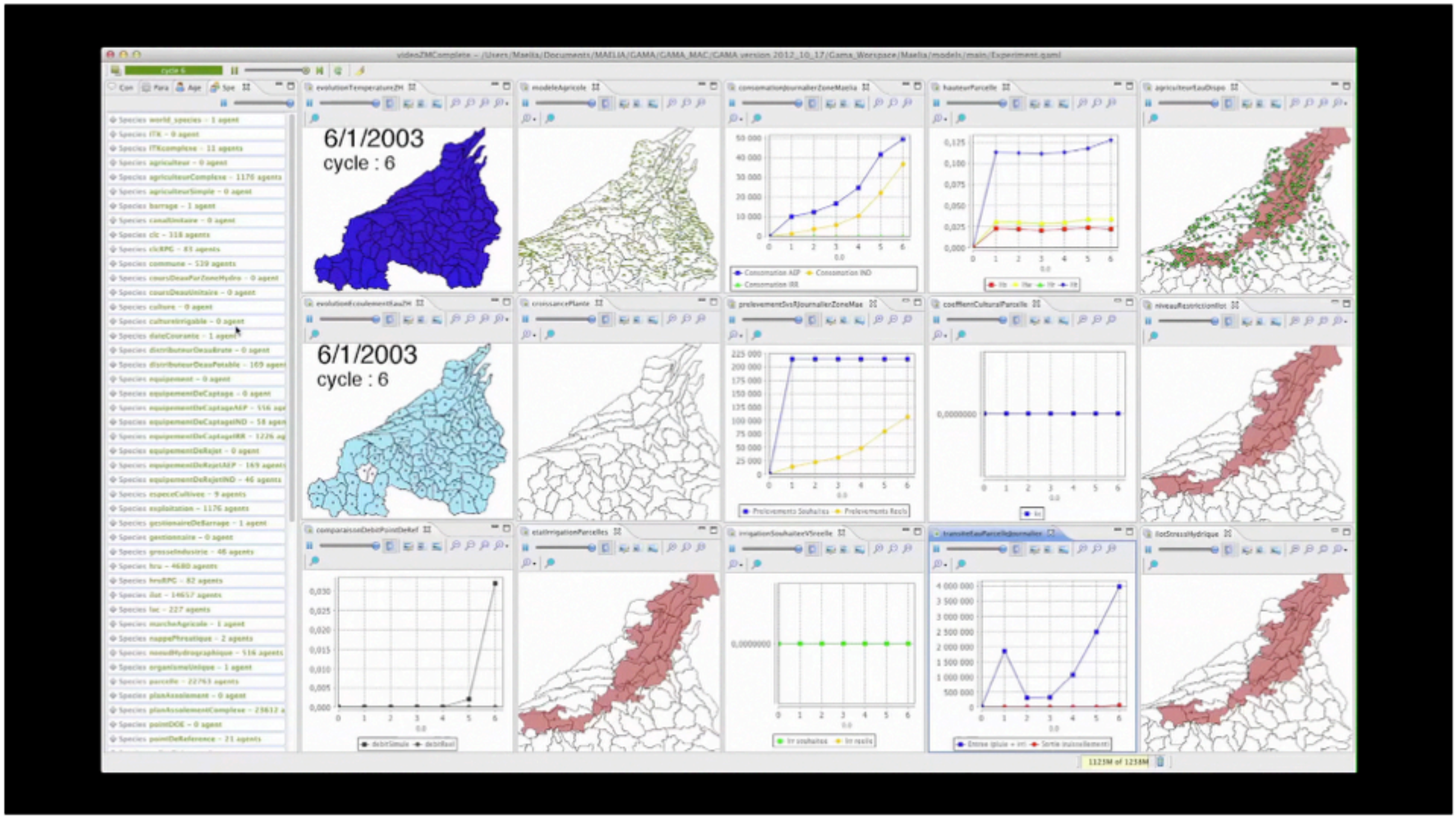
Example of real models : MAELIA

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



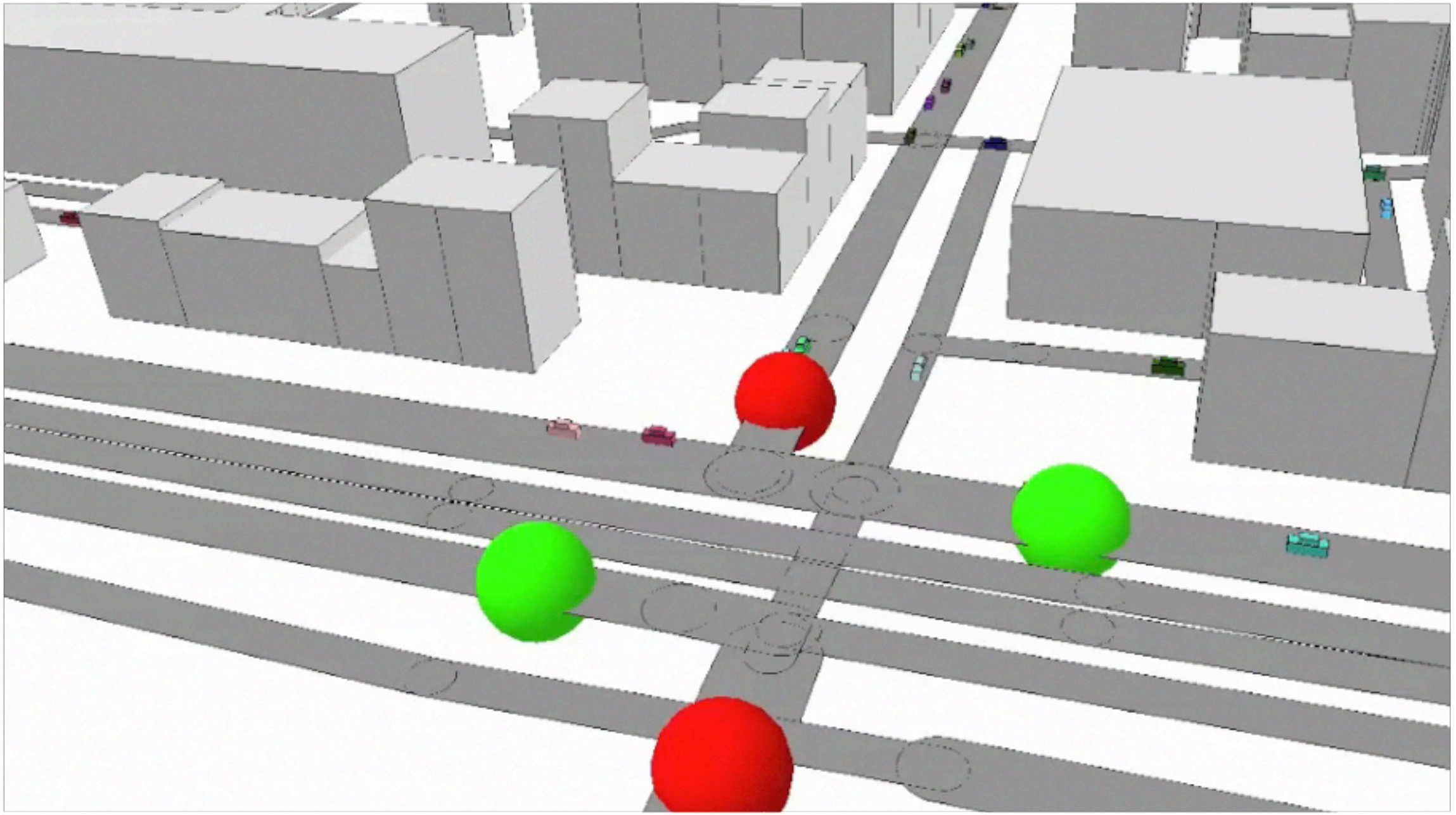
Example of real models : MAELIA

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



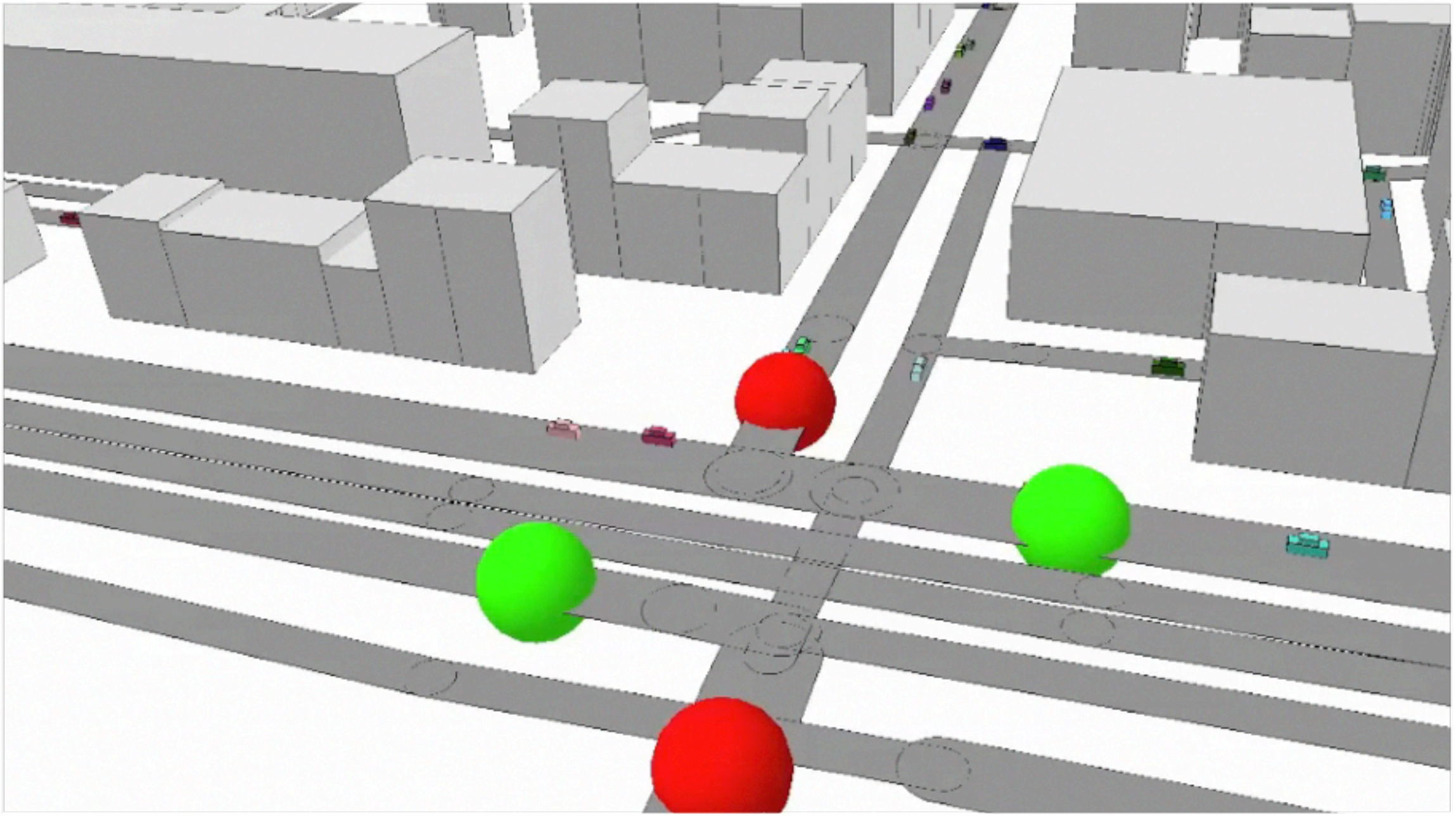
Example of real models : MOSAIIC

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



Example of real models : MOSAIIC

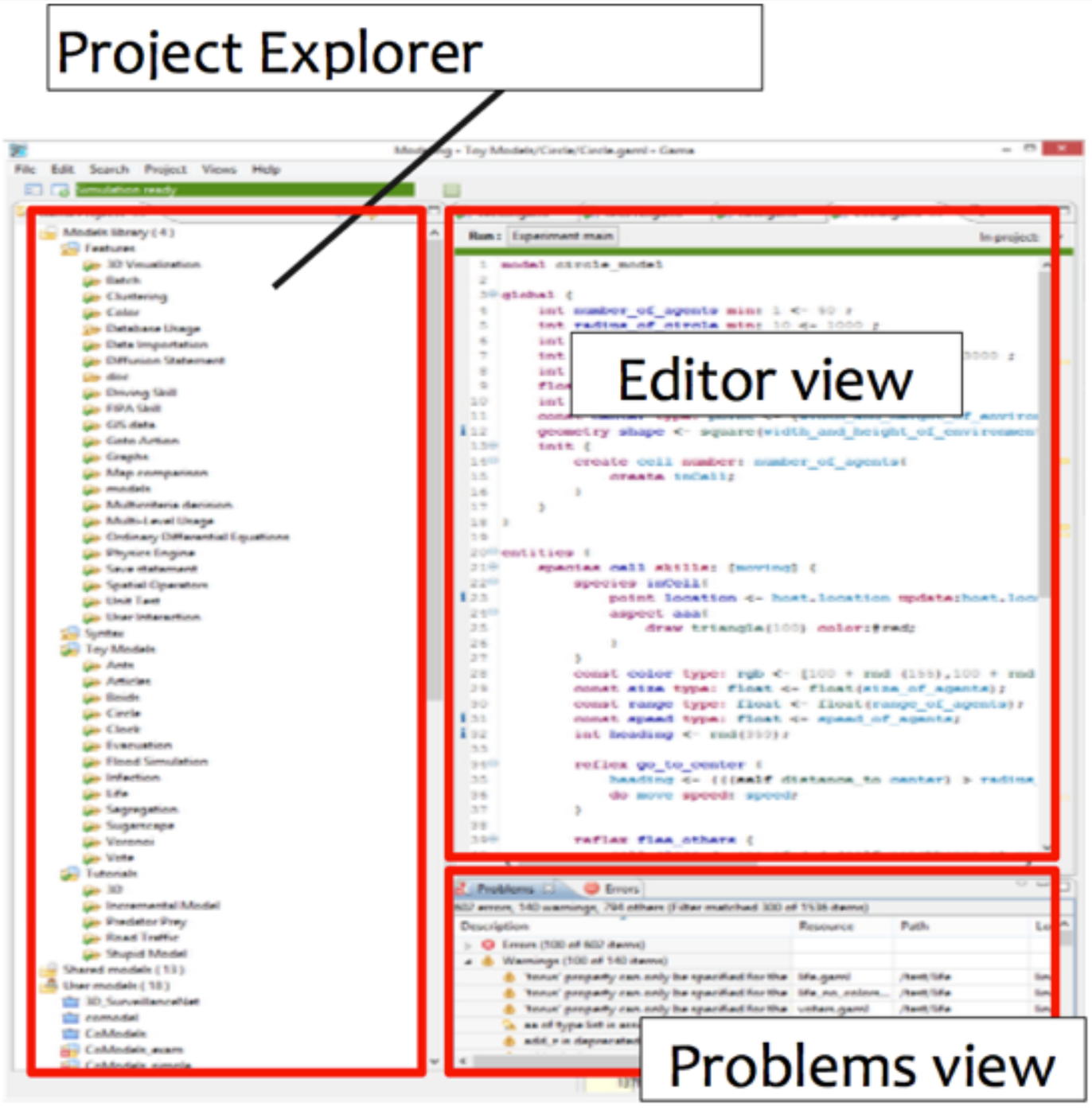
- **Rouen, France**: what is the impact of technological hazards on mobility?



- Each project may contain several models, as well as additional resources (GIS data, pictures, ...).

- Belonging to the same project allows models to use each other and have access to the same resources.

- Projects can be organized in any way, although a default layout (“models”, “doc”, ...) is proposed.

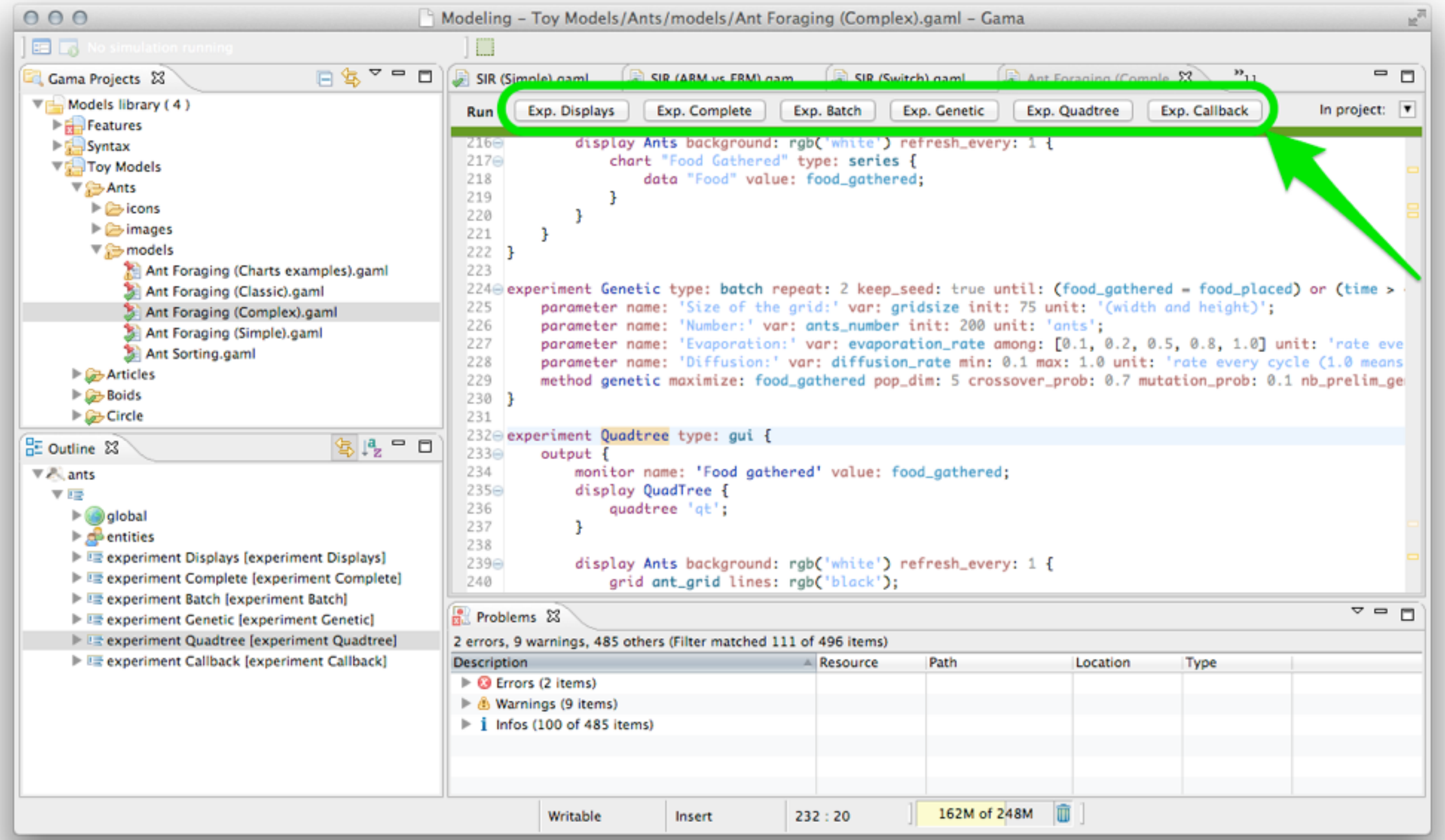


GAMA provides a library of models composed of 4 types of models:

- **Features**: simple models that show how to use some specific features (graph, GIS data, 3D….)
- **Syntax**: models that present the syntax of the GAML language
- **Toy models**: classic agents-based models (ant models, boids, Schelling…)
- **Tutorials**: models linked to online tutorials

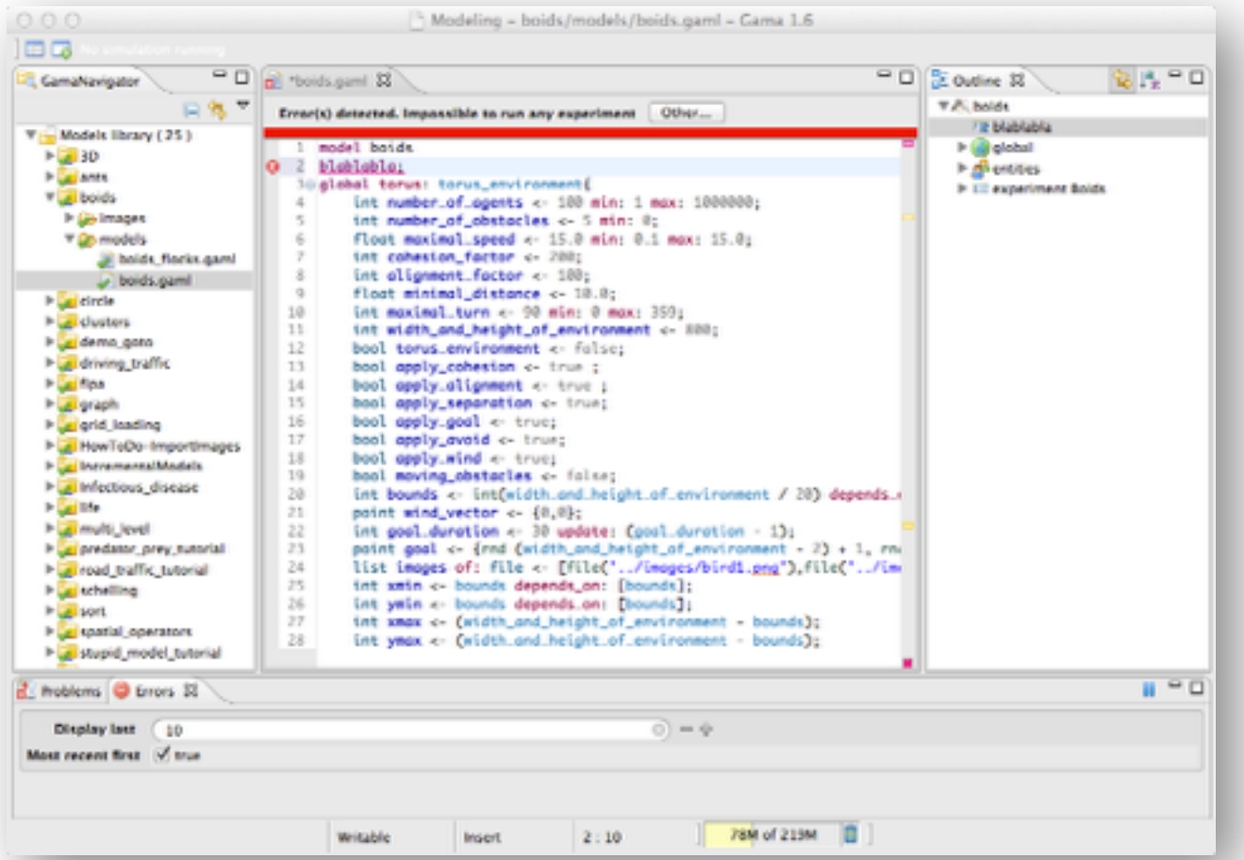
Loading an experiment

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

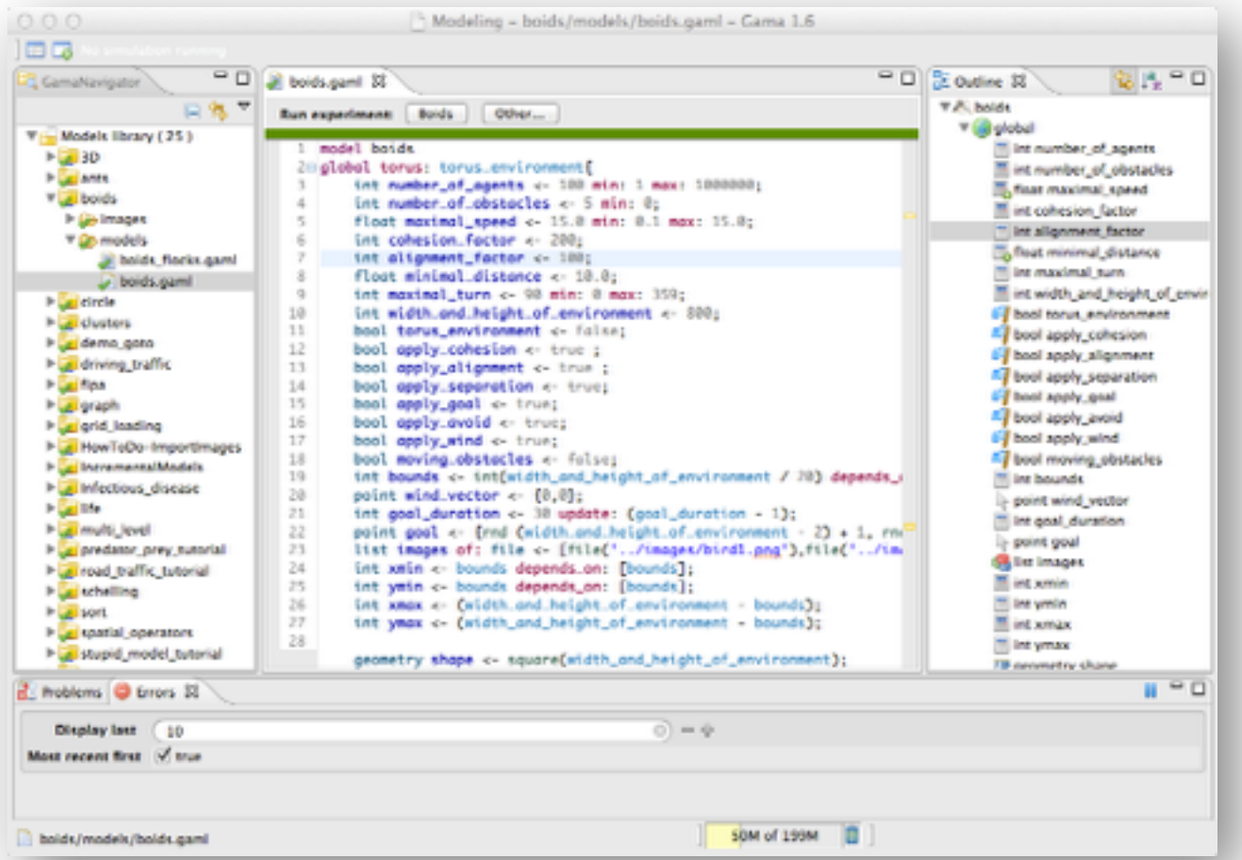


Loading an experiment: model errors

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

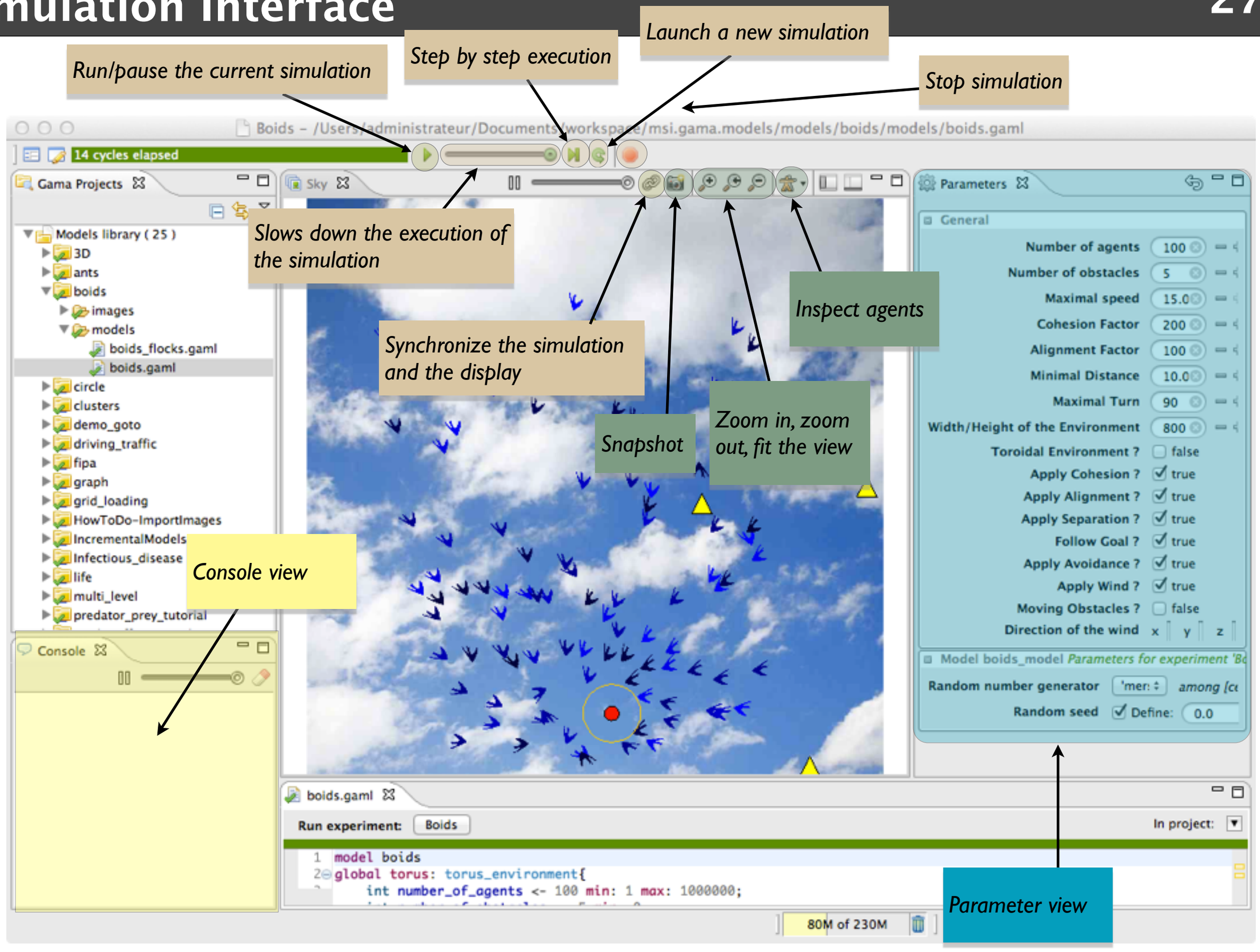


Red: Model with error(s)



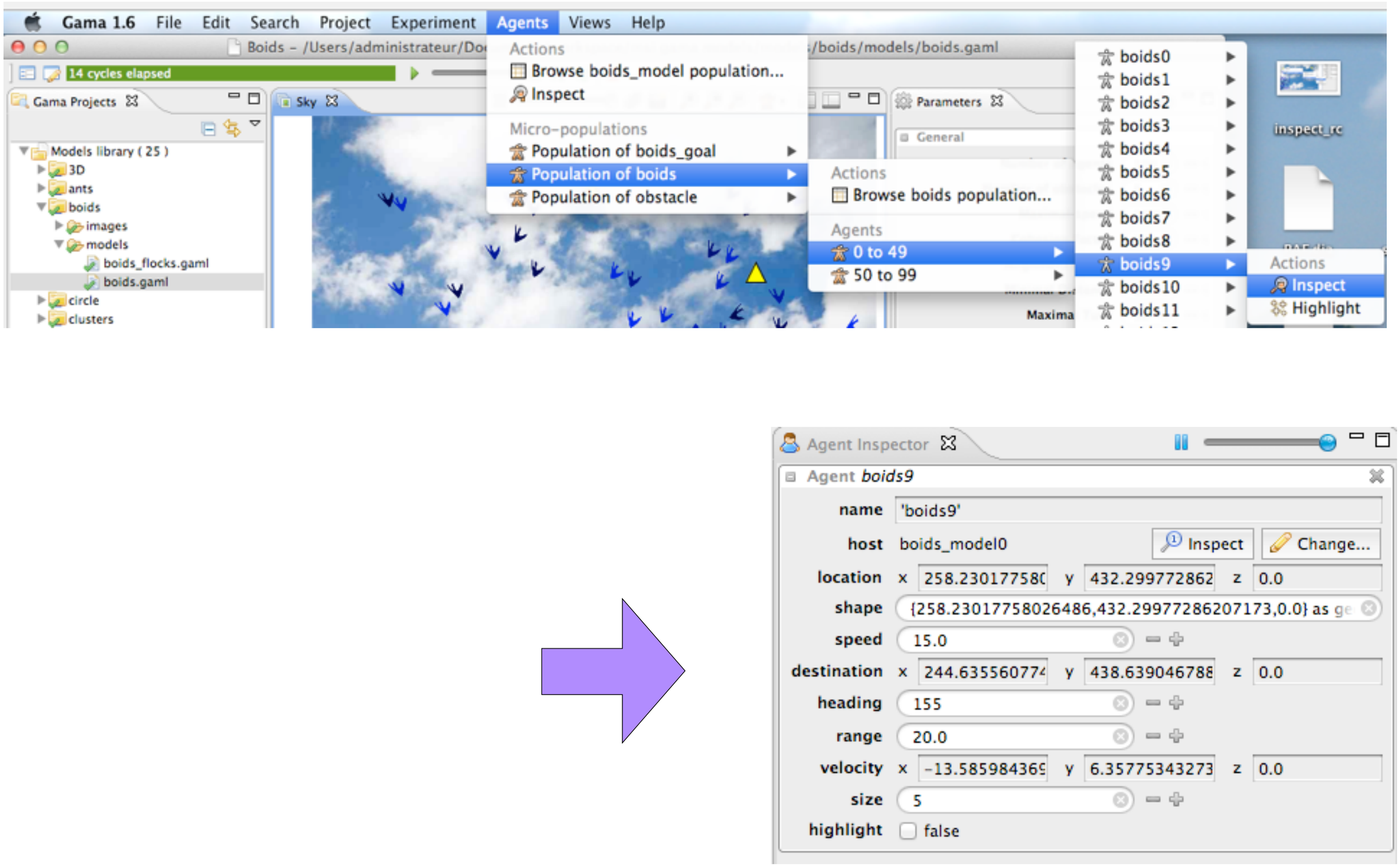
Green: Model without error(s) : ready to load !

Simulation Interface

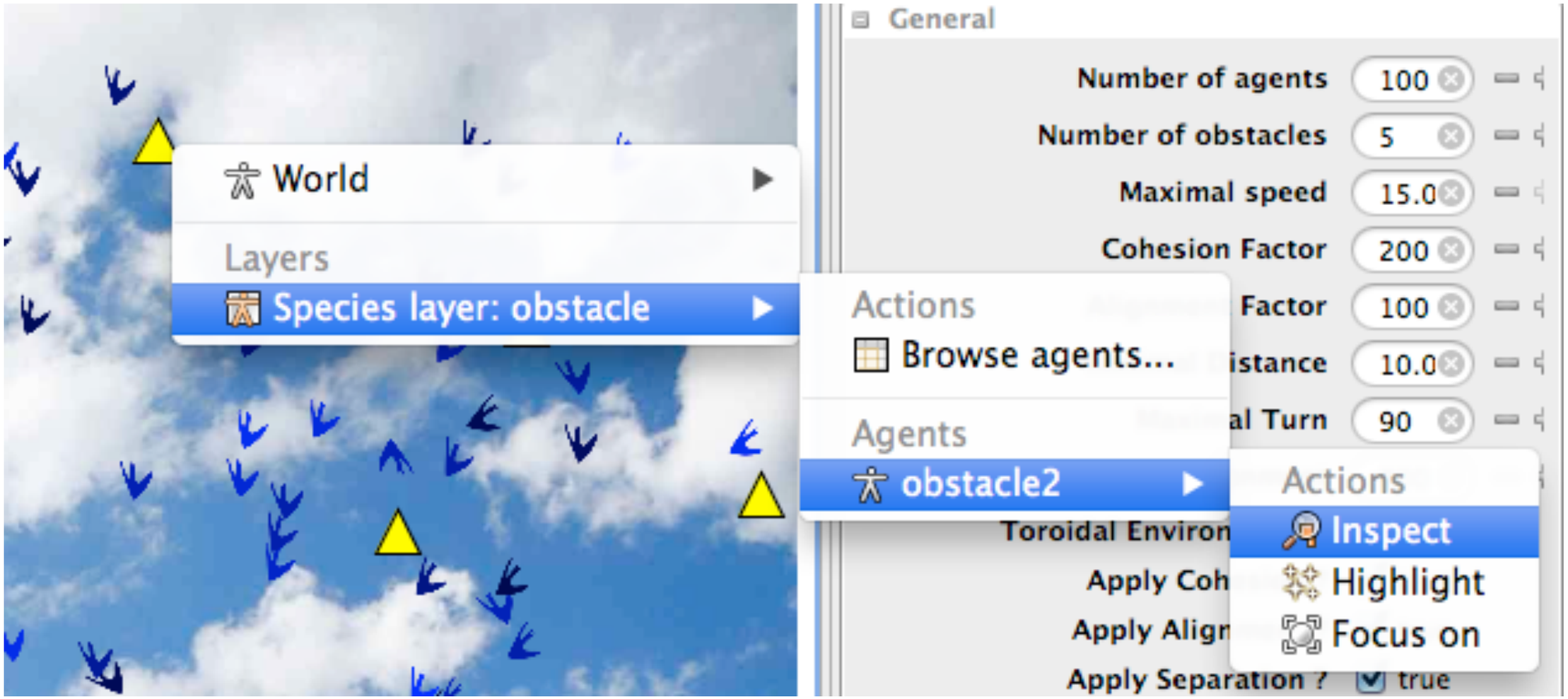


Inspector - 1

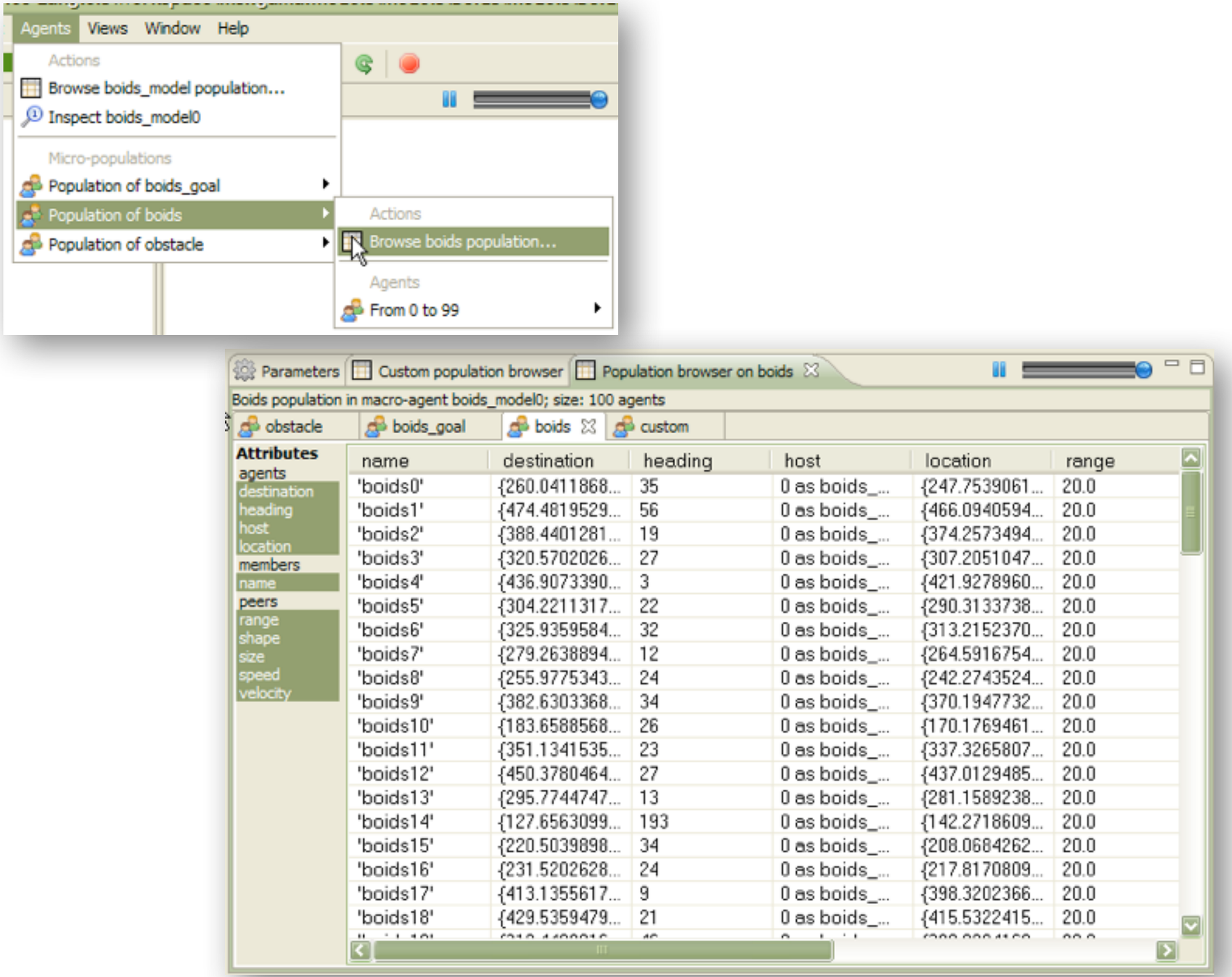
- **Inspector**: allows to obtain information on agents



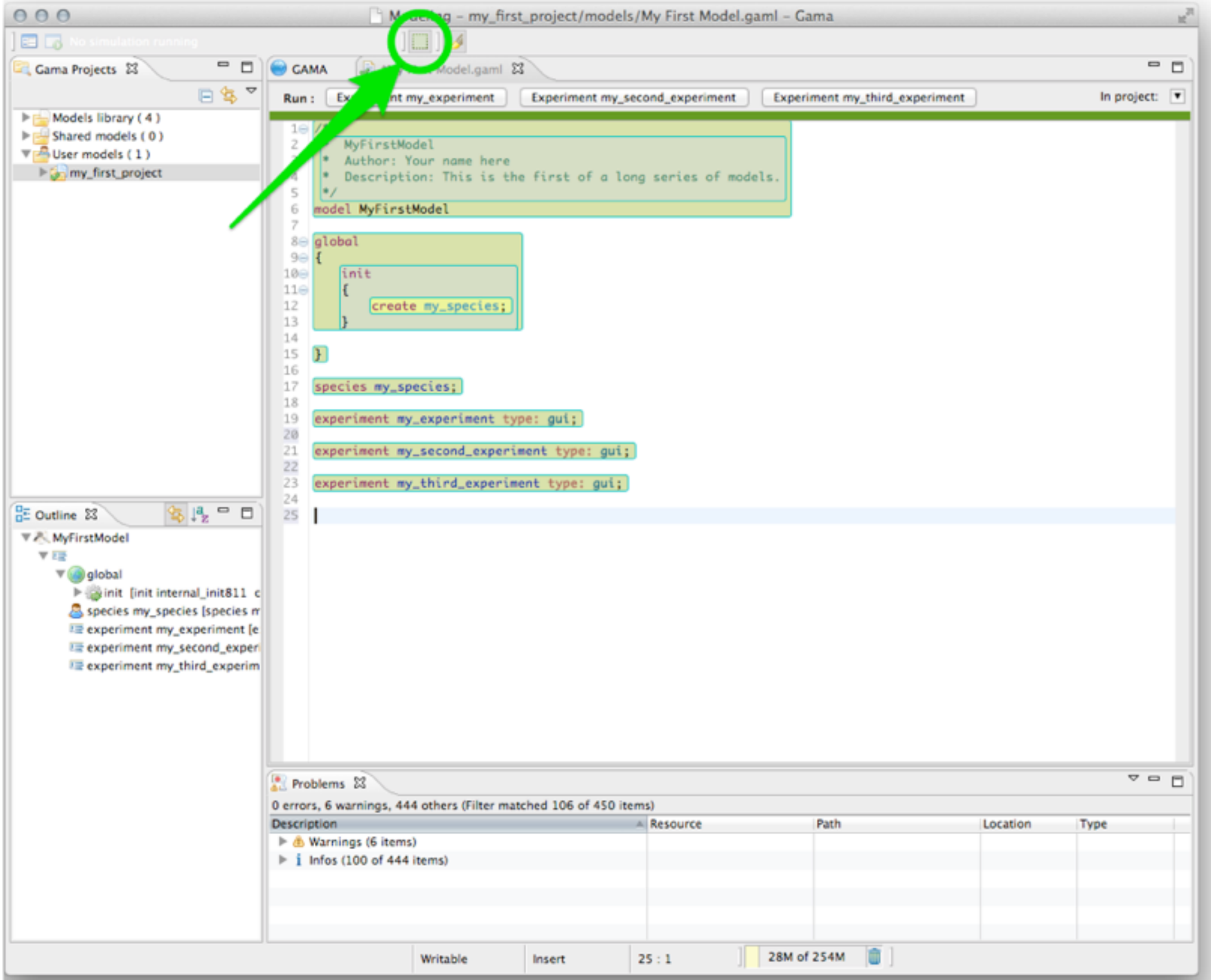
- **Inspector**: allows to obtain information on agents
  - It is possible to inspect an agent by right clicking on it (in a display)



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

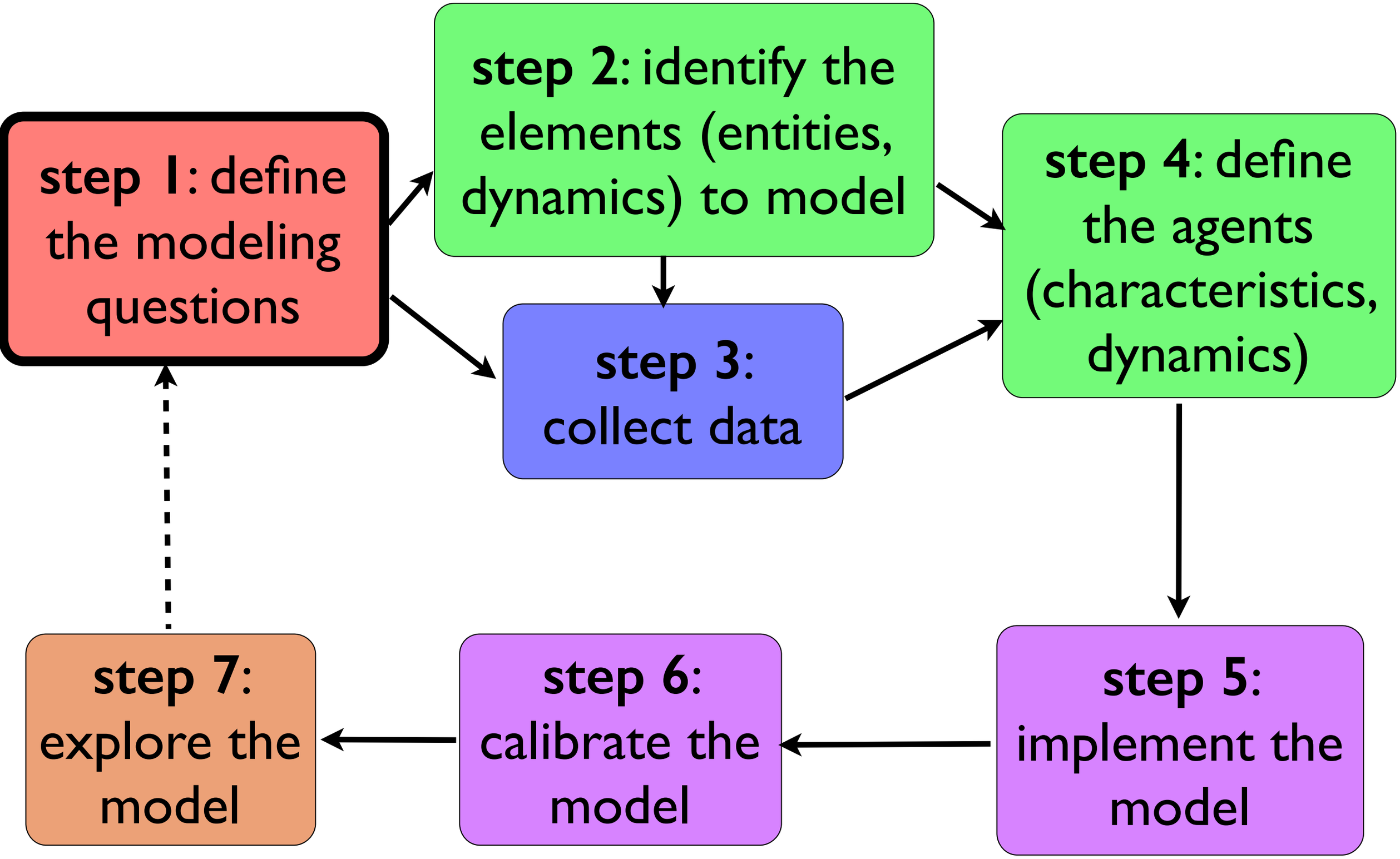


Modeling interface: structural highlighting



Phenomena to study: disease outbreak in a small city

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


step 1: 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 Luneray ?

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

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

- Duration of the study phenomena
  - Several hours

- Dynamic to take into account:
  - Movement of people
  - Disease spreading

- Elements to take into account:
  - The people
  - The buildings (where they are carrying out activities)
  - The roads used for moving

Modeling steps

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

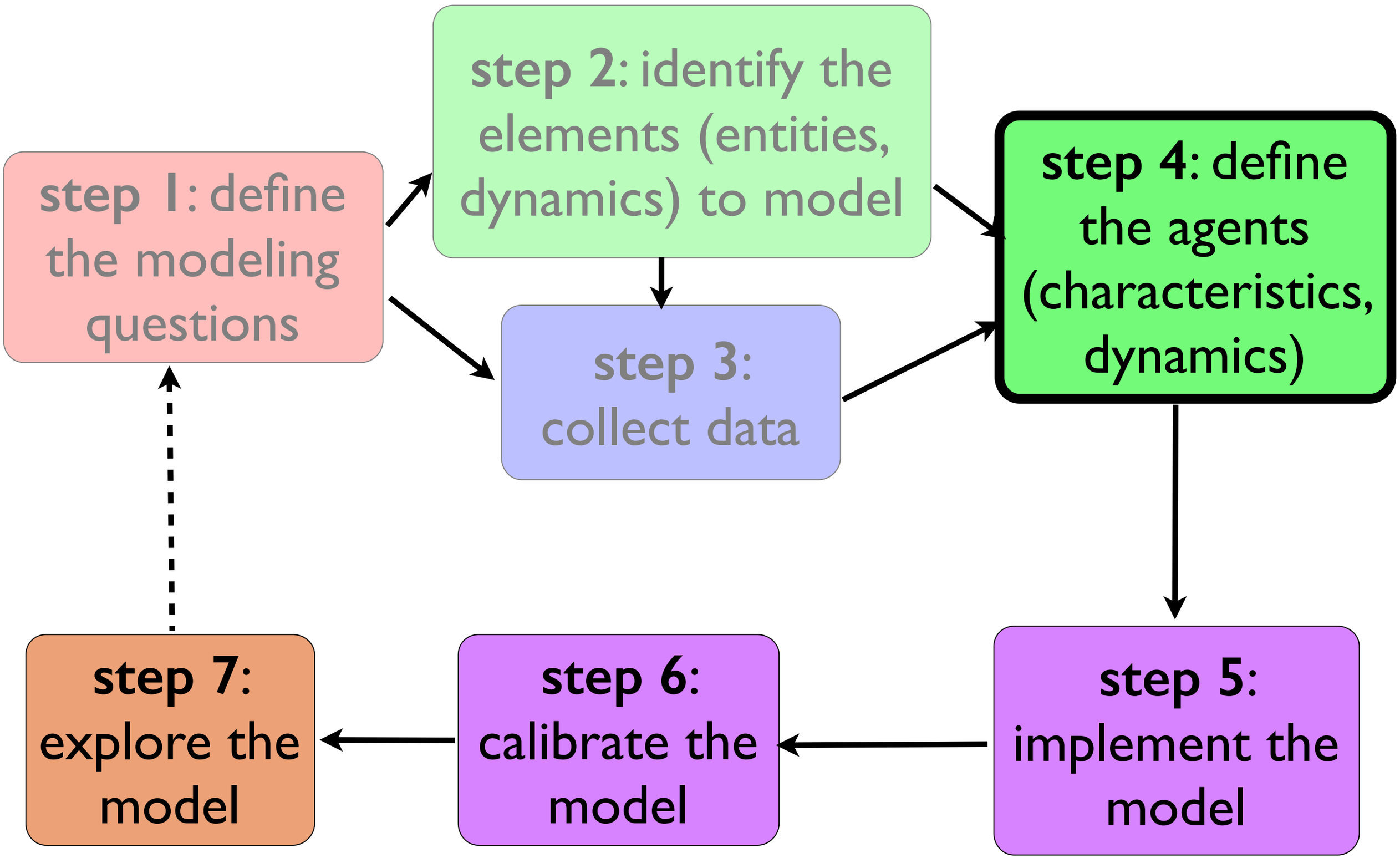
Step 3: data collection

Data available:

- Shapefile of the roads (OSM)

- Shapefile of the buildings (OSM)

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

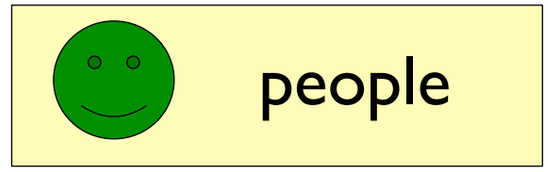

step 1: define the modeling questions
step 2: identify the elements (entities, dynamics) to model
step 3: collect data
step 4: define the agents (characteristics, dynamics)
step 5: implement the model
step 6: calibrate the model
step 7: explore the model

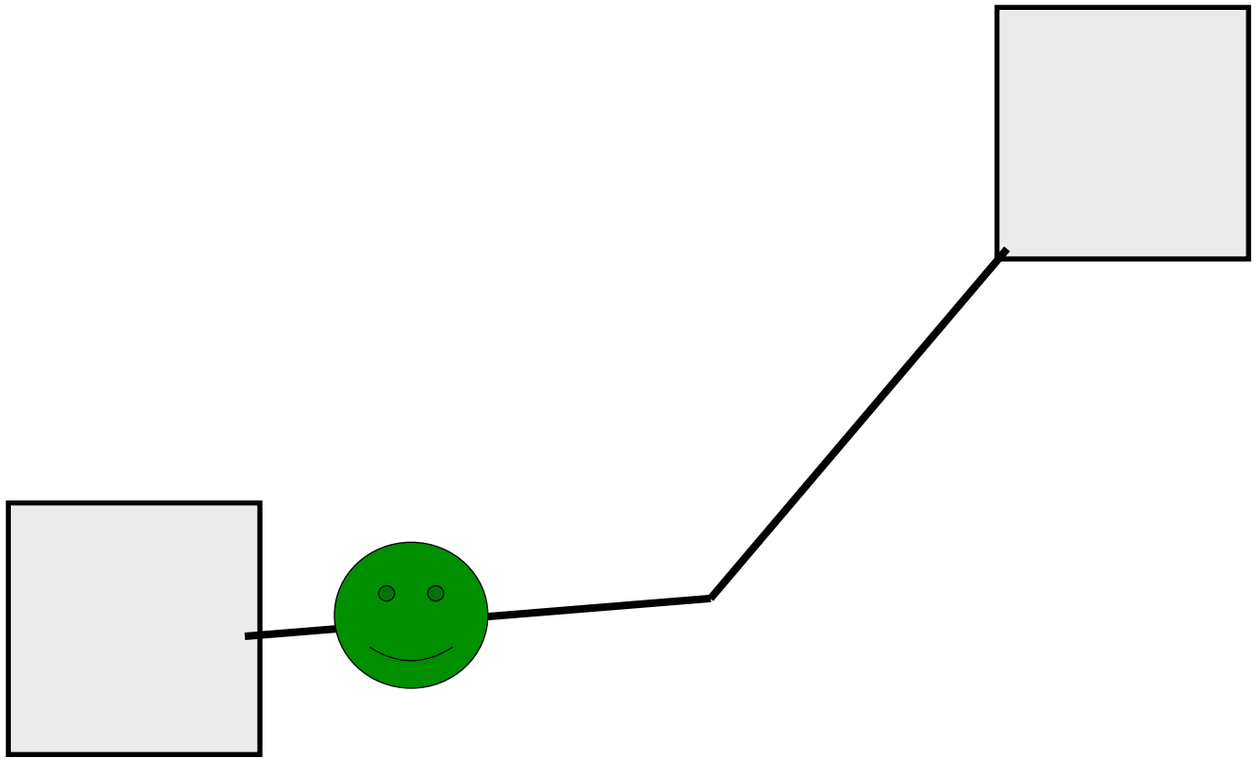
Step 4: Modeling choice

Modeling choices:

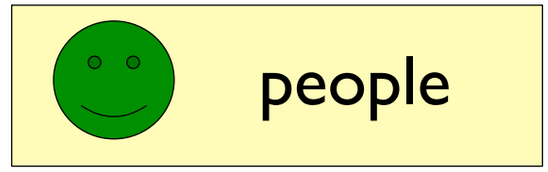
- Simulation step: 1 minute
- People are moving on the roads from building to building,
- Most of time people are moving to meet their friend then go back home
- People use the shortest path to move between buildings
- All people move at constant speed
- Each time, people arrived at a building they are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

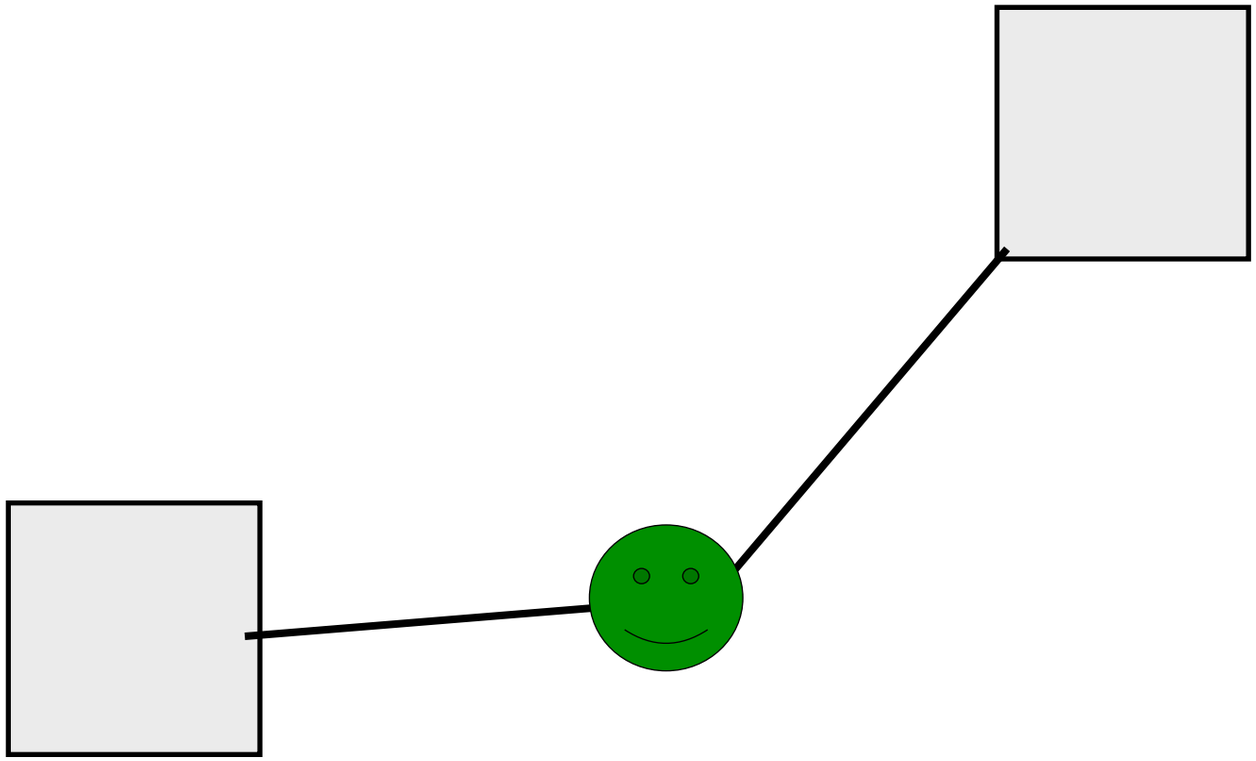
Moving dynamic



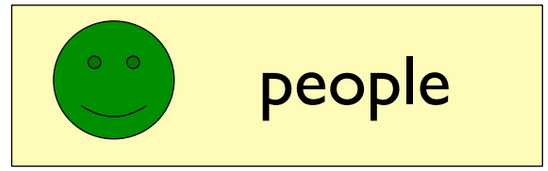


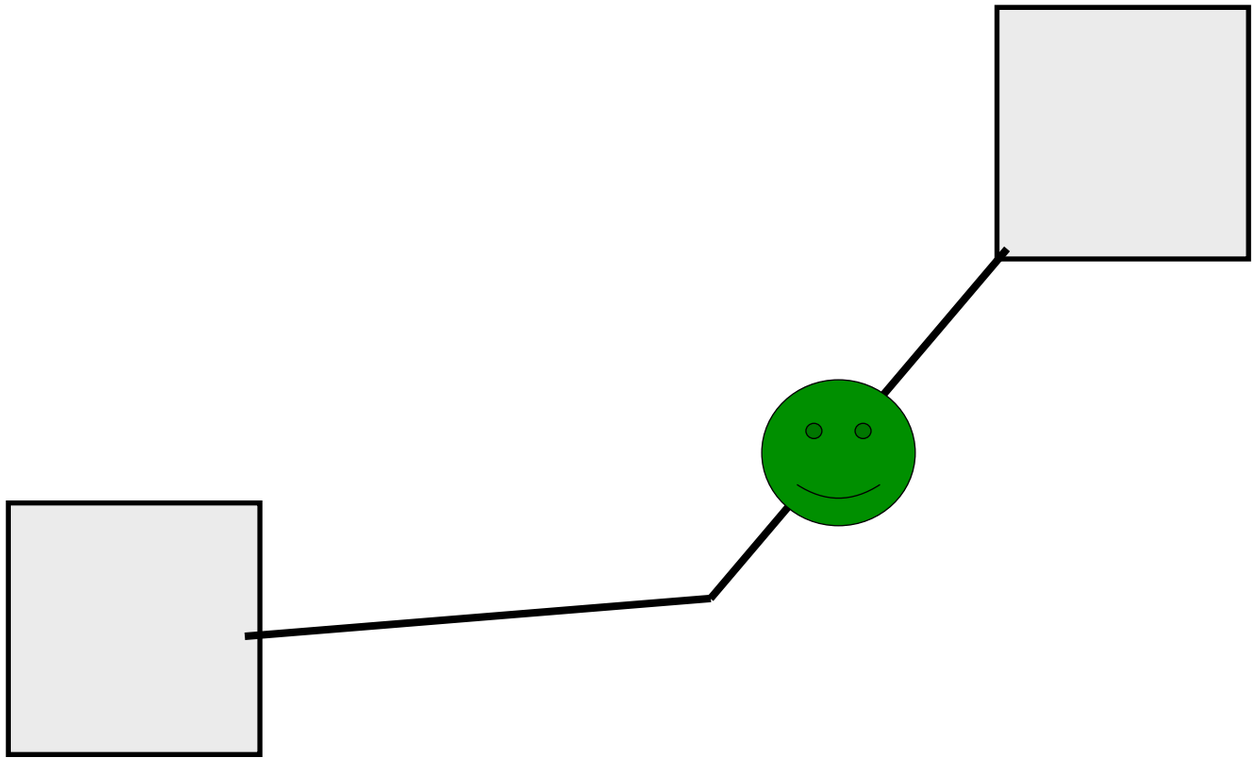
Moving dynamic



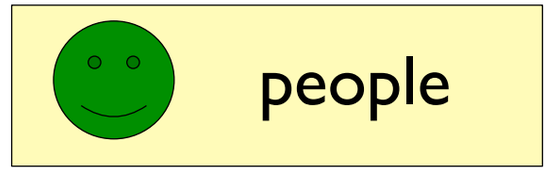


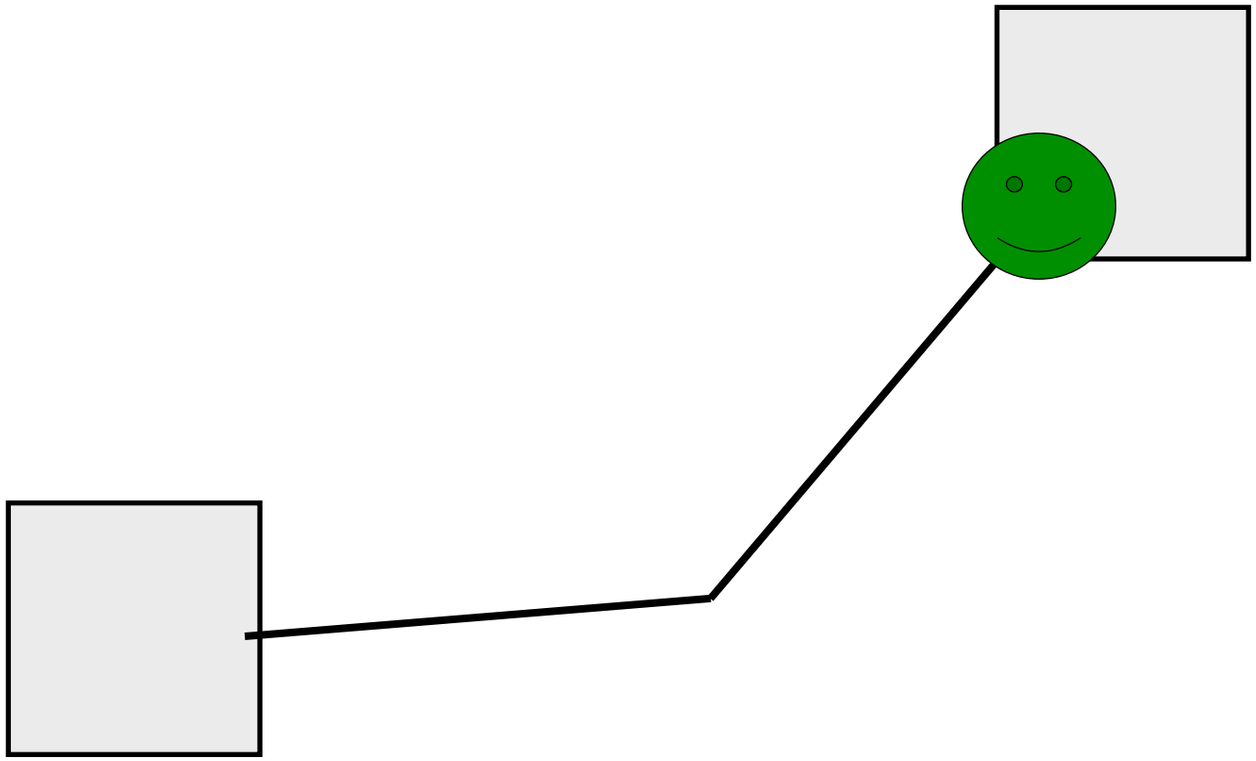
Moving dynamic



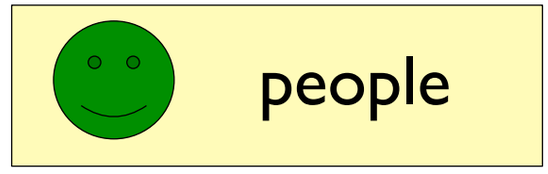


Moving dynamic

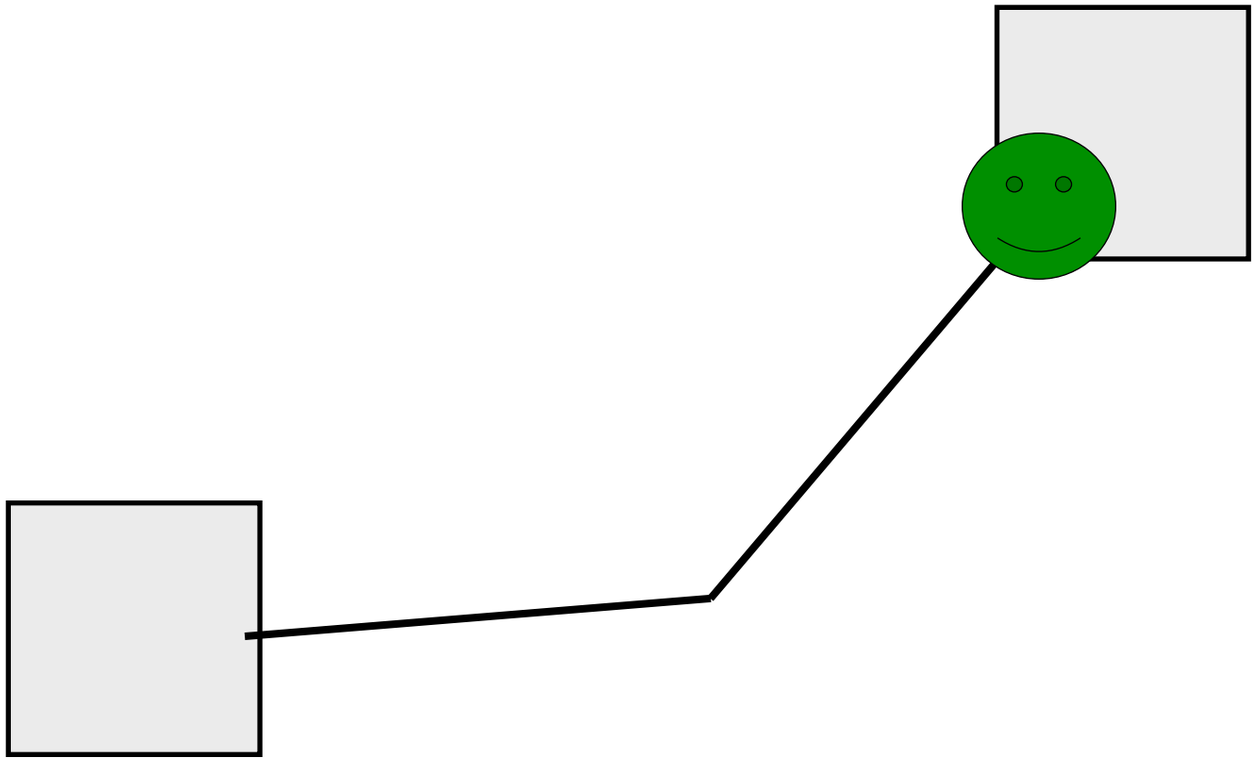




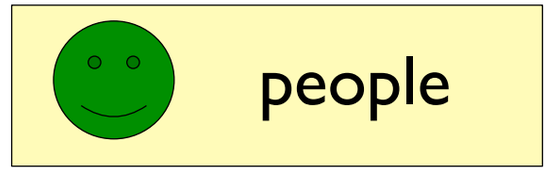
Moving dynamic



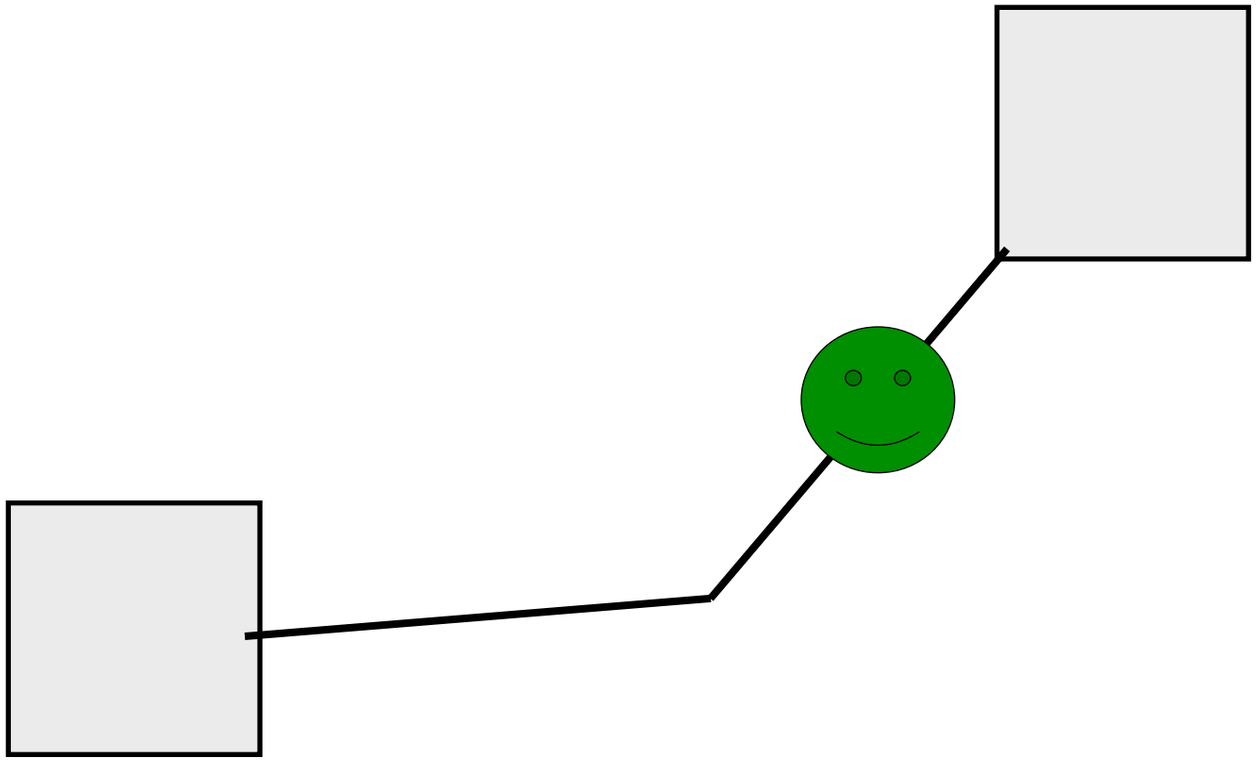
Stay for a certain time: at each simulation step, probability to leave:
if in their house: 0.01; otherwise: 0.1



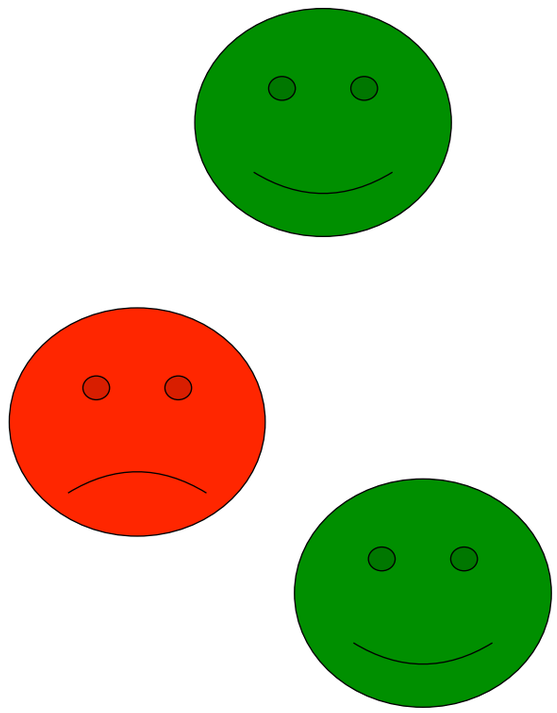
Moving dynamic



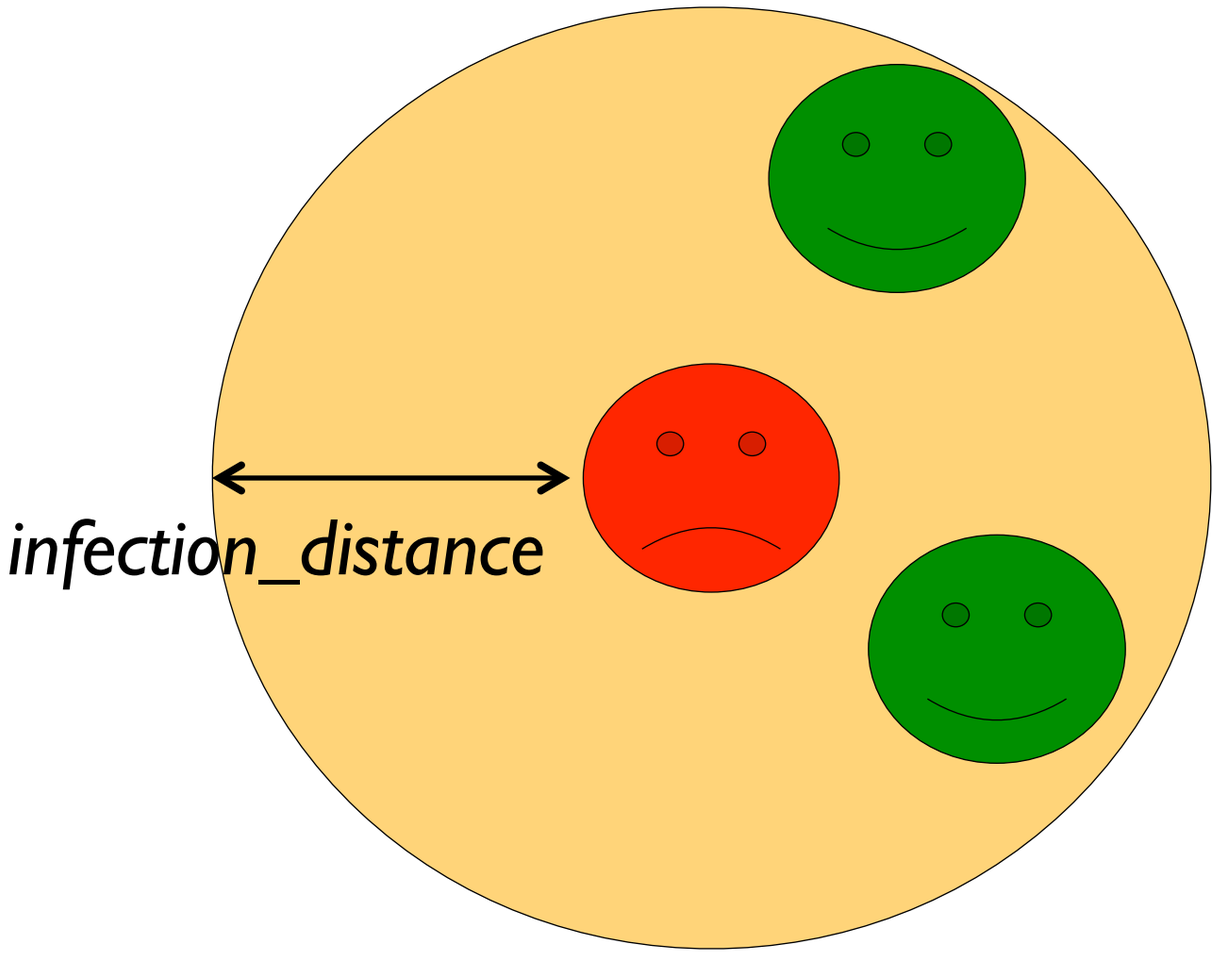
Stay for a certain time: at each simulation step, probability to leave:
if in their house: 0.01; otherwise: 0.1



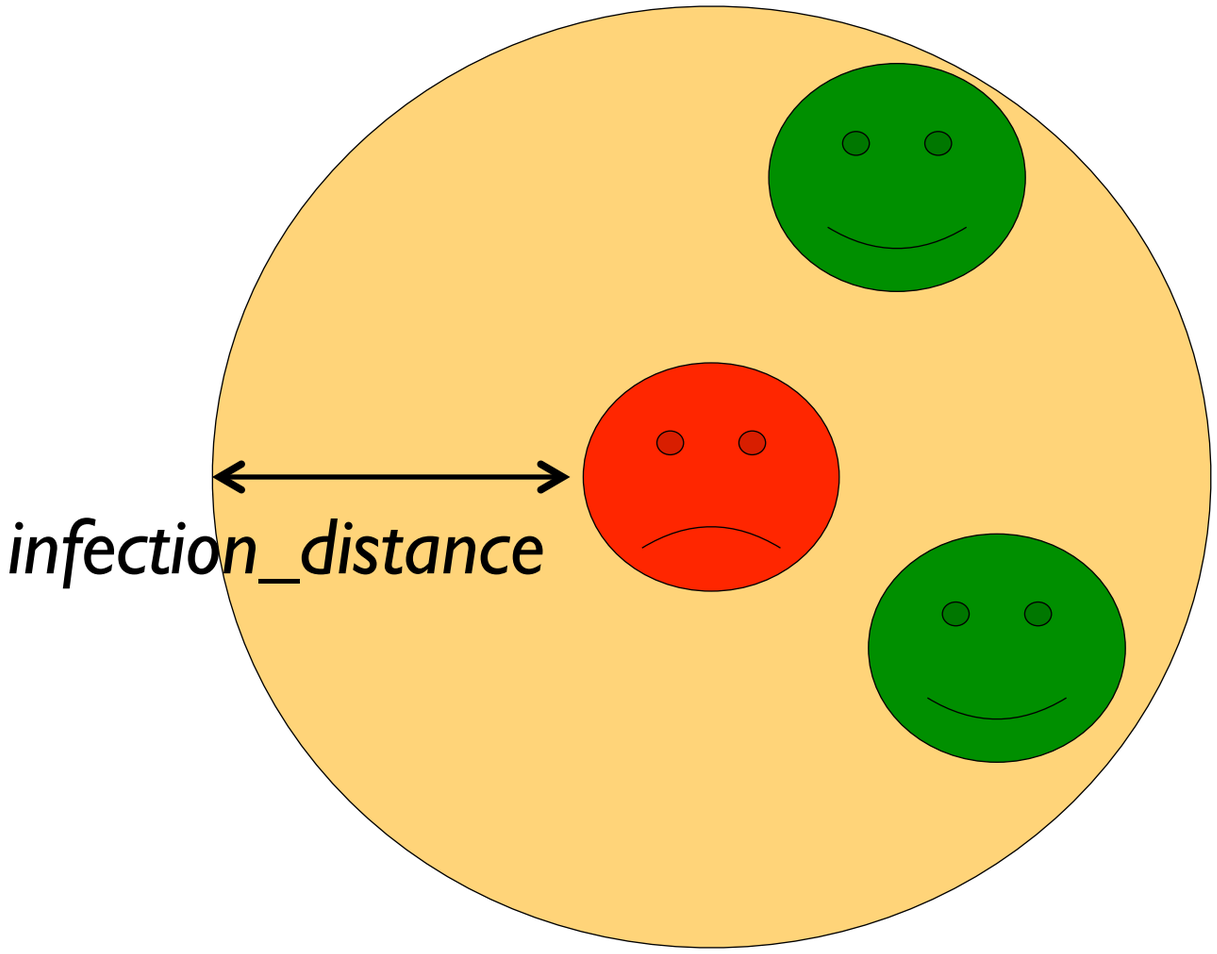
Infection dynamic



Infection dynamic



Infection dynamic

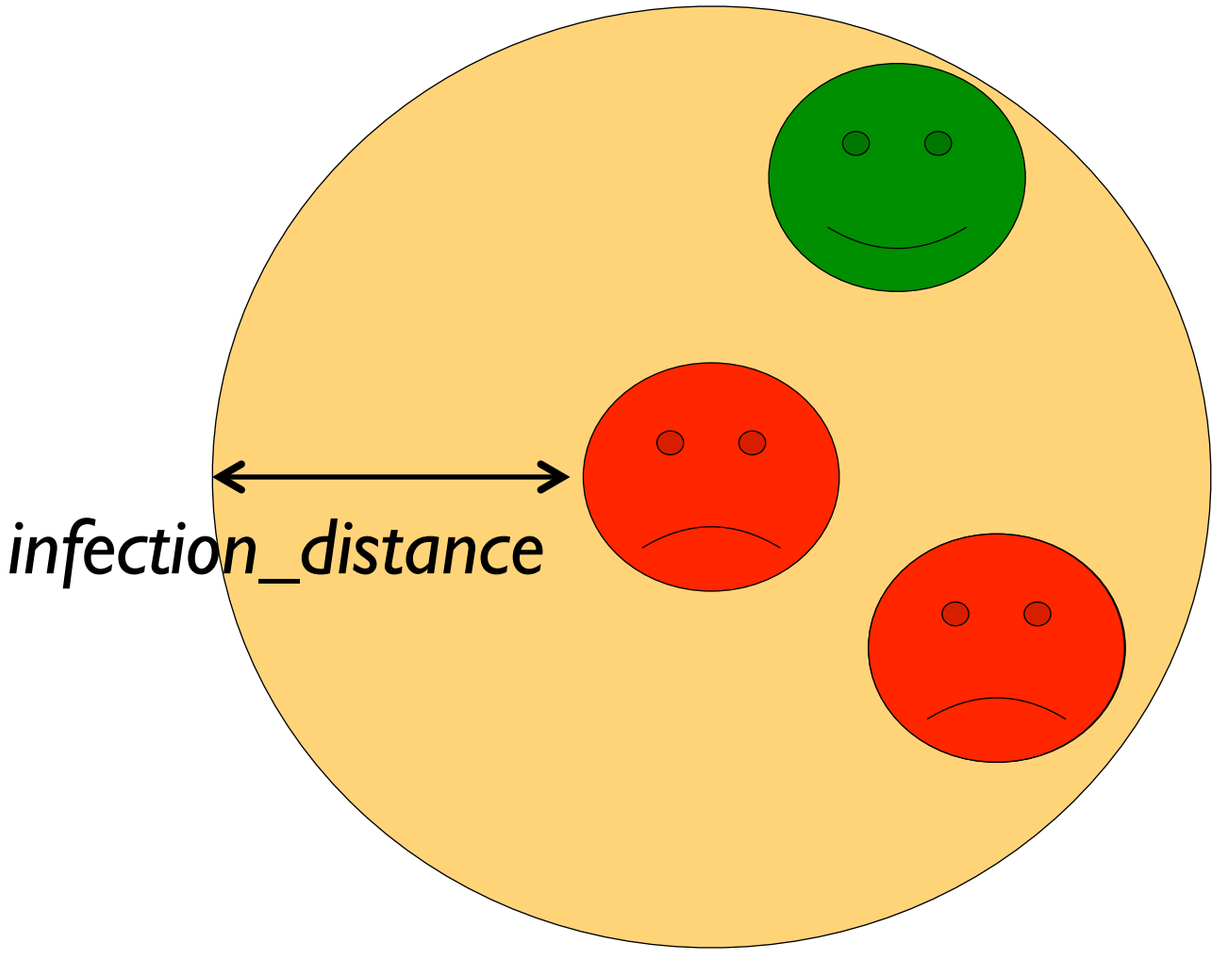


not infected people

Infected people

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

Infection dynamic

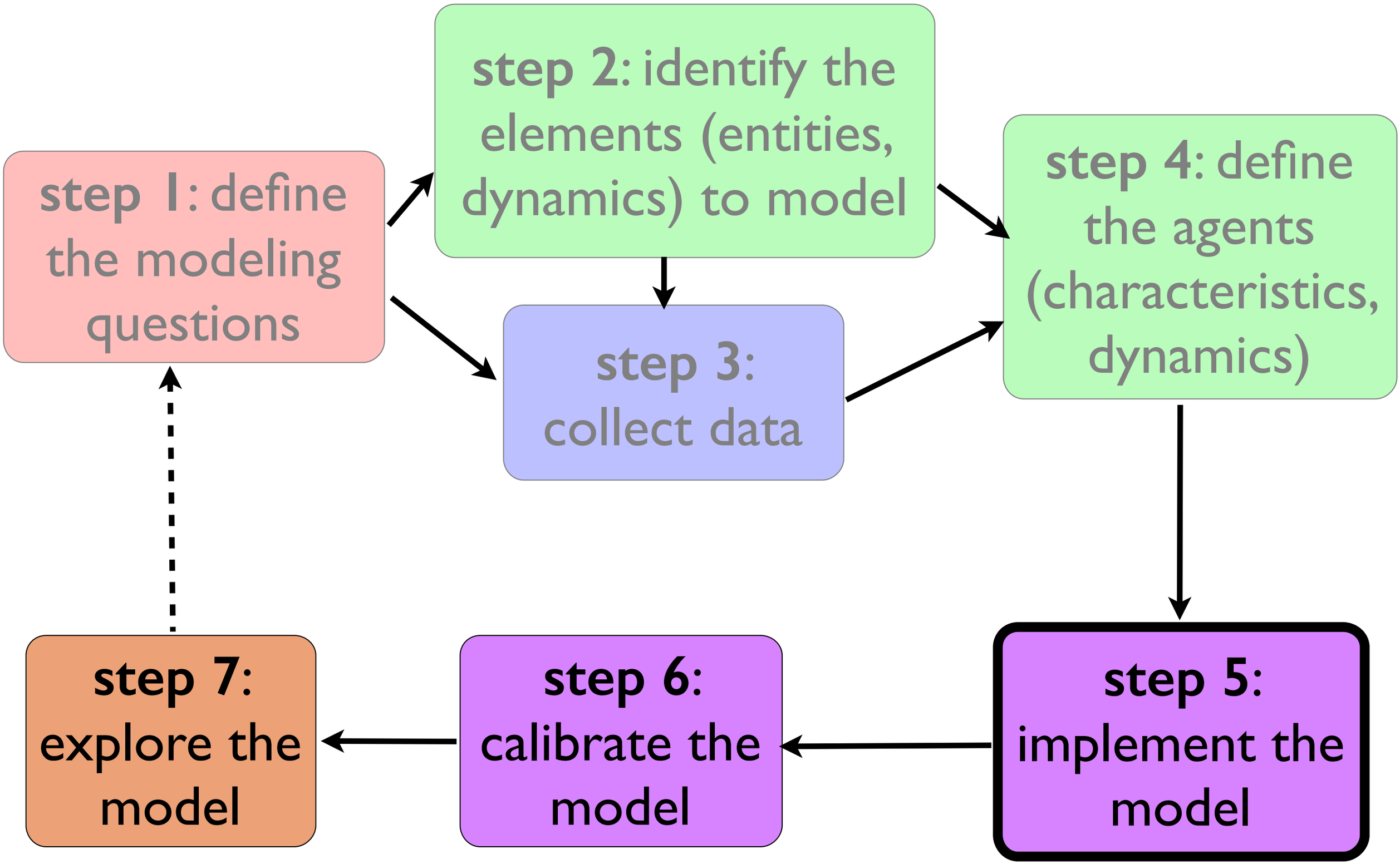


not infected people

Infected people

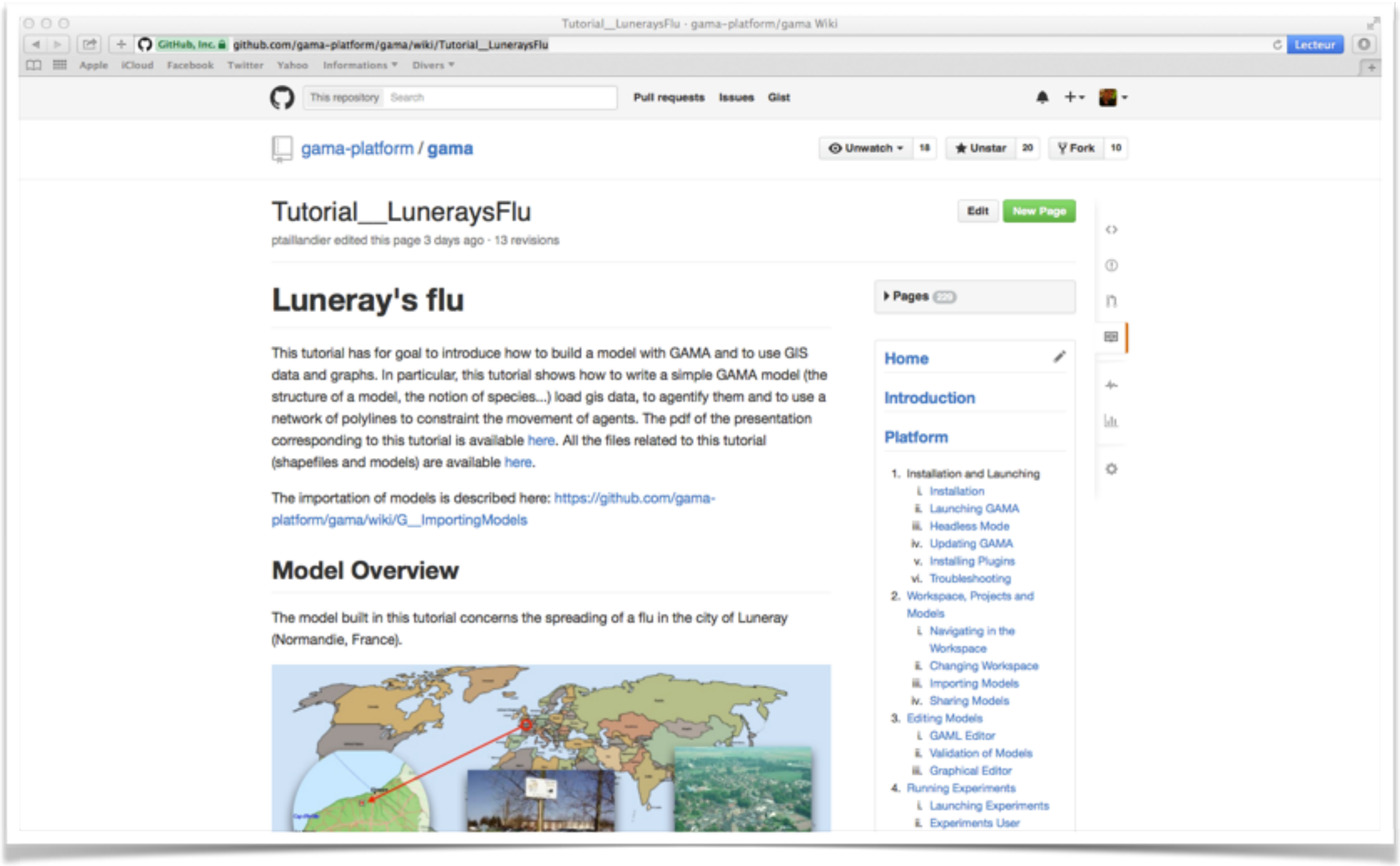
For each potential victims, probability to be infected: *proba_infection*

Modeling steps

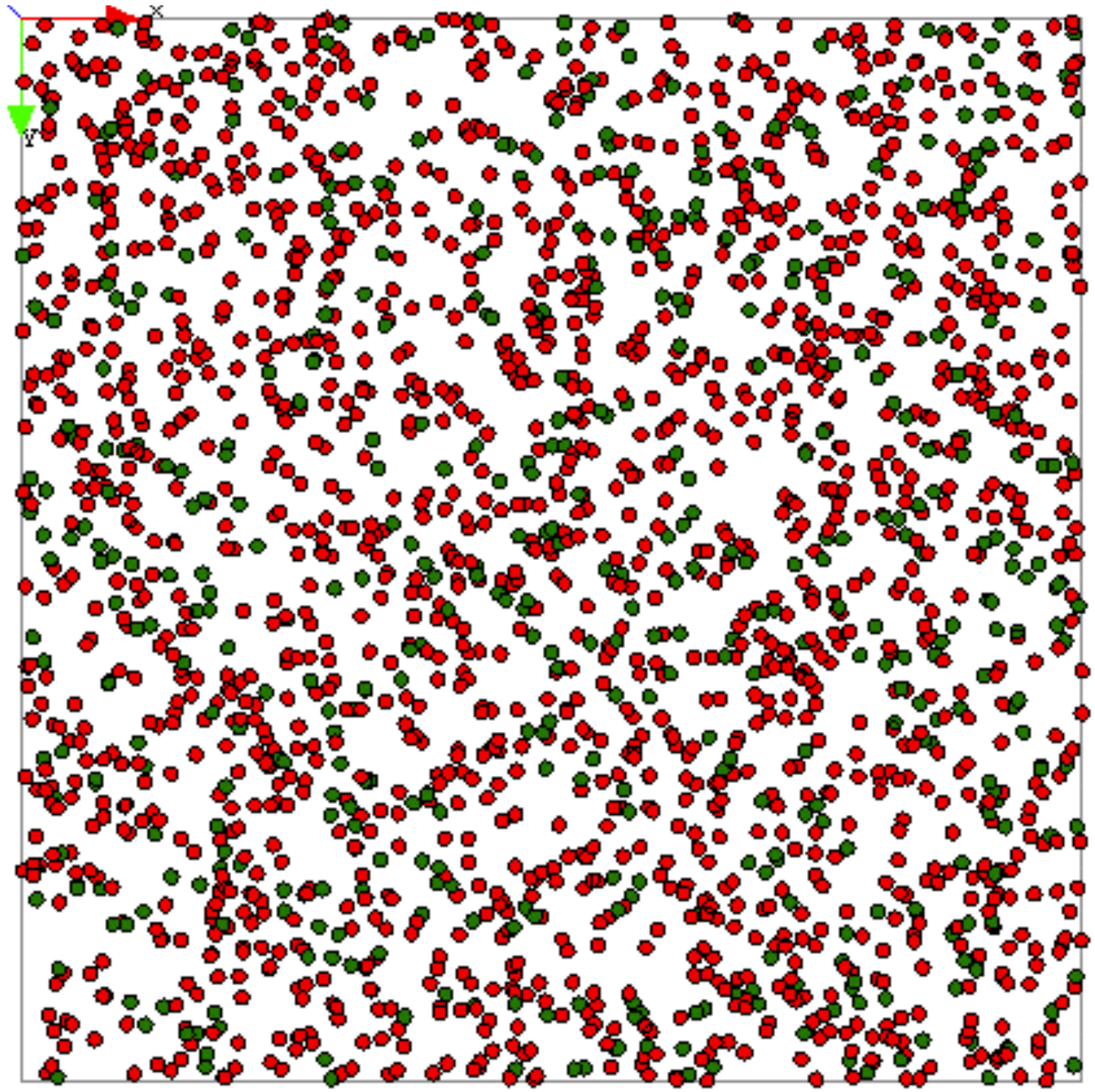


On-line version of the tutorial

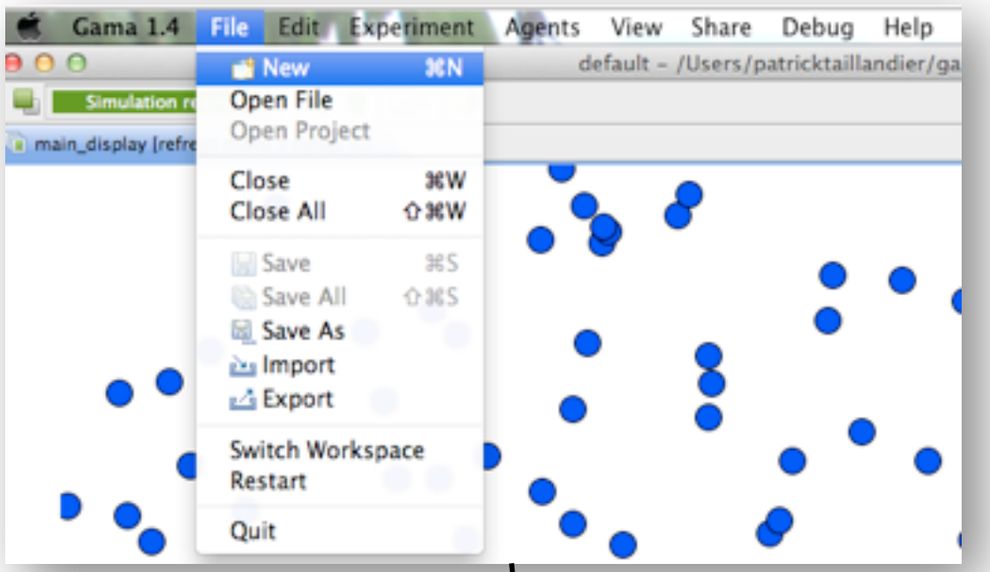
https://github.com/gama-platform/gama/wiki/Tutorial__LuneraysFlu

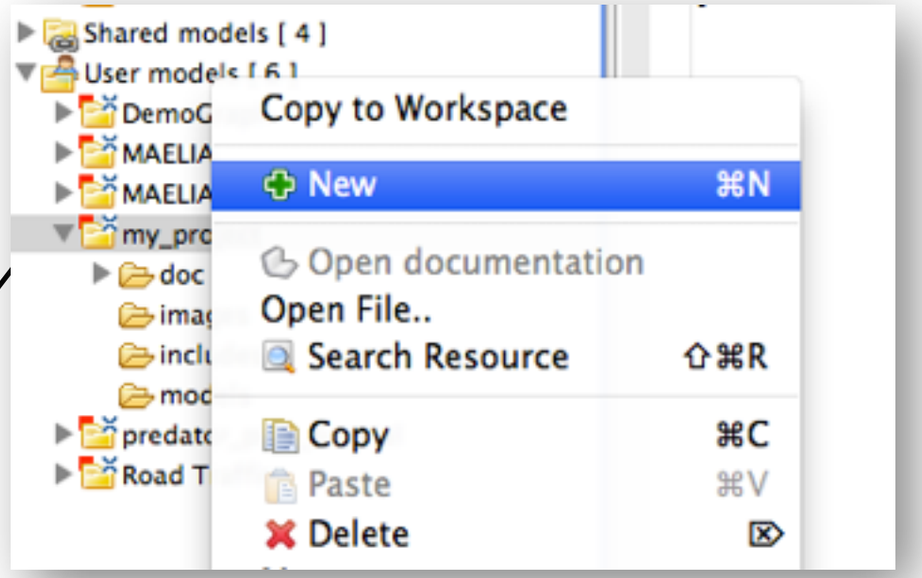


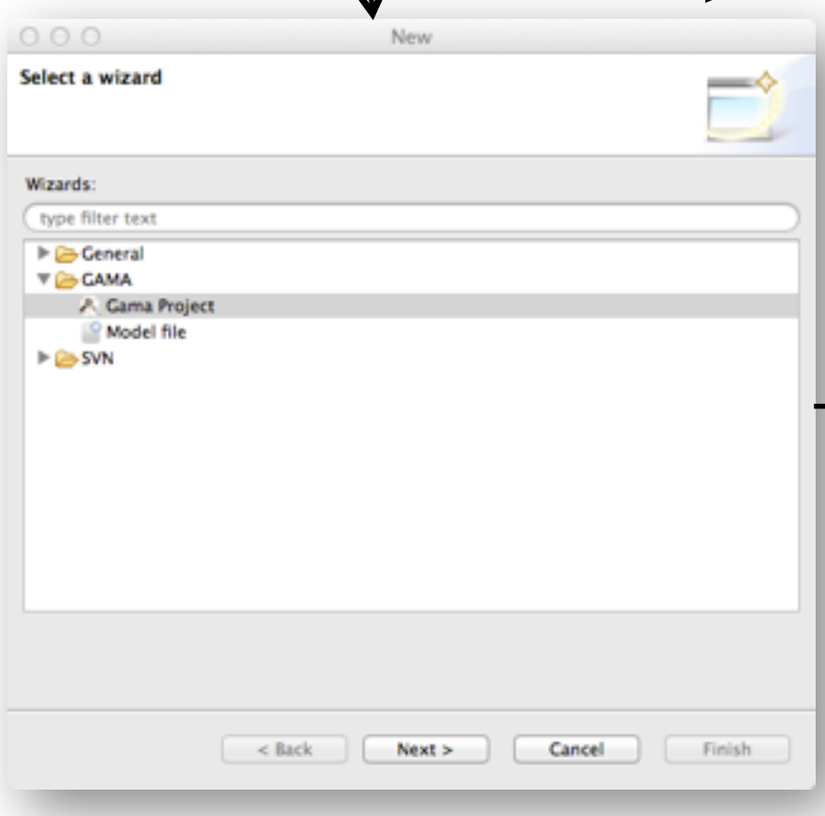
Definition of the structure of the model and of the basic behaviors of the people agents

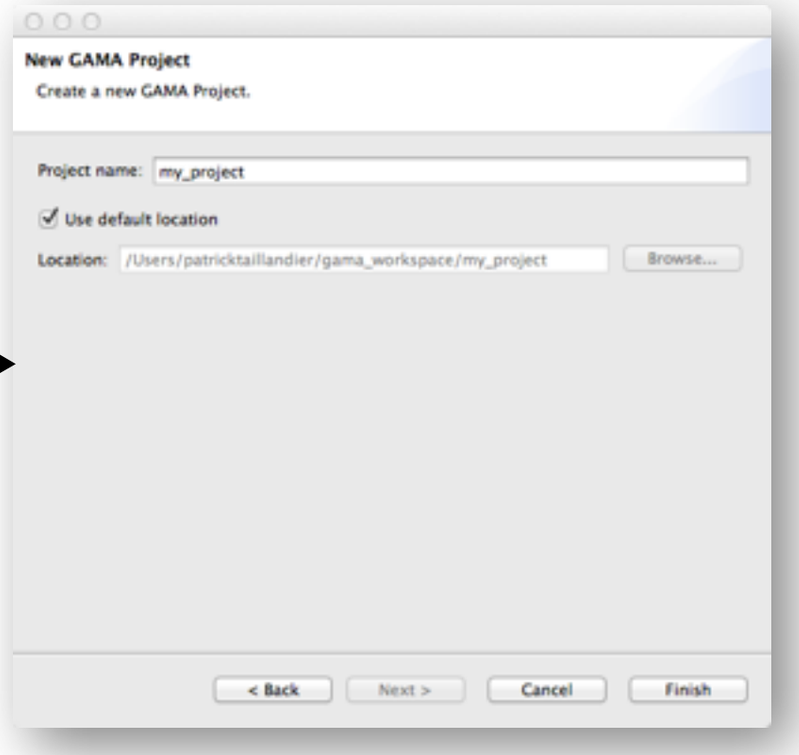


Creation of a new project

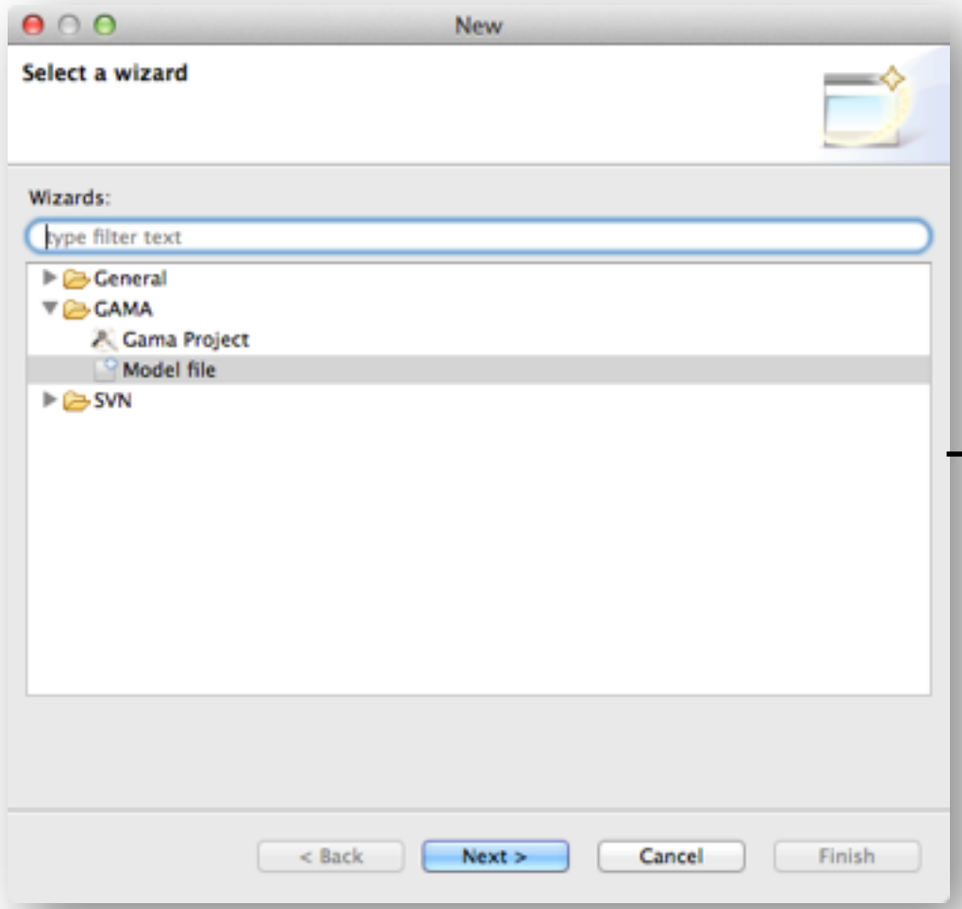


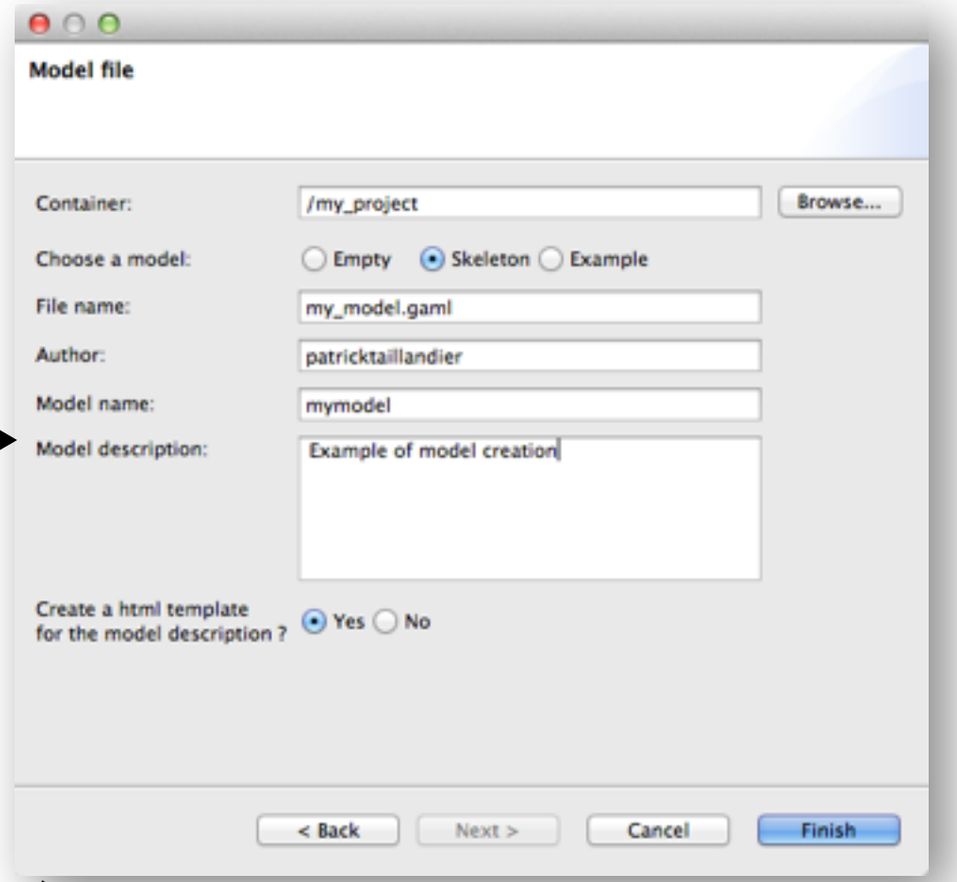






Creation of a new model






```
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 {
     }
}
```

Structure of a GAMA model

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

```
model my_model

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

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

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

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

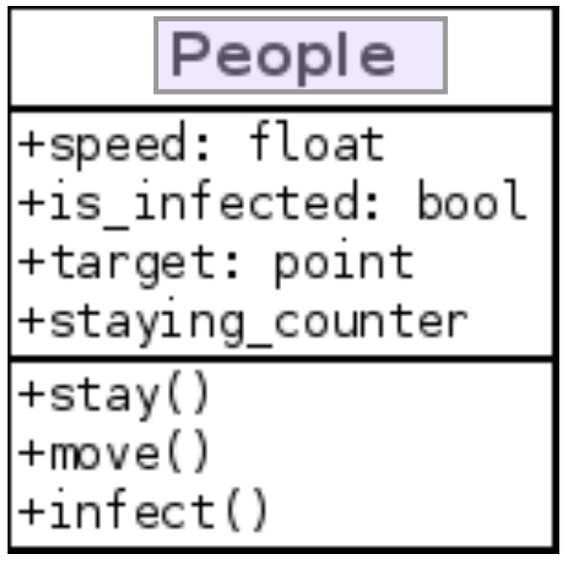
- //… : for one line. Example : //this is a commentary
- /* … */ : can be used for several lines. Example : /* this is as well a commentay */

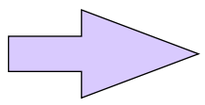
species and grid block: Species and Grid

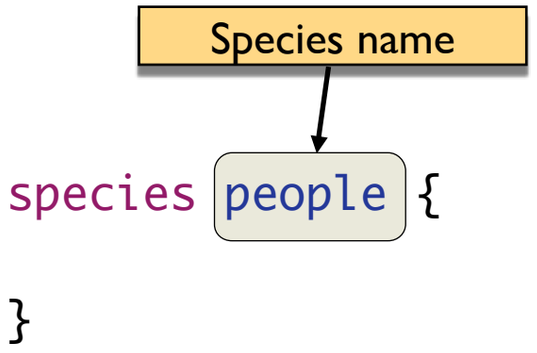
- **To define a species, 4 main elements can be defined :**
  - the internal state of the agents (attributes - variables)
  - their initialisation (init)
  - their behaviors (methods)
  - their display (aspects)

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : People species







species block: Species definition – skills

- 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*)

Color guide:

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

species block: Species definition - variables 1/3

- Variable definition : ***variable type*** + *nom*
  - ***type***: int, float, string, bool (boolean, can be either *true* or *false*), point, list, pair, map, file, matrix, agent species, rgb (Red, Green, Blue - color), graph, path…
  - Optional attributes:
    - **<-** : initial value,
    - ***update*** : value computed at each simulation step
    - ***function*** *or* **->** + ***{..}*** : value computed each time the variable is called
    - ***min*** : min value
    - ***max :*** max value

- GAMA allows to define **local variables**, i.e. variables that just exist inside a block. These variables are 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 <- (2 + rnd(3)) #km/#h;
    bool is_infected <- false;
}
```

The symbol # allows to precise the unity of a value

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

Color guide:

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

species block: Species definition – reflex 3/3

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

Model : People Reflex

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance 10 #m {
            if flip(0.05) {
                is_infected <- true;
            }
        }
    }
}
```

species block: Species definition – reflex 3/3

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

Model : People Reflex

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 10 #m {
            if flip(0.05) {
                is_infected <- true;
            }
        }
    }
}
```

species block: Species definition – reflex 3/3

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

Model : People Reflex

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 10 #m {
            if flip(0.05) {
                is_infected <- true;
            }
        }
    }
}
```

species block: Species definition – reflex 3/3

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

Model : People Reflex

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 10 #m {
            if flip(0.05) {
                is_infected <- true;
            }
        }
    }
}
```

species block: Species definition – reflex 3/3

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

Model : People Reflex

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 10 *m*

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance 10 #m {
            if flip(0.05) {
                is_infected <- true;
            }
        }
    }
}
```

species block: Species definition – reflex 3/3

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

Model : People Reflex

Reflex executed at each simulation step

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

Reflex activated only when the agent is infected

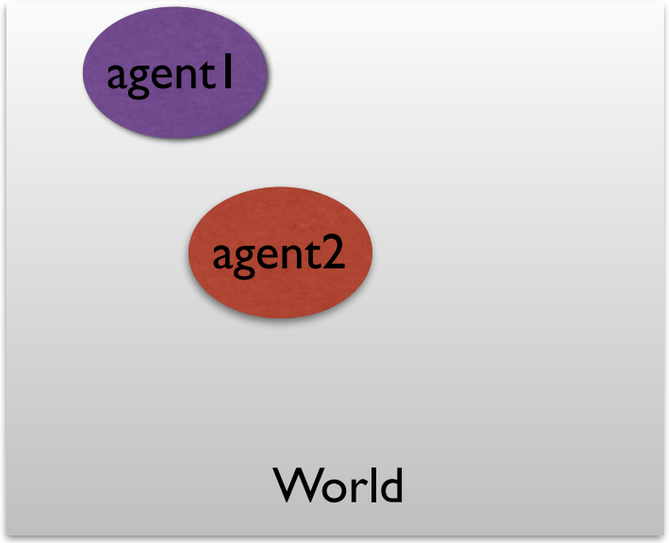
ask all the people agents located at a (euclidean) distance inferior or equal to 10 *m*

to draw a probability *0.05* to be infected

```
species people skills:[moving]{
    //variable definition
    reflex move{
        do wander;
    }
    reflex infect when: is_infected{
        ask people at_distance 10 #m {
            if flip(0.05) {
                is_infected <- true;
            }
        }
    }
}
```

species block: Species definition – reflex 3/3

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



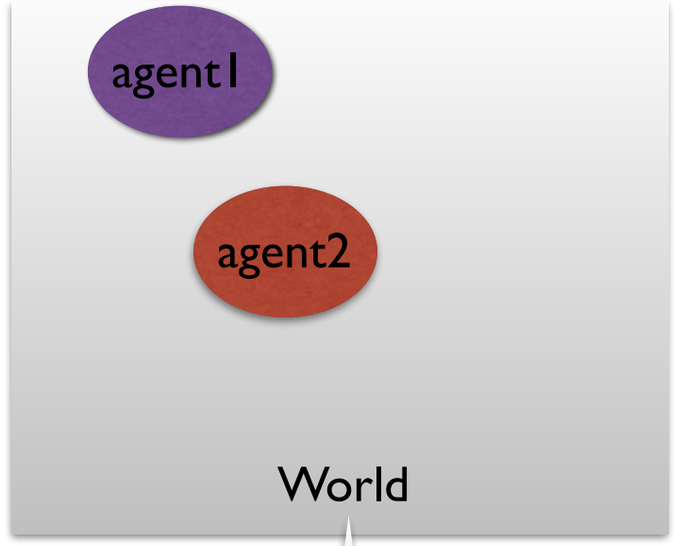
- Of course the Scheduler can be easily tuned through the GAML language:
  - ➡ modification of the order of activation of the agents (than can be dynamic)
  - ➡ Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action….)

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

species block: Species definition – reflex 3/3

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

- Of course the Scheduler can be easily tuned through the GAML language:
  - ➡ modification of the order of activation of the agents (than can be dynamic)
  - ➡ Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action….)

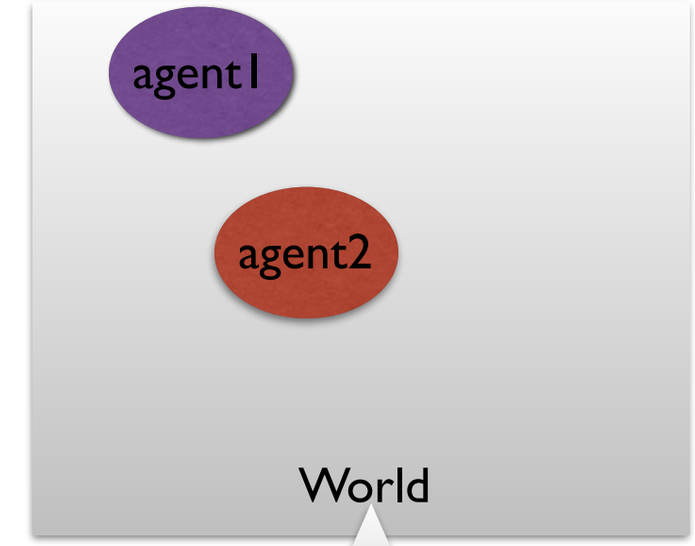


I) If I have attributes with a *update* facet, I compute their new value in their definition order

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

species block: Species definition – reflex 3/3

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



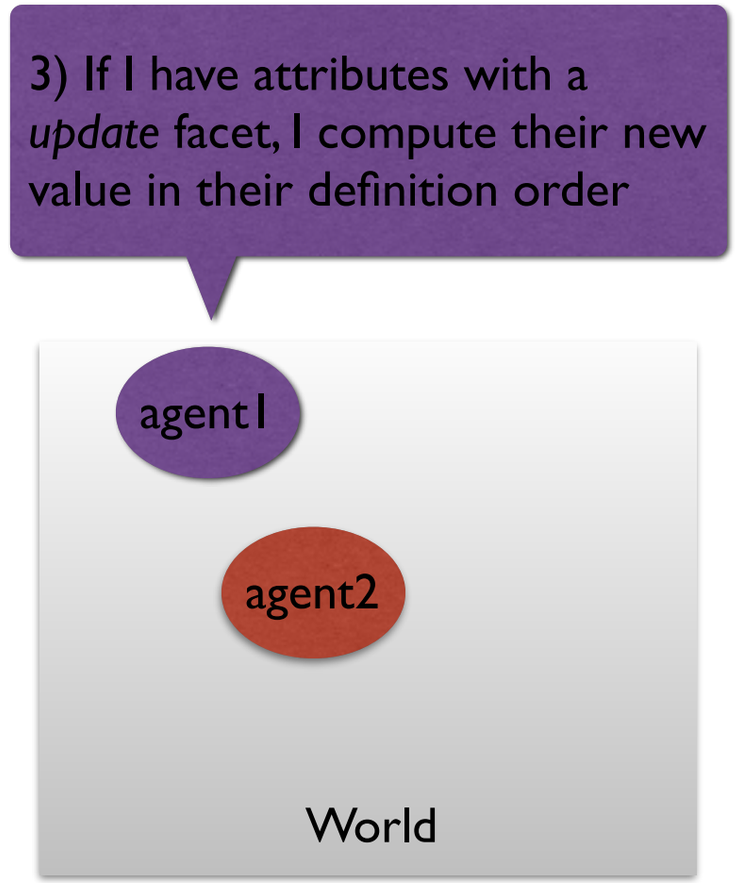
2) If I have reflexes, I activate them in their definition order

- Of course the Scheduler can be easily tuned through the GAML language:
  - ➡ modification of the order of activation of the agents (than can be dynamic)
  - ➡ Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action….)

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

species block: Species definition – reflex 3/3

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

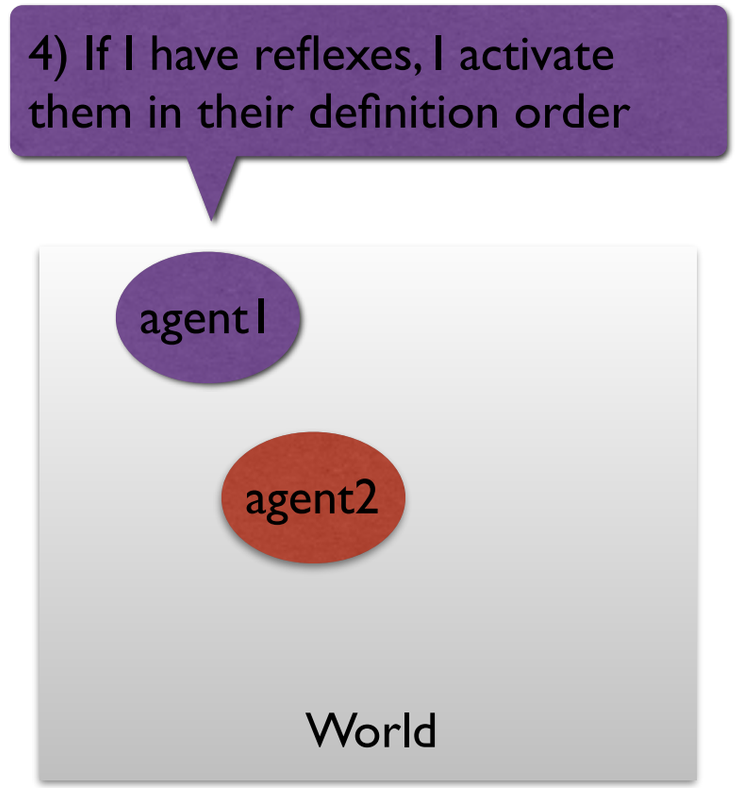


- Of course the Scheduler can be easily tuned through the GAML language:
  - ➡ modification of the order of activation of the agents (than can be dynamic)
  - ➡ Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action….)

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

species block: Species definition – reflex 3/3

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

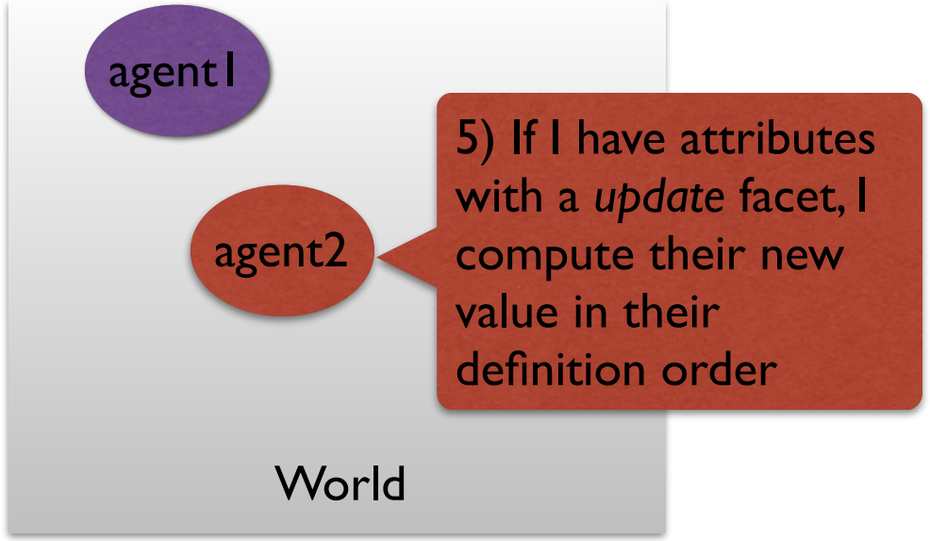


- Of course the Scheduler can be easily tuned through the GAML language:
  - ➡ modification of the order of activation of the agents (than can be dynamic)
  - ➡ Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action….)

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

species block: Species definition – reflex 3/3

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

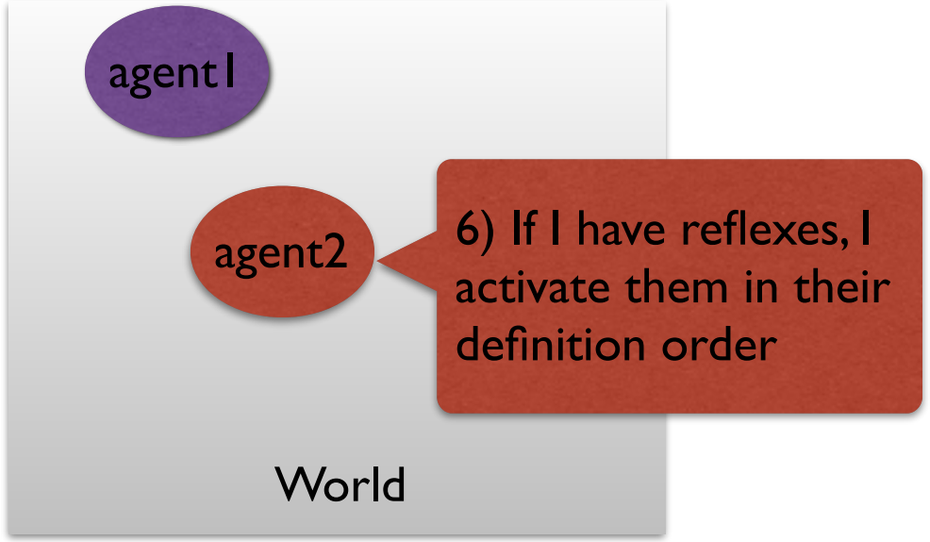


- Of course the Scheduler can be easily tuned through the GAML language:
  - ➡ modification of the order of activation of the agents (than can be dynamic)
  - ➡ Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action….)

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

species block: Species definition – reflex 3/3

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



- Of course the Scheduler can be easily tuned through the GAML language:
  - ➡ modification of the order of activation of the agents (than can be dynamic)
  - ➡ Fine activation of the agents using actions (e.g.: agent1 executes a first action, then agent2 executes an action, then agent1 executes again another action….)

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

species block: Species definition – aspect

- An *aspect* represent A possible display for a species of agents : ***aspect*** *aspect_name* ***{...}***
- In an *aspect* block, it is possible to display (as layers) :
  - a geometry/shape: for example, the agent shape
  - an image: for example, an icon
  - a text

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : aspect *circle* of the people agents

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

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

The symbol # allows

also to define a color

species block: Species definition – aspect

- An *aspect* represent A possible display for a species of agents : ***aspect*** *aspect_name* ***{...}***
- In an *aspect* block, it is possible to display (as layers) :
  - a geometry/shape: for example, the agent shape
  - an image: for example, an icon
  - a text

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : aspect *circle* of the people agents

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

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

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

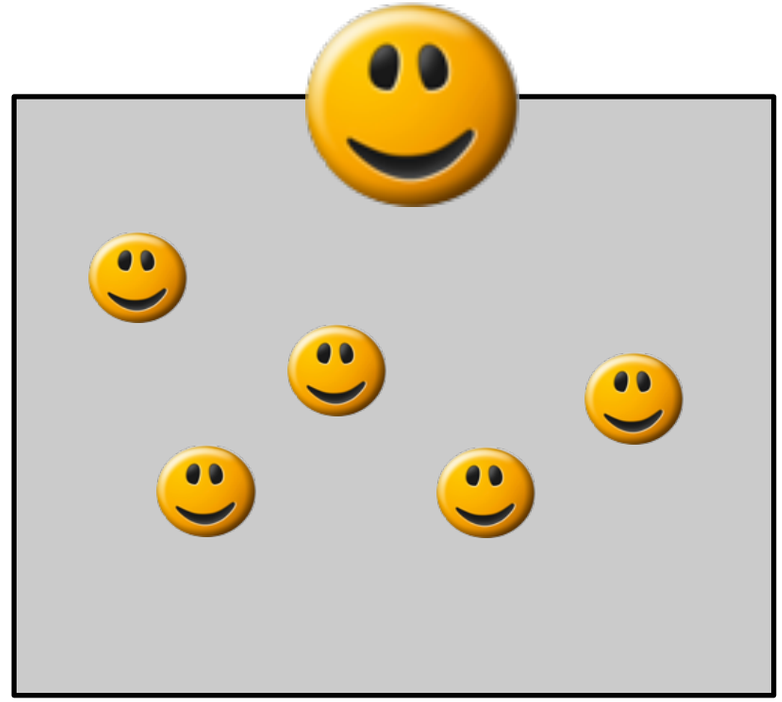
The symbol # allows

also to define a color

global block

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

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```



Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

Built-in variable: define the duration of simulation step

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

Built-in variable: define the duration of simulation step

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

Block global : agent creation

- Creation of agents : use of the statement: **create** species_nam
  - 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

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;
         }
    }
}
```

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

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;
        }
    }
}
```

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 an execution context of simulations
- Several experiment blocks can be defined
- Define by : **experiment** xp_name **type:** gui/batch **{…}**
  - gui : one simulation with graphical interface.
  - batch : experiment plan: set of simulations without graphical interface

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : Main experiment

```
experiment main_experiment type: gui {
}
```

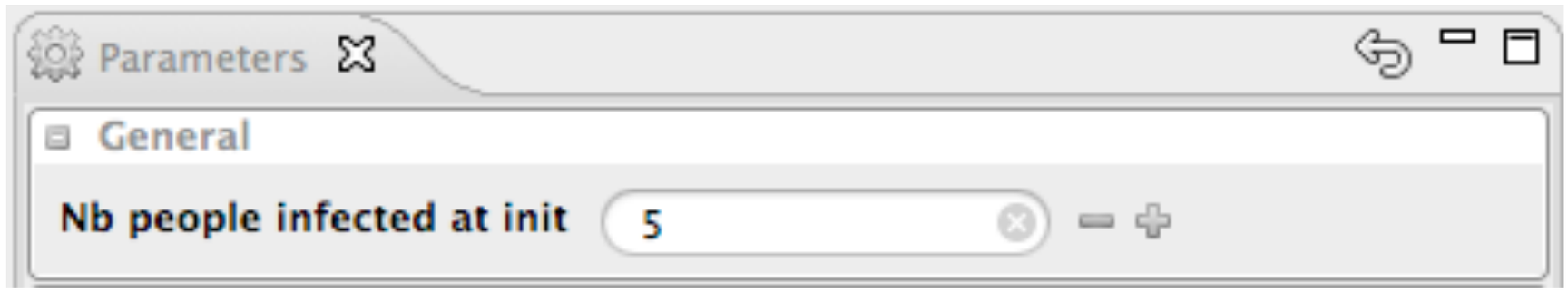
experiment block: parameter definition

- Parameter :
  parameter legend **var:** var_name **category:** my_cat;
  - Allow to give to the user the possibility to define the value of a global variable
  - legend: string to display
  - var_name: reference to a global variable
  - category: string (use to better organize the parameters) – optional

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : Main experiment

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

    output {
    }

}
```



experiment block: parameter definition

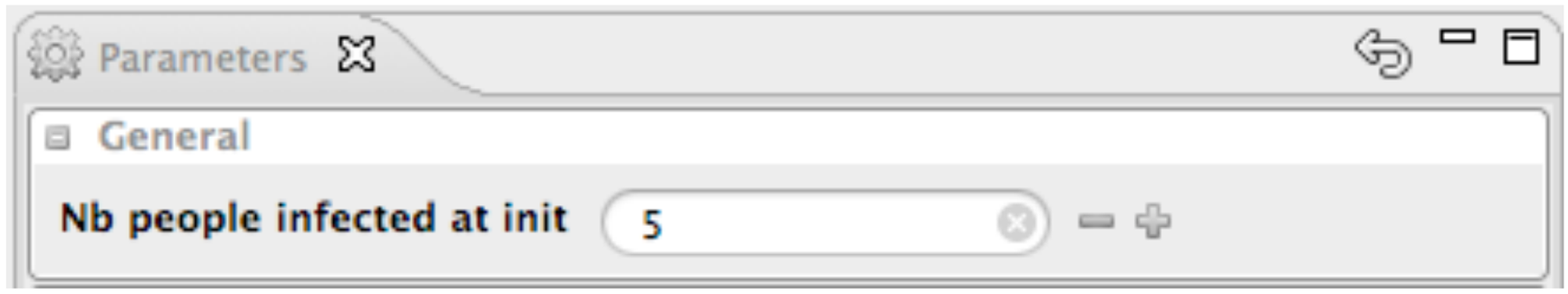
- Parameter :
  parameter legend **var:** var_name **category:** my_cat;
  - Allow to give to the user the possibility to define the value of a global variable
  - legend: string to display
  - var_name: reference to a global variable
  - category: string (use to better organize the parameters) – optional

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : Main experiment

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

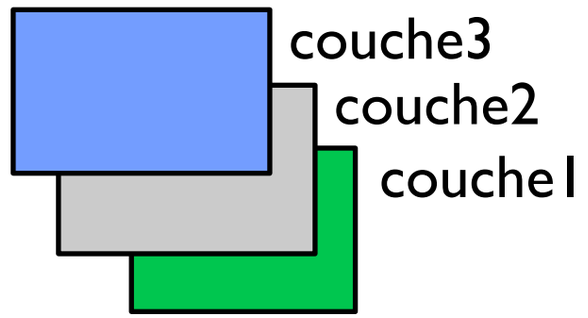
```
experiment main_experiment type:gui{
    parameter "Nb people infected at init" var: nb_infected_init min: 1 max: 2147;

    output {
    }

}
```



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

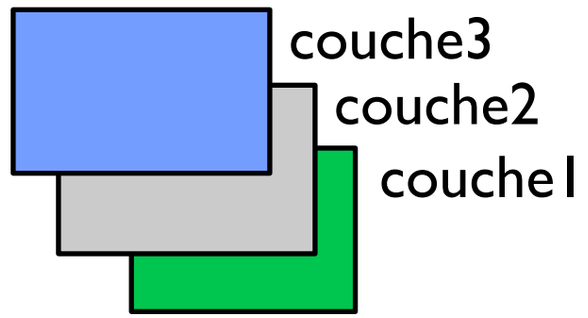
Model : experiment

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

  output {
        display map type: opengl{
            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

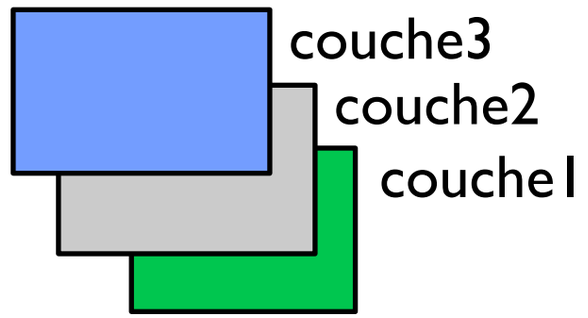
```
experiment main_experiment type: gui {
    ... //parameter definition

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

Allows to use optimized opengl displays

experiment block: output definition

- The *output* block has to be defined in an *experiment* block
- It allows to define displays:
  - A refreshing rate can be defined: facet **refresh_every:** *nb (int)*
  - Each *display* can contain different displays:
    - list of agents : **agents** *layer_name* **value:** *agents* **aspect:** *my_aspect;*
    - Agent species (all the agents of the species) : **species** *my_species* **aspect:** *my_aspect*
    - Grids: optimized display of grids: **grid** *grid_name* **lines**: *my_color;*
    - Images: **image** *layer_name* **file:** *image_file*;
    - Texts: **texte** *layer_name* **value: my**_*text*;
    - Charts: see later

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```



Model : experiment

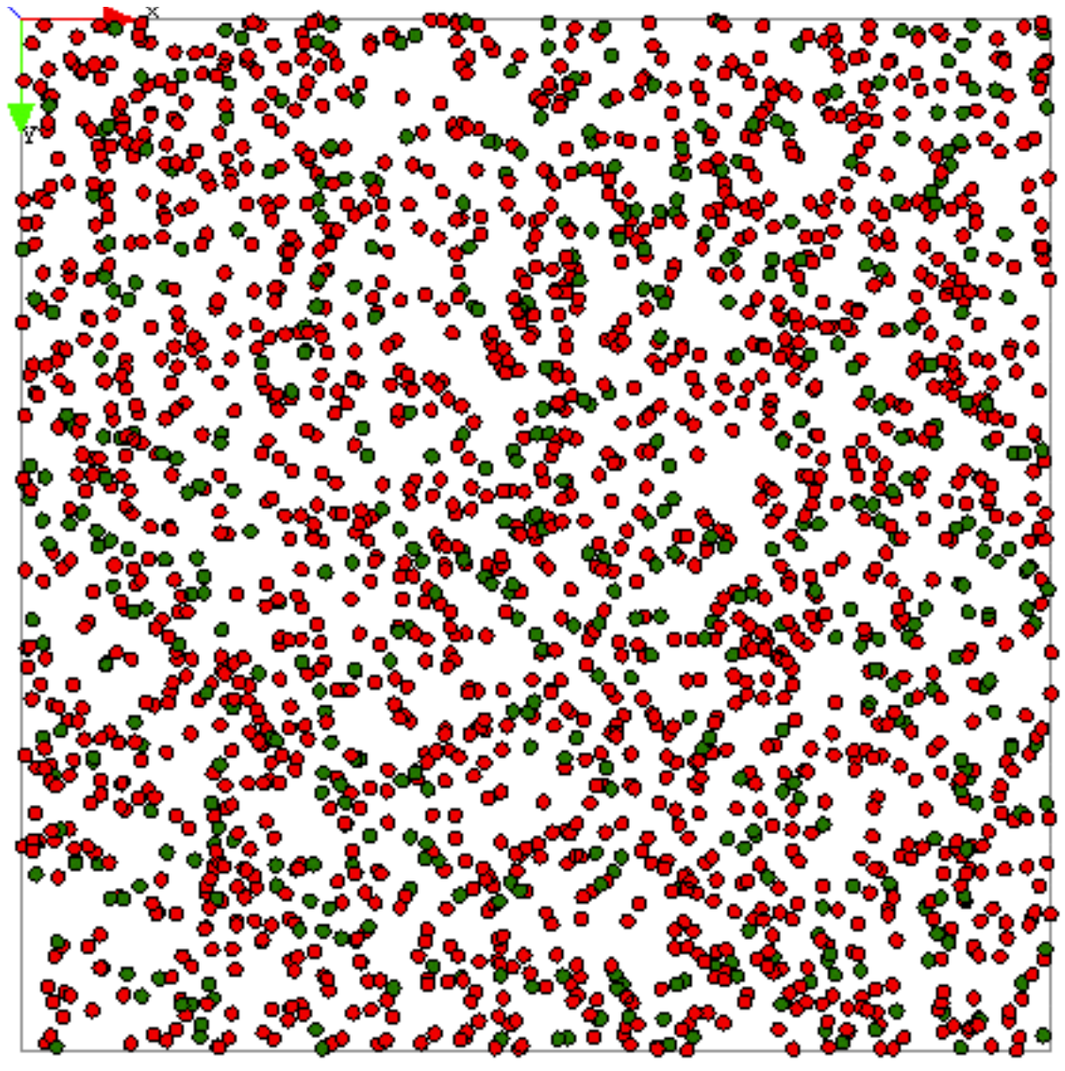
```
experiment main_experiment type: gui {
    ... //parameter definition

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

Allows to use optimized opengl displays

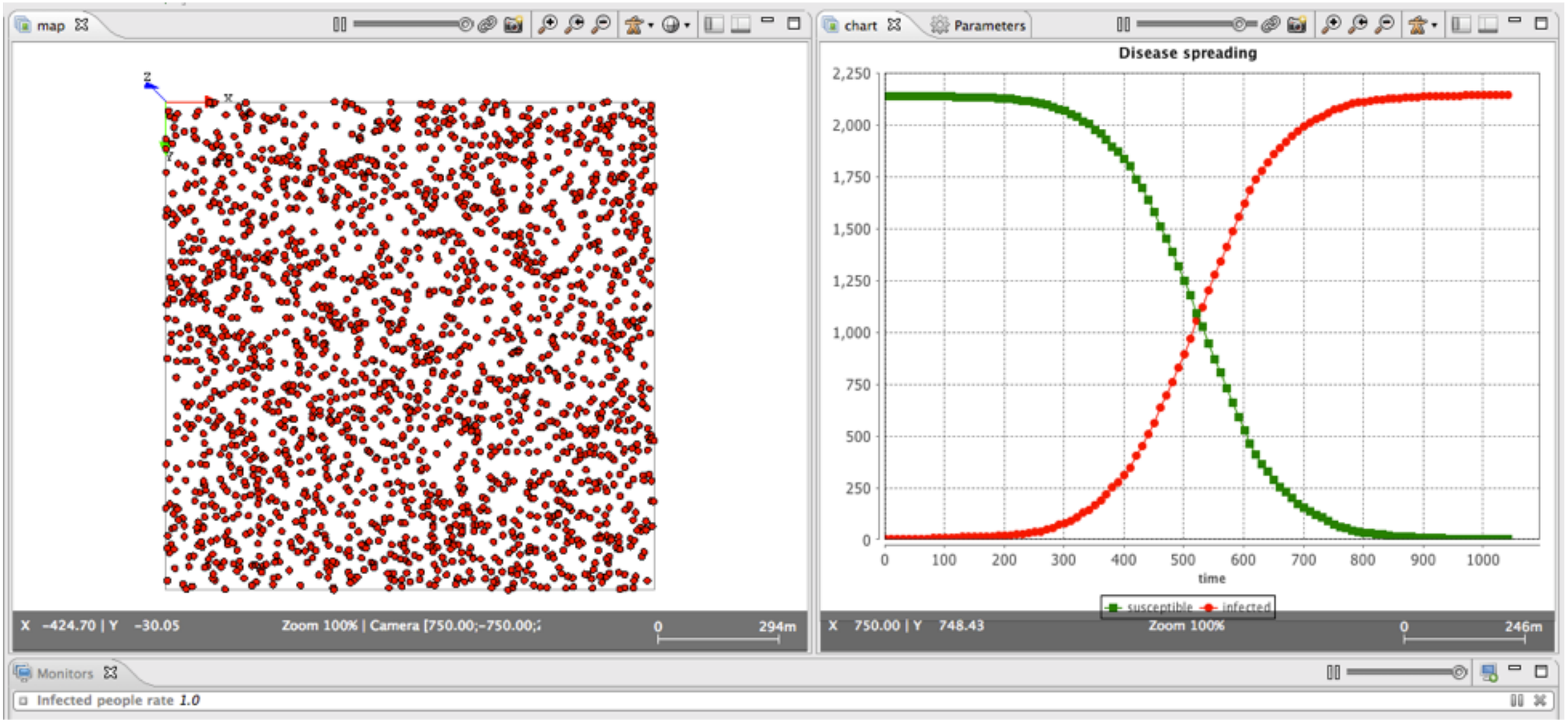
Layer displaying the people agents with their *circle* aspect

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

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



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



Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

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 {
}
```

count the number of infected agents among the people agents

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

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 {
}
```

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

count the number of infected agents among the people agents

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

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

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

count the number of infected agents among the people agents

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 pause;
    }
}
```

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 pause the simulation

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

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

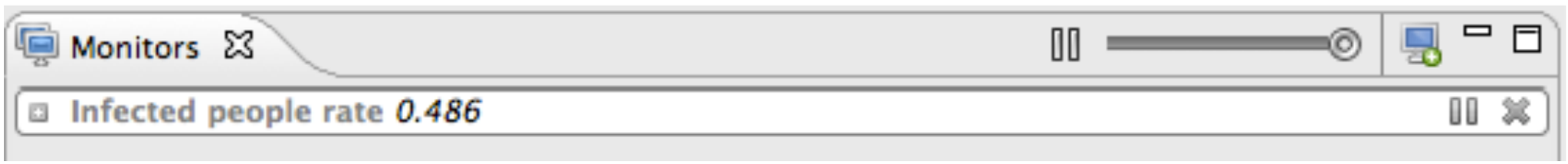
experiment block: monitor definition

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

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

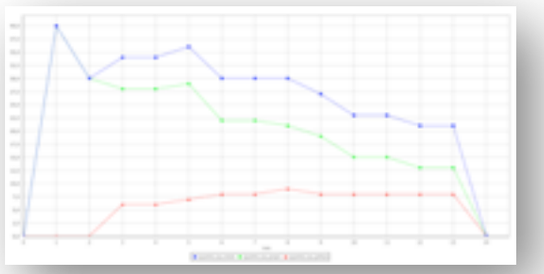
Model : monitor display

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

    //...display
    }
}
```

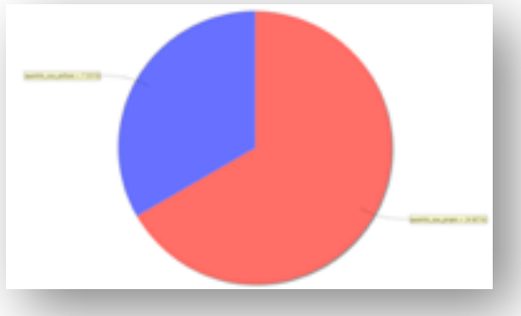


experiment block: chart definition

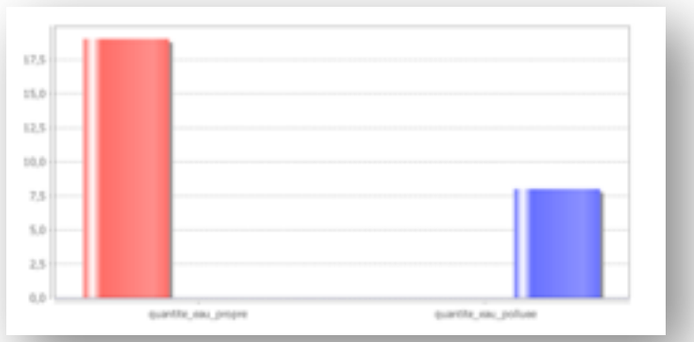
- GAMA allows to display several type of charts :
  - Series



  - Pie



  - H*istogram*



  - *XY chart*

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

experiment block: chart definition

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

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : chart display

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

        display chart_display refresh_every: 10 {
            chart "Disease spreading" type: series {
                data "susceptible" value: nb_people_not_infected color: #green;
                data "infected" value: nb_people_infected color: #red;
            }
        }
    }
}
```

experiment block: chart definition

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

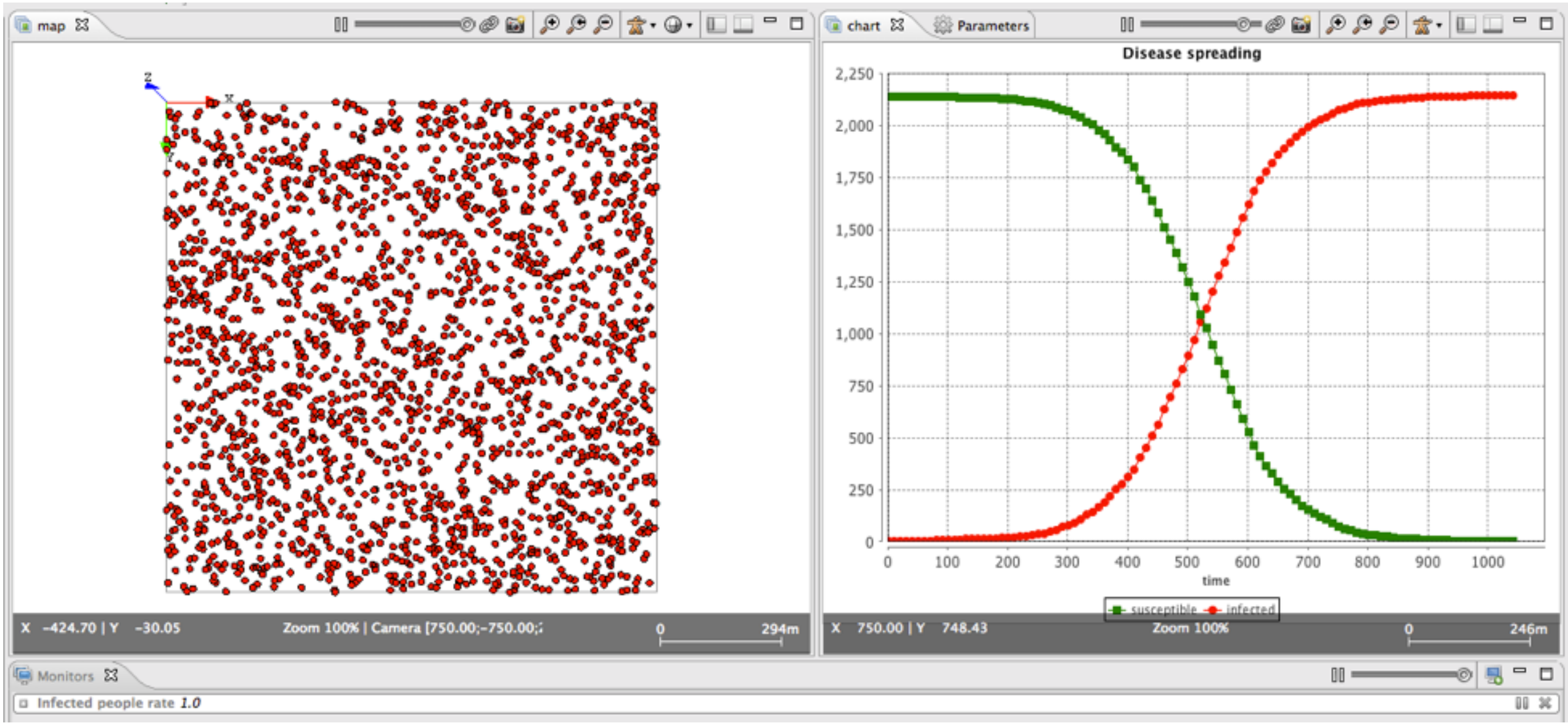
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : chart display

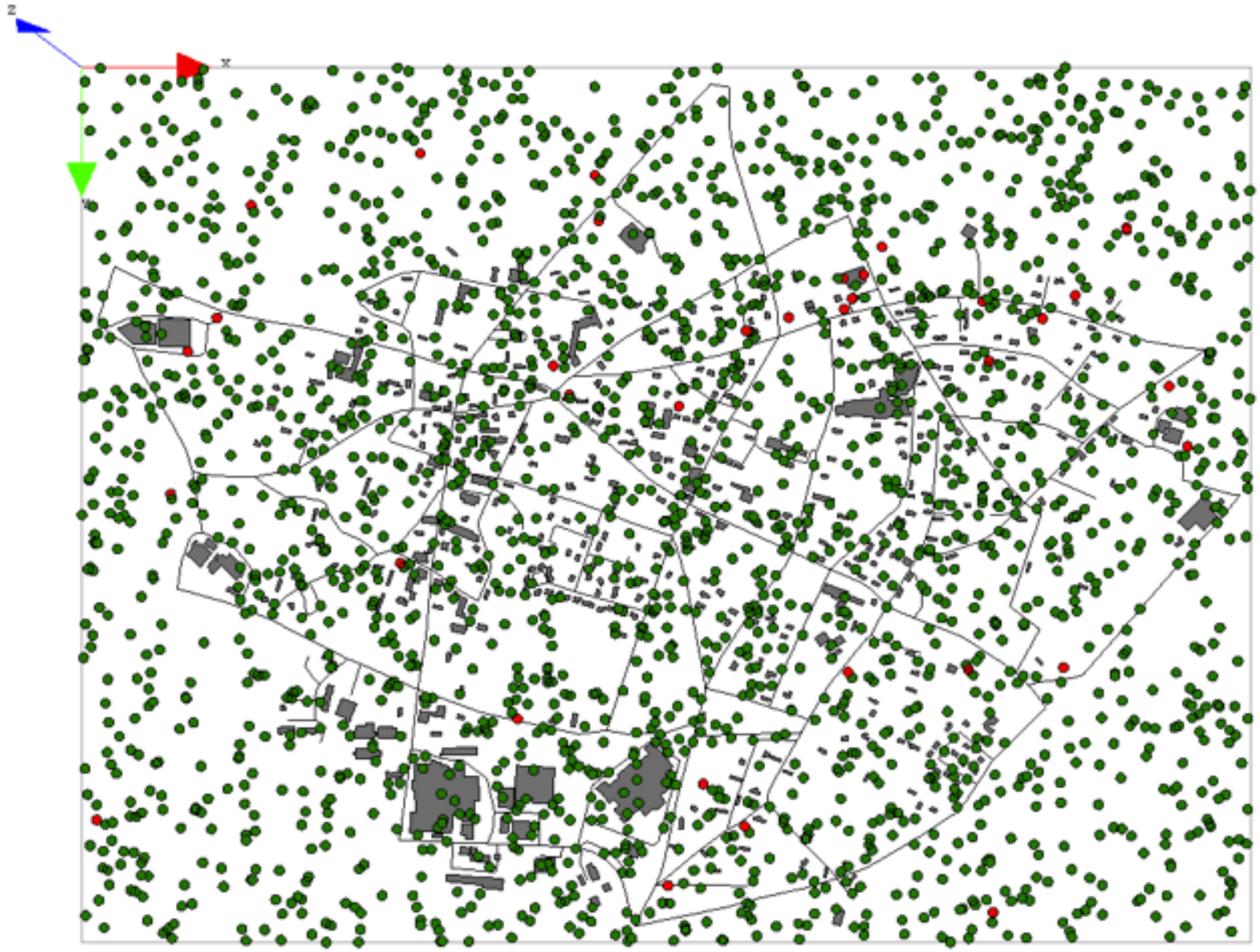
```
experiment main_experiment type:gui{
    //...parameters
    output {
        //...display and monitors

        display chart_display refresh_every: 10 {
            chart "Disease spreading" type: series {
                data "susceptible" value: nb_people_not_infected color: #green;
                data "infected" value: nb_people_infected color: #red;
            }
        }
    }
}
```

Display refreshed every 10 simulation steps

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



Loading of Geographical vector data to create buildings and roads



Model : road and building agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

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

Model : Add a house to people agent

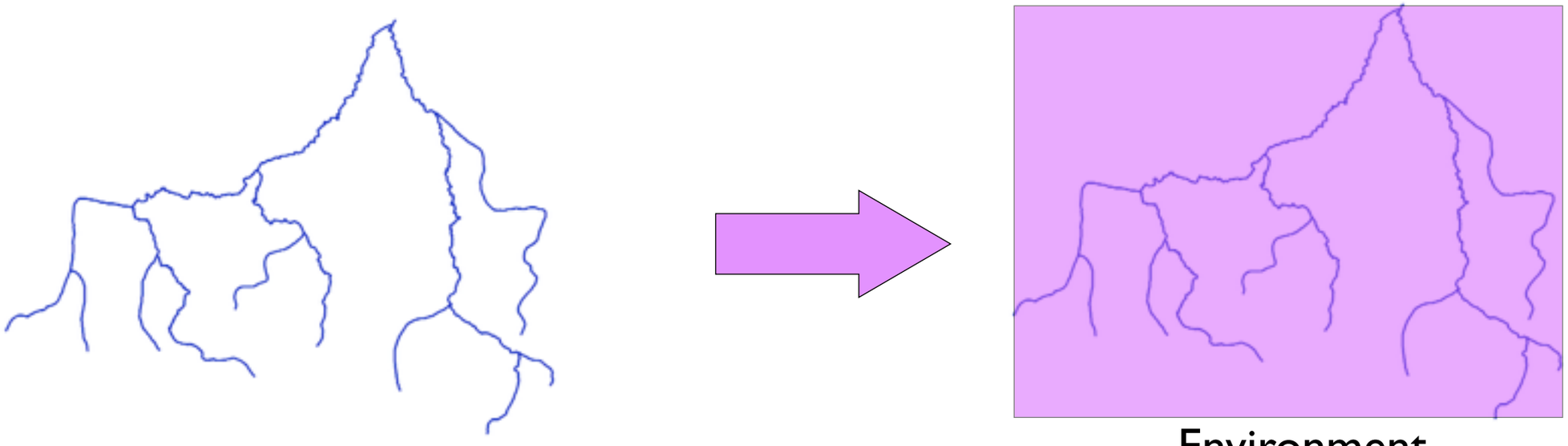
```
species people skills:[moving]{
    float speed <- (2 + rnd(3)) #km/#h;
    bool is_infected <- false;
    building my_house;
}
```

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

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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



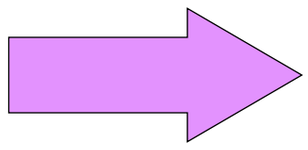
Environment

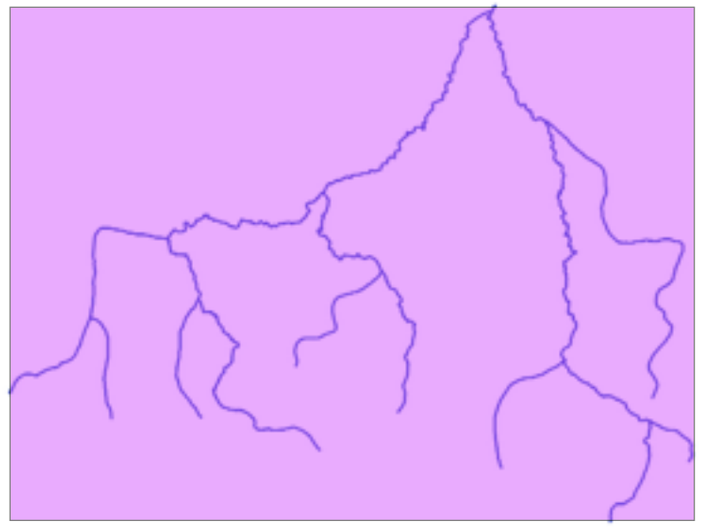
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model :World attributes

Shapefile of the roads

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





Environment

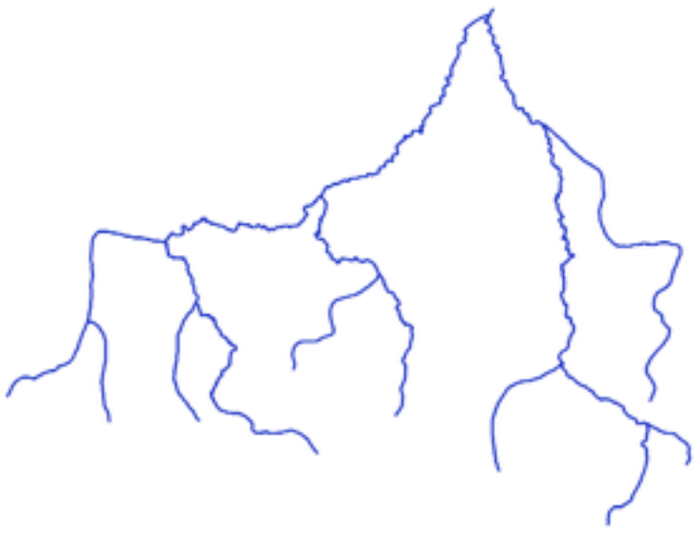
Model :World attributes

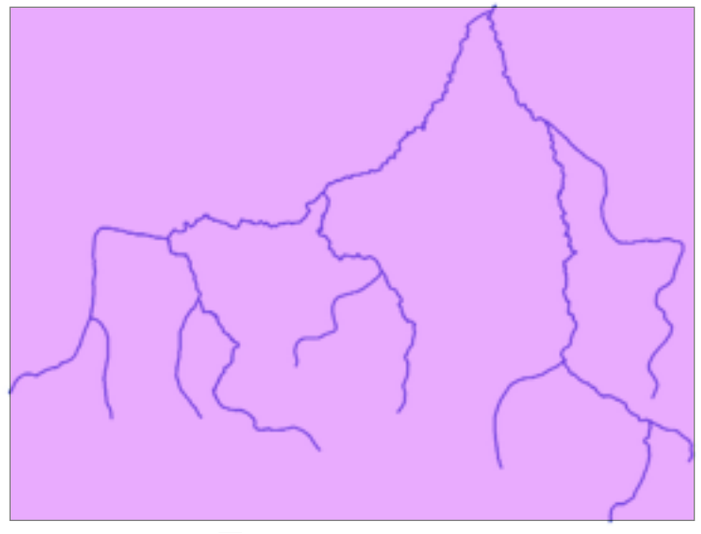
```
model my_model

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





Environment

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

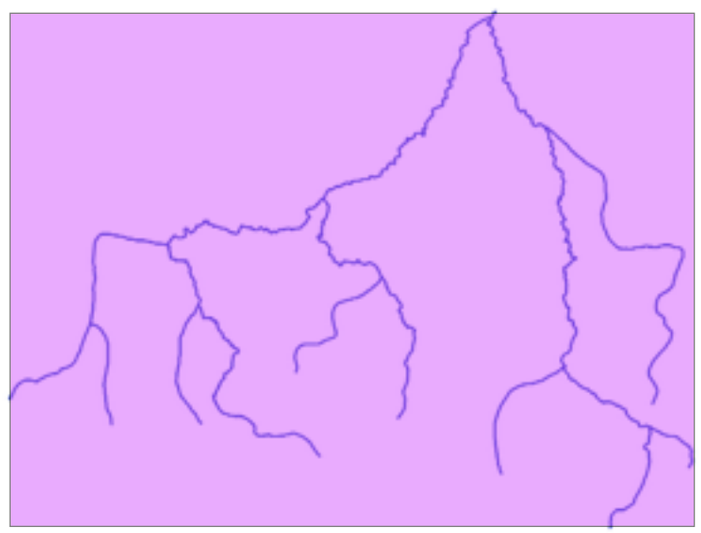
Model :World attributes

Shapefile of the roads

Shapefile of the buildings

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

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



Environment

Block global : agent creation from GIS data

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

Model :World init

```
global {
    // world variable definition

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

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 {
            my_house <- one_of(building);
            location <- any_location_in(my_house);       }
        }
        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 {
            my_house <- one_of(building);
            location <- any_location_in(my_house);        }
        }
        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 {
            my_house <- one_of(building);
            location <- any_location_in(my_house);
        }
        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 {
            my_house <- one_of(building);
            location <- any_location_in(my_house);
        }
        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

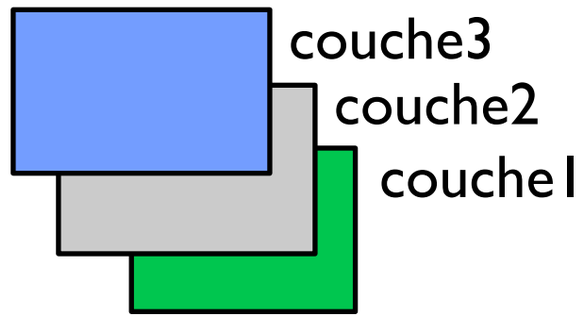
place the people agent on a random point in the selected building

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

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

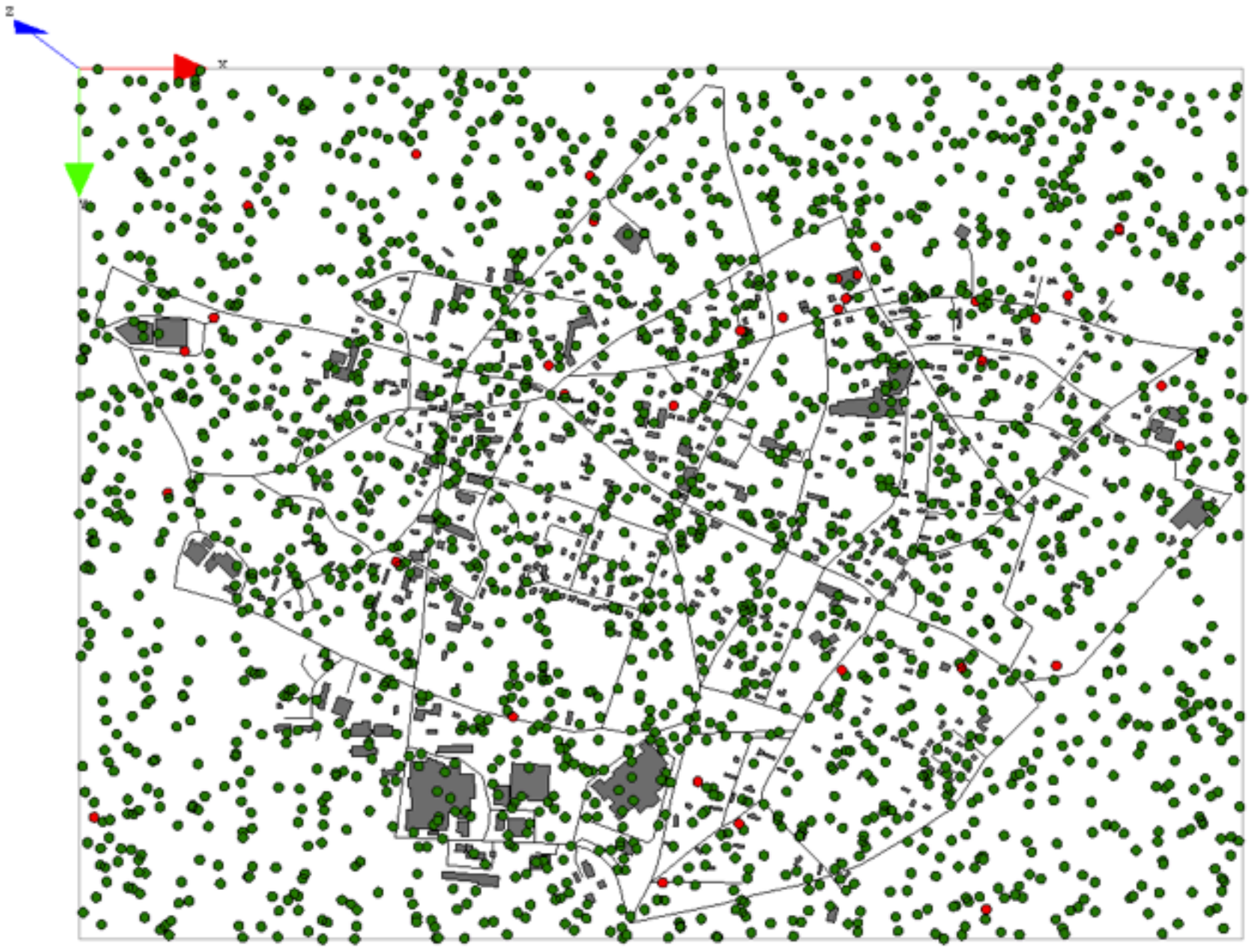
Model : experiment

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

    output {
    ... //monitor definition

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

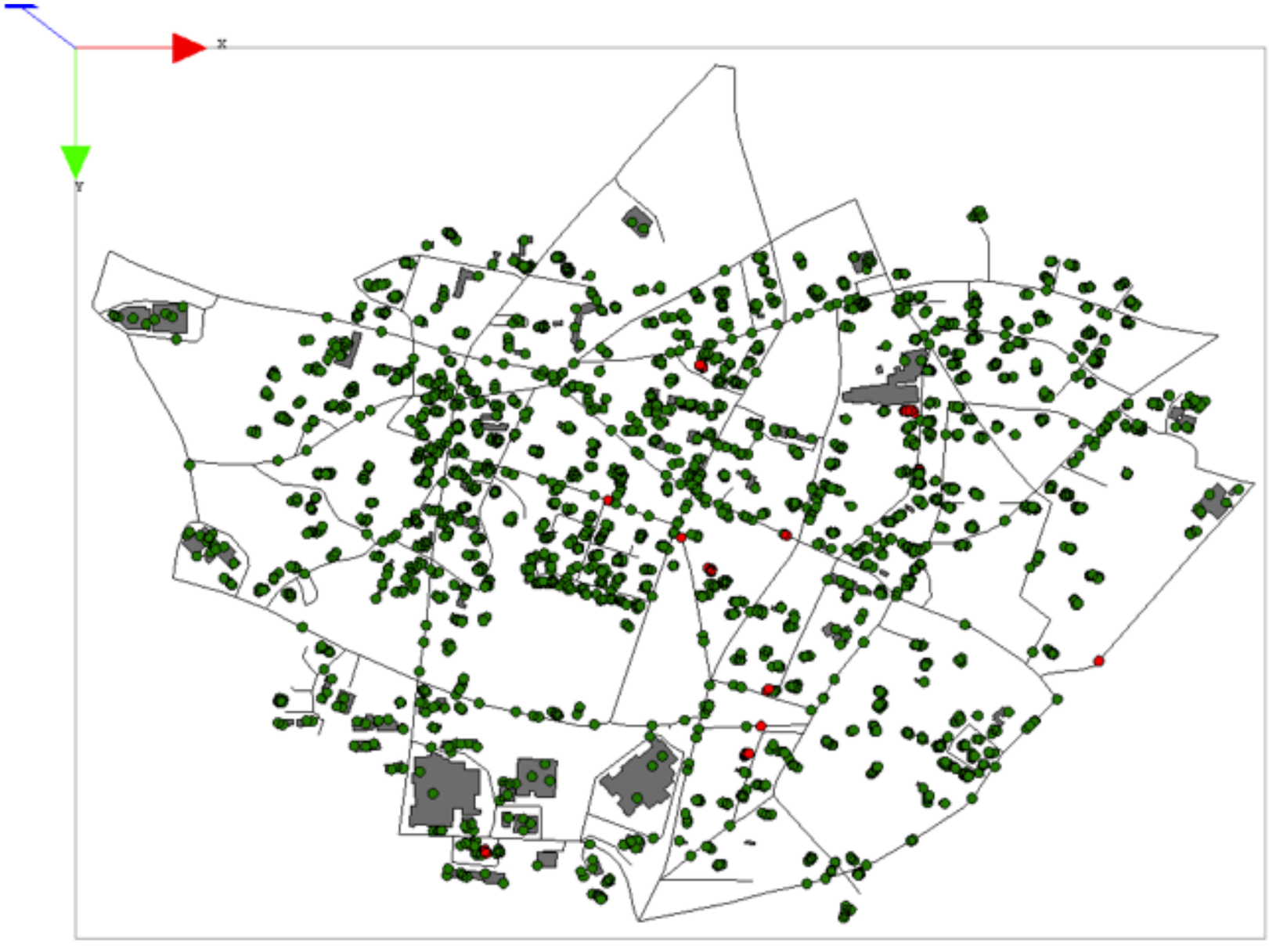


Note: in a *display*, the display order of the layer follows the layer definition

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;

    //... init
}
```

Model :World attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

Graph variable

Block global : graph building from road agents

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

Model :World init

```
global {
    // world variable definition

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

    }
  }
}
```

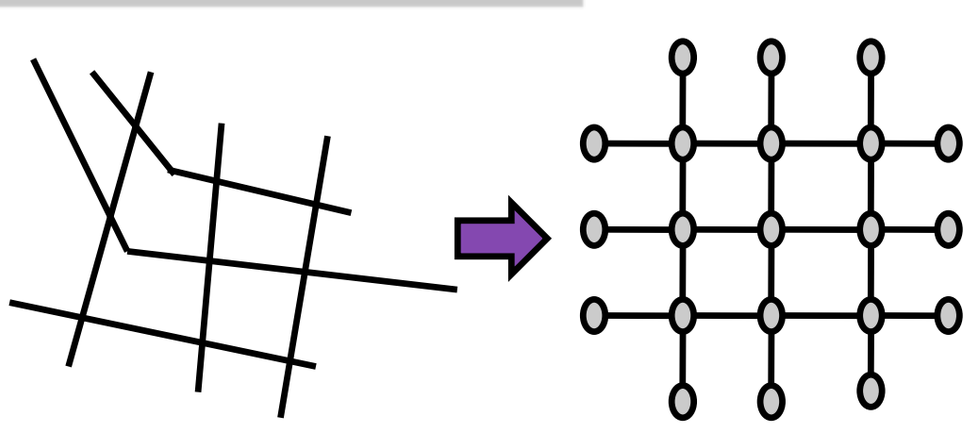
Block global : graph building from road agents

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

Model : World init

```
global {
    // world variable definition

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

    }
  }
}
```

Create a graph from the road geometries (polyline)



Model : People attributes

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

Model : Some reflexes of people

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        if flip(in_my_house ? 0.01 : 0.1) {
            building bd_target <- in_my_house ? one_of(building) : my_house;
            target <- any_location_in (bd_target);
            in_my_house <- not in_my_house;
        }
    }

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

}
```

Model : Some reflexes of people

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 {
        if flip(in_my_house ? 0.01 : 0.1) {
            building bd_target <- in_my_house ? one_of(building) : my_house;
            target <- any_location_in (bd_target);
            in_my_house <- not in_my_house;
        }
    }

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

}
```

Model : Some reflexes of people

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

test the probability to leave: if the agent is in its house: 0.01; 0.1 otherwise

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        if flip(in_my_house ? 0.01 : 0.1) {
            building bd_target <- in_my_house ? one_of(building) : my_house;
            target <- any_location_in (bd_target);
            in_my_house <- not in_my_house;
        }
    }

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

}
```

Model : Some reflexes of people

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

test the probability to leave: if the agent is in its house: 0.01; 0.1 otherwise

Choose a new target: can be either a point in one random building if the agent is already inside its house or a point inside its house otherwise

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        if flip(in_my_house ? 0.01 : 0.1) {
            building bd_target <- in_my_house ? one_of(building) : my_house;
            target <- any_location_in (bd_target);
            in_my_house <- not in_my_house;
        }
    }

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

}
```

Model : Some reflexes of people

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

test the probability to leave: if the agent is in its house: 0.01; 0.1 otherwise

Reflex activated only when the target is not nil

Choose a new target: can be either a point in one random building if the agent is already inside its house or a point inside its house otherwise

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        if flip(in_my_house ? 0.01 : 0.1) {
            building bd_target <- in_my_house ? one_of(building) : my_house;
            target <- any_location_in (bd_target);
            in_my_house <- not in_my_house;
        }
    }

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

}
```

Model : Some reflexes of people

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

test the probability to leave: if the agent is in its house: 0.01; 0.1 otherwise

Reflex activated only when the target is not nil

Choose a new target: can be either a point in one random building if the agent is already inside its house or a point inside its house otherwise

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

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        if flip(in_my_house ? 0.01 : 0.1) {
            building bd_target <- in_my_house ? one_of(building) : my_house;
            target <- any_location_in (bd_target);
            in_my_house <- not in_my_house;
        }
    }

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

}
```

Model : Some reflexes of people

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

test the probability to leave: if the agent is in its house: 0.01; 0.1 otherwise

Reflex activated only when the target is not nil

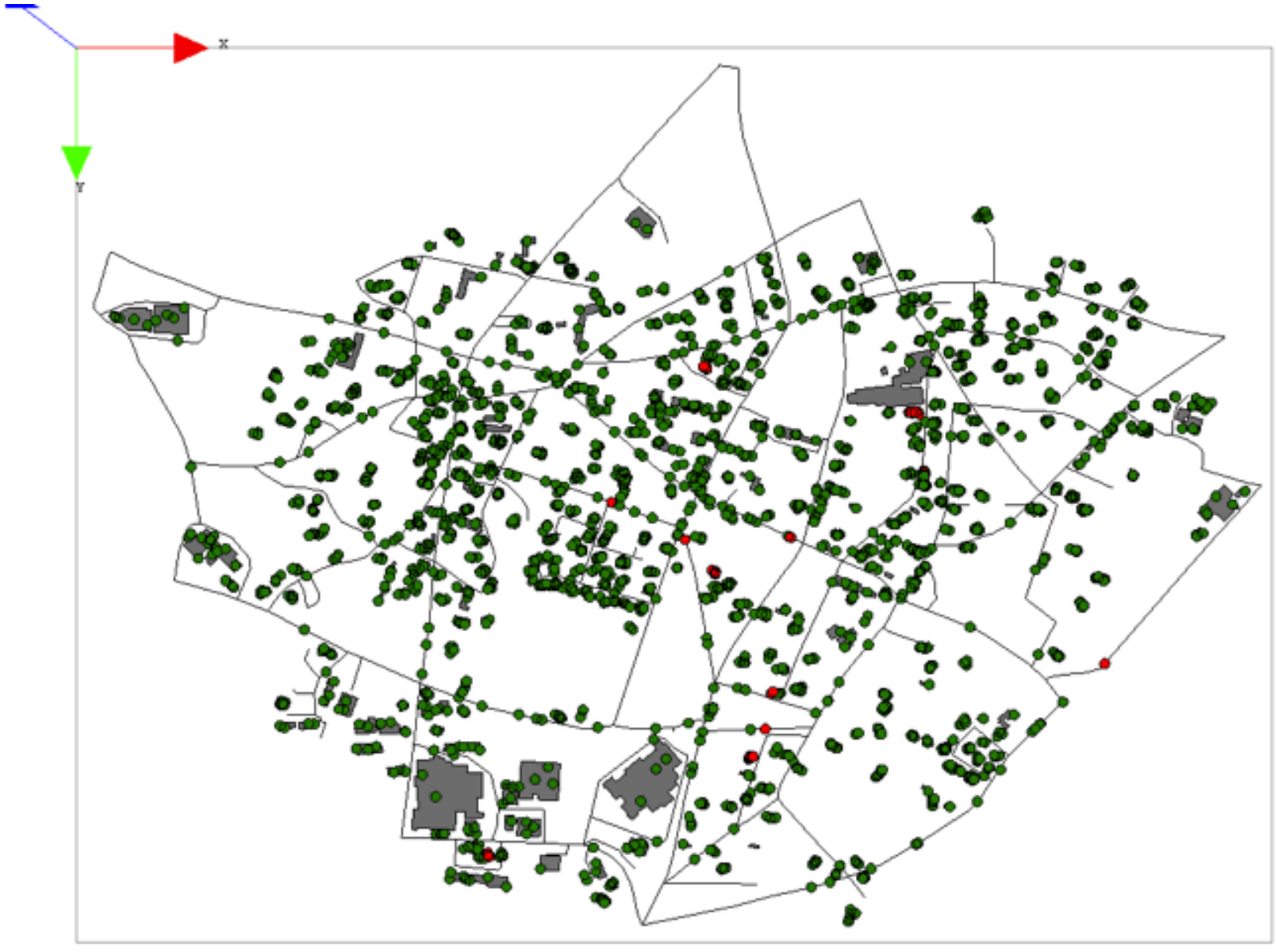
If the agent reaches its target, set target to nil

Choose a new target: can be either a point in one random building if the agent is already inside its house or a point inside its house otherwise

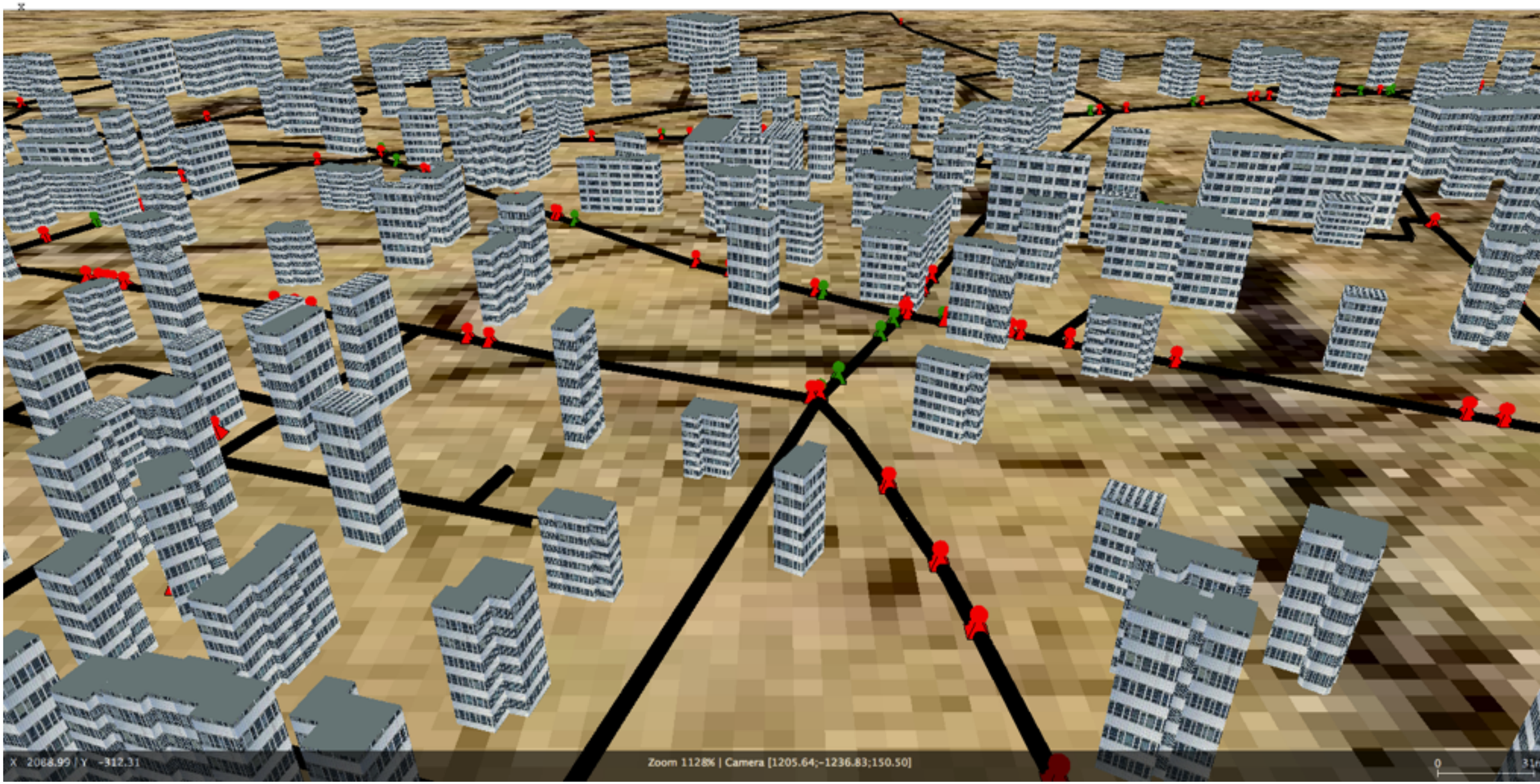
Move toward the target along the graph (shortest path computation). Use of the *goto* action of the moving skill

```
species people skills:[moving]{
    //variable definition
    reflex stay when: target = nil {
        if flip(in_my_house ? 0.01 : 0.1) {
            building bd_target <- in_my_house ? one_of(building) : my_house;
            target <- any_location_in (bd_target);
            in_my_house <- not in_my_house;
        }
    }

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

}
```

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



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



species block: Species definition

Model : road and building agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

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

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

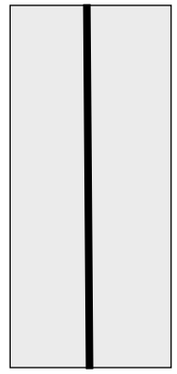
species block: Species definition

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) as tube of radius 2m

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

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

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



species block: Species definition

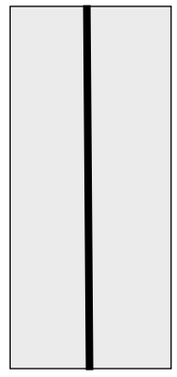
Model : road and building agents

```
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) as tube of radius 2m

New aspect: draw the display_shape geometry

```
species road {
    geometry display_shape <- line(shape.points, 2.0);
    //....
    aspect geom3D {
        draw display_shape color: #black;
    }
}

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



species block: Species definition

Model : road and building agents

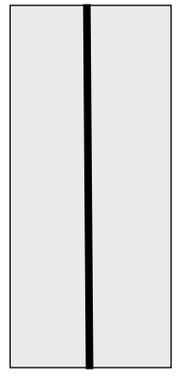
```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

New aspect: draw the display_shape geometry

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

```
species road {
    geometry display_shape <- line(shape.points, 2.0);
    //....
    aspect geom3D {
        draw display_shape color: #black;
    }
}

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



species block: Species definition

Model : road and building agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

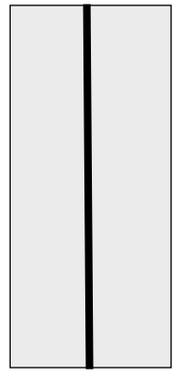
Definition of new variable of type geometry that represent the geometry of the road (polyline) as tube of radius 2m

New aspect: draw the display_shape geometry

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

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

```
species road {
    geometry display_shape <- line(shape.points, 2.0);
    //....
    aspect geom3D {
        draw display_shape color: #black;
    }
}

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



species block: Species definition

Model : people agents

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

species block: Species definition

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

Model : people agents

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

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

experiment block: output definition

Model : *new 3D display*

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

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

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

Model : *new 3D display*

```
model my_model

global {
}

species my_species{
}

experiment my_model type: gui {
}
```

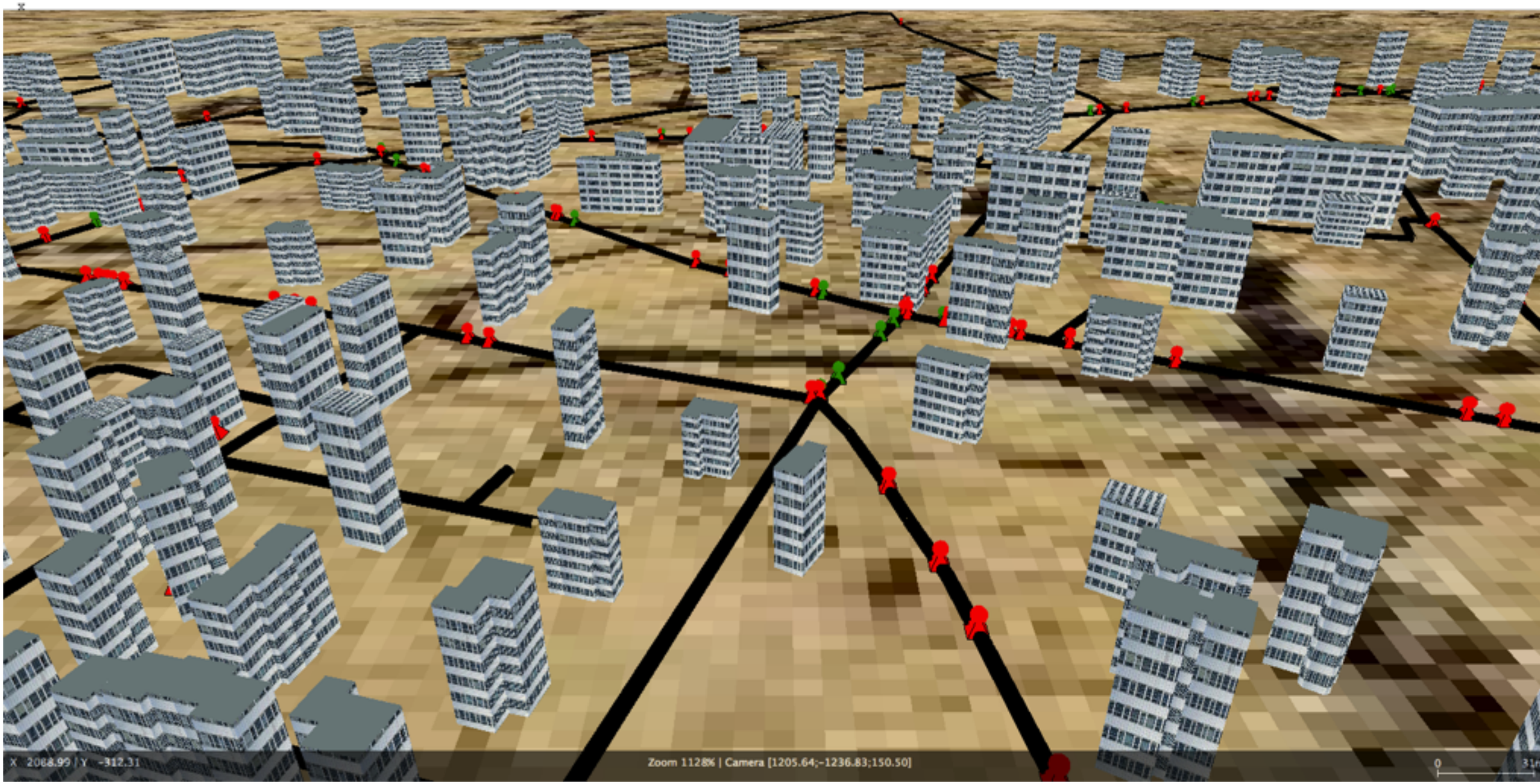
```
experiment main_experiment type: gui {
    output {
    // monitor and other displays
        display map_3D type: opengl ambient_light: 120 {
            image "../includes/soil.jpg" refresh: false;
            species road aspect:geom3D refresh: false;
            species building aspect:geom3D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

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

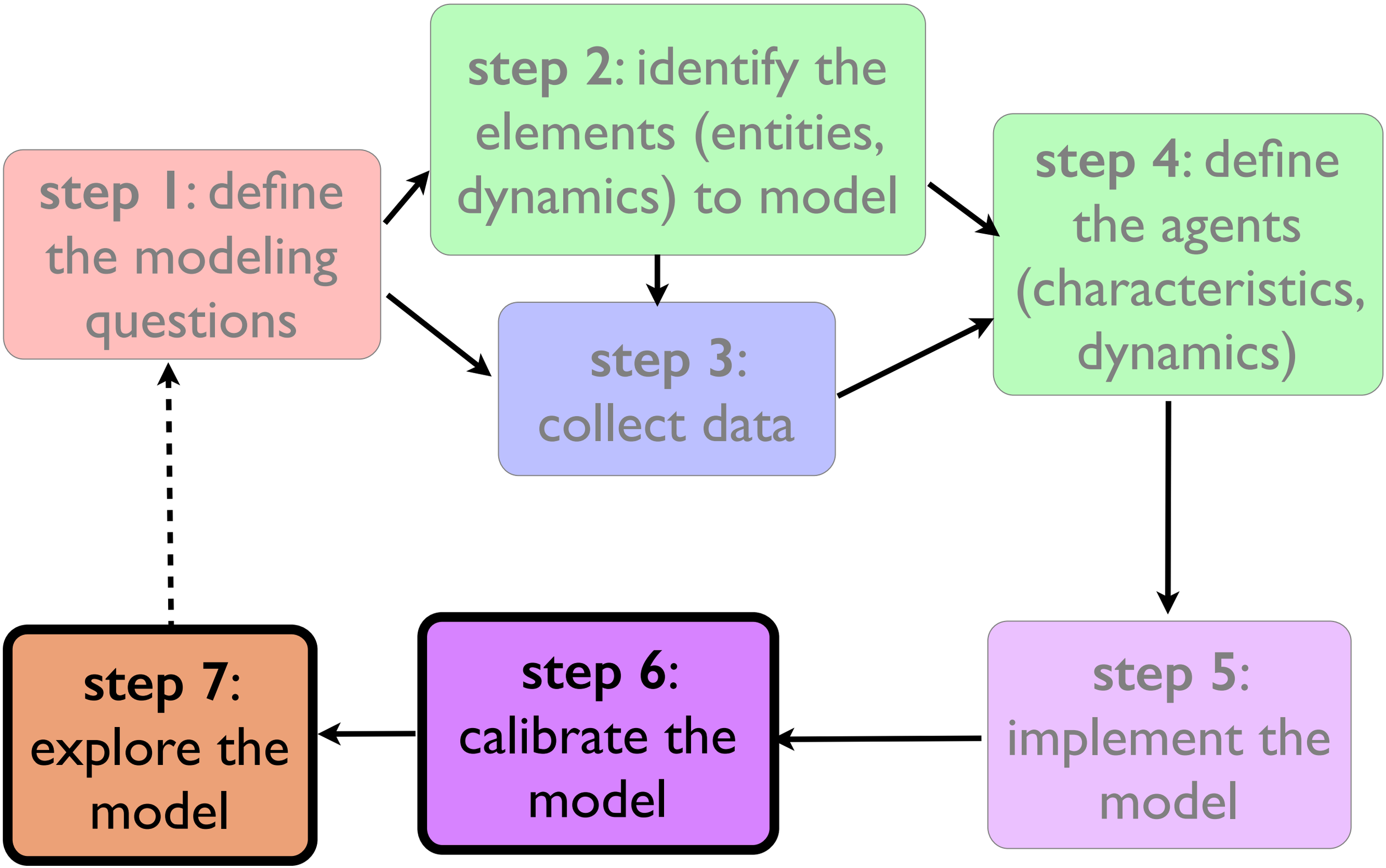
the **refresh: false** facet allows to specify that a layer does not have to be refresh and allows to optimize the display

End of the fith version of the model

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



Modeling steps



Exploration and calibration

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

https://github.com/gama-platform/gama/wiki/G__BatchExperiments

Conclusion

- GAMA is a generic open-source platform dedicated to the development and simulation of agent-based models

- More and more used (more than 3 000 downloads for the last versions, more than 180 research papers citing GAMA)

- In the next version of GAMA (December 2015):

➡ Performance improvement

➡ Bug fixes

➡ Improvement of the graphic user interface

➡ Co-modeling

➡ Possibility to read more file formats (in particular for 3D data)

➡ Better integration of BDI agent architectures