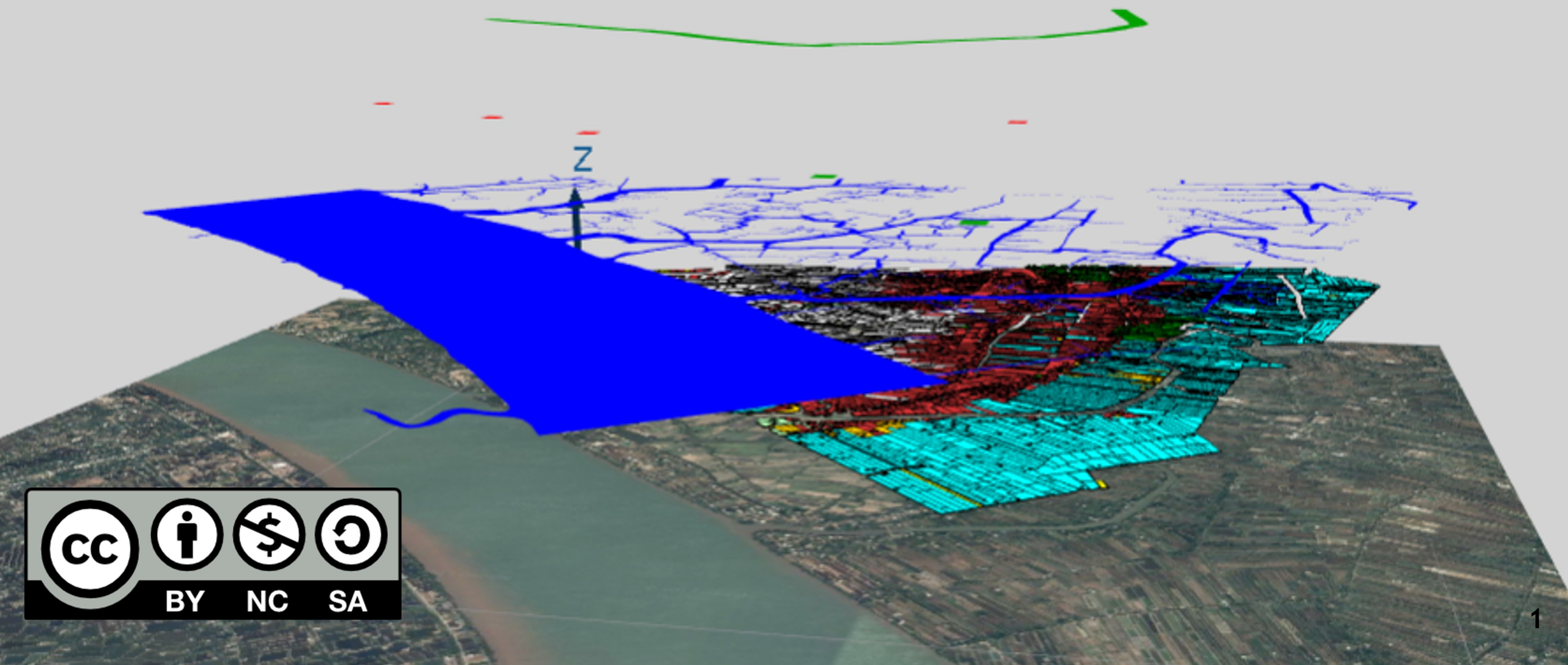
Introduction to simulation of complex systems

Baptiste Lesquoy, *IRD UMMISCO, Thuy Loi University; baptistelesquoy@protonmail.com*



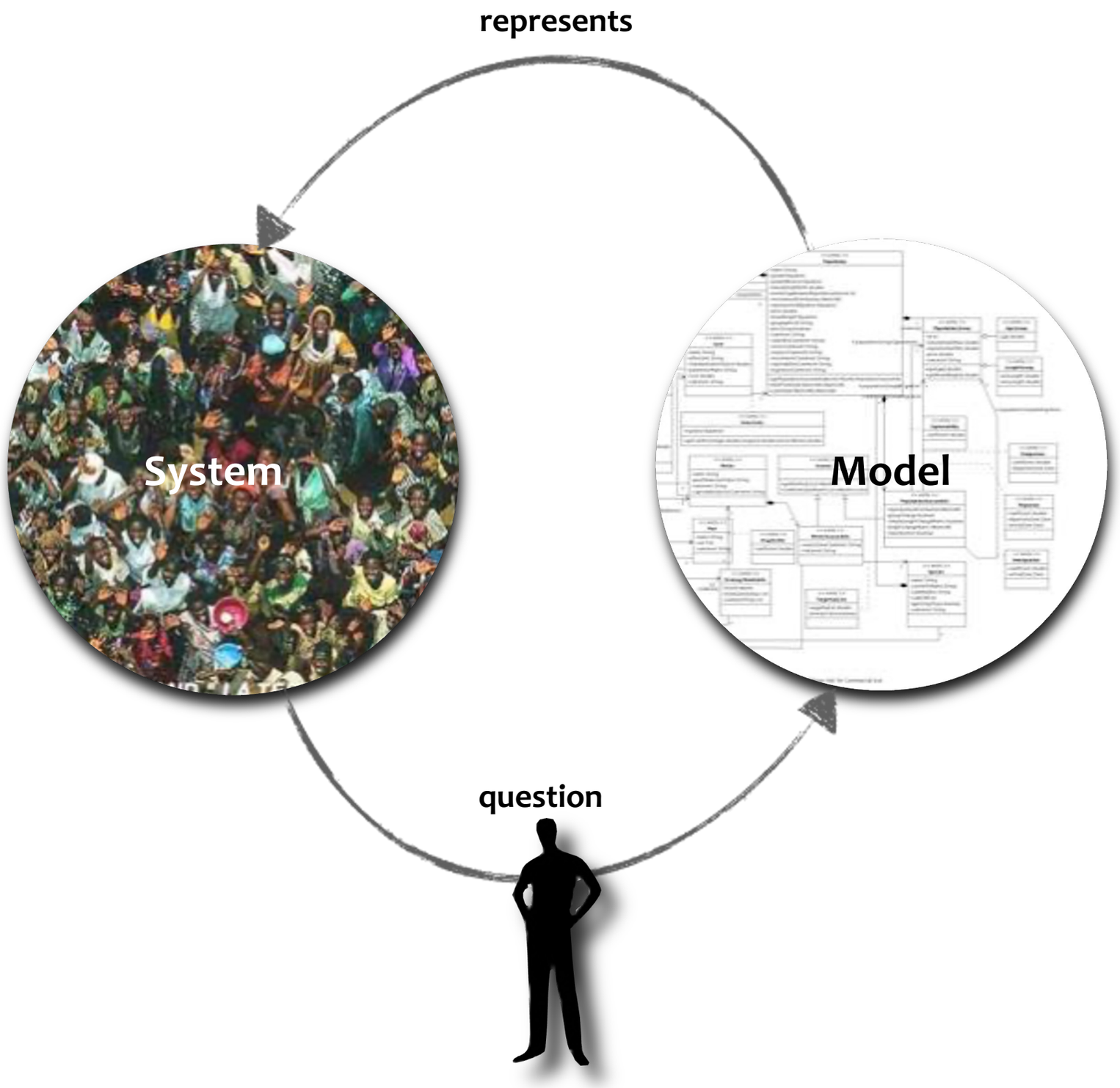
Course plan

- Week 1: Modeling session - GAMA platform

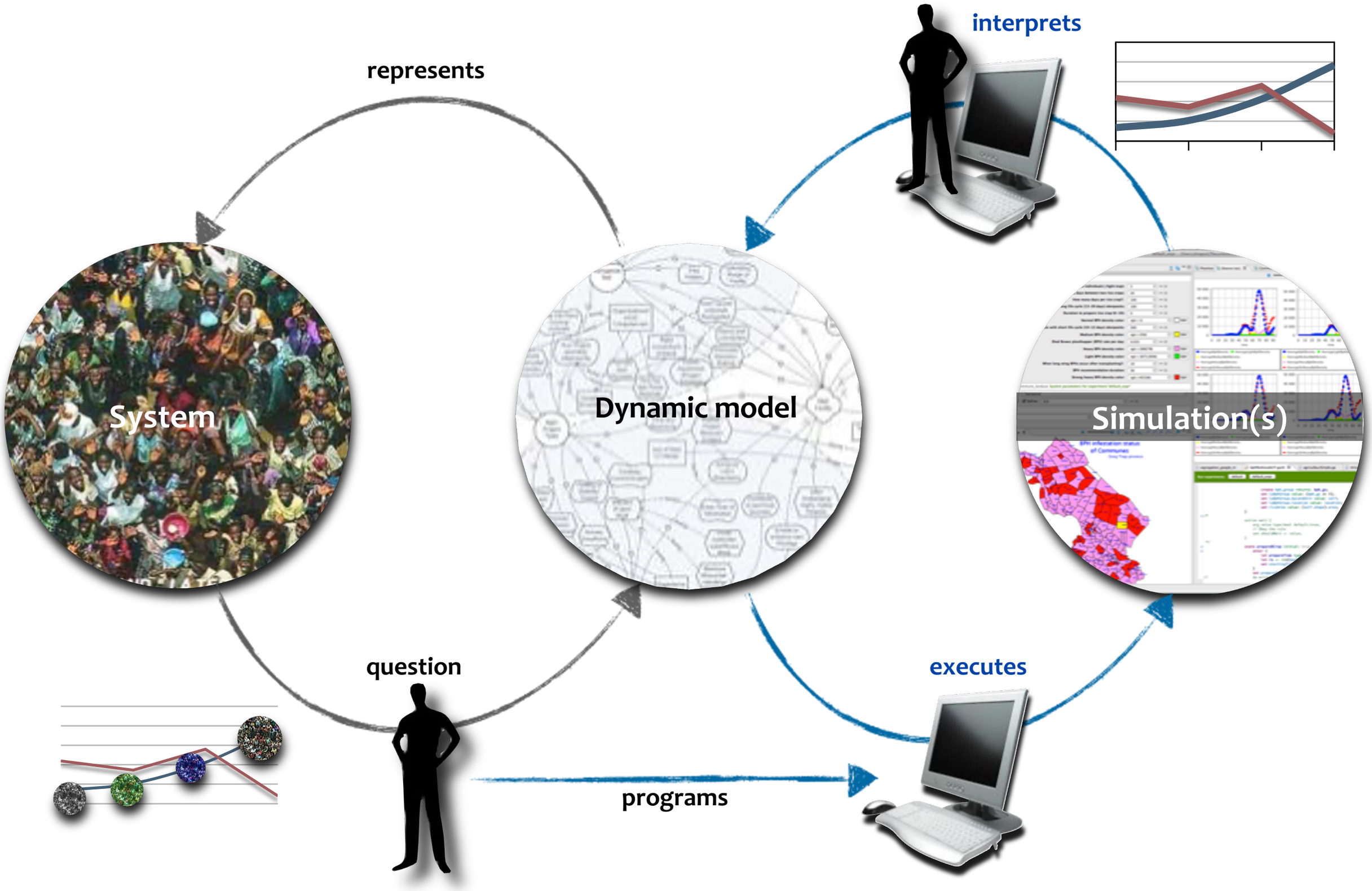
- Week 2: Simulation session
  - Monday: Exercice: multi-modal traffic
  - Tuesday: Introduction to simulation
  - Wednesday: Calibration
  - Thursday: Participation and interaction
  - Friday: Coupling gama and other softwares
  - Saturday: Project

Definitions

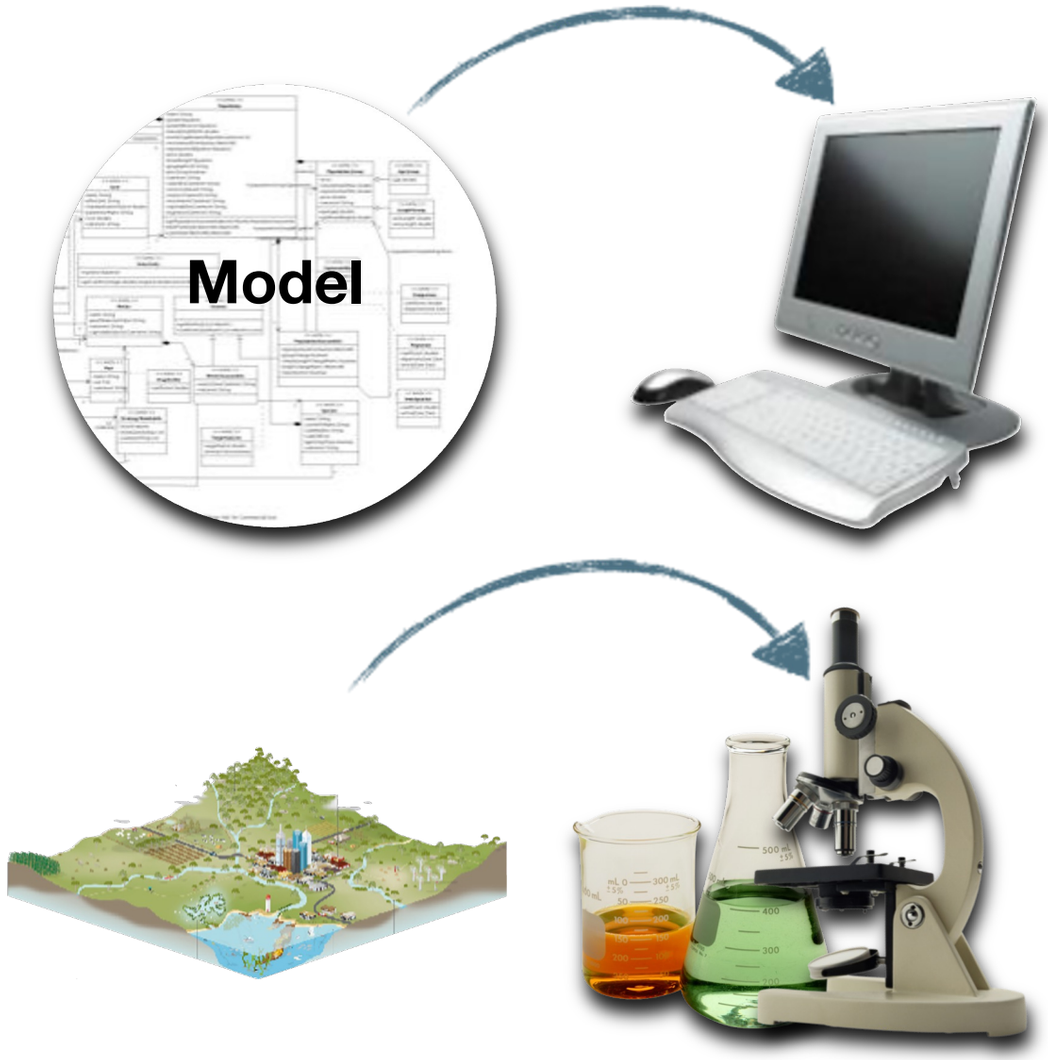
A model is a simplified representation of a reference system, designed to help answering a question on this system.

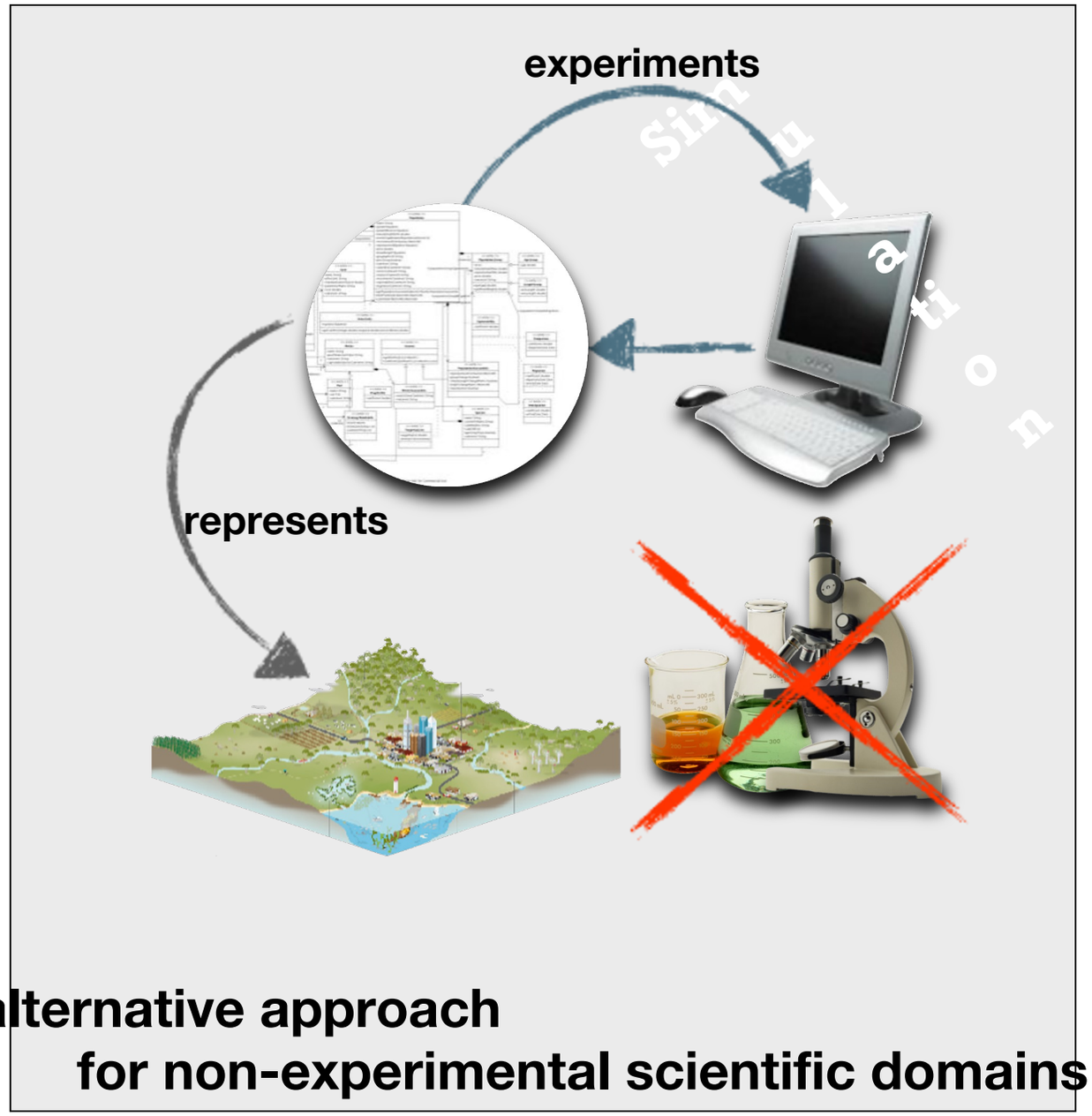


The execution of a (dynamic) computer model with a given set of parameters is called a simulation.



Simulations are to a model what experimentations are to the real system: controlled perturbations (changes of parameters, introduction of new structures) to answer specific questions

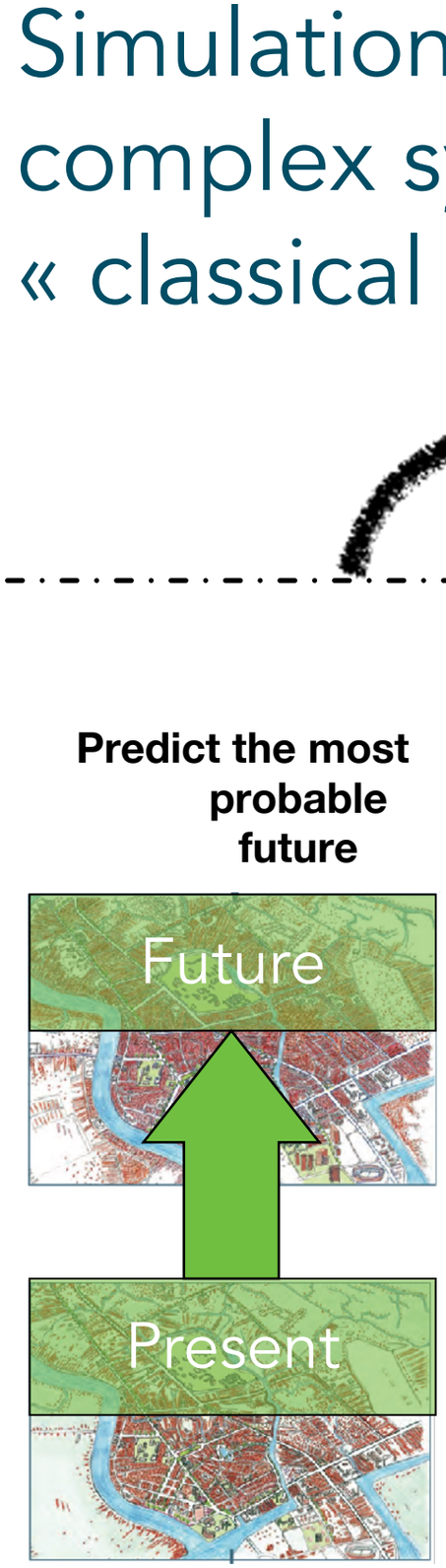




alternative approach
for non-experimental scientific domains

Simulations can be used for multiple purposes, but complex systems require often moving beyond their « classical uses » to explore multiple trajectories

Predict the most probable future

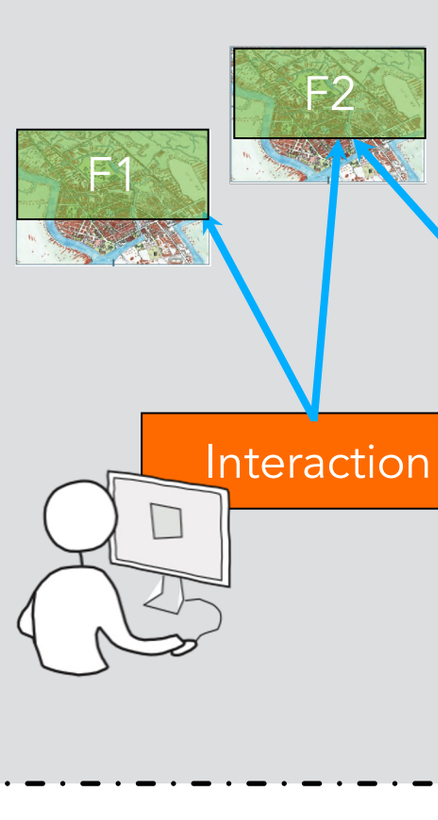


Best path to reach a desired future

Classical use of models in decision-support

Exploration

Generate, assess, explore and compare some scenarios of evolution



J.E.J. Ebert, D. T. Gilbert, T. D. Wilson, Forecasting and Backcasting: Predicting the Impact of Events on the Future, Journal of Consumer Research, Inc., Vol. 36, October 2009

A model is an object that can be manipulated or explored via simulation and on which we can launch experiments.

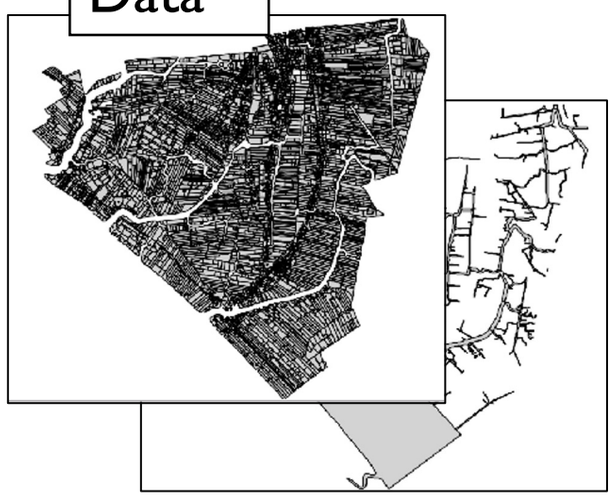
Parameters

Param 1
value between x1 and y1

Param 2
value between x2 and y2

Param 3
value between x3 and y3

Inputs
Data



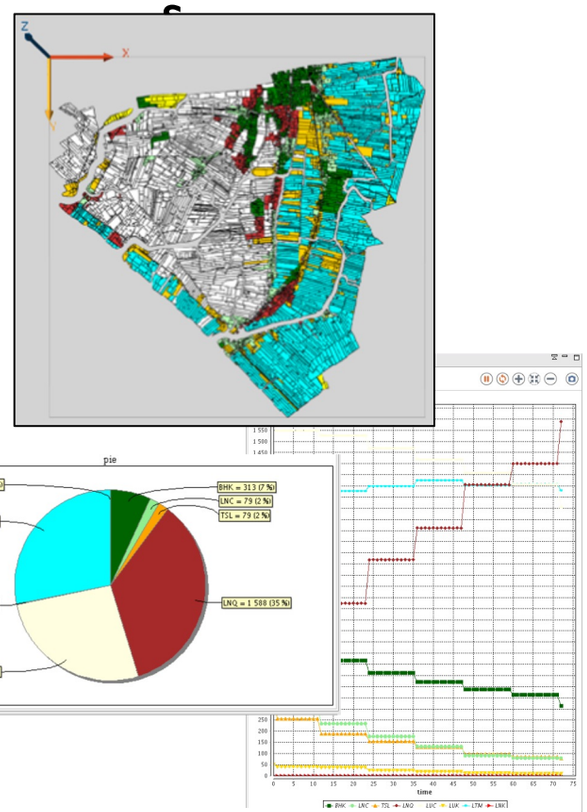
Experiments

Calibration | Charts
Sensibility | display_map

Model

```
8 model experimentationcoup
9 //import 'theoretic_network.gaml'
10 import 'theoretic_network3.gaml'
11 import 'theoretic_meteo.gaml'
12
13 global
14 {
15     //Step of 1#hour;
16     float step<-1#hour;
17     //Starts at 00:00:00 the 01/01/2016
18     date starting_date<-date([2016,1,1,0,0,0]);
19     file meteo_file <- csv_file("../includes/meteo.csv");
20     file building_file <- file("../includes/building.shp");
21     file road_file <- file("../includes/road.shp");
22     file water_file <- file("../includes/waterway.shp");
23     list<building> houses;
24     list<building> commercials;
25     list<consumer> households -> {consumer where (each.type = "household")};
26     list<consumer> companies -> {consumer where (each.type = "company")};
27     meteo_handler the_meteo_handler;
28     map<string,geometry> possible_place ;
29     geometry shape <- envelope(road_file);
30     bool new_decision_time update:cycle > 1 and (cycle mod (24*30)) = 0 ;
31     bool new_cycle_time update:cycle > 1 and (cycle mod (24*30)) = 1 ;
```

Output



Kappa indicator: 0.005719475416811799

The model will be only considered as a black box.

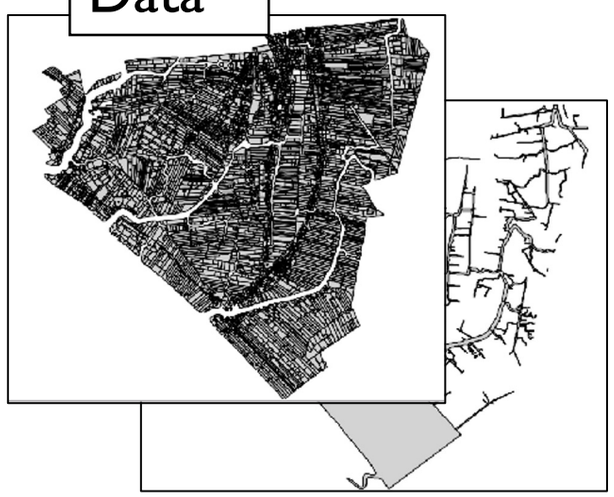
Parameters

Param 1
value between x1 and y1

Param 2
value between x2 and y2

Param 3
value between x3 and y3

Inputs
Data

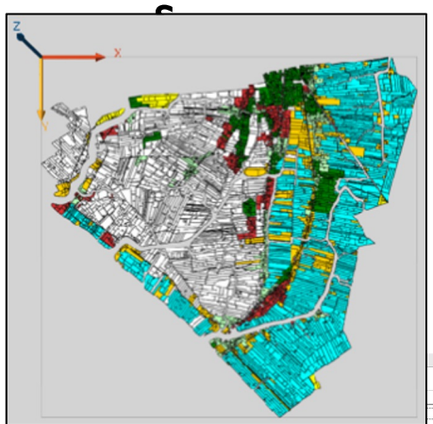


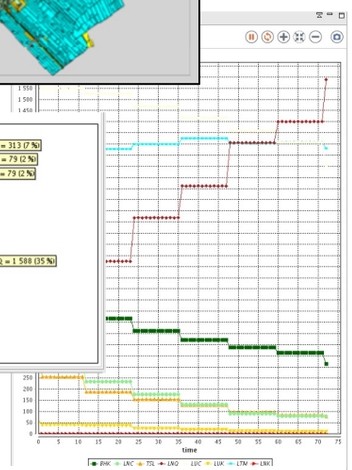
Experiments

Calibration | Charts
Sensibility | display_map

Model

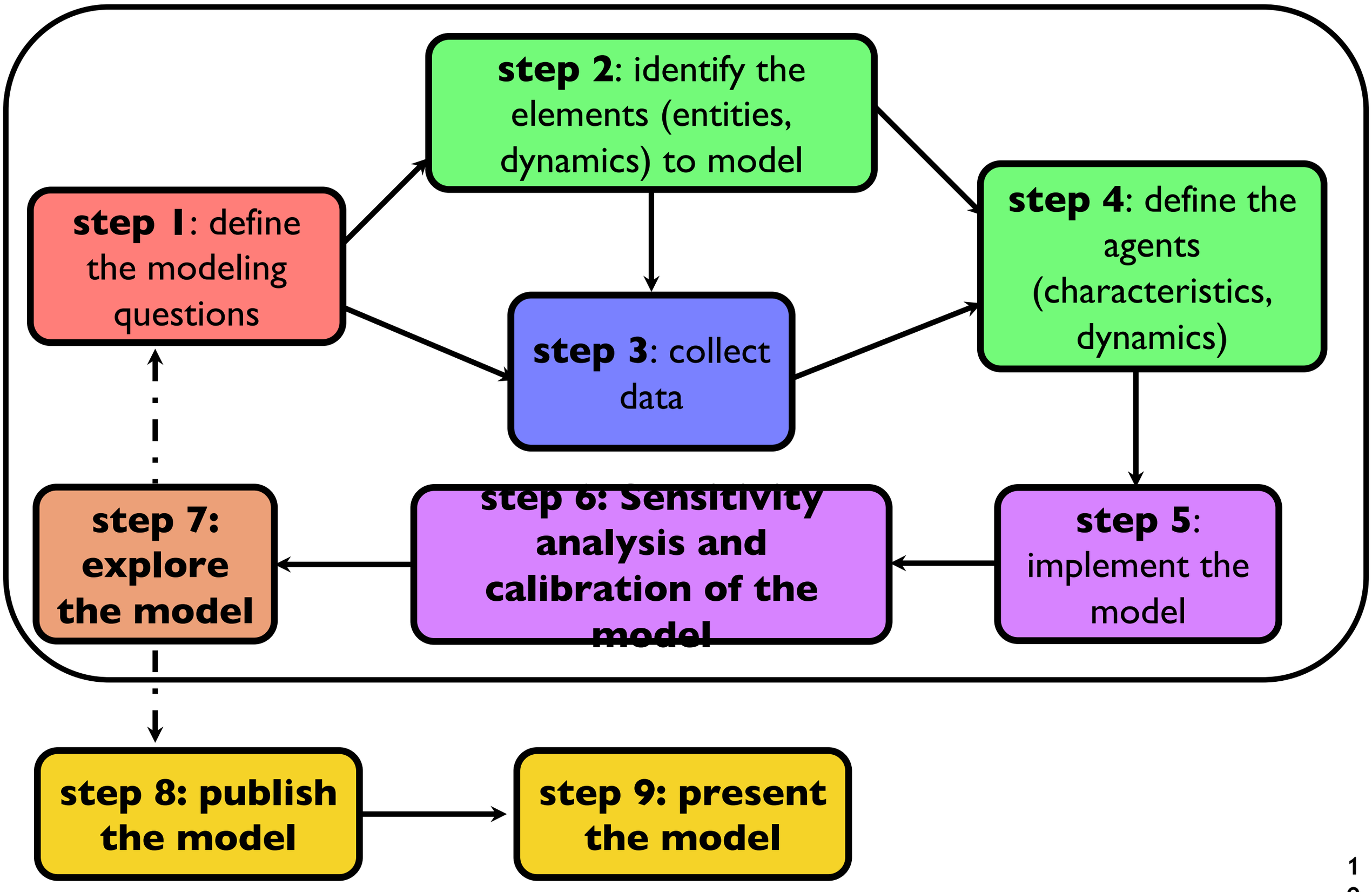
Output





Kappa indicator: 0.005719475416811799

Modelling is a multi-step cycle.



Managing simulations in GAMA

GAMA proposes 2 ways to run simulations = 2 kinds of experiments.

GUI experiments:

Gui experiments allow the modeler to observe indicators evolution during a simulation

Batch experiments:

batch experiments allow model to observe the evolution of indicators of the simulation (computed at the end of the simulation) over the change of parameters.

```
model integrated
global { ... }
...
experiment "Debug mode" type: gui / batch
{ ... }
```

```
experiment "Debug mode" type: gui {
   [parameters]
   output {
      ...
   }
}
```

```
experiment Optimization type: batch
      [additional facets] {

   [parameters]
   [exploration method]
}
```

Note: there exists 1 more type of experiment: **test** for unit tests.

Several experiment can be defined for 1 model.

```
experiment display_map type:gui {
  // some code here ...
}

experiment Calibration type:batch  {
  // some code here ...
}

experiment Sensibility type:batch {
  // some code here ...
}
```

We can define inputs of an experiment via parameter.

Each parameter must be defined inside the experiment.

Each parameter should refer to a global variable present in the global section of the model.

```
parameter "title" var:global_variable [possible_values] ;
```

General definition:

There are 2 ways to describe the value range :

- Explicit list: **among:** *values_list*

```
parameter 'my_variable_value' var: my_var among: [1,5,10];
```

my_var can take three values among 1, 5 and 10.

- Range : **min:** *min_value* **max:** *max_value* **step:** *increment_step*

```
parameter 'my_variable_value' var: my_var min: 10 max: 30 step: 1;
```

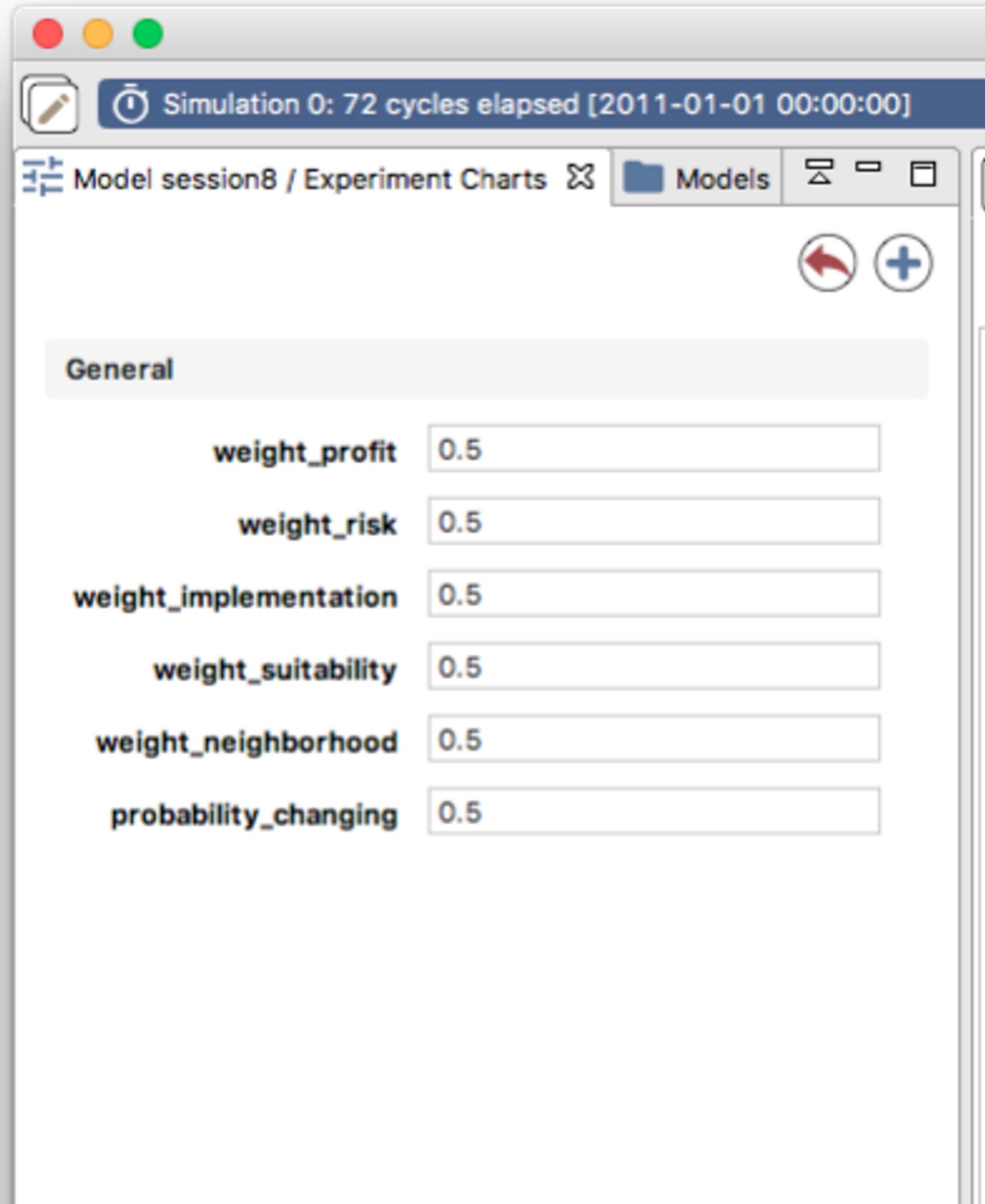
my_var could be equals to the following values : 10, 11, 12..., 29, 30

Examples of parameter definitions

```
global
{
        ...
      float weight_profit <- 0.5 parameter:true;
       float weight_risk <-0.5 parameter:true;
       float weight_implementation <-0.5 parameter:true;
       float weight_suitability <-0.5 parameter:true;
       float weight_neighborhood <-0.5 parameter:true;
       float probability_changing <- 0.5 parameter: true;

      ...
}
```

Another way to define a global variable as parameter

```
experiment Charts type: gui {
        parameter weight_profit  var:weight_profit
      min: 0.0 max: 1.0 step: 0.1;
        parameter weight_risk  var:weight_risk
      min: 0.0 max: 1.0 step: 0.1;
        parameter weight_implementation  var:weight_implementation
      min: 0.0 max: 1.0 step: 0.1;
        parameter weight_suitability  var:weight_suitability
      min: 0.0 max: 1.0 step: 0.1;
        parameter weight_neighborhood  var:weight_neighborhood
      min: 0.0 max: 1.0 step: 0.1;
        parameter probability_changing var:probability_changing
      min: 0.0 max: 1.0 step: 0.1;
...
}
```



Exercice: add parameters to an existing model

In the ForestFire model

- create a new GUI experiment called “test parameters” that displays the grid
- add a parameter to set different numbers of fire sources at the beginning of the simulation
- make that parameter defined between 1 and 25

In GAMA, even simulations are agents!

See the ForestFire model

- 1 simulation is automatically created. We can create new ones!

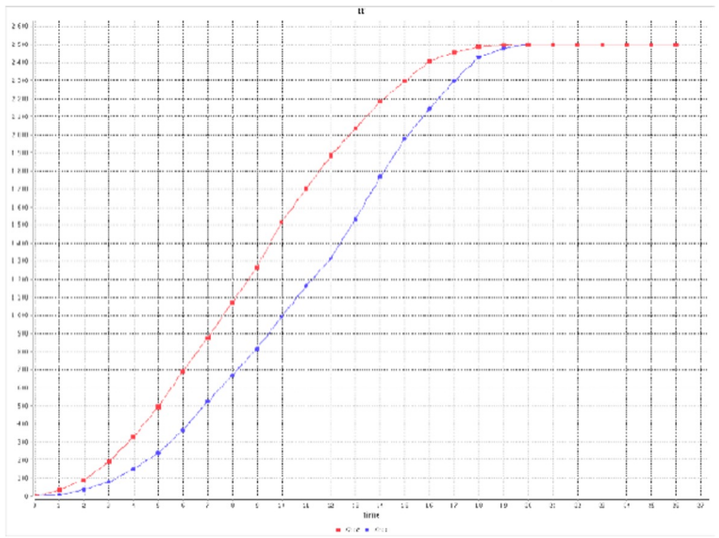
```
experiment ForestFireModelExp type: gui {

    init {
        create simulation ;
    }

    output {
        display d type: opengl{
                grid parcel border: #black;
            }
        }
    }
```

- Remark:
  - all the simulations are similar when they share the same random number generator seed (default behavior).

```
experiment MultiForestFireModelExp type: gui {
    init {
        create simulation ;
        create simulation with:[seed::2];
    }
}
```

Let’s manipulate all the simulations.

- In the experiment, the keyword simulations refers to the list of simulations.



- So we can plot in a single chart, the number of parcels on fire for each simulation!

- What is defined in the output block will be duplicated for each simulation, but what is defined in the permanent block are displays that are created only once, whatever the number of simulations.

```
experiment ForestFireModelExp type: gui {
 // …
        permanent {
                display c {
                        chart "ff" type: series {
                                loop simu over: simulations {
                                        data "fire"+int(simu) value: simu.parcel count(each.color =
#red);
                                }
                        }
                }
        }
}
```

Reflex in simulations: save data

- As any other species, simulations can have reflexes.
- In gui experiment, reflexes are executed after each step. In batch experiment, they are called after the end of each set of replications (cf. later).
- Modeler uses the save statement, to save data into a file:

```
experiment saveData type: gui {
	init {
		write "init file";
		save ["simulation","seed","cycle","#parcelOnFire"]
		to: "file_result.csv" format: "csv" rewrite: true header: false;
	}

	reflex saveDataFile {
		ask simulations {
			save [self.name,self.seed,self.cycle,self.parcel count(each.color = #red)]
		to: “file_result.csv" format: "csv" rewrite: false ;
		}

	}

	output {
		display d {
			grid parcel lines: #black;
		}
	}
}
```

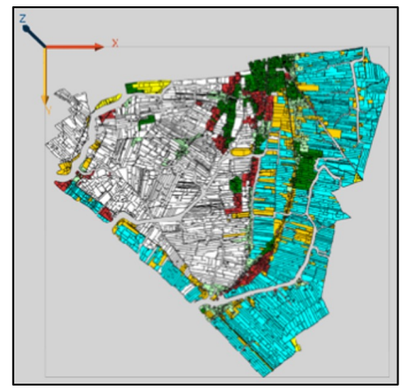
Notes about simulation visualisations

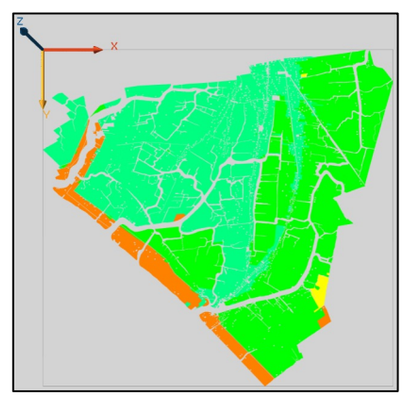
The way to display agents of a species is defined in the species in aspect block(s).
A species can have several aspects.

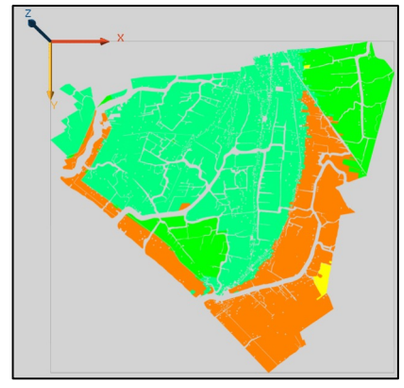
```
species parcel {
   …
        aspect land_use
        {
            draw shape color: color_map[my_land_use.lu_code] border: #black;
        }

        aspect salinity
        {
            draw shape color: hsb(0.4 - 0.4 * (min([1.0, (max([0.0, current_salinity - min_salinity])) / max_salinity])), 1.0, 1.0);
        }

        aspect salinity2010
        {
            draw shape color: hsb(0.4 - 0.4 * (min([1.0, (max([0.0, salinity_years[2010] - min_salinity])) / max_salinity])), 1.0, 1.0);
        }

        aspect land_use2010
        {
            draw shape color: color_map[lu_years[2010]] border: #black;
        }
}
```

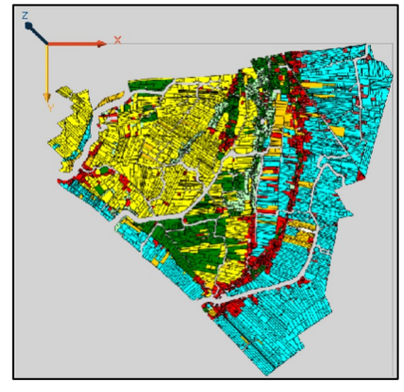
The way to display agents of a species is defined in the species in aspect block(s).
A species can have several aspects.

```
species parcel {
   …
        aspect land_use
        {
            draw shape color: color_map[my_land_use.lu_code] border: #black;
        }

        aspect salinity
        {
            draw shape color: hsb(0.4 - 0.4 * (min([1.0, (max([0.0, current_salinity - min_salinity])) / max_salinity])), 1.0, 1.0);
        }

        aspect salinity2010
        {
            draw shape color: hsb(0.4 - 0.4 * (min([1.0, (max([0.0, salinity_years[2010] - min_salinity])) / max_salinity])), 1.0, 1.0);
        }

        aspect land_use2010
        {
            draw shape color: color_map[lu_years[2010]] border: #black;
        }
}
```







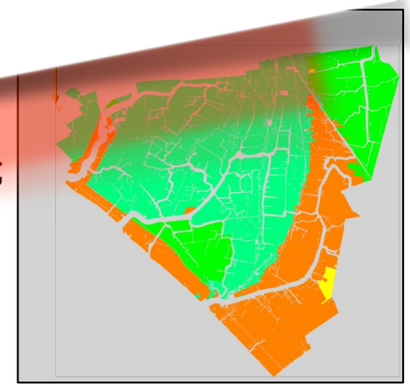


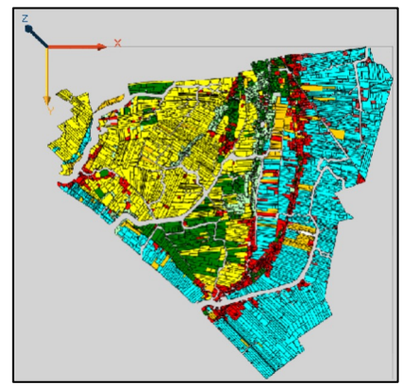
The way to display agents of a species is defined in the species in aspect block(s).
A species can have several aspects.

```
species parcel {
    …
        aspect land_use
        {
            draw shape color: color_map[my_land_use.lu_code] bord
        }

        aspe

        {
            draw shape color: color_map[lu_years[2010]] border: #black;
        }
}
```

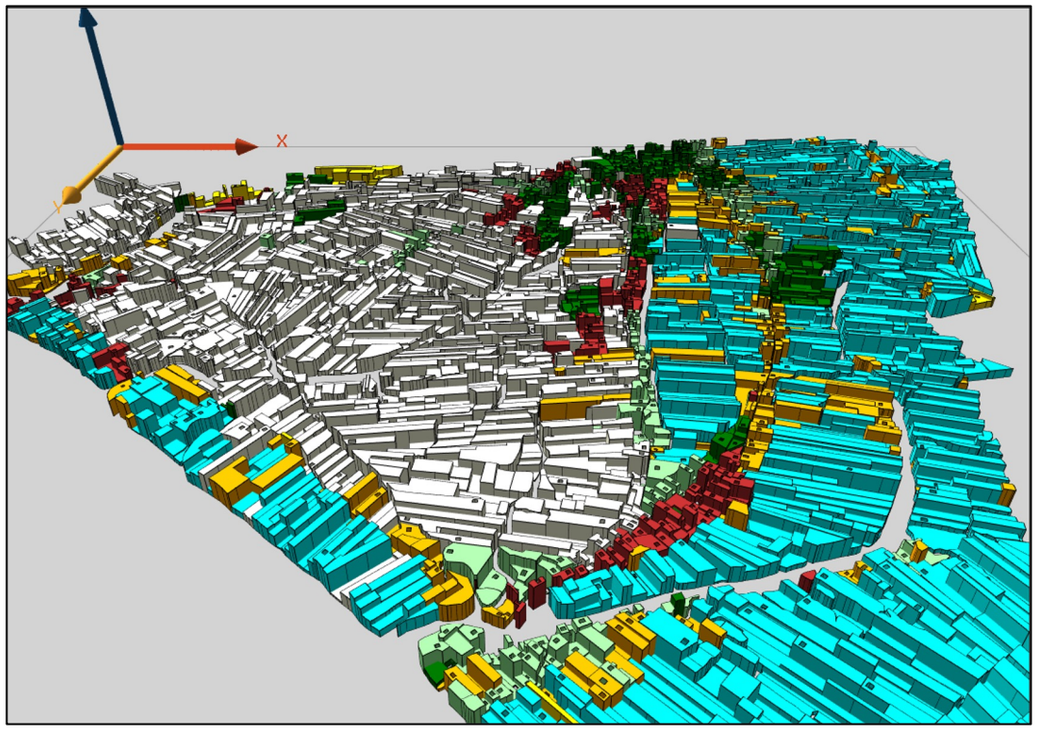
Warning:

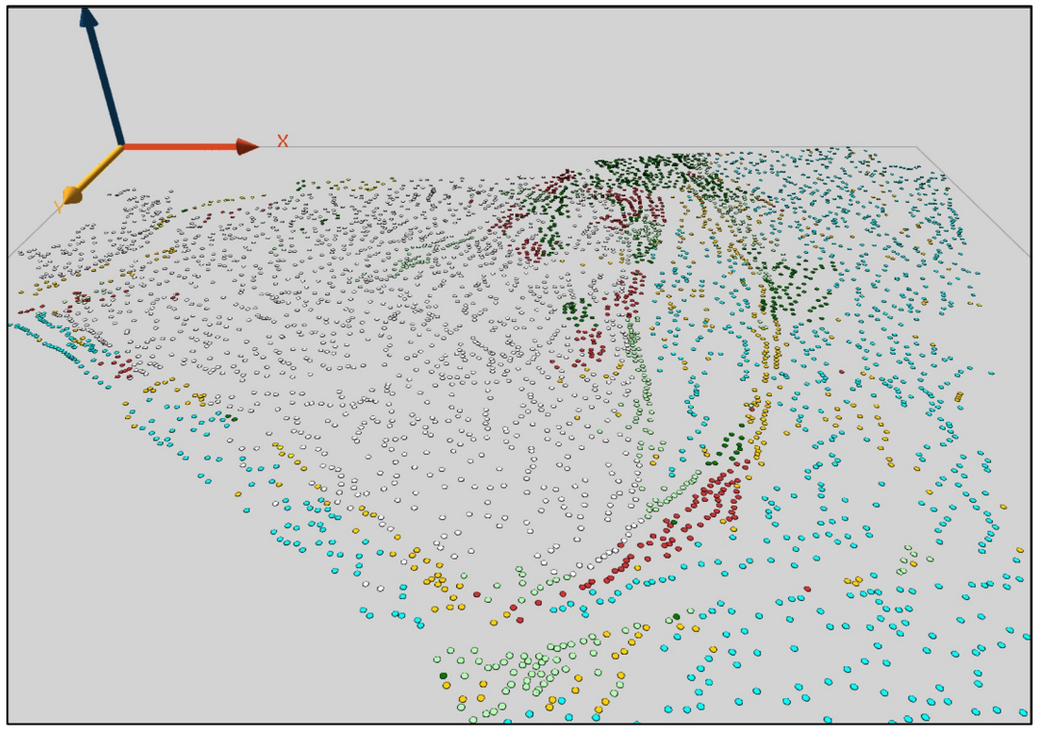
the way agents are displayed can be totally different to their actual shape or location.

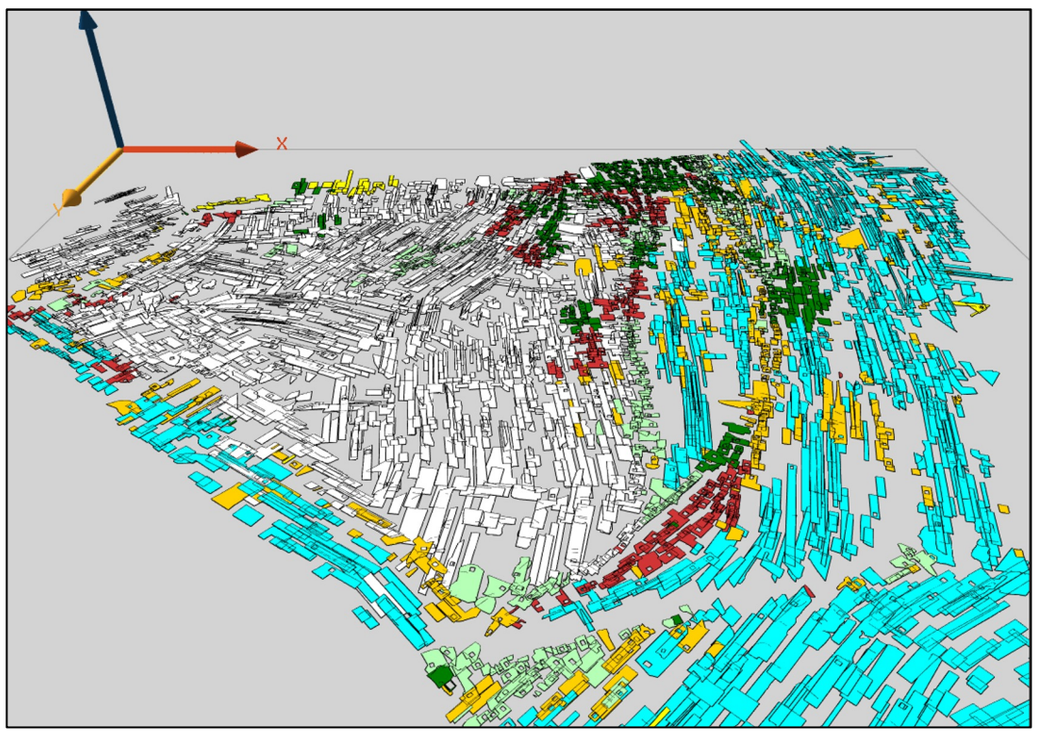


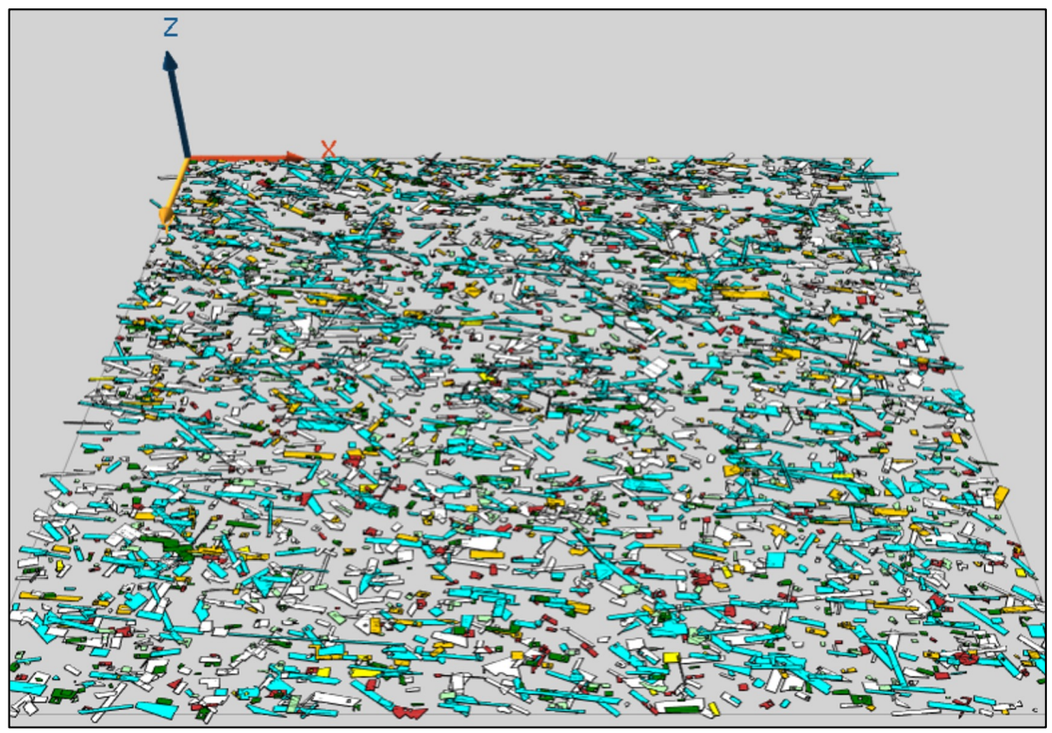


WARNING: the way agents are displayed can be totally different to their actual shape or location.

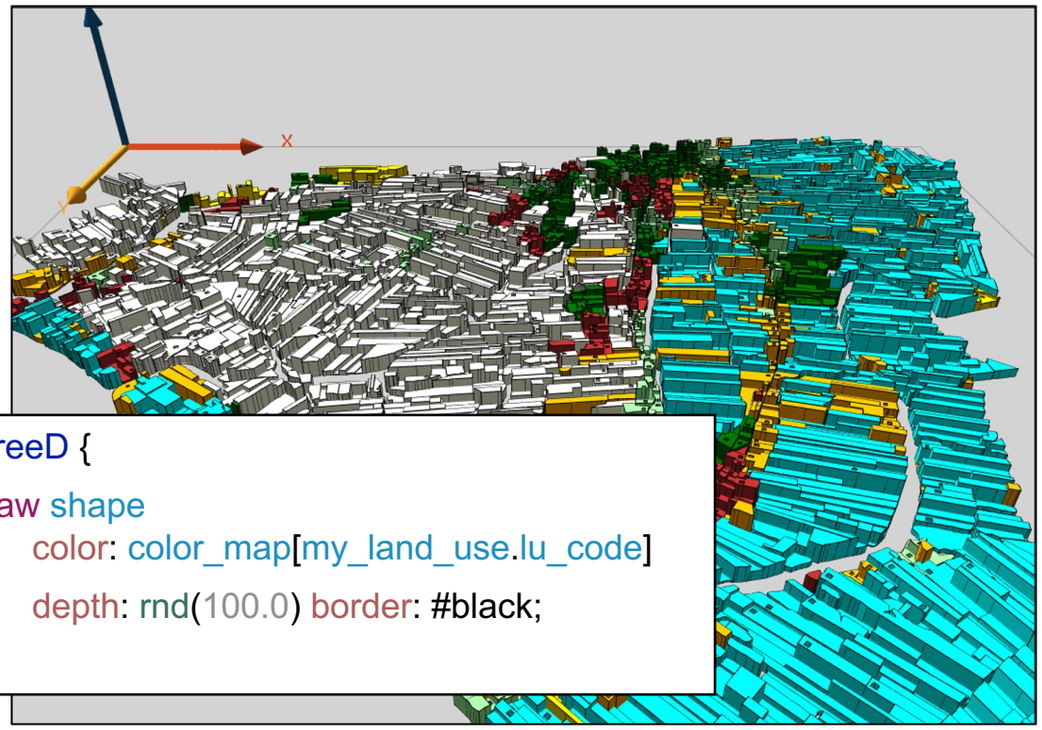




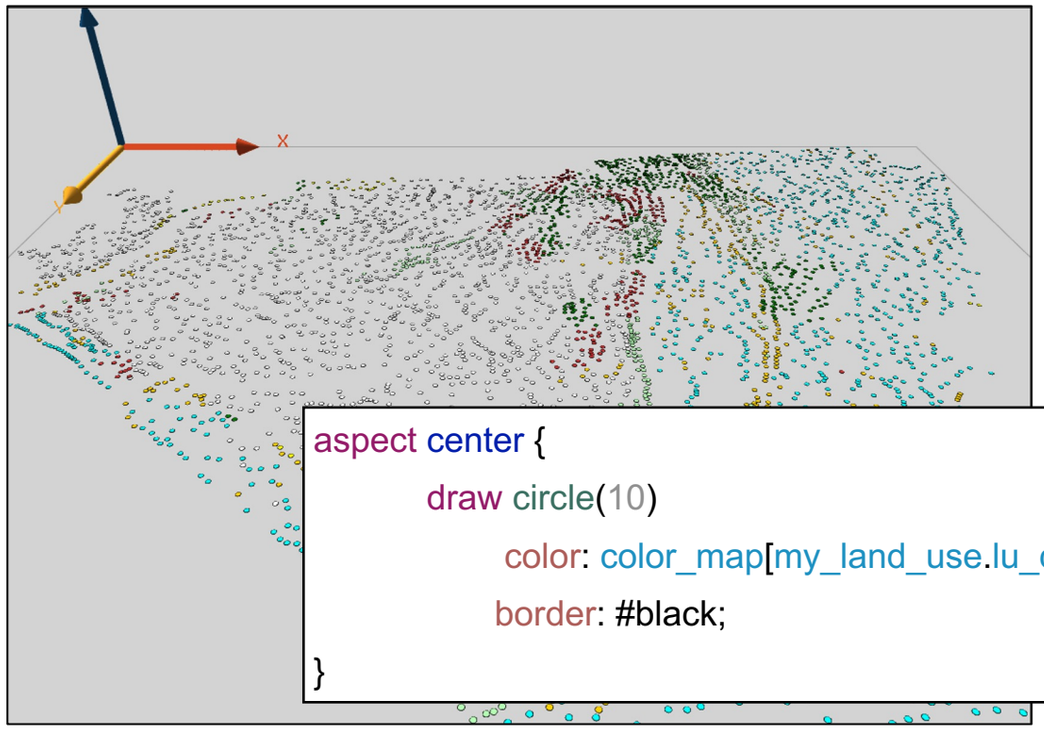




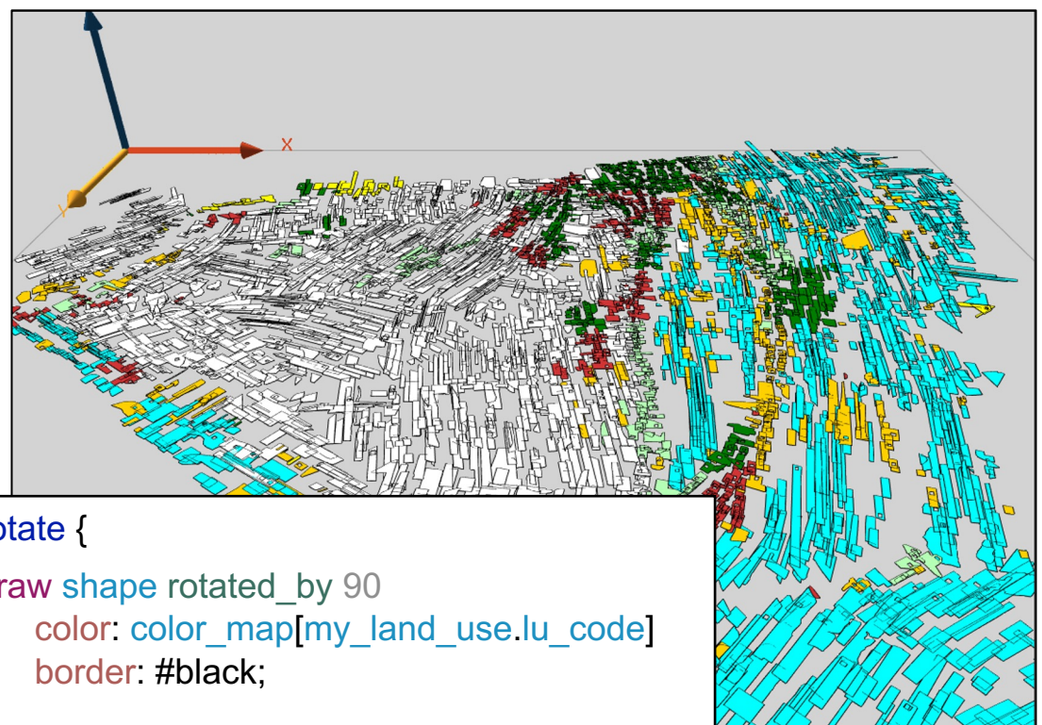
WARNING: the way agents are displayed can be totally different to their actual shape or location.



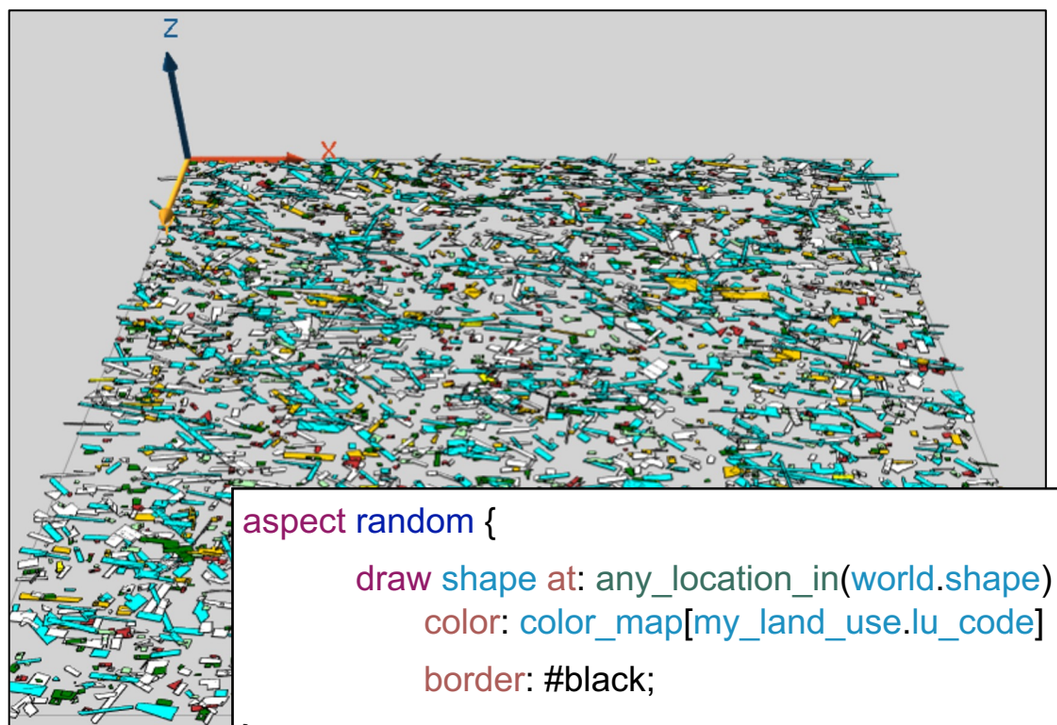
```
aspect threeD {
        draw shape
            color: color_map[my_land_use.lu_code]
            depth: rnd(100.0) border: #black;
}
```



```
aspect center {
        draw circle(10)
            color: color_map[my_land_use.lu_code]
            border: #black;
}
```



```
aspect rotate {
        draw shape rotated_by 90
            color: color_map[my_land_use.lu_code]
            border: #black;
}
```



```
aspect random {
        draw shape at: any_location_in(world.shape)
            color: color_map[my_land_use.lu_code]
            border: #black;
}
```

In GUI experiments, species are displayed in layers. Additional information (text, images) can be added.

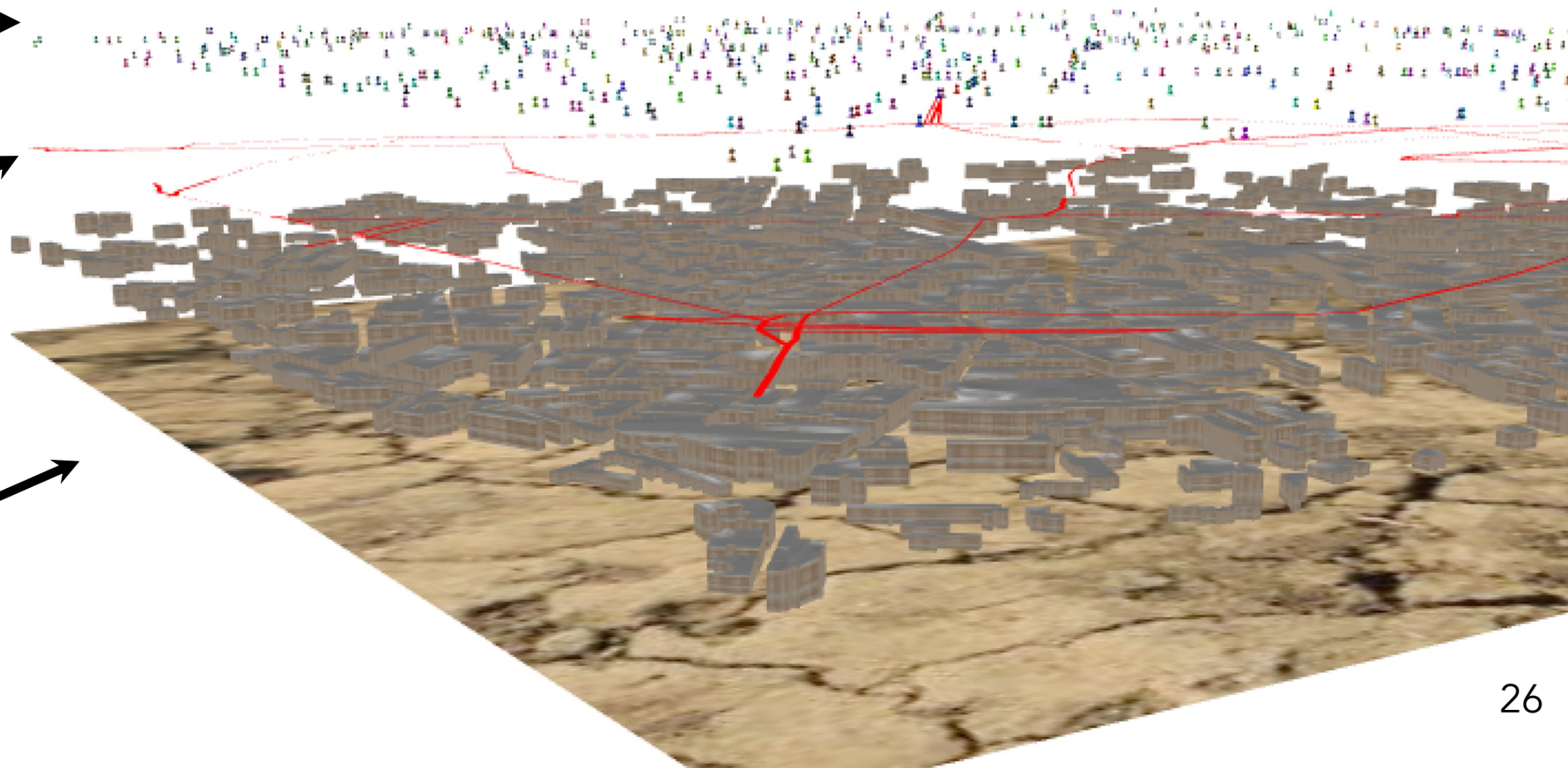
```
display map type: opengl {
        image "../includes/soil.jpg" refresh: false;
        species building aspect: geom refresh: false;
        species road aspect: geom ;
        species inhabitant aspect: threeD;
}
```

Species inhabitant

Species road

Species building

Background image



Additional information (text, images) can be added using the graphics layer statement.

- A graphics is a layer in which modeler can draw whatever it wants (without introducing new species).

```
experiment traffic type: gui {
        output {
                display map type: opengl {

                        image "../includes/soil.jpg" refresh: false;
                        species building aspect: geom refresh: false;
                        species road aspect: geom ;
                        species inhabitant aspect: threeD;

                        graphics g {
                                draw "" + current_date font: font("Arial", 32, #bold) at: {50,100} color: #black;
                        }
                }
        }
}
```

In GUI Experiment, we can define **monitor** (to monitor the current value of a variable) and various kinds of **chart** (series, pies, histogram...).

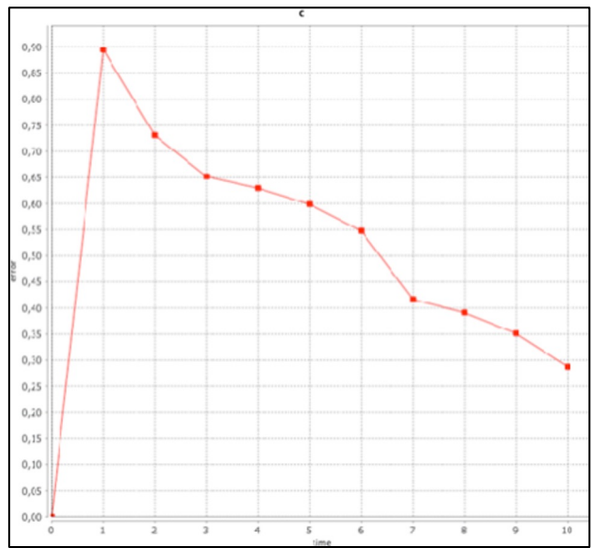
Variable to monitor.

Monitor

```
output {
  monitor "Error" value: error;
}
```

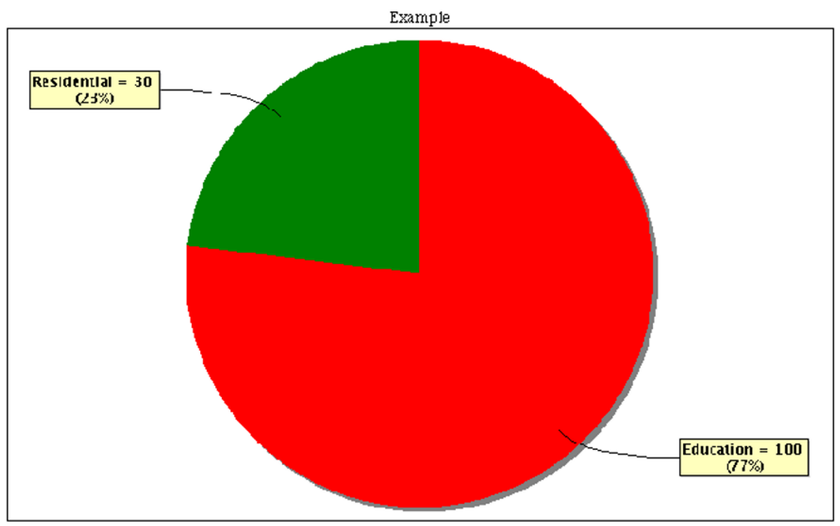
Series

```
output {
  display series {
    chart "c" type: series {
                data value: error legend: "error" color: #red;
              }
  }
}
```



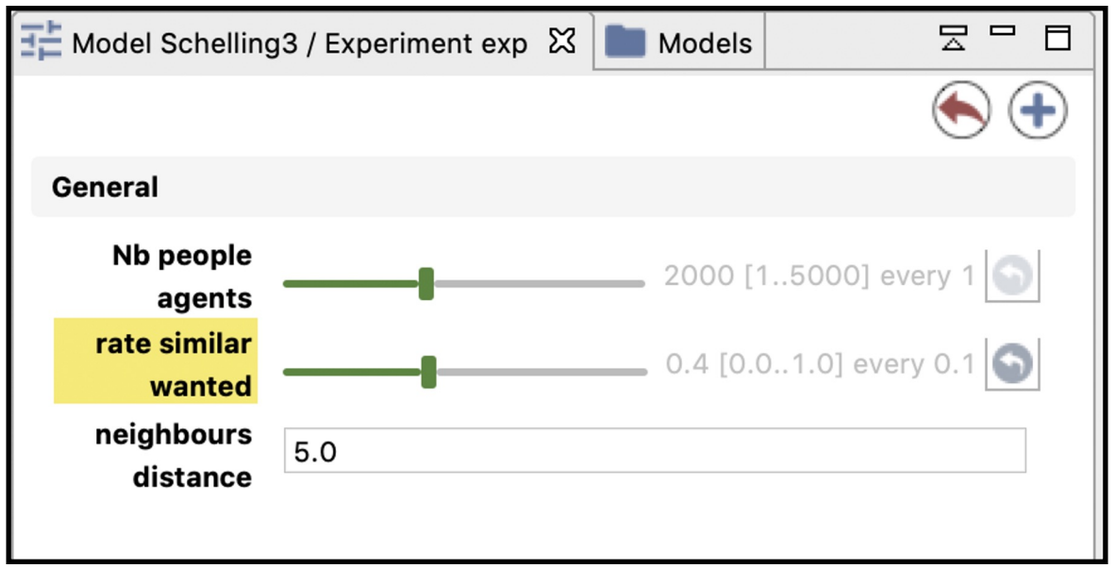
Pie

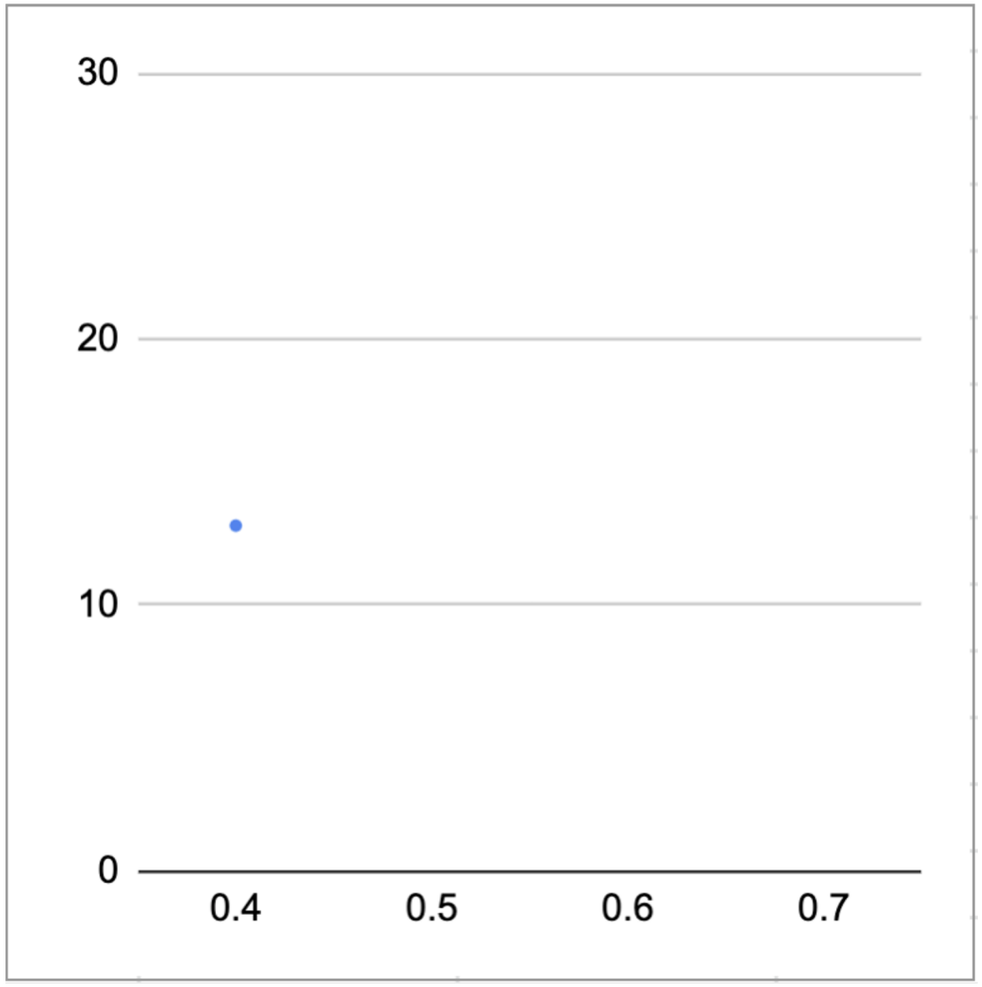
```
output {
  display pies {
              chart "Example" type: pie {
                data value: valueA legend: "ValueA" color: #red;
                data value: valueB legend: "ValueB" color: #green;

              }
  }
}
```



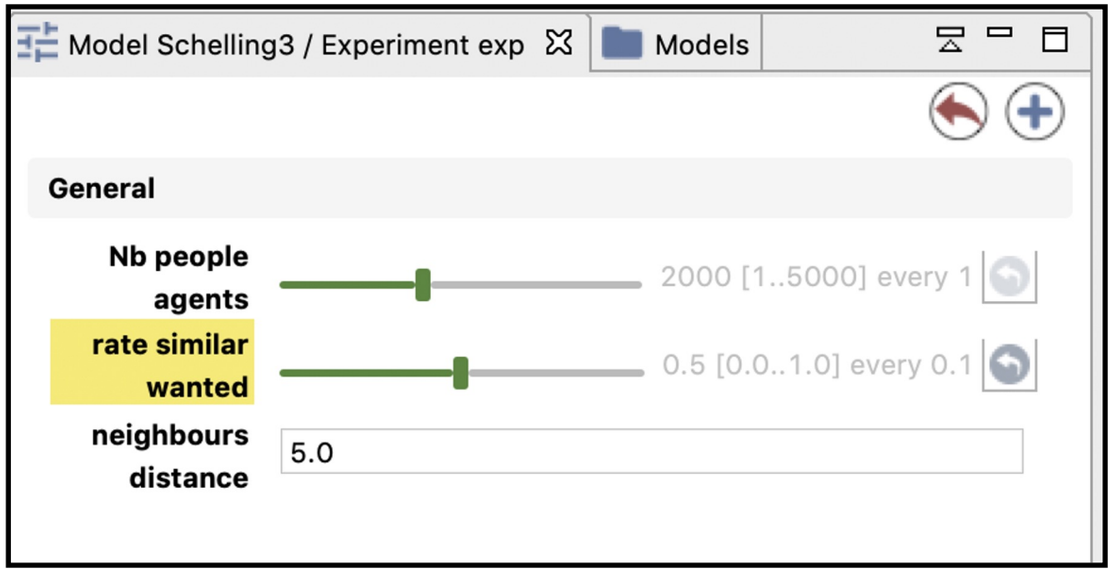
Exploration methods

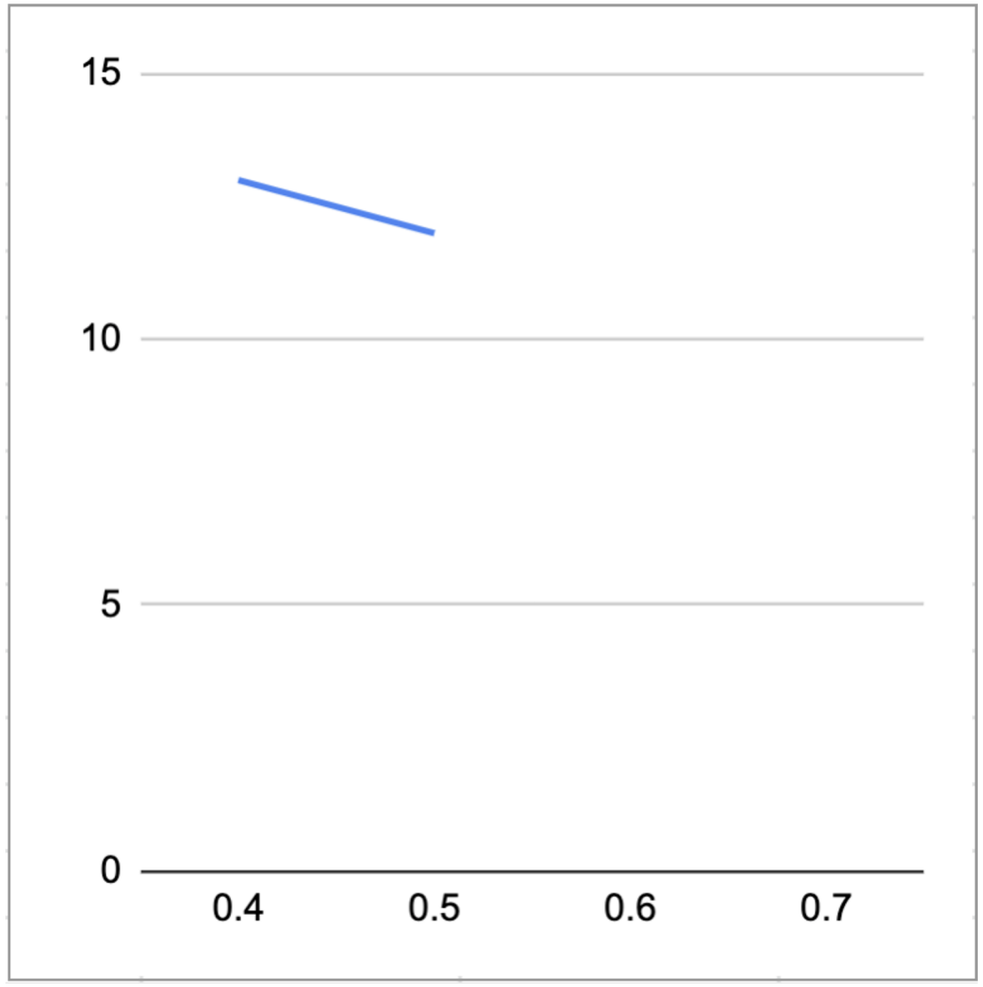
GUI experiments allow to test various parameters values and observe the outputs... which can be long and fastidious at hand to explore a model with several parameters...



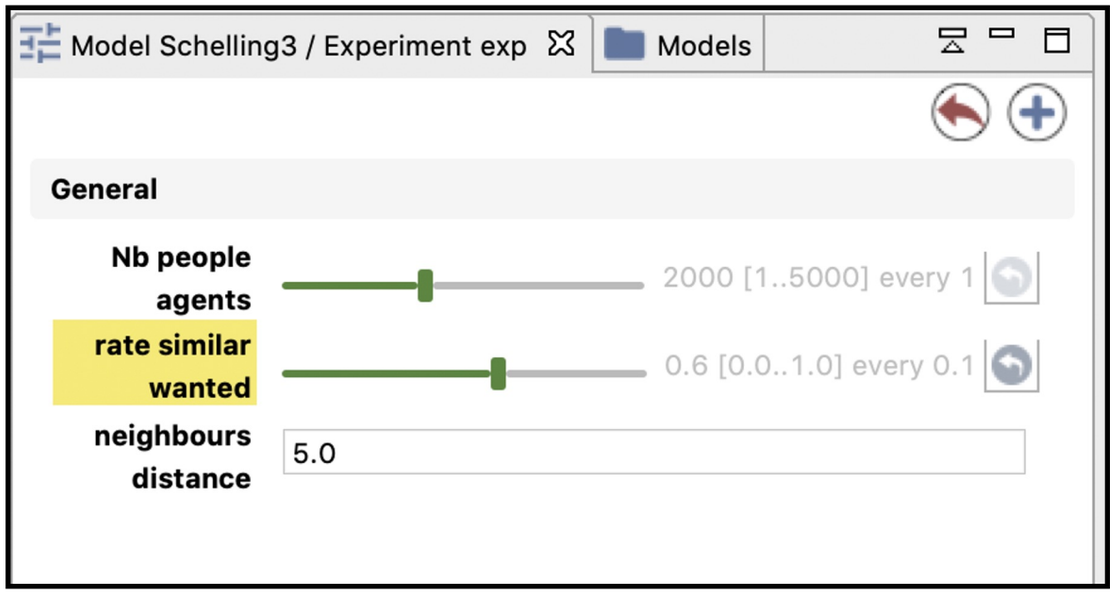


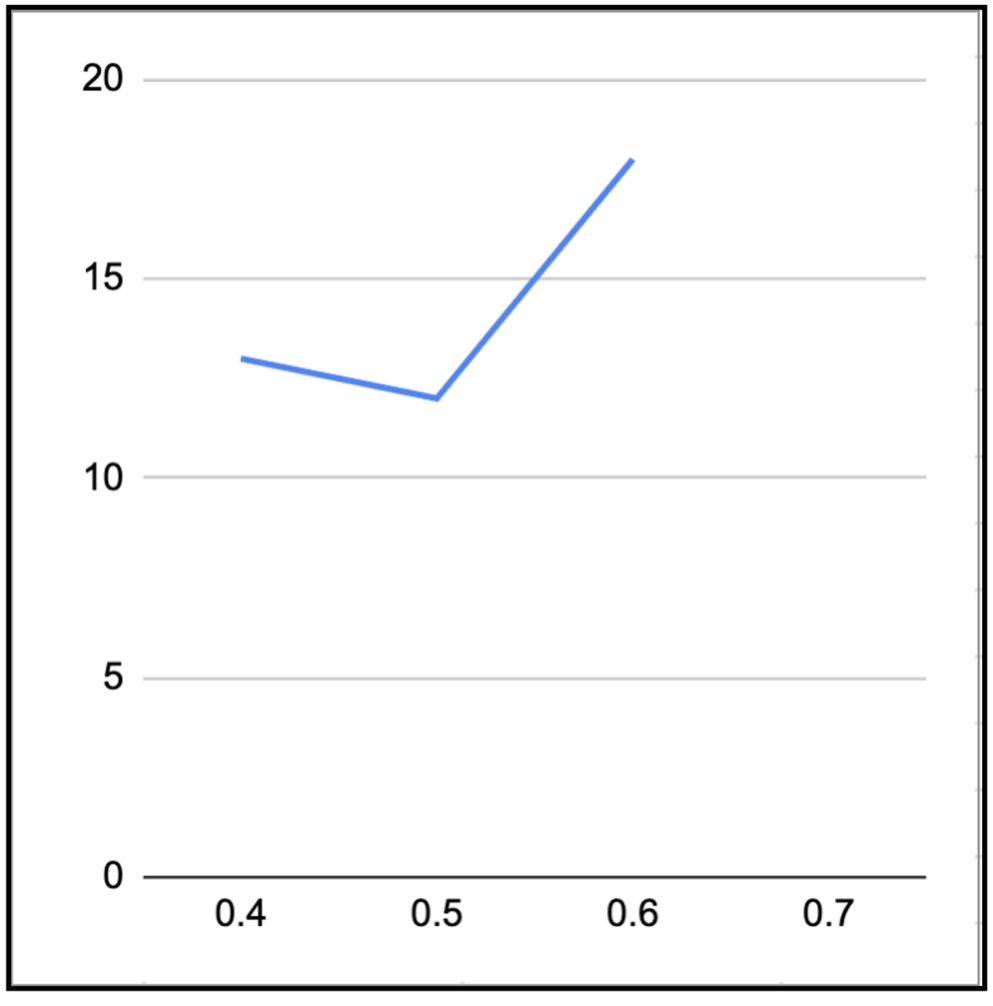
GUI experiments allow to test various parameters values and observe the outputs... which can be long and fastidious at hand to explore a model with several parameters...



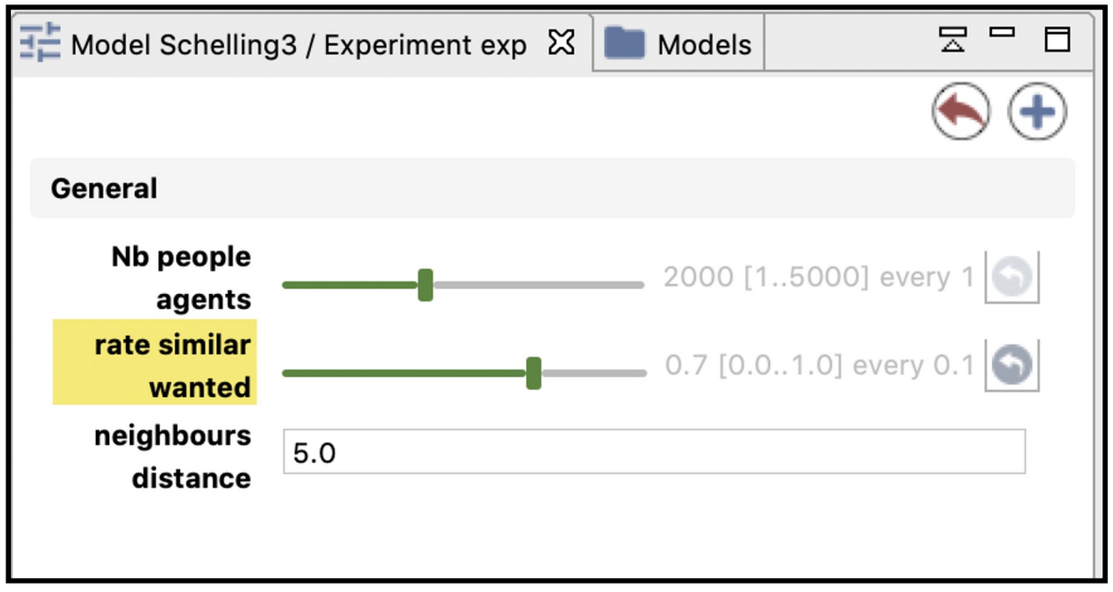


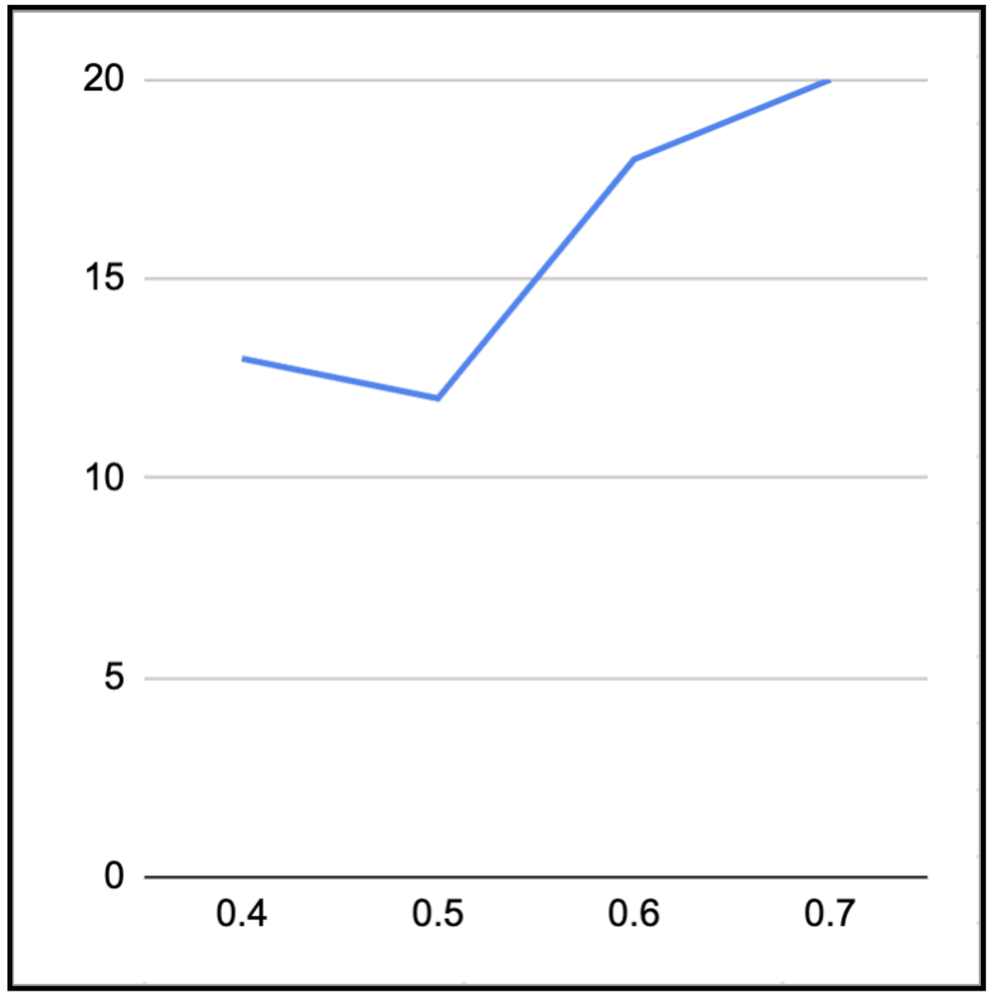
GUI experiments allow to test various parameters values and observe the outputs... which can be long and fastidious at hand to explore a model with several parameters...



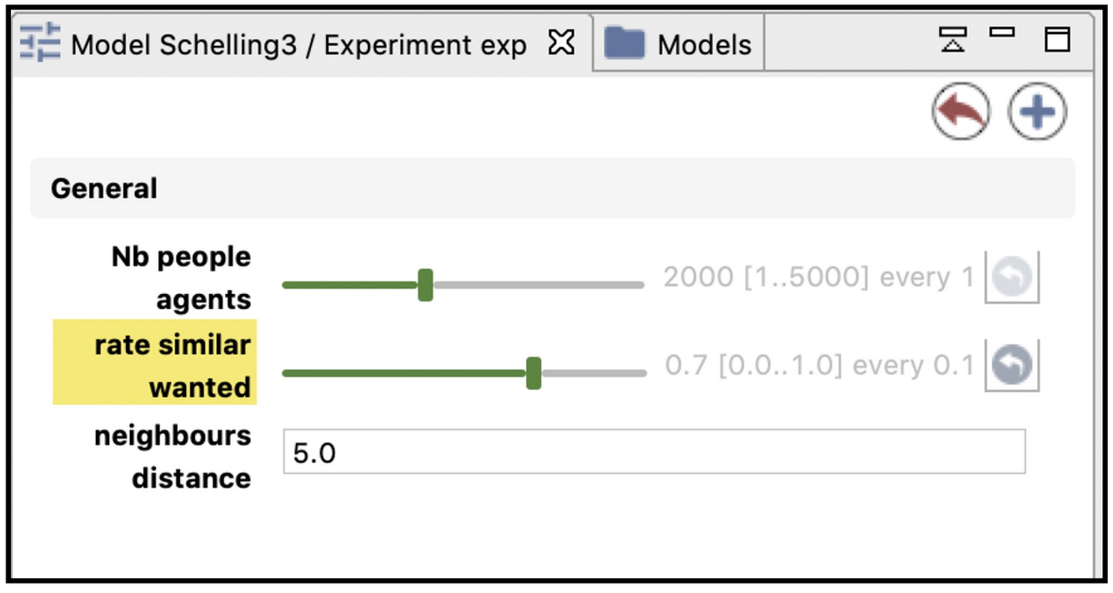


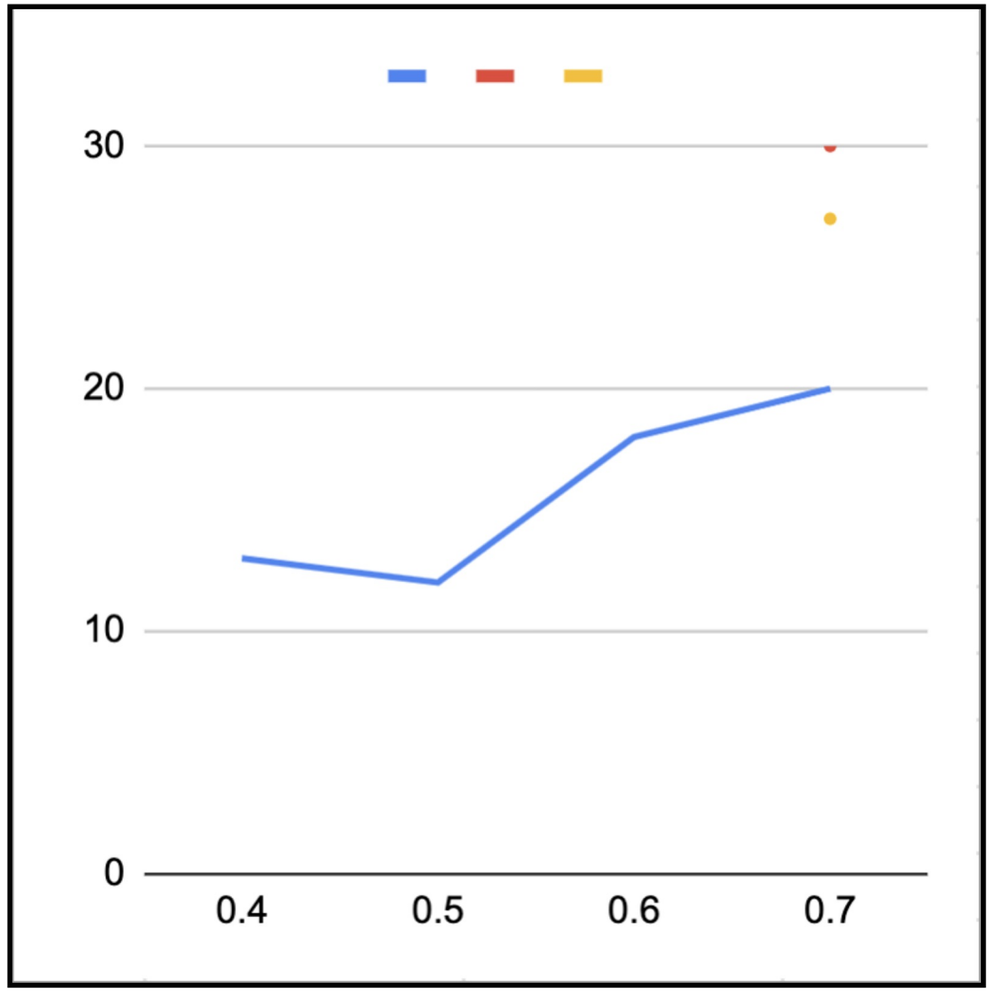
GUI experiments allow to test various parameters values and observe the outputs... which can be long and fastidious at hand to explore a model with several parameters...



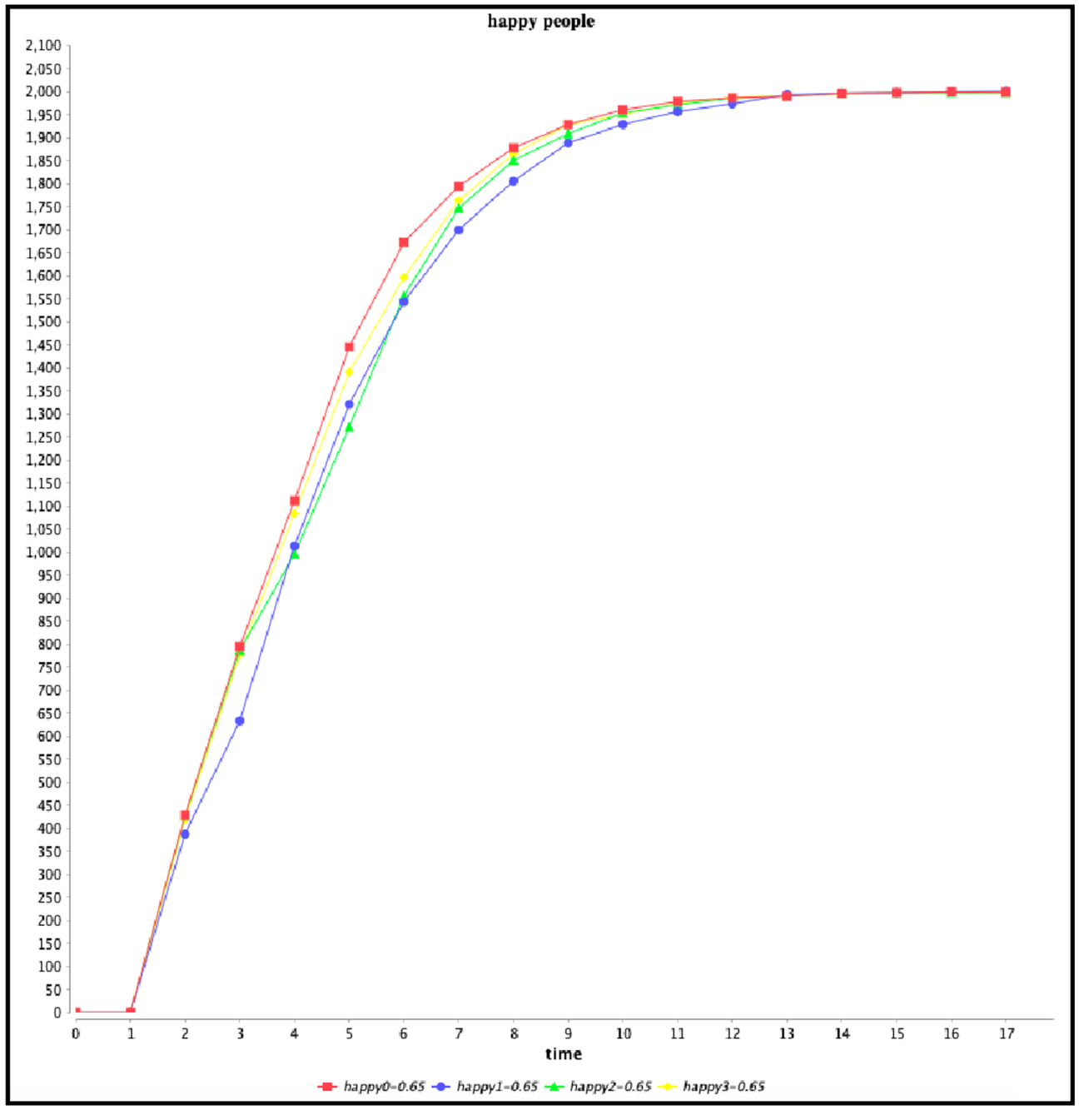


But even for the same value of a parameter, results can be different! (due to stochasticity)





In addition for the same values of parameters several replications should be run, due to randomness in the model.



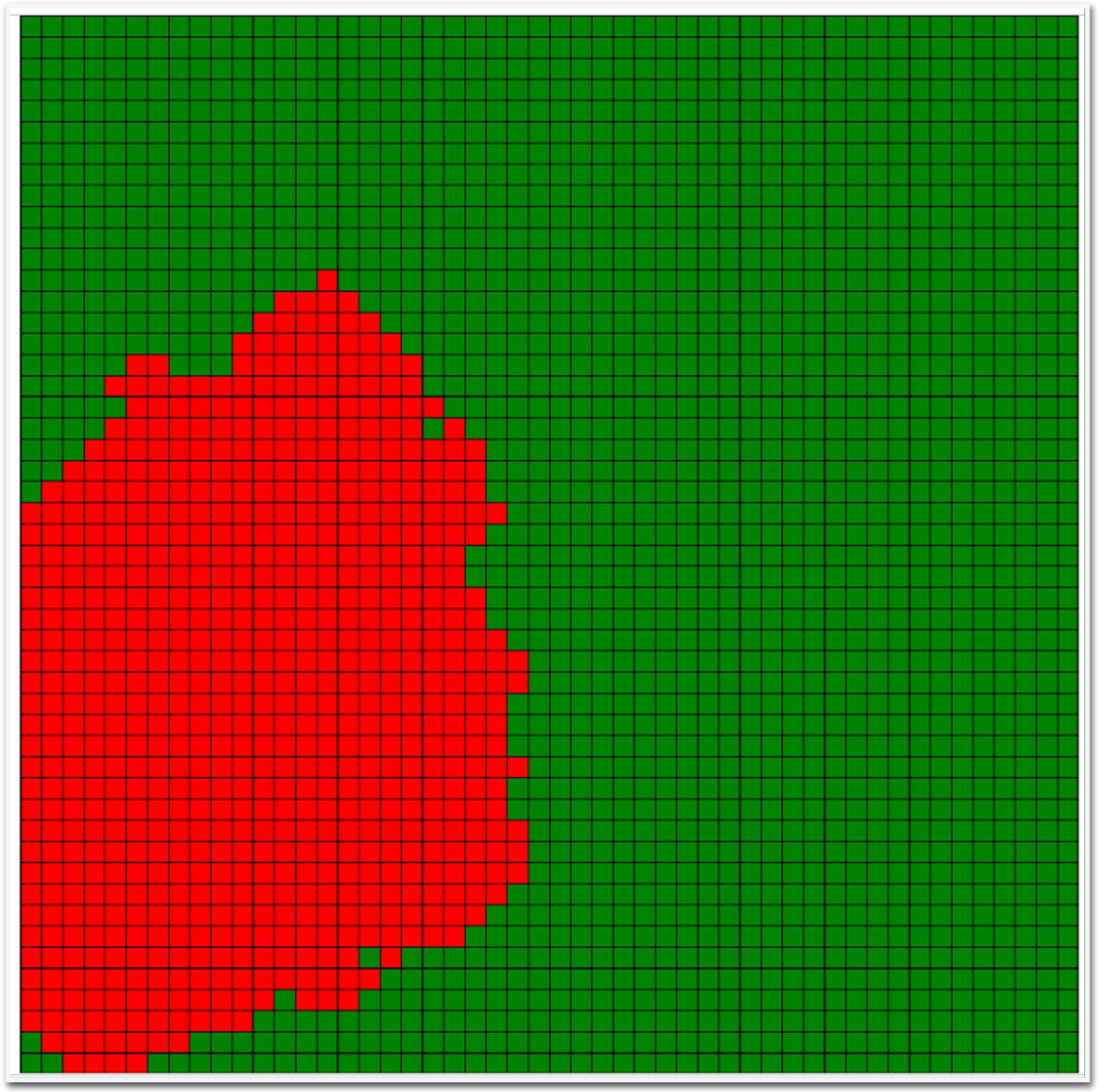
Example of the influence of stochasticity and scheduling on results: the Forest Fire model.

The forest is represented by a grid of forest cells (e.g. 50x50) of color green.

Initially 1 forest cell is set on fire (its color becomes red).

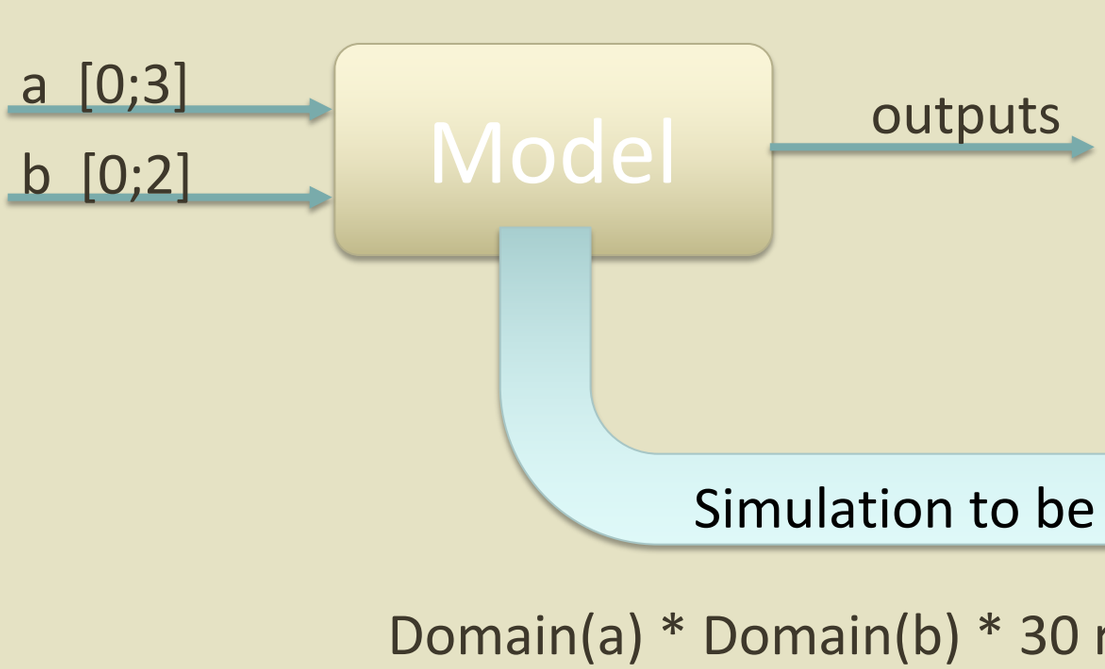
At each step, a red (on fire) forest cell diffuses the fire to all of its neighbors.

- Observe the effect of stochasticity (plot the number of red forest cell)



An exhaustive exploration is resources consuming (time and memory)

- **Run many and many simulations**
  - Store huge amount of data
- **Analyze important indicators**
  - Mean
  - Variance
  - Standard deviation



a	b
0	0
0	1
0	2
1	0
1	1
1	2
2	0
2	1
2	2
3	0
3	1
3	2

360 simulations are required for this simple example

if in our model we have just 6 parameters with 11 possible values, it makes 1 771 561 possible combinations (to be multiplied by the number of replications).

The batch mode allows to automatically execute parameter space exploration.

- Exploring the parameters space of a model can be used to:
- Correct the model,
- Do sensibility analysis,
- Calibrate it,
- Discover emerging properties,
- …

A necessary step!

Steps in an experiment design.

- General method:

  - Define an initial situation and a stop criterion (of 1 simulation)
  - Select the set of parameters to explore and the indicators
  - Define a strategy of exploration
  - Execute, collect and save the data
  - Analyze the collected data

Get results from exploration: save into file

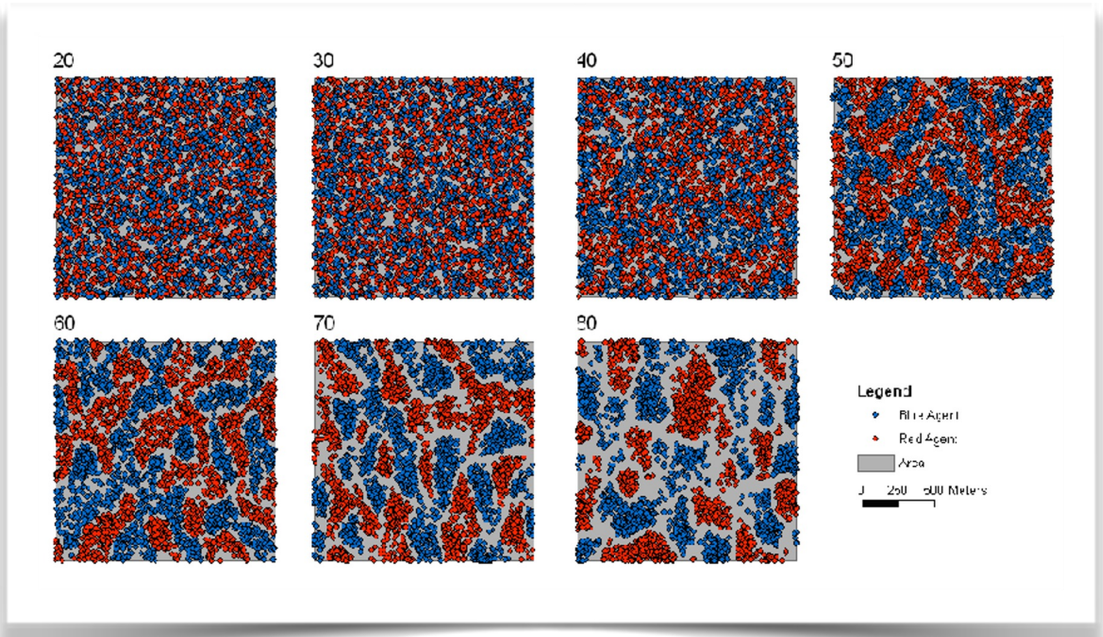
- Using an exploration method, requires to:
- use a batch experiment
- Define a stop condition
- Define a number of repetitions
- Choose the exploration method

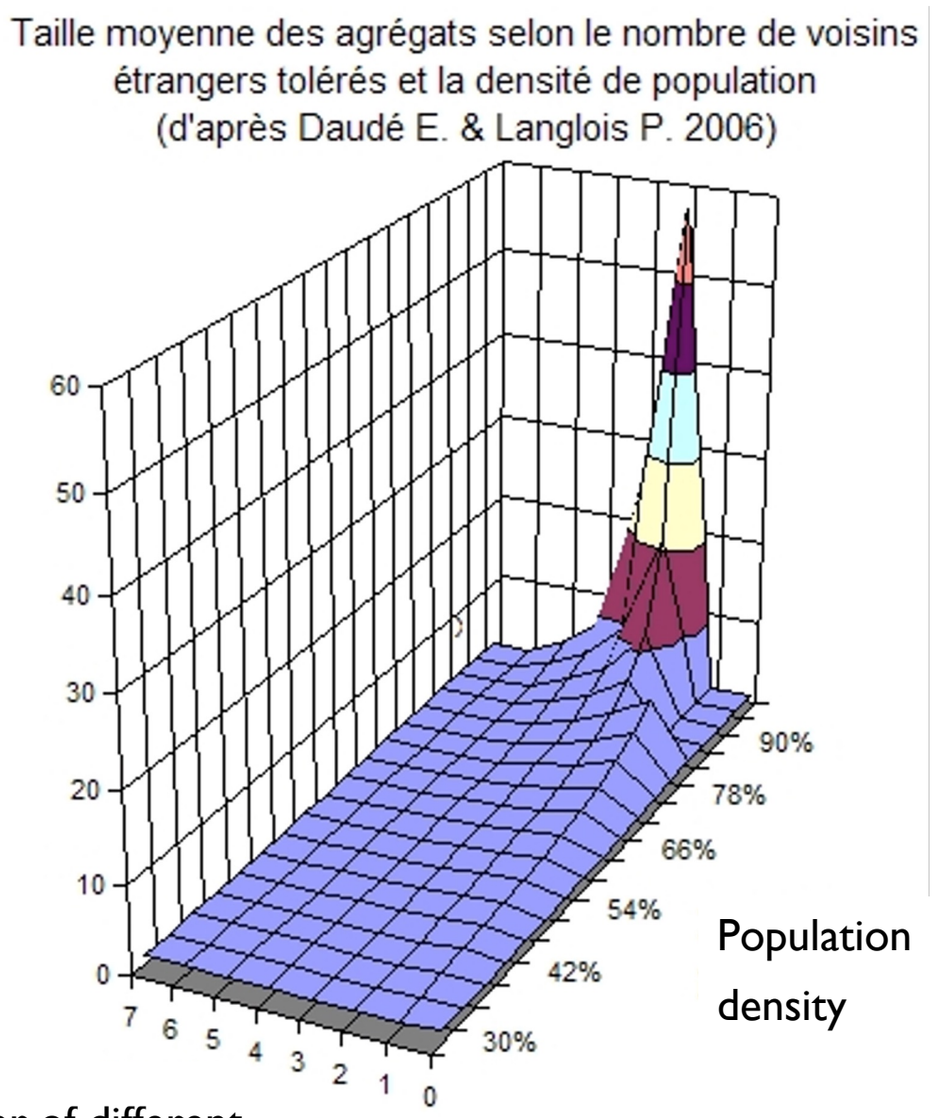
- Note: a model run in batch mode should not contain any stop condition in the model.

Explore the model behavior through the exploration of the parameters space.

- Aim: to have a deep knowledge of the behavior of the model (and in particular its sensitivity to parameters variations)
- ==> the range of values to which the simulation is very sensitive -> discretization need (step= 1, 0.1 or 0.0001 …)

- Example on the Schelling's model:





«Exhaustive» exploration of the Schelling's model

Let’s try on the Schelling model

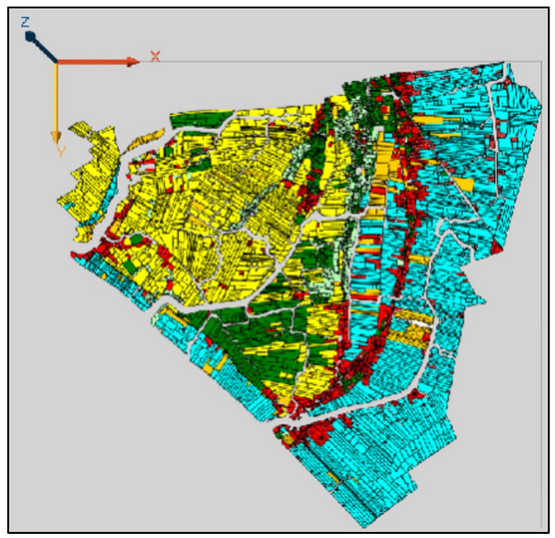
- The default method is “exploration” (previously called exhaustive) method allows to define an experiment design exploring all the combination of parameter values.

- On the Schelling model, what are the parameters ? Their possible values ?

- Run exploration !

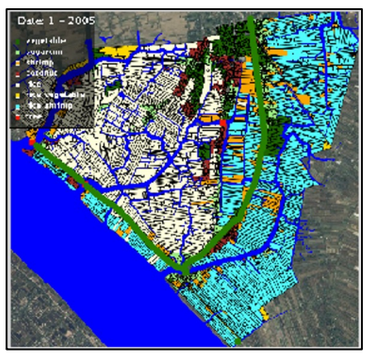
The calibration allows to find parameters values making the simulation results fitting to real data.

- Objective:
- What are the values of parameters:
  - that allow to obtain a «realist» model ?
  - that minimize the difference between real and simulated data (variable **error** in the model) ?



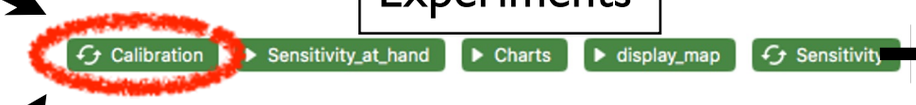
error = difference between simulated data and real data

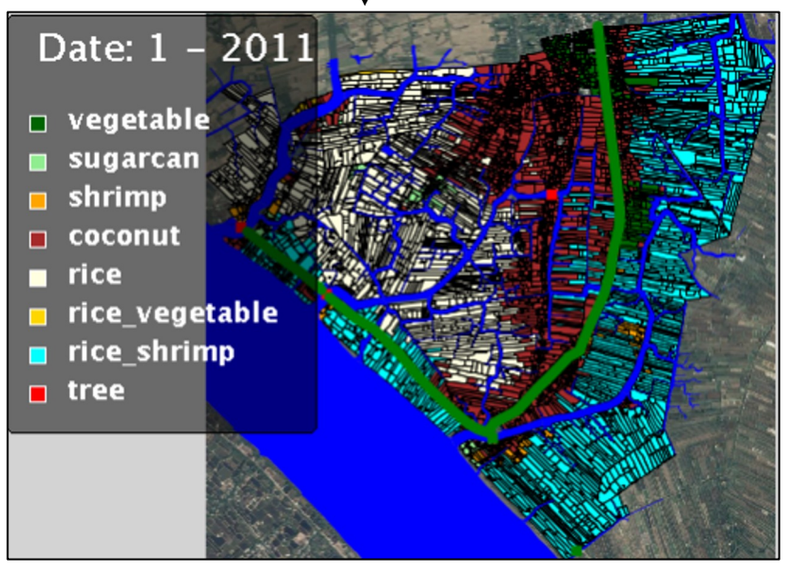
weight_profit = ??
weight_risk = ??
weight_implementation = ??
weight_suitability = ??
weight_neighborhood. = ??
probability_changing = ??



Real data in 2005

Experiments





Model exploration: exploration method

- The statement method defines the exploration type to use
- method method_name +
  - maximize : optional, expression to maximize (i.e. *fitness*)
  - minimize : optional, expression to minimize (i.e. *fitness*)

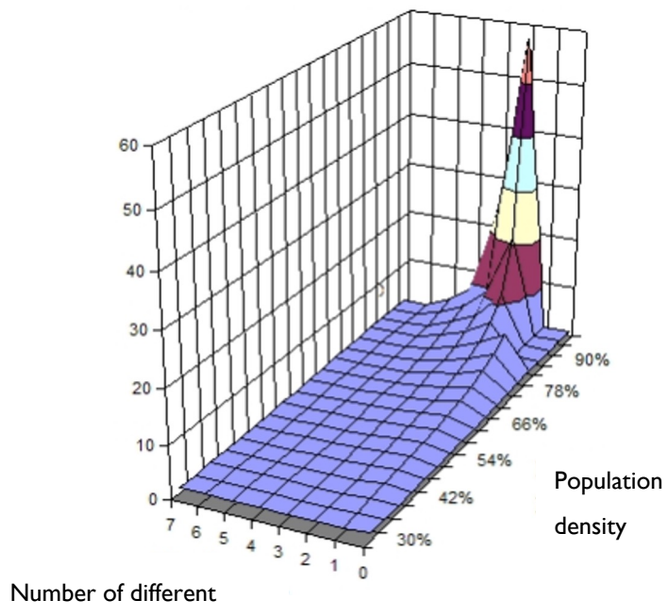
} One of the two

- Existing exploration methods
  - **exhaustive** (in older versions, now default mode)
  - **genetic** (genetic algorithm)
  - **hill_climbing**
  - **annealing** (simulated annealing)
  - **tabu** (local tabu search)
  - **reactive_tabu** (local reactive tabu search)

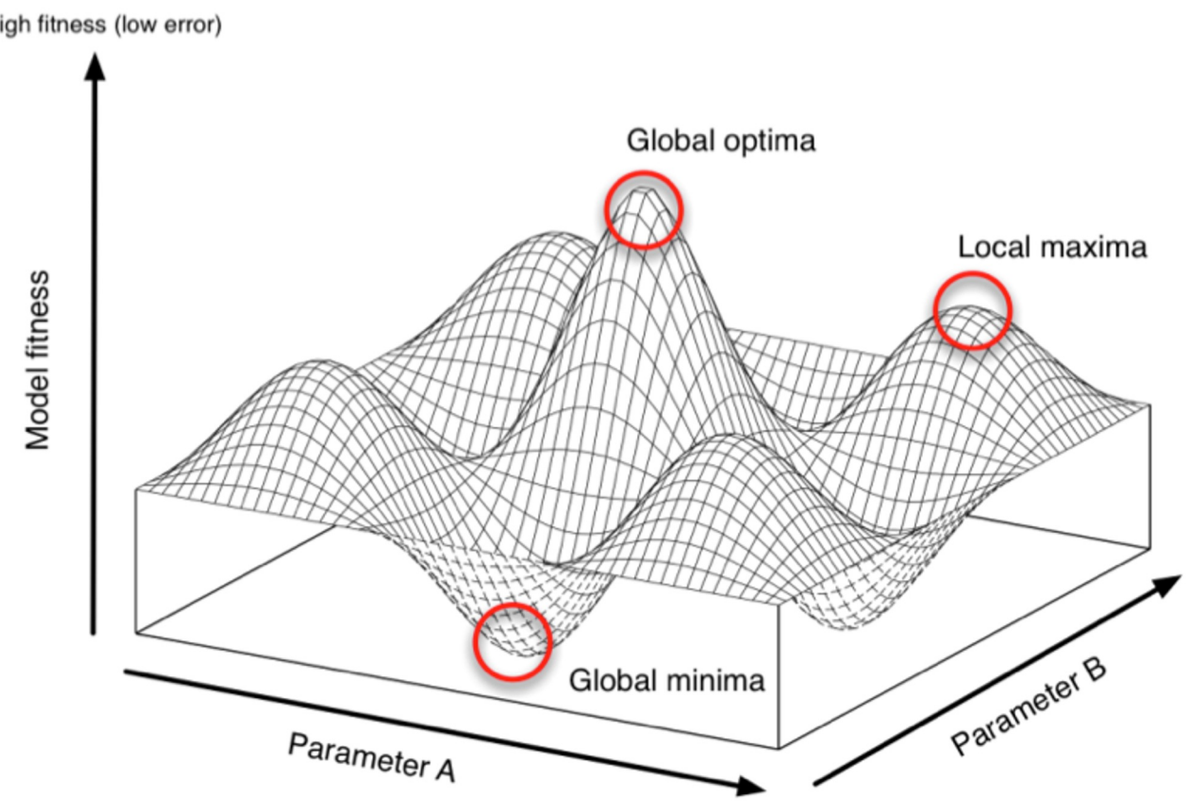
```
experiment Optimization type: batch keep_seed: true repeat: 2 until: ( date = 2010 ) {
   [parameters]
        method genetic minimize: error
         pop_dim: 5 crossover_prob: 0.7 mutation_prob: 0.1 nb_prelim_gen: 1 max_gen: 500 ;
}
```

Issues linked to the calibration.

- Find a good indicator (often named fitness) to represent the fact that a simulation is «good» (e.g. the difference between real and simulated data):



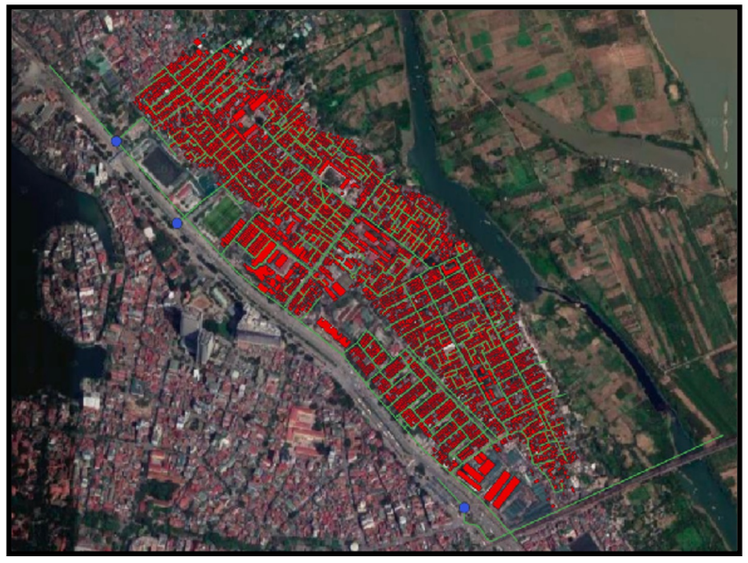
- To find the parameters closed to an optimization problem (minimize a value):
- lot of methods exist (exhaustive, Genetic algorithms, ...)



http://nickmalleson.co.uk/wp-content/uploads/2013/04/parameter_space.png

Once the model has been calibrated, we can explore the result of the simulation to various scenario.

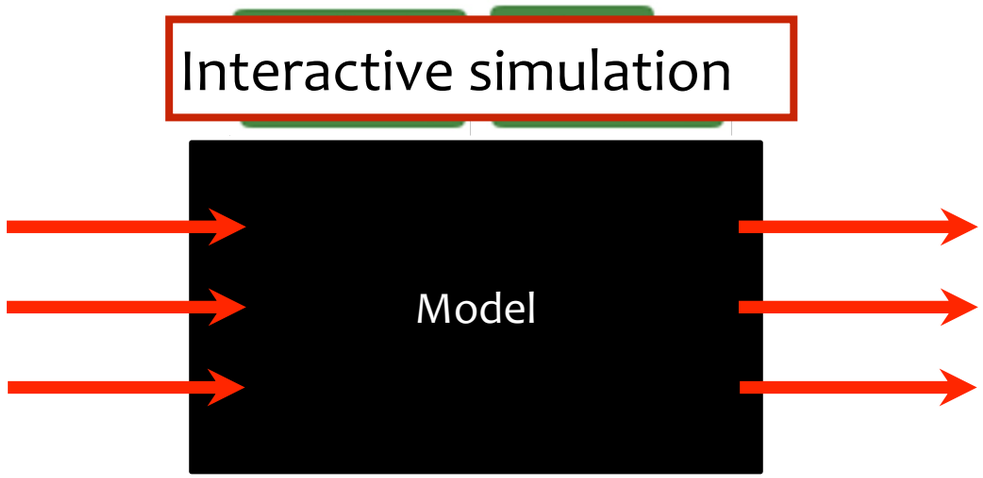
- We can define scenario that we want to explore:
  - building of a new infrastructure (new exit ?)
  - Test possible alert strategies



What-If new infrastructure are built ?

Summary

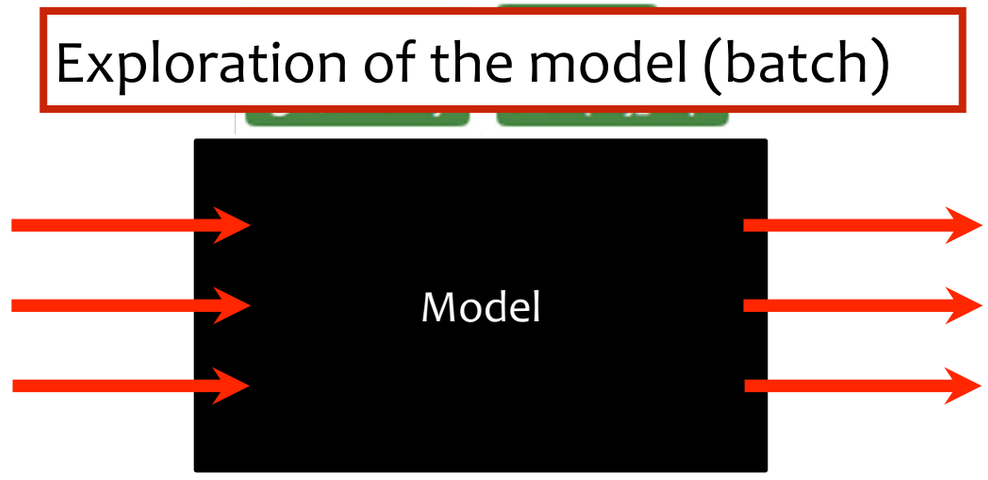
weight_profit = x1
weight_risk = x2
weight_implementation = x3
weight_suitability = x4
weight_neighborhood. = x5
probability_changing = x6



Land Use distribution = ??

Kappa indicator = ??

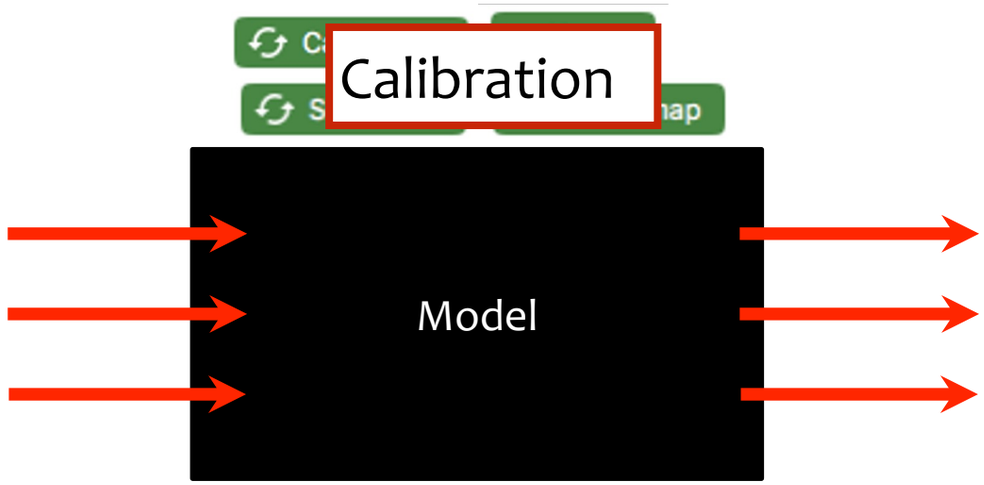
weight_profit = [x1… y1]
weight_risk = [x2… y2]
weight_implementation = [x3… y3]
weight_suitability = [x4… y4]
weight_neighborhood. = [x5… y5]
probability_changing = [x6… y6]



Land Use distribution = [??… ??]

Kappa indicator = [??… ??]

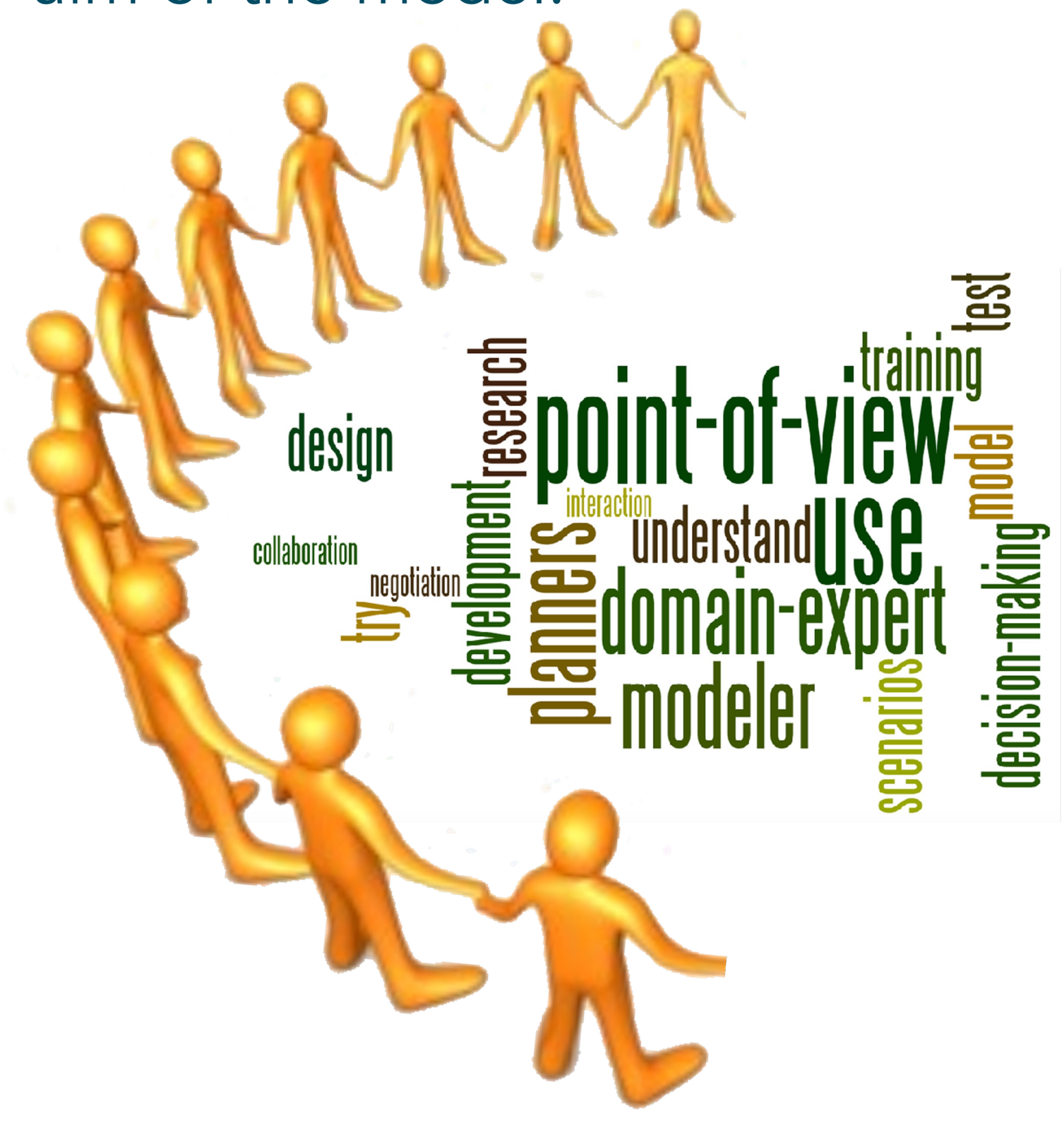
weight_profit = ??
weight_risk = ??
weight_implementation = ??
weight_suitability = ??
weight_neighborhood. = ??
probability_changing = ??

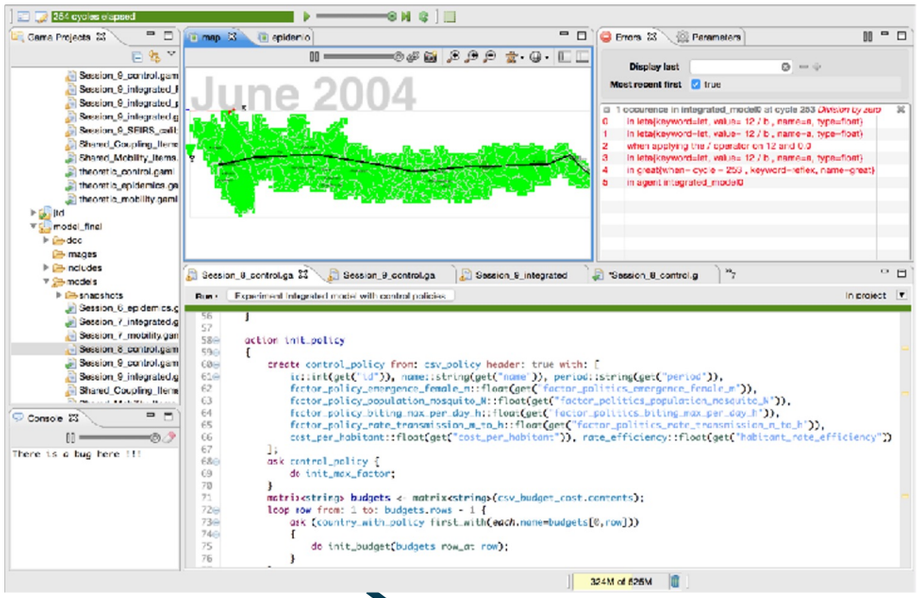


Land Use distribution = X1

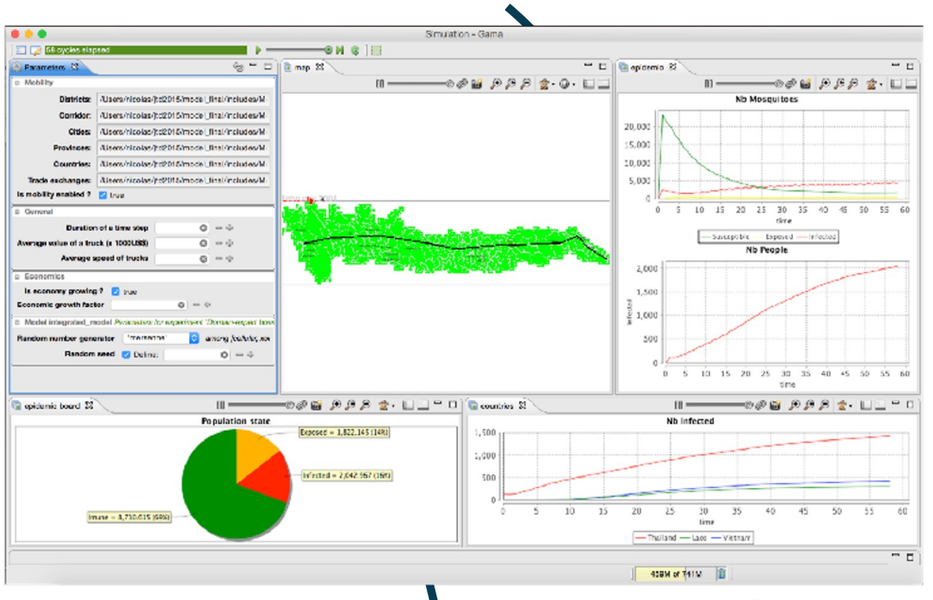
Kappa indicator = Y1

Identify experiments according to end users and the aim of the model.





Developper visualisation

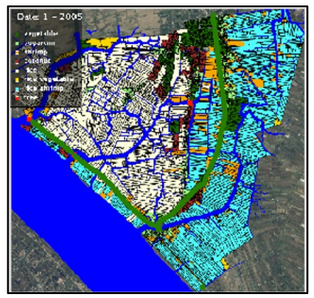


Domain-expert visualisation

Decision making data

Explore a model imply to run many and many simulations with various parameter sets

Simplified model



Collaborative/participative model

Large scale model

Interactive mode

Users interact with the simulator to select parameters and observe results

Batch mode

An algorithm interacts with the simulator to conduct the exploration (select parameter sets, execute them, ...)