

# Simulation Explorer

When opening this window, The *Model Editor* and the *Parameters Editor* windows become in read-only mode.

In the course of each simulation, each actor try to get the higher value for its aim, defined as  $aim(a, s) = (1 - abs(GI(a))) * satisfaction(a, s) + GI(a) * (influence(a, s) - influence(a, a, s))$

where

$s = (s_{r1}, \dots, s_{rn})$  is any state of the organisation

$satisfaction(a, s) = \sum_{c \in A} \sum_{r \in R} solidarity(a, c) * stake(c, r) * effect_r(c, s_r)$ , ce qu'il reçoit

$influence(a, b, s) = \sum_{r \in R; a \text{ controls } r} \sum_{b \in A} satisfaction(b, s)$ , ce qu'il donne

## **Initial states** panel:

For each relation, select the initial value and the lower and upper bounds to be used for the simulation.

## **Actors' parameters** panel:

**Distance min/max satisfaction:** provides the distance between the situations providing lowest and the highest aim to the actor (measured as the Euclidian distance between the states of the relations, weighted by their respective stake)

**Scope:** determines the actors' capacity to distinguish the situations for the selection of the applicable rules. If  $scope = 1$ , any rule is applicable in any case; if  $scope = 3$ , one may consider that the actor has rules for bad, neural and good situations. High scope increases the size of the actor's rule base and the length of simulations.

**Tenacity:** more the tenacity is high, more the actor will explore the state space of the organization, before to resolve to exploit the acquired knowledge. High tenacity tends to make simulations longer and to provide higher satisfactions.

**Repartition of reward:** distribution of the reward between the last and the penultimate applied rules.

### **Type of rules:**

**Self-learning:** rules are rewarded in proportion of the amount of gain or loss they have brought and according to the rate of exploration wrt exploitation.

**Simple:** rules have a fixed **reward**, without regard for the amount of the gain or loss they have brought, until an upper bound is reached. Rules are slowly forgotten of the **oblivion** value.

**Oblivion:** see above simple rules.

**Reward:** see above simple rules.

**Action range:** (only for simple rules) reference value of importance of the actors' actions. More the range is high, more the actor tends to explore.

Do not forget to **accept** to validate the values.

## Upper left Frames

**Fuzzy stakes:** select randomly the values of stakes within the bound given in the *fuzzy stakes* panel of the *Parameters Editor* window. The sum of stakes of each actor is kept to 10.

**Fuzzy solidarities:** idem

**With constraints:** if not checked, constraints between relations are not applied.

**Number of steps:** maximum number of step of the simulation. A simulation is said to **converge** whether it ends before this number.

**Number of runs:** number of simulations to be runned.

Do not forget to **accept** to validate the values.

**Run:** launch the simulations. You have to specify the directory where a new directory <name of the model>-simulation-<date\_Hour> will be created; simulation results will be stored there.

**Open previous simulation:** allow to load the results of a previous simulation by selecting the <name of the model>-SimulationInitialParameters.xls file, within the directory containing the simulation results.

**View results:** provide various curves on the relations states and actors' satisfactions drawn from the simulation results.

**Synthesis results** synthetic data about the simulation results: mean values and standard deviations over all the runs.

**State analysis:** Open the *State Analysis* windows; in the *Significant states* frame, *Convergence's state* corresponds to the mean value of the relations states.

**Save report:** generates a file including the value of the parameters, the synthetic data and the curves added by the user <name of the model>\_(date)(heure)Synthesis.rtf.

**Clear report:** remove the added curves.

## View Results window

Results of imulations that do not converge are discarded.

**Convergence Option :** the proportion of simulation that have converged.

## Actor Option

**Experiment:** to get detailed curves, choose either *toutes les simulations* and a single one actor, or one of the simulations with *tous les acteurs*.

If you choose *toutes les simulations* and *tous les acteurs*, you get bar charts.

**Variable:** according to the selected variable, a bar chart or a curve is displayed.

On curves, the x-axis corresponds to the steps of the simulation(s), the y-axis to the actor(s)' *satisfaction* or *SeuilSatisfaction* (his ambition threshold).

With the "*toutes les simulations*" choice, the red curve is the means of the others.

A  $\oplus$  mark indicates the end of a simulation.

Pointing the cursor gives the curve number and position of the point.

**Relation Option:** very similar to the Actor option.

**Add to report:** add the display to the report.

**ZZZ:** Long simulations of models including more than 10 actors can produce a stack overflow (cf. message on the console).

## Specific Run

Calls the `core.orgNew.specificRun_10` function, that you can program at your need.

## The Self-learning algorithm

```
repeat      // The Global Simulation Loop
  foreach actor a:      //all actors see the same world
    a.action = a.selectAction ( )
  foreach actor a:      //they don't act in turn
    performAction (a.action)
Until (foreach actor a:  $a.ambition_t \leq a.aim_t$ )

//SelectAction
      // Perception of aim and updating gap
 $aim_t(a, s) = (1 - abs(GI(a))) * satisfaction(a, s) + GI(a) * (influence(a, s) - influence(a, s))$ 
  deltaAim =  $aim_t - aim_{t-1}$ 
  gap =  $(ambition_{t-1} - aim_t) / (ambition_{t-1} - minAim)$ 
      // Updating ambition
  if ( $ambition_{t-1} > aim_t$ )
    if ( $(gap < (11 - tenacity) / 10) \ \&\& \ (deltaAim == 0)$ )
       $ambition_t = ambition_{t-1} - ((1 - ExpR_{t-1}) * (ambition_{t-1} - aim_t + 1)) / 100$ 
    else  $ambition_t = ambition_{t-1} - ((1 - ExpR_{t-1}) * gap / 50$  //about
  else      //a is already satisfied
     $ambition_t = ambition_{t-1} + ((aim_t - ambition_{t-1}) / 100)$ 
      // Updating exploration rate
   $ExpRIns = 0.1 + (0.8 / (1 + e^{slope * (gap - abscissa)}))$ 
   $ExpR_t = ExpR_{t-1} * ExpR_{t-1} + (1 - ExpR_{t-1}) * ExpRIns$ 
      // Updating action_range
   $action\_range = 2 * ExpR_t$  //about
      // Updating strength of last and penultimate selected rules
      //  $SR_t$  stands for the Rule Selected at time t
   $SR_{t-1}.strength_t = (1 - ExpR_t) * SR_{t-1}.strength_{t-1} + ExpR_t * autonomy * deltaAim$ 
   $SR_{t-2}.strength_t = SR_{t-2}.strength_{t-1} + ExpR_t * (1 - autonomy) * deltaAim$ 
      // Forgetting bad rules
  if ( $SR_{t-2}.strength_t < 0$ ) RuleBase.remove( $SR_{t-2}$ )
      // Selecting the set M of rules applicable at time t
  M.clear ( )
  foreach rule R in RuleBase
    if ( $distance(R.situation, CurrentSituation < closenessThreshold)$ )
      M.add(R)
      // Selecting an action
  if (M.isEmpty())
     $SR_t = (CurrentSituation(), (atRandom( ) * action\_range), 0)$ 
    RuleBase.add ( $SR_t$ )
  else
     $SR_t = ChooseOneOfThreeRulesWithMaximumStrength (M)$ 
  return ( $SR_t.action$ )
```