# Participatory Modeling and Simulation with the GAMA platform

**Patrick Taillandier[1], Arnaud Grignard[2], Nicolas Marilleau[3], Damien Philippon[3], Quang-Nghi Huynh[5], Benoit Gaudou[6], Alexis Drogoul[3,4]**

[1]*MIAT, University of Toulouse, INRA, 24 Chemin de Borde Rouge, 31326 Castanet Tolosan Cedex, France*
[2]*Media Lab, Massachussets Institute of Technology, Cambridge, MA, United States*
[3]*UMI UMMISCO, IRD/SU, 32 rue Henri Varagnat, 93143 Bondy Cedex, France*
[4]*ICTLab, USTH, VAST, 18 Hoang Quoc Viet, Cau Giay, Hanoi, Vietnam*
[5]*DREAM-CICT/IRD, Can Tho University, Campus II, 3/2 street, Ninh Kieu District, Can Tho City, Vietnam*
[6]*IRIT,University of Toulouse, CNRS, INPT, UPS, UT1, UT2J, 2 rue du Doyen-Gabriel-Marty, 31042 Toulouse, France*
Correspondence should be addressed to *patrick.taillandier@gmail.com*

*Journal of Artificial Societies and Social Simulation* 22(2) 3, 2019
Doi: 10.18564/jasss.3964 Url: http://jasss.soc.surrey.ac.uk/22/2/3.html

Received: 16-01-2018 Accepted: 01-02-2019 Published: 31-03-2019

**Abstract:** In recent years, agent-based simulation has become an important tool to study complex systems. However, the models produced are rarely used for decision-making support because stakeholders are often not involved in the modeling and simulation processes. Indeed, while several tools dedicated to participatory modeling and simulation exist, these are limited to the design of simple KISS models, thus reducing their potential impact. In this article, we present the new participatory tools integrated within the GAMA modeling and simulation platform. These tools, which take advantage of the GAMA platform with respect to the definition of rich KIDS models, allow the modelers to build models graphically and develop distributed serious games easily. Several application examples illustrate their use and potential.

**Keywords:** Agent-Based Simulation, Participatory Modeling, Participatory Simulation, Serious Game

## Introduction

**1.1** Nowadays, agent-based modeling has not only become increasingly significant for the study of complex systems, but has also opened up promising perspectives for the inclusion of rich and complex models in the socio-environmental decision-making processes (design of public policies, investment planning, community adaptation to climate change, etc.). This is especially true in the case of researchers from fields other than computer science (geography, environmental sciences, social sciences, health, etc.). The adoption of such models, which was partly fueled by the emergence of modeling platforms that ease the work of the modelers, like NetLogo (Tisue & Wilensky 2004), GAMA (Taillandier et al. 2018) or Cormas (Bousquet et al. 1998), has led to an increase in the number of models and to an abundant literature. However, only a handful of these models have really been used to assist in the decision-making process. Many reasons can explain this situation, beginning with the fact that agent-based modeling is still a "research" approach. However, we claim that the main reason lies in the lack of implication of stakeholders in the model design and exploration (simulation) processes. Indeed, most of the time, models are developed and explored only by researchers.

**1.2** An approach that can overcome this difficulty is participatory modeling. Participatory modeling aims at using modeling in support of a decision-making process involving stakeholders (Voinov & Bousquet 2010). The stakeholders can be involved in one or several stages of the modeling process: problem definition, conceptualization, calibration or scenario definition (Barreteau et al. 2014). Participatory simulation is a special case of participatory modeling that concerns the last step of the modeling process, in which participants are invited to interact with a simulated environment, often through a serious game.

**1.3** The models developed in the participatory context can be more or less complex depending on their objective. Typically, it is a common practice to start a participatory modeling process by developing a simple stylized

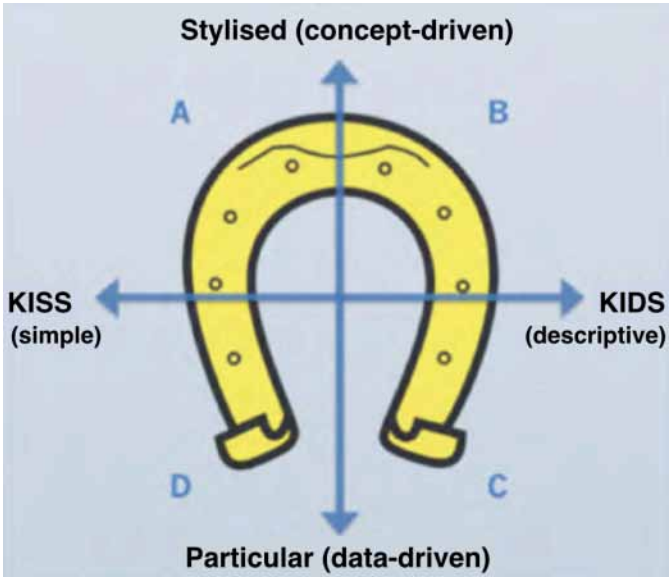



Figure 1: "Horseshoe" reading template proposed by Banos & Sanders (2013) to classify models in geography.

prototype at first and then gradually making it more complex. Indeed, a model designed in a participative way by a group of people with diverse views on a system will often need to be more descriptive to comply with these differing visions.

**1.4** To characterize the variety of models that can be developed, Banos & Sanders (2013) proposed the "Horseshoe" reading template with 2 axes (Figure 1): the simplicity and the abstraction levels. The simplicity level refers to the opposition between the KISS (Keep It Simple, Stupid (Axelrod 1997)) and the KIDS (Keep It Descriptive, Stupid (Edmonds & Moss 2004)) approaches. The KISS approach favors simple models, with very simple agent behavior to reproduce complex systems: the model is simple, but the simulation results are complex. On the other hand, the KIDS approach advocates descriptive models, which remain explanatory. The second axis is related to the abstraction level of the model: does the model represent a stylized fact or a particular given phenomenon? These two axes define four quadrants, the horseshoe illustrating the easy and natural path between these quadrants.

**1.5** More recently, Le Page & Perrotton (2017) introduced the KILT (Keep It a Learning Tool) approach to describe the type of models appropriate for participatory simulations: models should not be abstract (as in the KISS approach) so that stakeholders can identify with the case study and not specify (as in the KIDS approach) too many details to prevent them from focusing on specificities. While this general principle makes sense, we argue that the simplicity and abstraction levels of a model should depend on its objective. In some contexts, to be accepted by stakeholders, the serious game has to be as close as possible to the real system.

**1.6** Some generic modeling and simulation platforms such as Repast, NetLogo or Cormas propose tools dedicated to participatory modeling or development of serious games. However, these tools are generally adapted to the design of simple models with little data (KISS or simple KILT models). We consider that, among the tools mentioned previously, there is none adapted to modelers who are not experts in computer science. Indeed, either they are too complex to be used by non-computer scientists or too limited in terms of implementable models.

**1.7** This article presents the new tools integrated in the GAMA modeling and simulation platform that we developed in order to overcome this shortcoming. GAMA is an open-source modeling and simulation platform for building spatially explicit agent-based simulations. It offers a complete and comprehensive modeling language (GAML: GAma Modeling Language) and an integrated development environment to support the definition of large-scale models. The richness of GAML comes from the numerous optimized operators that it integrates. In particular, GAMA provides modelers with a native integration of GIS data (Taillandier et al. 2010), high-level visualization tools (Grignard & Drogoul 2017), cognitive agent architectures (Caillou et al. 2017; Bourgais et al. 2016), multi-level models (Vo et al. 2012), and co-modeling (Drogoul et al. 2016). Like NetLogo, GAMA allows the user to switch very easily between a modeling perspective (writing the model code) and a simulation perspective (running the simulation). Indeed, with few lines of codes, GAMA allows us to define a model and visualize the simulation results. This possibility makes it particularly suitable for rapid prototyping necessary in the KILT approach: rather

than waiting to have a complete model before running simulations, GAMA enables modelers to quickly experiment the impact of the modifications in the model's code, and consequently favors a test-and-try modeling approach.

1.8 The article is organized as follows: Section 2 presents the new participatory modeling tools integrated into GAMA, more particularly, its graphical modeling plug-in. Section 3 is dedicated to the presentation of the new participatory simulation tools, which concern user-interaction, communication between models and advanced visualization. Lastly, Section 4 concludes and presents perspectives.

# Participatory Modeling

## Context

### Platforms for participatory modeling

2.1 Participatory modeling consists in implicating stakeholders in the model design process (Becu et al. 2015a). A classic approach to achieve this objective is to organize a workshop during which the modelers and stakeholders build a model together using a graphical modeling tool. Most of the modelers conducting such workshops tend to use a specific graphical modeling tool that is not directly related to an agent-based modeling platform.

2.2 Indeed, while there are a few open-source agent-based modeling platforms for developing models through a graphical interface (Gaudou et al. 2010), they are either too complex for non-computer scientists or too limited to allow development of KIDS models.

2.3 To be more precise, some of these platforms such as StarLogo (Resnick 1996) and Modelling4All (Kahn & Noble 2009) are mostly pedagogical software limited to the development of very simple models.

2.4 Another platform that offers such type of tools is Repast Symphony. It proposes to define models in three ways: using Java, the ReLogo language or a graphical modeling language (North et al. 2013). While the ReLogo language and the graphical modeling tools can be used for simple models (or rapid prototyping), developing a KIDS model with this platform requires knowledge of Java.

2.5 The Cormas platform (Bousquet et al. 1998), which is specialized in participatory approaches to address renewable natural resource management issues (Le Page et al. 2012), also provides some graphical modeling tools (definition of activity diagrams). However, this platform, which is more adapted to the development of KISS models, does not offer the same richness as GAMA or Repast Symphony in terms of model development, in particular for developing models based on complex spatial environments or cognitive agents.

2.6 Finally, the MAGeo platform (Langlois et al. 2015) allows us to merely define a model through a dedicated graphic interface. It proposes to formalize agent behavior as an aggregation of basic behaviors with a simple grammar. This grammar is perfectly adapted to the definition of KISS models, but does not allow development of KIDS models. In addition, the number of inbuilt operators is very low in comparison to the ones proposed by the GAMA platform.

2.7 To sum up, there is still a lack of tools, which could be used by a large audience and which would enable graphical definition of KIDS models at the same time.

### Graphical modeling language

2.8 An important issue for developing graphical modeling tools is the choice of a graphical language. The most used for modeling purposes is UML. In the context of agent-based modeling, some works have shown the significance of using such graphical language for communication (Bersini 2012). However, some authors have pointed out that the use of UML as an agent-oriented modeling language is inappropriate (Beydoun et al. 2009).

2.9 Other graphical languages based on UML and dedicated to multi-agent systems have been proposed: the most famous ones are AUML (Bauer et al. 2001) and AML (Cervenka et al. 2005). These languages make it possible to introduce some specific features linked to the agent paradigm. However, their scope goes beyond agent-based simulation and covers all multi-agent aspects, which can make these languages difficult to apprehend for non-computer scientists and not adapted to the context of modeling complex systems.

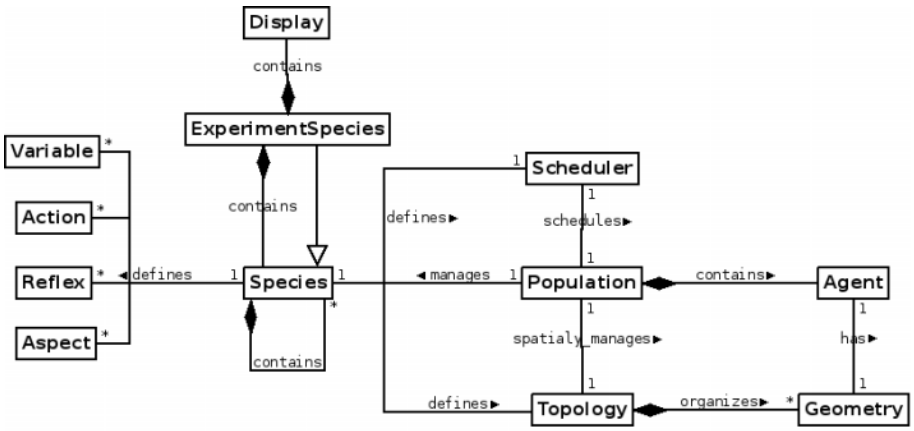



Figure 2: Meta-model of GAMA.

**2.10** The last modeling language that needs to be mentioned is the one proposed by the MAGeo platform. This language is based on the AOC (Actor - Organization - Behavior) meta-model (Daudé et al. 2010). This graphical language is close to UML and respects most of the properties of OOP (Object Oriented Programming). In addition, it allows one to natively define multi-level models. However, this language imposes a lot of constraints due to the limitations of the MAGeo platform. In addition, no distinction is made between what an agent can do and what it is going to do (capabilities versus behaviors).

**2.11** Our goal for the GAMA platform was thus to propose a modeling language, which is simple enough to manipulate (with a small number of concepts), which allows one to develop KIDS models, and which is as close as possible to implementation in order to avoid a gap between the conceptual model and the implemented one. In order to achieve this objective, we identified several properties that our modeling language had to respect:

- Properties of OOP
- Differentiation between capabilities and behaviors
- Native handling of multi-level modeling
- Possibility of defining elements related to simulation visualization

## Graphical modeling with GAMA

### Overview

**2.12** The purpose of our graphical modeling tool is to address the need for participatory tools that would help define KIDS models. It allows GAMA users to graphically define their models and eventually translate them into the GAML language. In addition, this tool enables translation of a GAML model into a graphical one. This feature aims at facilitating the discussion (and communication) pertaining to a model. The tool is based on the Graphiti plugin of Eclipse (Graphiti 2018) and is integrated in a dedicated GAMA plug-in that can be directly downloaded and installed through GAMA.

**2.13** The modeling process with this tool consists first in defining a conceptual model represented by an entity-relationship diagram, then filling in all the defined entities through dialog boxes.

### Definition of the conceptual model

**2.14** We chose to base the definition of the conceptual model on the GAMA meta-model. Although many agent-oriented meta-models were proposed in the literature (see (Beydoun et al. 2009) for a presentation of the most famous ones), most of them are not directly used for simulation purposes and are very difficult for non-computer scientists to grasp. Another advantage of using the GAMA meta-model is that it allows us to limit the gap between the conceptual model and the implemented model.

**2.15** Figure 2 presents the meta-model of GAMA. The main component of this meta-model is the **Species**. A **Species**, like a class in OOP, defines the characteristics common to all the agents of a population. In particular, it defines

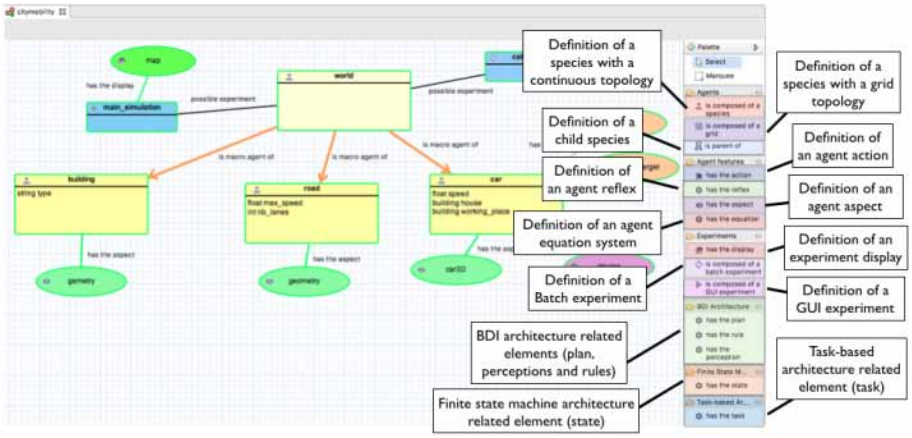



Figure 3: Graphical User Interface of the graphical modeling tool.

their variables, actions, reflexes and aspects. An **Action** is a capability that the agents of the population have, i.e. something that the agents can do. A **Reflex** is a behavior, i.e., something that the agents of the population are going to do (if some conditions are respected). An **Aspect** represents a possible display of the agents. Note that a species can specify several actions, reflexes and aspects. In addition, a species specifies the spatial topology and scheduling of the agent population. A containment relationship between species helps to describe the hierarchical levels of an agency. Finally, a specialization relationship between species helps to define links of inheritance between them.

**2.16** An **ExperimentSpecies** represents a context of execution of a model. It is a particular species of agents that contains a set of species (the one defined in the model) and a set of displays.

**2.17** More details about the GAMA meta-model can be found in (Vo et al. 2012). The use of this meta-model makes it possible to respect the four properties defined in Section 2.

**2.18** Figure 3 presents the modeling graphical framework of the tool. The right palette allows the modelers to select the type of elements to add to the diagram. This framework proposes all the classic features of graphical editors (undo, drag and drop, etc).

**2.19** Table 1 presents all the elements that can be added to the conceptual diagram.

**2.20** When a graphical model is created, a first species of agents is automatically created: the world species. The world species corresponds to the first level of agency that describes the global spatial topology of the model, its basic scheduling, its parameters and global behaviors. It is the host for the populations of agents described by the species defined by the modeler.

**2.21** Thus, development of the conceptual model consists in defining all the species (with their chosen topology: continuous, grid) living in the world, their capabilities (actions), their behaviors (reflexes, tasks, states or BDI elements) and possible displays (aspects). Note that the inheritance relation can be used between species. In addition, definition of the conceptual model consists in determining the possible contexts of execution of the simulation (experiments) and the corresponding outputs (displays) for each of these. Every time the diagram is modified by the user, it is validated: if there is no error in the diagram, its components appear with green borders, and buttons corresponding to each defined experiment appear in the top of the editor (for example, see the *my_GUI_xp* button in Figure 3). By clicking on one of the experiment buttons, the user can load it (and run the corresponding simulation(s)). When there are errors in the diagram, the problematic components appear with red borders.

**2.22** As an example, Figure 4 presents the conceptual model of a simple predator-prey model. In this model, four species of agents live in the world:

- *vegetation_cell*: species with a grid topology that will be used as a spatial environment for the other agents. This species has only one behavior (reflex): *grows*.
- *animal*: species with a continuous topology that will be used as the generic species to define the predators and preys. This species has 4 behaviors (reflex): *eats*, *moves*, *reproduces* and *dies*. It also has one action called *eating* and one aspect called *circle*.
- *prey*: species with a continuous topology that inherits from the *animal* species. This species overrides the *eating* action.

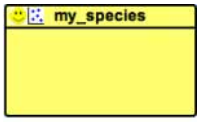
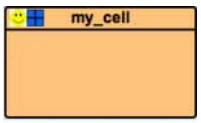
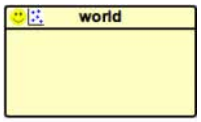
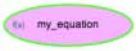
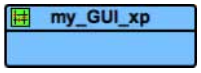
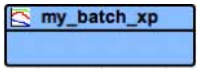
| Symbol | Source | Description |
|---|---|---|
|  | A species | **Species**: A species of agents with a continuous topology. |
|  | A species | **Grid**: A species of agents with a grid topology |
|  | - | **World**: the first level of agency. It contains all other species of agents. |
| | A species | **Action**: A capability that the agents have. |
| | A species | **Reflex**: A behavior (sequence of statements) that will be activated at each simulation step (according to a given condition). |
|  | A species | **Equation**: a differential equations system that can be used to describe the evolution of some of the agent attributes |
| | A species | **Aspect**: A possible display for the agents. |
|  | The world | **GUI Experiment**: load only one simulation with the graphical user interface |
|  | The world | **Batch Experiment**: load a set of simulations without the graphical user interface |
| | A GUI Experiment | **Display**: frame allowing to display outputs (map, charts...) |
| | A species with a BDI architecture | **Plan**: sequence of statements that will be executed in order to fulfill a particular intention. More details about the GAMA BDI architecture and the plan statement can be found in several works (Taillandier et al. 2016; Caillou et al. 2017; Bourgais et al. 2017, 2016) |
| | A species with a BDI architecture | **Rule**: function executed at each iteration to infer new desires or beliefs from the agent's current beliefs and desires |
| | A species with a BDI architecture | **Perception**: function executed at each iteration that updates the agent's belief base according to its perception |
| | A species with a finite state machine architecture | **State**: sequence of statements that will be executed if the agent is in this state (an agent has a unique state at a time). |
| | A species with a task-based architecture | **Task**: sequence of statements that can be executed by the agent at each time step. If an agent owns several tasks, the scheduler chooses a task to execute according to their priority. |

Table 1: Entities of the graphical modeling language

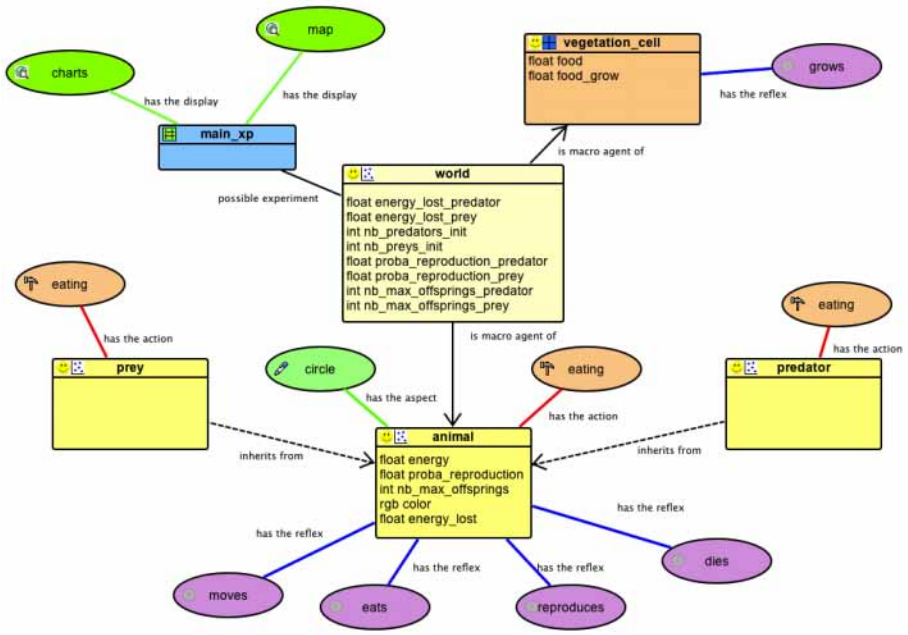



Figure 4: Conceptual model of the predator-prey model.

- *predator*: species with a continuous topology that inherits from the *animal* species. This species overrides the *eating* action.

**2.23** In addition, we define one GUI experiment called *main_xp* that has a display called *map* and a display called *charts*.

### Definition of the parameters and processes

**2.24** Once the conceptual model is defined, the next step consists in describing the properties of each defined entity.

**2.25** When the user clicks on an entity, a new dialog box allowing parameterization appears. It is through these dialog boxes that the modeler will be able to transform his/her conceptual model into a simulation. Sometimes, the parameterization will just consist in making a choice between different options, but sometimes it will consist in writing GAML instructions.

**2.26** The most important entity to parameterize is the species. The species dialog box helps to define some of the properties of the species with a minimal amount of code (see Figure 5). In particular, it allows the modeler to define the variables (attributes) of the species. For each variable, the modeler has to define its name and its type (among many types such as integer, float number, string, list, matrix, map, point, geometry, graph, path...). In addition, the modeler can define optional facets for each variable such as its initial value, a minimum, a maximum, an expression that will be used to re-compute the variable at each simulation step or a function that defines how the variable will be computed each time it is requested. The modeler can also give skills to the species. A skill is a predefined set of variables and actions coded in Java. For instance, the moving skill provides the species with the variables *speed*, *heading* and *destination* and the actions *move*,*goto*,*wander* and *follow*. The modeler can also choose the architecture of the species. By default, all agents in GAMA have a reflex control architecture (behaviors activated according to a certain condition), but it is possible to add another architecture to the agent: a BDI architecture, a state-machine architecture or a task-based architecture. The modelers can also define the scheduler of the species and its frequency of activation. Lastly, the modeler can define an *init* block that represents the constructor of the species, i.e. it defines what will happen when agents of this species are created.

**2.27** The dialog boxes for the world species and the grid definition are very similar. For the grids, the dialog box only allows the user to additionally define the number of cells in the rows and in the columns, and the type of neighborhood: Moore, van Neumann or Hexagonal. For the world species, the dialog box allows the modeler to additionally determine whether the environment is torus or not.

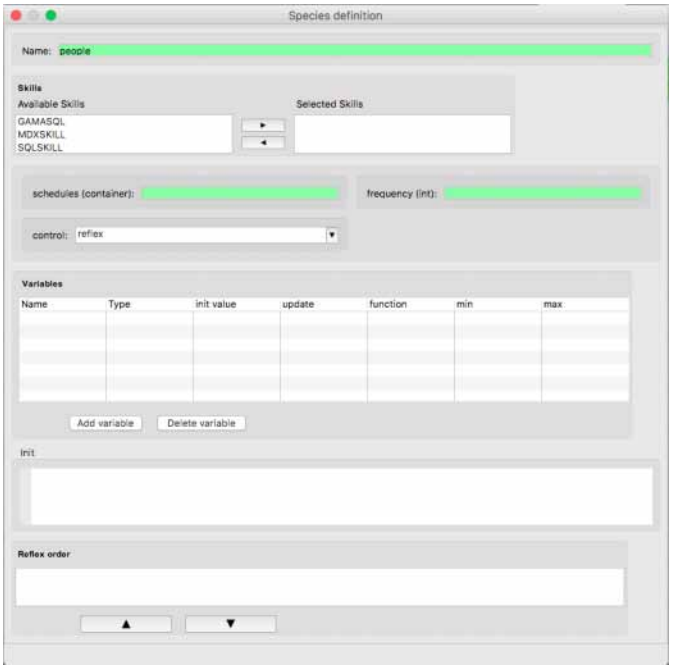



Figure 5: Dialog box for defining Species.

**2.28** Concerning the parameterization of reflexes, the dialog box makes it possible to choose the condition of the reflex activation and its effect. The activation condition and the effect are described using the GAML language. The action dialog box is very similar to the reflex one except that no condition can be defined, but the modeler can add arguments and a return value.

**2.29** For the aspect definition, the dialog box enables the definition of the layers composing the aspect (and their order). These layers are defined through a dialog box, in which the modeler can choose the shape to display (a simple shape such as a circle, a square, a rectangle..., an icon, a text or a complex geometry such as a polyline or a polygon), its color, and some specific properties (rotation, filled/empty shape...) or directly through the GAML language.

**2.30** The experiment definition dialog box makes it possible to define the parameters that the user will be able to modify through the simulation interface.

**2.31** Concerning the display definition, the dialog box allows the modeler to define the layers composing the display (and their order), to choose a color for the background and the refreshing rate. The layers are defined through a dialog box, in which the modeler can choose the elements to display (a list of agents, a chart, an image, a text...), the level of transparency of the layer, its size and its position.

**2.32** Finally, concerning the equation, state, task and BDI-related elements, the modelers will have the possibility of filling some specific fields (facets of these statements) and describing their effects in GAML code.

## Conclusion

**2.33** To sum up, the graphical modeling tool of GAMA enables graphical definition of a complete and functional GAMA model with a minimal quantity of GAML code.

**2.34** This tool offers many advantages for a participatory modeling context. The choice made in this tool to use a graphical meta-model that covers the more important concepts of the GAML language, allows modelers to graphically define a whole model. As a consequence, there is no cognitive effort to switch between the graphical modeling language and the implementation language. On the other hand, the model created graphically can be directly executed in GAMA: this upholds one of the main advantages of the modeling and simulation platforms, i.e. the possibility of executing the model being implemented very quickly and switching between the model and the simulation, thus contributing to the understanding of the model. In addition, the graphical modeling language enables a description of the core components of the model and also helps to determine how to simulate it: it allows the modeler to define the experiments, i.e. the parameters that can be modified during

the experiment and the various displays or visualizations of the simulation. This last aspect is often missing from participative modeling and simulation tools.

**2.35** The tool is already used in several projects and lectures, which makes the discussions around a model easier or allows for a collective construction of a new model from scratch.

# Participatory simulation

## Context

**3.1** Nowadays, more and more modeling projects integrate participatory simulation, often with the help of a serious game. Participatory simulations invite human participants to interact with a simulated environment. Such simulations can be used to convey messages efficiently (Klabbers 2009), to serve as communication media between participants or to engage the public in a decision-making process (Noyman et al. 2017). A classic example is Fish Banks (Meadows et al. 1986), which allows participants to play the role of a fishing company manager and to interact with a computer simulation of renewable fish stocks.

**3.2** The main difficulty in participative simulation is gaining the confidence of stakeholders. It implies: (i) the development of an empirical model showing realistic dynamics and (ii) reproduction, in the serious game, of the workspace that the stakeholders are used to (e.g., available policies, budget, governmental restriction, social events). The Graphical User Interface has thus to be designed according to the player role.

**3.3** To facilitate the design and implementation of such serious games, some generic agent-based platforms propose dedicated tools.

**3.4** One of the most commonly used platforms for serious games is NetLogo. This platform allows users to directly interact with different agents of a simulation through agent inspectors and the observer. In addition, it allows addition of some components (sliders, field area, button) to the interface allowing a user to interact with the simulation. Moreover, it allows creation of a dialog to ask the user to fill in a specific variable value. Finally, with its Hubnet extension (Blikstein et al. 2005), NetLogo allows several instances of the same models communicating with each other through a local network. This last feature is particularly interesting when implementing distributed serious games. However, NetLogo is more adapted to the development of simple KISS models rather than complex KIDS models. In addition, the presence of a unique display of the environment and the general simulation panel are important drawbacks for the implementation of serious games.

**3.5** 3.5 Another platform that proposes tools for participatory simulations is Cormas. This platform offers many features to develop and manage such simulations (Becu et al. 2016). In particular, like NetLogo, Cormas allows participants to directly interact with the various agents (or group of agents) though the agent inspector. Using Cormas, one can also directly manipulate one or several agents (for instance, execute some of their actions) through a dedicated interface. Cormas can distribute the control of a simulation through a network on several computers (Becu et al. 2015b). More precisely, it gives the control on the same simulation (running on a server) to different clients who can have different views on the same simulation and interact with it. In addition, unlike NetLogo, Cormas allows the modeler to define several displays. Finally, Cormas gives the possibility of going back in time: by just clicking on the backward button, the user can backtrack to the previous step or replay a simulation. While Cormas is particularly well-adapted for use in participatory contexts, developing models, which require using the Smalltlak language and the VisualWorks framework (Brauer 2015), can be complex for non-computer scientists. In addition, like NetLogo, the platform is not adapted to the development of complex KIDS models (no real integration of GIS data, simple representation of the environments, etc.).

**3.6** To sum up, NetLogo and Cormas propose many tools that can be used in a participatory simulation context, but they are both limited with respect to the development of KIDS models, which explains why most participatory simulations are still based on simple KISS or KILT models.

**3.7** As a consequence of these limitations, several projects have chosen to use GAMA to develop ambitious interactive models.

**3.8** A first example is Sprite [1] (Adam et al. 2016; Taillandier & Adam 2018), which allows the participant to play the role of the mayor and manage Oleron Island for a certain number of years, with the mission of finding an appropriate balance between popularity, economy, attractiveness, safety and ecology. Sprite proposes a precise representation of the island through GIS data and integrates an advanced submersion model. Integration of

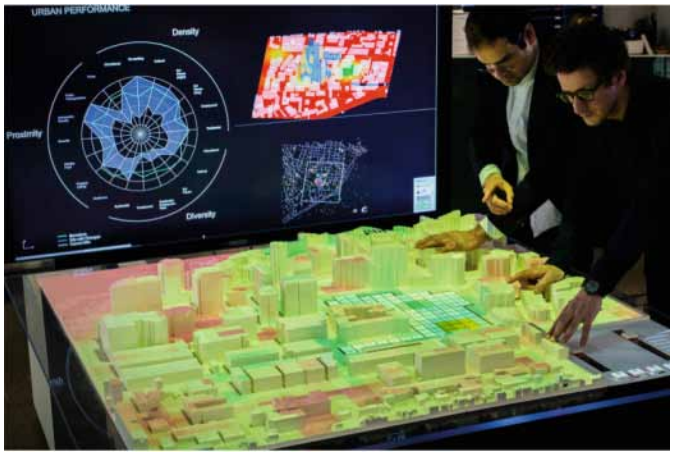



Figure 6: Tangible interface of the Cityscope model (taken from Grignard et al. (2018)).

GIS data would have been very difficult to achieve with Cormas, whereas the multiples views of the territory offered by the model would not have been possible with NetLogo.

**3.9** Another example is Littosim[2] (Becu et al. 2017), which also illustrates the management of the risk of submersion. In Littosim, participants play the role of the land-planning manager of a coastal municipality prone to marine submersion risk. It integrates several sub-models such as water rise and socio-economic evolution models. One of the features of Littosim is its use of distributed controls (each player playing with a dedicated interface from a computer tablet) and of a projector to propose a global shared display showing the impact of flooding. Like for Sprite, the heavy use of GIS data would be been very difficult to achieve with Cormas or NetLogo, whereas the distributed control over several simulations is much more robust in the face of network disconnections than in NetLogo. This prevents running the risk of having to restart the entire game in case of a single disconnection.

**3.10** A last example that concerns urban design is CityScope[3] (Grignard et al. 2018; Alonso et al. 2018). In this model, which simulates the daily life of the inhabitants of a city, in particular their mobility, used as a tangible interface to let stakeholders interact with the simulation (Figure 6). With this interface, stakeholders can modify the city plan and get a direct feedback of the impact of these modifications on the simulation. Two displays are used to show these impacts: a first one projected on the LEGO bricks that provides information about the buildings and inhabitant mobility, and a second one projected on a screen that presents different sub-displays related to urban performance. The model illustrates several features of GAMA in addition to GIS data management: connection of a GAMA simulation with another software (the one used to analyze the LEGO city model in this case), definition of several rich displays composed of sub-displays and management of the projection of a display on a specific physical surface (wall, table, etc...).

**3.11** These three examples illustrate the new features that we developed for the GAMA platform linked to the three main components of the development of rich participatory simulations: visualization, user interaction and simulation interconnection.

## Visualization with GAMA

**3.12** Visualization is one of the main components of agent-based simulations, in particular in the context of participatory simulations (Dorin & Geard 2014). For many modelers, visualization is not only their first point of entry when building a model, but also an increasingly prevalent way of designing, verifying and validating models. The back and forth between writing and visualizing a model is an integral part of the daily life of modelers.

**3.13** This visualization is part of an integrated modeling approach that gives the possibility of intuitively verifying agent states and refining individual behaviors of the agents and the expected collective or emerging structures (Grignard et al. 2013). This practice is a key point of participatory simulation approaches, where the visualization of the model serves as a mediation between actors and as a support for their decision-making. In order to speak to a large and heterogeneous audience and define graphical interface to a specific role, it is necessary to propose different views of the same simulation.

**3.14** GAMA enables modelers to easily differentiate a model from its visualization by offering the possibility of defining different displays composed of several layers per simulation. The user, depending on his/her role and

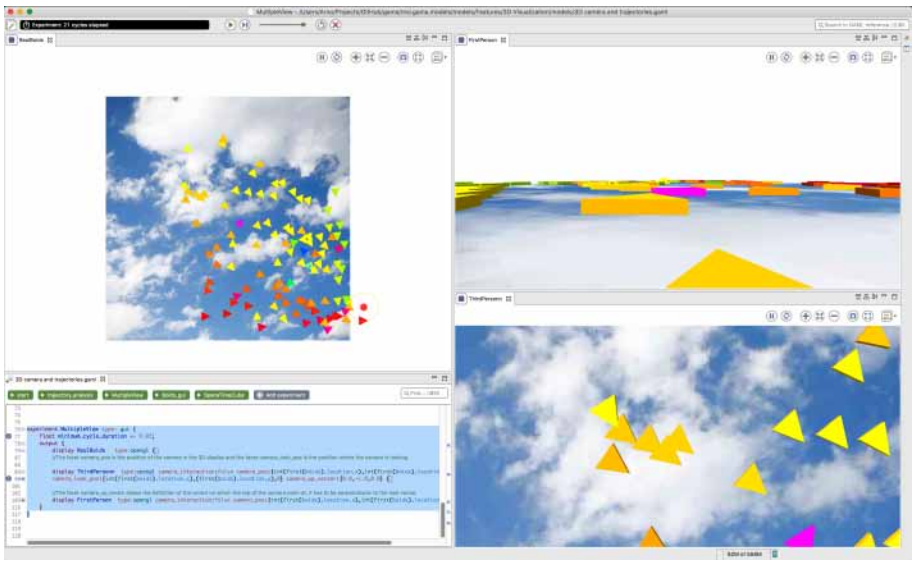


Figure 7: The same model observed with 3 different perspectives.

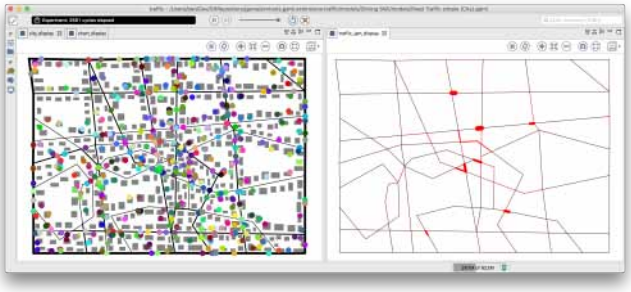


Figure 8: A same traffic model observed through 2 different points of views.

his/her level of expertise, will then visualize the simulation in a specific way and interact with its representation without altering the initial model. Figure 7 shows a classical boids model[4] observed from 3 different perspectives (3 different positions of the camera). To go further, Figure 8 shows a simulation of traffic through 2 points of view: the first display shows the individual vehicle agents moving on the network and generating traffic (jams) and a second display drawing the traffic jams themselves)[5]. In an **experiment**, the user can define as many **displays** as he/she wants. Each display is composed of different **layers**. For each **layer**, the modeler can configure visibility, transparency, position and size of the layer.

**3.15** As noted by Allan (2009) and Railsback et al. (2006), the use of 3D is still uncommon in the world of agent-based simulation, whereas using the third dimension can significantly increase the immersive property of a simulation. GAMA offers advanced visualization features pertaining to 3D visualization with textures, complex lights, and custom dynamic cameras through simple GAML statements. For example, the CityScope model uses these 3D features for some sub-displays.

**3.16** Finally, as presented in the previous section, GAMA offers tools to manage the projection of a display on a specific physical surface (wall, table, etc...) to avoid any resulting image deformation, which is a big advantage not only when engaging the community but also when communicating the results of a model as in some projects (Becu et al. 2017). Figure 9 shows an example of the use of this feature for the MarrakAir project[6] (Emery et al. 2017): in this project, which aims at alerting the general public to the air pollution of the city of Marrakech, a GAMA simulation is displayed on the physical model of Marrakech (built with a 3D-printer). This tangible interface makes it easier for spread awareness and understanding among a wide audience about the effects of pollution in urban areas.

## User Interaction

**3.17** Like NetLogo and Cormas, GAMA provides many features for defining user interaction, i.e. the possibility for

Figure 9: Picture of the MarrakAir installation.

participants to interact with a simulation. While some of these features such as the possibility of modifying the value of the agent's variables through the agent inspector are directly available for all simulations, some others can be specified by the modeler using the GAML language. The next sections describe all these features.

### Event layer

**3.18** In contrast to platforms such as NetLogo, GAMA permits the definition of as many displays as necessary. In addition, a modeler can specify in a display a specific layer called the event layer allowing an action to be triggered when an event occurs. The possible events considered are: mouse up, mouse down, mouse move, mouse enter, mouse exit, or a character key (keyboard event).

**3.19** Once the event is triggered, the action linked to this event is requested and can modify the state of the world. The modeler can directly access the location of the mouse in a display (in terms of coordinates in the world) through a dedicated keyword. From this location, and thanks to GAMA's spatial operators, the modeler can obtain the list of agents in the neighborhood or overlapping the mouse location.

**3.20** For example, the Sprite and Littosim models use this type of events to detect the players' actions on the display. An older example that uses a former version of event layer is presented in Chu et al. (2008): in this model concerning the coordination of a rescue team (ambulances, cops, firefighters) after an earthquake, the user can modify, at runtime, the target of the different members of the rescue team by just clicking on them and the desired target, and observe the result.

### User command

**3.21** In addition event layers, the modeler can define actions that can be directly activated through the GUI interface during a simulation: a user command.

**3.22** A user command can be defined in a GUI experiment or in a species of agents.

**3.23** If the user command is defined in a GUI experiment, the implemented action appears as a button on the top of the parameter view (Figure 10)[7].

**3.24** If the user command is defined inside a species scope (either a global or a regular one), the user can access the action during execution in two ways:

- When the agent is inspected, the user commands appear as buttons above the agents' attributes (Figure 11).
- When the agent is selected by a right-click in a display, these commands appear in the pop-up menu (Figure 12).

**3.25** For example, one of the instances of the CityScope model uses user commands to enable stakeholders to directly modify the type of mobility allowed on a road (e.g. pedestrian, bike, car).

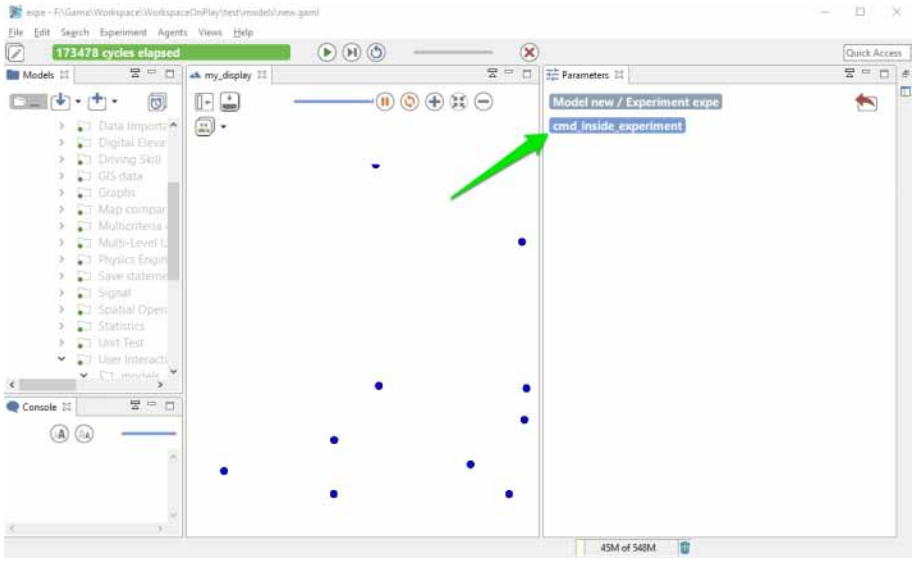



Figure 10: Interface with a user command called *cmd_inside_experiment* defined in the experiment

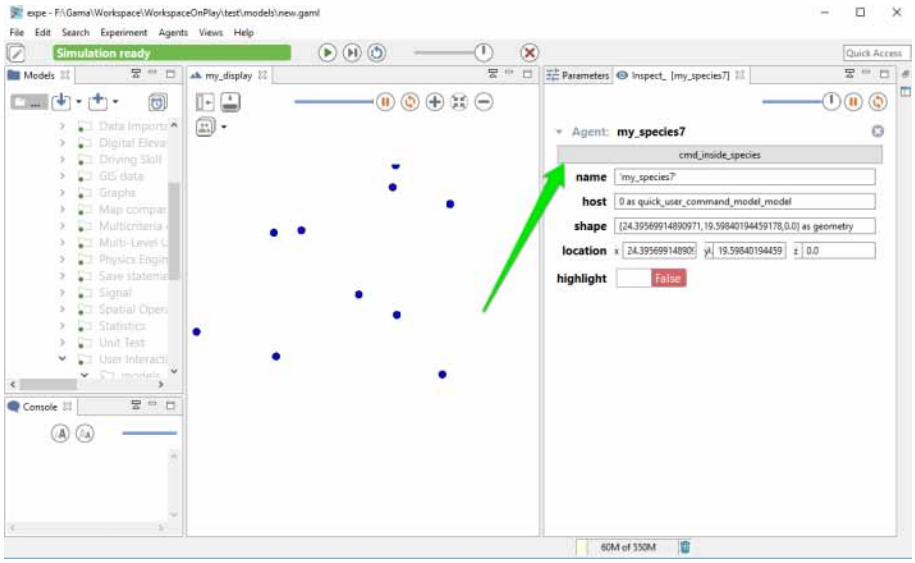



Figure 11: Interface with a user command called *cmd_inside_species* defined in the *my_species* species – inspector view.

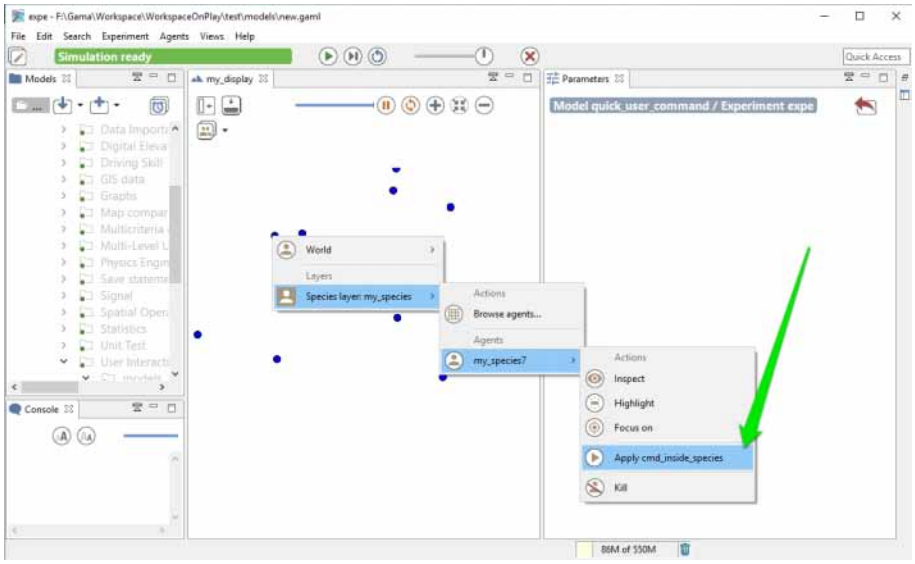



Figure 12: Interface with a user command called *cmd_inside_species* defined in the *my_species* species – right-click on the display.

### User input

**3.26** Like in NetLogo, GAMA allows the modeler to define a dialog to ask the user to give a value to one or several variables through a specific operator. This operator displays a dialog asking the user to define the values of the corresponding variables. The modeler can also add a text as an argument by the operator, which will be displayed as a title for the dialog pop-up (Figure 13). The dialog is modal and will interrupt the execution of the simulation until the user has either dismissed or accepted it. It can be used, for instance, for the purpose of initialization to force the user to input new values instead of relying on the initial parameter values. It can also be used for participative models where each user runs a client model and inputs values to the server model.

**3.27** The Sprite model uses this feature to ask for information from the player (first of all, the player's name).

### Control architecture

**3.28** Another way to define user interactions is to use the user control architecture that allows users to take control over an agent during the course of the simulation through a user-controlled panel.

**3.29** This control architecture is a specialization of the Finite State Machine Architecture (FSM) where agent behaviors can be defined by using a new construct called *user panel*. This *user panel* translates, in the interface, into a semi-modal view that waits for the user to choose action buttons, change attributes of the controlled agent, etc.

**3.30** As *user panel* is a specialization of state, the modeler has the possibility of describing several panels and choosing the one to open depending on a few conditions, using the same syntax as the one used for finite state machines. This ensures great flexibility in the design of the user interface, as it can be adapted to various stages of the simulation.

## Simulation interconnection

**3.31** As illustrated by the work presented in Becu et al. (2017) and Grignard et al. (2018), GAMA enables a simulation to communicate with other programs through different protocols (TCP, UDP, MQTT).This feature allows, for example, a GAMA simulation to communicate with another GAMA simulation or with various terminals such as smartphones, tablets and sensors to export displays or to collect data. The basic idea is that any agent in the model can get the capability (through a dedicated GAML skill) to connect to a particular server and to send messages to any other agent in any other simulation that is connected to this server. Sent data is automatically and internally serialized before being sent as content of messages; this allows an agent to send data of any kind through the network.

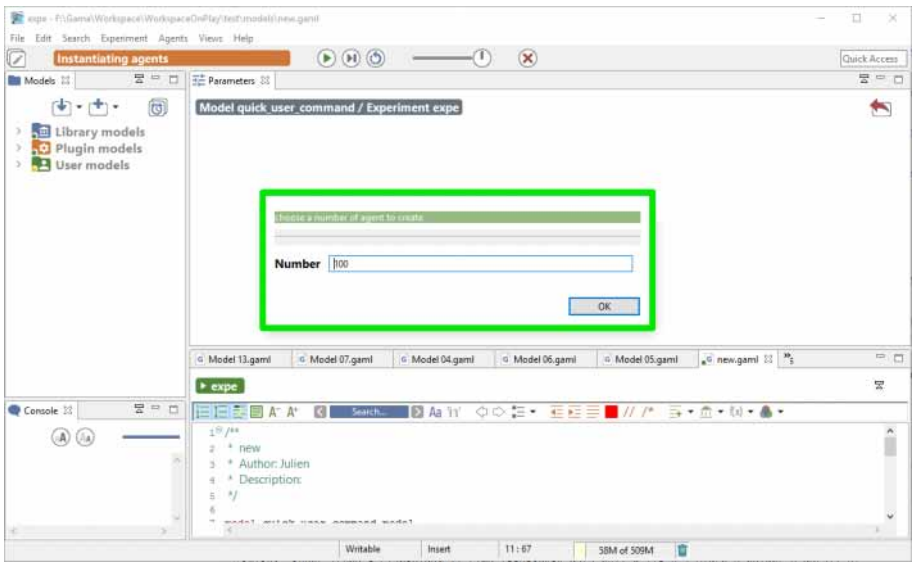



Figure 13: Interface with an input command that asks the user to fix the number of agents to be created.

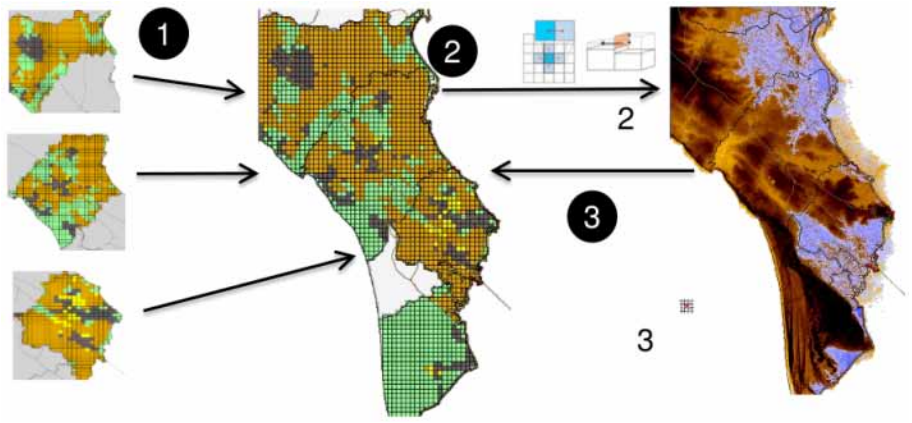



Figure 14: Processes of Littosim: 1) data is sent by the clients (players); 2) the sever sends the data to the flooding model; 3) the flooding model is sent back to the server.

**3.32** From a technical point of view, extensive tests have been made using an ActiveMQ message broker (MQTT protocol) with 5 to 10 GAMA simulations connected on it. One of the main advantages of using such an architecture is that it the negative effects of unstable Wifi connections. In addition, it can accept connections from any type of devices such as mobile devices (based on Android or IOS) or captors. For instance, this architecture allows the modeler to connect GAMA simulations running on different computers with an android interface deployed on a mobile phone controlling the game.

**3.33** For example, the Littosim project (presented in Becu et al. 2017) uses the following type of architecture (Figure 14):

1. Each client is controlled by a player, data is sent to the server model.
2. Elevation and land cover rugosity data is updated and sent to the flooding model (external program).
3. Flood simulation results are sent back to the server model and each player can evaluate damages in his/her territory.

## Conclusion

**4.1** In this article, we presented the new features we developed for the GAMA platform dedicated to participatory modeling and simulation. As shown, these features enable GAMA to simplify the work of modelers in a par-

ticipatory context. Several real application cases have already shown the usability of these features to create complex serious games.

**4.2** GAMA is continuously evolving and new features are constantly being added to the platform. One of the new features that is currently under development concerns the possibility of saving the different steps of a simulation to give the possibility of backtracking to previous steps or replay simulations. While this feature is already available in some platforms such as Cormas, its usability for KIDS models composed of thousands of agents with complex variables (3D geometries, graph…) is a real challenge.

**4.3** Furthermore, in the future, we plan to couple GAMA with a game engine such as Unity to be able to propose high-quality 3D displays and rich real-time interactions with a 3D environment. The challenge will then be to retain the simplicity of GAMA modeling, even with complex 3D environments.

## Acknowledgements

This work is partially supported by the public grants overseen by the French National Research Agency (ANR) as part of the program PRC (reference: ACTEUR ANR-14-CE22-0002).

## Notes

[1]The source code of the Sprite model can be downloaded here: `https://goo.gl/n4SRYL`.

[2]The source code of the Littosim project is available on the following GitHub repository: `https://github.com/LittoSim`.

[3]The source code of the CityScope project is available on the following GitHub repository: `https://github.com/CityScope`.

[4]This model is available in the GAMA Model Library: *Library Models / Features / 3D Visualization / 3D Camera and trajectories.gaml*.

[5]This model is available in the GAMA Model Library: *Plugin Models / Driving Skill / Road Traffic simple (City).gaml*.

[6]The source code of the MarrakAir project can be found on the GitHub repository: `https://github.com/gnoubi/MarrakAir`.

[7]Figures 10, 11, 12 and 13 are snapshots of the execution of models available in the GAMA Model library: they can be found in *Library Model / Features / User Interaction/*.

## References

Adam, C., Taillandier, F., Delay, E., Plattard, O. & Toumi, M. (2016). SPRITE – Participatory simulation for raising awareness about coastal flood risk on the oleron island. In *International Conference on Information Systems for Crisis Response and Management in Mediterranean Countries*, (pp. 33–46). Berlin/Heidelberg: Springer

Allan, R. J. (2009). Survey of agent based modelling and simulation tools. Tech. rep.

Alonso, L., Zhang, Y. R., Grignard, A., Noyman, A., Sakai, Y., ElKatsha, M., Doorley, R. & Larson, K. (2018). Cityscope: A data-driven interactive simulation tool for urban design. Use case Volpe. In A. J. Morales, C. Gershenson, D. Braha, A. A. Minai & Y. Bar-Yam (Eds.), *Unifying Themes in Complex Systems IX*, (pp. 253–261). Cham: Springer

Axelrod, R. (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton, NJ: Princeton University Press

Banos, A. & Sanders, L. (2013). Modéliser et simuler les systèmes spatiaux en géographie. In F. Varenne & M. Silberstein (Eds.), *Modéliser & simuler*, vol. 2, (p. 2). Paris: Edition Matériologiques

Barreteau, O., Bousquet, F., Étienne, M., Souchère, V. & d'Aquino, P. (2014). Companion modelling: A method of adaptive and participatory research. In *Companion Modelling*, (pp. 13–40). Berlin/Heidelberg: Springer

Bauer, B., Müller, J., Odell, J. & Arbor, A. (2001). Agent UML: A formalism for specifying multiagent interaction. *Agentoriented Software Engineering*, *1957*, 91–103

Becu, N., Amalric, M., Anselme, B., Beck, E., Bertin, X., Delay, E., Long, N., Marilleau, N., Pignon-Mussaud, C. & Rousseaux, F. (2017). Participatory simulation to foster social learning on coastal flooding prevention. *Environmental Modelling & Software*, *98*, 1–11

Becu, N., Amblard, F., Brax, N., Gaudou, B. & Marilleau, N. (2015a). How to involve stakeholders in the modeling process. In A. Banos, C. Lang & N. Marilleau (Eds.), *Agent-Based Spatial Simulation with NetLogo*, (pp. 223–252). Amsterdam: Elsevier

Becu, N., Bommel, P., Le Page, C. & Bousquet, F. (2016). Cormas, une plate-forme multi-agent pour concevoir collectivement des modèles et interagir avec les simulations. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA)*. Cépaduès

Becu, N., Frascaria-Lacoste, N. & Latune, J. (2015b). Experiential learning based on the newdistrict asymmetric simulation game: Results of a dozen gameplay sessions. In *Hybrid Simulation & Gaming in the Networked Society: The 46th ISAGA Annual Conference 2015*

Bersini, H. (2012). UML for ABM. *Journal of Artificial Societies and Social Simulation*, *15*(1), 9

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J. & Gonzalez-Perez, C. (2009). FAML: A generic metamodel for mas development. *Software Engineering, IEEE Transactions on*, *35*(6), 841–863

Blikstein, P., Abrahamson, D. & Wilensky, U. (2005). NetLogo: Where we are, where we're going. In *Proceedings of the Annual Meeting of Interaction Design and Children*

Bourgais, M., Taillandier, P. & Vercouter, L. (2016). An agent architecture coupling cognition and emotions for simulation of complex systems. Social Simulation Conference 2016

Bourgais, M., Taillandier, P. & Vercouter, L. (2017). Enhancing the behavior of agents in social simulations with emotions and social relations. In *The 18th Workshop on Multi-agent-based Simulation-MABS 2017*

Bousquet, F., Bakam, I., Proton, H. & Le Page, C. (1998). Cormas: Common-pool resources and multi-agent systems. In A. P. Del Pobil, J. Mira & M. Ali (Eds.), *Tasks and Methods in Applied Artificial Intelligence*, (pp. 826–837). Berlin/Heidelberg: Springer

Brauer, J. (2015). The VisualWorks development environment. In J. Brauer (Ed.), *Programming SmallTalk – Object-Orientation from the Beginning*, (pp. 77–96). Berlin/Heidelberg: Springer

Caillou, P., Gaudou, B., Grignard, A., Truong, C. Q. & Taillandier, P. (2017). A simple-to-use BDI architecture for agent-based modeling and simulation. In W. Jager, R. Verbrugge, A. Flache, G. de Roo, L. Hoogduin & C. Hemelrijk (Eds.), *Advances in Social Simulation 2015*, (pp. 15–28). Berlin/Heidelberg: Springer

Cervenka, R., Trencansky, I. & M., C. (2005). Modeling social aspects of multiagent systems: The AML approach. In J. P. Müller & F. Zambonelli (Eds.), *Agent-Oriented Software Engineering VI 6th International Workshop, AOSE 2005, Utrecht, The Netherlands, July 25, 2005. Revised and Invited Papers*, (pp. 28–39)

Chu, T.-Q., Boucher, A., Drogoul, A., Vo, D.-A., Nguyen, H.-P. & Zucker, J.-D. (2008). Interactive learning of expert criteria for rescue simulations. In *Pacific Rim International Conference on Multi-Agents*, (pp. 127–138). Berlin/Heidelberg: Springer

Daudé, E., Langlois, P., Blanpain, B. & Sapin, E. (2010). AOC, une ontologie formelle pour la modélisation de systèmes complexes en géographie. In *Outils, méthodes et modèles en géomatique pour la production de connaissances sur les territoires et le paysage*

Dorin, A. & Geard, N. (2014). The practice of agent-based model visualization. *Artificial Life*, *20*(2), 271–289

Drogoul, A., Huynh, N. Q. & Truong, Q. C. (2016). Coupling environmental, social and economic models to understand land-use change dynamics in the mekong delta. *Frontiers in Environmental Science*, *4*, 19

Edmonds, B. & Moss, S. (2004). From KISS to KIDS – An 'anti-simplistic' modelling approach. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, (pp. 130–144). Berlin/Heidelberg: Springer

Emery, J., Marilleau, N., Martiny, N., Thévenin, T., Nguyen-Huu, T., Badram, M., Grignard, A., Hbdid, H., Laatabi, A.-M. & Toubhi, S. (2017). Marrakair: une simulation participative pour observer les émissions atmosphériques du trafic routier en milieu urbain. In *Treizièmes Rencontres de Théo Quant*

Gaudou, B., Marilleau, N. & Ho, T. V. (2010). Toward a methodology of collaborative modeling and simulation of complex systems. In *Intelligent Networking, Collaborative Systems and Applications*, (pp. 27–53). Berlin/Heidelberg: Springer

Graphiti (2018). Graphiti. `http://www.eclipse.org/graphiti/`

Grignard, A., Alonso, L., Taillandier, P., Gaudou, B., Nguyen-Huu, T., Gruel, W. & Larson, K. (2018). The impact of new mobility modes on a city: A generic approach using ABM. In *International Conference on Complex Systems*, (pp. 272–280). Berlin/Heidelberg: Springer

Grignard, A. & Drogoul, A. (2017). Agent-based visualization: A real-time visualization tool applied both to data and simulation outputs. In *The AAAI-17 Workshop on Human-Machine Collaborative Learning*, (pp. 670–675)

Grignard, A., Drogoul, A. & Zucker, J.-D. (2013). Online analysis and visualization of agent-based models. In *International Conference on Computational Science and Its Applications*, (pp. 662–672). Berlin/Heidelberg: Springer

Kahn, K. & Noble, H. (2009). The modelling4all project a web- based modelling tool embedded in web 2.0. In *International Conference on Simulation Tools and Techniques*

Klabbers, J. H. G. (2009). *The Magic Circle: Principles of Gaming and Simulation*. Rotterdam: Sense

Langlois, P., Blanpain, B. & Daudé, E. (2015). Magéo, une plateforme de modélisation et de simulation multi-agent pour les sciences humaines. *Cybergeo: European Journal of Geography*

Le Page, C., Becu, N., Bommel, P. & Bousquet, F. (2012). Participatory agent-based simulation for renewable resource management: The role of the Cormas simulation platform to nurture a community of practice. *Journal of Artificial Societies and Social Simulation*, *15*(1), 10

Le Page, C. & Perrotton, A. (2017). KILT: A modelling approach based on participatory agent-based simulation of stylized socio-ecosystems to stimulate social learning with local stakeholders. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, (pp. 156–169). Berlin/Heidelberg: Springer

Meadows, D., Fiddaman, T. & Shannon, D. (1986). Fish banks, ltd. *Laboratory for Interactive Learning*

North, M., Collier, N., Ozik, J., Tatara, E., Macal, C., Bragen, M. & Sydelko, P. (2013). Complex adaptive systems modeling with Repast simphony. *Complex Adaptive Systems Modeling*, *1*(1), 3

Noyman, A., Holtz, T., Kröger, J., Noennig, J. R. & Larson, K. (2017). Finding places: HCI platform for public participation in refugees accommodation process. *Procedia Computer Science*, *112*, 2463–2472

Railsback, S. F., Lytinen, S. L. & Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, *82*(9), 609–623

Resnick, M. (1996). Starlogo: An environment for decentralized modeling and decentralized thinking. In *Conference Companion on Human Factors in Computing Systems*, (pp. 11–12)

Taillandier, F. & Adam, C. (2018). Games ready to use: A serious game for teaching natural risk management. *Simulation & Gaming*, *49*(4), 441–470

Taillandier, P., Bourgais, M., Caillou, P., Adam, C. & Gaudou, B. (2016). A BDI agent architecture for the GAMA modeling and simulation platform. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, (pp. 3–23). Berlin/Heidelberg: Springer

Taillandier, P., Gaudou, B., Grignard, A., Huynh, Q.-N., Marilleau, N., Caillou, P., Philippon, D. & Drogoul, A. (2018). Building, composing and experimenting complex spatial models with the GAMA platform. *GeoInformatica*

Taillandier, P., Vo, D.-A., Amouroux, E. & Drogoul, A. (2010). GAMA: A simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *International Conference on Principles and Practice of Multi-Agent Systems*, (pp. 242–258). Berlin/Heidelberg: Springer

Tisue, S. & Wilensky, U. (2004). NetLogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, (pp. 16–21)

Vo, D.-A., Drogoul, A. & Zucker, J.-D. (2012). An operational meta-model for handling multiple scales in agent-based simulations. In *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on*, (pp. 1–6)

Voinov, A. & Bousquet, F. (2010). Modelling with stakeholders. *Environmental Modelling & Software*, *25*(11), 1268–1281