MSEL
Modelling in Science Education and Learning

Modelling in Science Education and Learning
*Volume* •, *No. 1*, •.
Instituto Universitario de Matemática Pura y Aplicada

# *Experiences with a Socio-Environmental Modeling Course*

**Pedro R. Andrade**
**Gilberto Camara**
National Institute for Space Research (INPE).
pedro.andrade@inpe.br,
gilberto.camara@inpe.br

**Tiago G. S. Carneiro**
Federal University of Ouro Preto (UFOP).
tiagogsc@gmail.com

**Raian V. Maretto**
**Antonio Miguel V. Monteiro**
National Institute for Space Research (INPE).
raian@dpi.inpe.br, miguel@dpi.inpe.br

**Flavia F. Feitosa**
Federal University of ABC (UFABC).
flavia.feitosa@ufabc.edu.br

**Abstract**

*In a social-environmental modeling course, students need to learn complementary skills that include the conceptualisation of a model, different modeling paradigms, computer programming, and the process of rigorously converting ideas and data into a computational program using a given toolkit. Such topics need to be taught in parallel in order to keep a heterogeneous audience motivated. Based on the experience with multidisciplinary audiences, this paper describes a socio-environmental modeling course that explores three modeling paradigms:* System dynamics, Cellular automata, *and* Agent-based modeling. *We also present a small tutorial with some examples developed for the course.*

Keywords: modeling paradigms, computer simulation, system dynamics, cellular automata, agent-based modeling, TerraME.

## 1.1 Introduction

There is an increasing awareness of the need to investigate real-world problems through approaches that overcome the arbitrariness and artificiality of representing social and ecological systems separately (Glaser, Krause, Ratter, & Welp, 2012). Such recognition, which claims for the study of so-called 'social-ecological systems (SESs)', imposes the challenge of bridging the gap between social and natural sciences and, therefore, combining the knowledge of researchers from different fields and traditions.

In the recent years, computational simulation models have been seen as a promising tool to represent the particularities of social-ecological systems, which are complex and adaptive, being characterised by biophysical and social agents that interact at multiple spatial and temporal scales (Janssen & Ostrom, 2006). More than simple representations of a system, models representing SESs have also the potential to allow researchers with different backgrounds to cross the boundaries of their disciplines by sharing insights and experiences. In other words, models can be built and understood as "boundary objects" (Star & Griesemer, 1989; Feitosa & Monteiro, 2012).

In a social-environmental modeling course, students need to learn complementary skills that include the conceptualisation of a model, different modeling paradigms, computer programming, and the process of rigorously converting ideas and data into a computational program using a given toolkit. Such topics need to be taught in parallel in order to keep a heterogeneous audience motivated. Based on the experience with multidisciplinary audiences, this paper describes a socio-environmental modeling course that explores three modeling paradigms: *System dynamics*, *Cellular automata*, and *Agent-based modeling*.

Following this introduction, the paper is organised as follows: First, we provide an overview about models and the three modeling paradigms explored in the course. Second, we describe the course and its technological aspects, including the toolkit adopted along the course. Afterwards, we present a small tutorial with some examples of our teaching experience. Finally, we conclude with some teaching experiences and final remarks.

## 1.2 Models and modeling paradigms

A model is generally understood as a purposeful and simplified representation of some system or any other aspect of the real world (Starfield, Smith, & Bleloch, 1993). There are many types of models, including, among others, verbal, mathematical, and graphical representations. Building models is a well-known way of improving our understanding of the world.

Computational simulation is a particular type of model. Unlike mathematical models, simulations are run rather than solved. Therefore, by overcoming many limitations imposed by mathematical tractability, simulation models allow researchers to address issues that require less simplified representations of the world (Railsback & Grimm, 2011). Such models introduce new possibilities to investigate coupled natural and human systems, including the ability of simulating the emergence of complex behaviors from relatively simple activities.

There are different world views, also called *paradigms*, to develop computational models for coupled natural and human systems. The system under study is then represented with the definitions available for that paradigm. The modeler needs to take into account the advantages and the limitations of the available paradigms in order to choose one that best addresses the problem under study. The socio-environmental modeling course presented in this paper explores three modeling paradigms: *Systems dynamics*, *Cellular automata*, and *Agent-based modeling*, which are presented as follows.

### 1.2.1 System dynamics

System dynamics is the most basic paradigm upon which all others are based. This paradigm starts from the assumption that a system under study can be isolated from the rest of the world. In this sense, we would have only linear dynamics between the system and the rest of the world. If the dynamics is not linear, it would need to be simplified to a linear dynamics or to be included in the model.

This paradigm views the world as a set of *stocks* and *flows*, assuming that everything that can be measured can also be modelled (Ford, 1999). Stocks can be anything, from the area covered by a lake to the number of individuals that have a given disease, from the amount of carbon stored in the atmosphere to the fat of a body. Flows describe mathematically how to transform or move stocks, making the system dynamic. They can connect two stocks, a stock with itself, or a stock with the external world, being always directed (Angerhofer & Angelides, 2000). One ore more flows can build a feedback loop that allows the system to present complex behavior. A model of System dynamics usually has two or more feedbacks that compete between them for the control over the simulation.

Using System dynamics models, it is possible to work with what-if scenarios to investigate the behavior of the system according to changes in its inputs or even changes in its internal behavior (Stave, 2003; Meadows, Meadows, Randers, & Behrens III, 1972). These scenarios might be developed from scratch, can be result of proposed public policies (Stave, 2002; Ghaffarzadegan, Lyneis, & Richardson, 2011), or based on outcomes of other models such as the ones developed by IPCC (Field, Barros, Dokken, et al., 2014). One important property these scenarios can investigate is whether the system is capable of adapting itself to exogenous changes, keeping its internal state stable. This property is called *homeostasis*.

The main disadvantage of this paradigm comes from its simplicity in representing the world as stocks. Individuals have to be aggregated in quantities, percentages, or averages. Space is constrained to measures of volume or area, with everything distributed homogeneously within the study area. The connections between stocks cannot change along the simulation. These constraints can be feasible in some studies, but they might become too much restricting if the spatial distribution of stocks is important in the problem under study, of when the individuals need to take their own decisions autonomously. These constraints are relaxed by the next paradigms.

### 1.2.2 Cellular automata

The Cellular automata paradigm views the world as a space with a finite set of two-dimensional discrete and contiguous cells (Von Neumann, 1966). Each cell has a homogeneous internal content, which includes zero or more stocks and an internal state machine, called automaton. The simulation then runs as a sequence of discrete steps. In each step, the automaton state changes according to transition rules, defined based on its own state and the state of its neighbors. These changes occur in parallel in space, which means that, when a cell changes its state in a given time step, this change cannot cause any effect in the decision of the other cells in the same time step.

The Neighborhood is an important concept for Cellular automata, since it determines the interaction between the elements that build up the model (Hagoort, Geertman, & Ottens, 2008). If two cells are neighbors, it means that one exerts some sort of influence over the state of the other. Depending on the process being modelled, the neighborhood can be homogeneous or heterogeneous in space. Examples of homogeneous neighborhoods are the well-established von Neumann and Moore Neighborhoods. However, in some cases, these neighborhoods are not

enough to represent the complexities of the spatial relations, which may demand, for example, the definition of neighborhoods that include transportation networks as input, i.e., neighborhoods based on topologies.

This paradigm has become increasingly popular among environmental modelers because of its ability to simulate complex spatial patterns that emerge from simple parallel local interactions between neighboring cells (Couclelis, 2000; Wolfram, 1984). In addition, due to its explicit spatial representation, it has a direct compatibility with geospatial representations of the world used by Geographic Information Systems (GIS). During the last decades, it has been widely used in the studies of phisical processes (Hesselbarth & Göbel, 1991) and social process, like urban dynamics (Batty & Xie, 1994; Batty, 2007; Clarke & Gaydos, 1998) and epidemiology (Medeiros et al., 2011; Slimi, El Yacoubi, Dumonteil, & Gourbière, 2009).

Despite its ability to simulate the emergence of complex spatial patterns from simple local rules, the main disadvantage of this paradigm comes from the impossibility of representing mobile objects, once the cells are fixed in the space. Models that use Cellular automata for studying human behavior anthropomorphise the state variables of a cell, simulating human behavior as physical processes. However, these adaptations might not be enough to represent the complexity of human behavior.

### 1.2.3 Agent-based Modeling

The agent-based modeling paradigm views the world as multiple autonomous agents interacting within an environment. Agents can be understood as entities "situated within and a part of an environment, that senses that environment and acts on it, over time, in pursuit of its own agenda" (Franklin & Graesser, 1997). This definition emphasizes agent's features that have been conventionally identified as important: *autonomy*, *social ability*, *reactivity*, and *proactivity* (Wooldridge & Jennings, 1995). Like cellular automata, agents are autonomous. They are a separate locus of control, fully responsible for their actions and in charge of accomplishing their role. Although centralized authorities may exist as environmental constraints, there is no global or external flux of control dictating the agent's actions. This 'self-organization' of autonomous agents is what promotes the emergence of global patterns from the bottom-up (Macy & Willer, 2002). Second, agents have social ability and are able to interact with each other. Third, agents are reactive and capable of responding to stimuli coming from their environment. In addition, agents are proactive, which means that they exhibit goal-directed behavior by taking their own initiative (Wooldridge & Jennings, 1995; Zambonelli, Jennings, & Wooldridge, 2001).

The environment defines a space in which agents operate, serving as a support to their actions. The meaning and role of an environment depends on the system that is being modeled. In some situations, it may be neutral, with minimal or no effect on the agents or, in analogy to the real world, the environment may have an active role in providing the context for agents to perform their actions, to acquire information about the problem they have to solve, and to communicate with each other (Gilbert, 2008; Weyns, Schumacher, Ricci, Viroli, & Holvoet, 2005). In the latter case, the environment can be specified as an independent piece of software, such as cellular automata, that encapsulates its own roles in the ABM, including particular characteristics and dynamics that directly influence the agent's behavior and the emergence of complex structures (Gilbert, 2008; Weyns et al., 2005).

Interactions are also fundamental for an ABM, as they represent the main feature that distinguishes ABM from other paradigms. The agents' potential to locally interact with each other and their environment is the key to the simulation of the emergent properties of complex systems (Axelrod, 2003; Holland, 1998). For this reason, all ABM include some sort of interaction that involves transmission of knowledge or materials that can affect the behavior of the

recipients (Gilbert, 2004). The nature and sophistication level of these interactions may vary substantially depending on the roles assumed by the agents in a simulated system. In some cases, agents interact by simply perceiving the presence of their pairs in the surroundings, while other situations demand interactions based on the development and use of complicated communication means (Gilbert, 2008; Zambonelli et al., 2001). In general, ABM can present direct agent-agent interactions, indirect agent-agent interactions, and agent-environment interactions.

The agent-based modeling paradigm separates the spatial representation from the individuals that take decision, which makes it more complex than the other two paradigms. It means that this paradigm is particularly powerful for representing the complexity of coupled natural and human systems, but also more challenging and time consuming in terms of development.

## 1.3 Overall description of the course

The course presented in this paper teaches the three modeling paradigms presented in last section. It focuses on a multidisciplinary audience, supposing that most of the students do not have any modeling or programming background. Therefore, each lecture needs to tackle three challenges: modeling itself, computer programming, and toolkit concepts. The idea is to learn the three challenges in parallel in order to keep the audience motivated.

Every lecture has a modeling concept, a modeling problem, and one or more toolkit concepts that can be used to solve the problem. We start from the easiest (System dynamics) to the most difficult paradigm (Agent-based modeling). The first models are implemented from scratch with the students, running them each time a new piece is implemented. It helps the students to learn details about the programming language and how to formalize the knowledge about the dynamics of the system under study. At the end of the lecture, small variations of the model are presented to explore the effects of changing parts of the source code. We enforce that each small part of the source code is an explicit assumption of the model. The modeling concepts and the toolkit functionalities learned in the previous lectures are then reinforced at each class meeting.

During the course the students are invited to select a published paper using one of the three paradigms to be replicated as final project. At the end of the course, we compare the three paradigms, presenting their advantages and disadvantages. The modeller should keep in mind the properties of each one and choose the one that best addresses the problem under study. The complete description of the course is presented in Table 1. The tutorial presented in the Section 1.4 is a simplified version of the course.

Given its simplicity, it is easy to implement a System Dynamics model and there are visual tools, such as STELLA (Roberts, Andersen, Deal, Garet, & Shaffer, 1983), Vensim (Eberlein & Peterson, 1992), and Simile (Muetzelfeldt & Massheder, 2003), that help one to develop models following this paradigm without needing to learn any programming language. Several works have discussed the challenge of teaching programming, which some claim that the main difficulty found by novice programmers is not to learn the programming concepts, but how to apply them in practical situations (Lahtinen, Ala-Mutka, & Järvinen, 2005; Hadjerrouit, 1998; Milne & Rowe, 2002). In order to practice since the basic concepts, we decided to start using a programming language even for System dynamics because more complex models will need to be implemented along the course. It allows the students to be more comfortable with the development environment, learning concepts related to the programming and executing cycle, syntax errors, and language specificities.

In this course, we adopted TerraME (Terra Modeling Environment) as toolkit to implement the models presented along the course (Carneiro, Andrade, Câmara, Monteiro, & Pereira,

[1]These three concepts will not be presented in the tutorial.

Table 1 – Main topics and concepts of the environmental modeling course.

| Paradigm | Modeling Concepts | Models | Toolkit Concepts |
|---|---|---|---|
| System Dynamics | Discrete simulation | Water in the tube | Cell |
| | Feedback | Thermostat | Event |
| | Calibration | Population growth | Timer |
| | Homeostasis | Disease propagation | Observer |
| | Scenarios | | |
| Cellular Automata | Autonomy | Fire in the forest | CellularSpace |
| | Parallel processes | Hydrology | Neighborhood |
| | Emergence | Deforestation | Trajectory[1] |
| | | | Legend |
| Agent-based Modeling | Heterogeneity | Population growth | Agent |
| | Rationality and its limits | Schelling | Society |
| | Relations | Sugarscape | Environment |
| | | BAR problem | Group[1] |
| | | | SocialNetwork[1] |

2013). It is an open-source tool distributed under the GNU LGPL license and is available at www.terrame.org. In TerraME, models can be implemented using different modeling paradigms, or even combining them. It provides concepts that work as building blocks for model development, allowing the user to specify the spatial, temporal, and behavioral parts of a model.

There are other toolkits that could be used for this course, such as NetLogo (Tisue & Wilensky, 2004) or RePast (North, Collier, & Vos, 2006), but some design decisions of TerraME were thought to facilitate the learning process of programming a model, as it is the most difficult step for those that do not have strong programming background. We can cite the most important ones:

**Multiple paradigms into a single toolkit:** The user does not need to switch between toolkits in order to learn a new paradigm. It makes the modeler more confident and comfortable as it always have something that the modeller already knows when learning a new paradigm.

**Encapsulate complex programming concepts:** Allow the modeler to focus on the description of the model rather than creating data structures to represent data, functions to transverse it, and explicitly control the simulation. Avoid the need to define explicit loop structures and to access directly multi-dimensional data structures.

**Small set of concepts:** Following the idea presented by (Sloman, 1971), TerraME has a small core of concept to minimize the conceptual distance between the modeler's mental representation of concepts and the representation that the computer will accept. TerraME adopted the strategy of having little syntax to describe well-established models, avoiding the modeler to use advanced definitions to implement them. This guarantees a smooth learning curve to the modeler, who will learn and use new concepts, functions and parameters only when necessary.

**Data sources:** Data interface is one of the most time-consuming tasks a modeler needs to execute during the development of a model. In order to deal with real world geospatial data, TerraME provides an interface with an open source geospatial library called TerraLib (Câmara et al., 2008), which allows TerraME to integrate the models with geospatial data stored in different data sources. TerraLib supports open source database management

systems such as MySQL and PostgreSQL. The library has functions to read data in different formats and convert them into regular or irregular cellular spaces. It also ensures persistent storage and retrieval of modeling data.

**Scripting language:** TerraME was developed as an extension of Lua, an open-source interpreted language with extensible semantics (Ierusalimschy, de Figueiredo, & Celes, 1996). In the design of TerraME, Lua was chosen because it provides a clear and expressive language, easy to learn and use by modellers that do not have advanced expertise in computer programming. No need to compile the source code, automatic garbage collector.

**Visualization:** Built-in functions to query and visualize data, using charts, tables, maps, and databases. The visualization tools can be activated or deactivated without needing to change the structure of the model. The user does not need to control the execution loop to acquire signals and deliver them.

## 1.4 A small tutorial

This section presents a short tutorial based on the course introduced in last section[1]. We describe one model for each of the three modeling paradigms, implement them in TerraME, and investigate their outcomes.

### 1.4.1 Water in the tube

The first model of this tutorial was initially proposed by Donella Meadows and is called *water in the tube* (Meadows, 2008). This very simple model uses System dynamics to describe how water flows out of a tube. Initially the tube is filled with 40 gallons of water. During the simulation, water flows out of the tube at a constant rate of 5 gallons per second. The idea is to implement a model to simulate the amount of water in the tube over time.

In TerraME, a stock is described using the concept of **Cell**. It represents a spatial location which can have persistent as well as runtime attributes. Persistent attributes are loaded from databases or external files, while runtime attributes need to be initialized along the simulation. In the tube model, there is a single runtime attribute to represent the amount of water. Figure 1 shows a declaration of a cell named *tube* in TerraME with an attribute *water*.[2] Note that the source code does not represent units of measurement explicitly. It is up to the modeler to define that a given stock is measured as gallons, liters, or any other unit, using or converting it accordingly when defining flows. In the case of Figure 1, *water* is measured in gallons.

```
tube = Cell{water = 40}
```

Figure 1 – A tube with 40 galoons of water.

Before describing the flow, we need to indicate how the stock will be observed along the simulation. TerraME has a concept of **Observer** to indicate which and how outputs of the simulation will be visualized or saved to files or databases. Figure 2 describes an observer of *type* chart to plot the attribute water from the *subject* tube. Every time the subject *notifies* the observer is updated, as shown in the last line. It is necessary to notify in time zero to draw the initial amount of water in the chart.

[1] This tutorial supposes the reader already has some knowledge about Lua language. For a short introduction see the *Lua for TerraME* report, available at the webpage of TerraME.

[2] The source code presented in this paper was implemented using TerraME under development, which will be released as version 1.4.

```
o = Observer{
    subject = tube,
    type = "chart",
    attributes = {"water"}
}

tube:notify(0)
```

Figure 2 – Declaring an observer for the stock.

The behavioral part of the model is described using the concepts of **Event** and **Timer**. Event is a time instant when the simulation engine executes operations. It contains a function named *action* that describes part of the behavior of the model. Timer is a clock that registers, manages, and executes a set of events over time. It manages an event queue ordered according to their priorities and timestamps. It is also used to run the simulation. In the case of System dynamics models, we use them to describe flows. Figure 3 declares a timer with a single event that reduces the amount of water in the tube and notifies the observer of the tube. The action takes the event itself as argument and uses it to notify the observer with the current simulation time. In the end, the timer runs until it executes the event eight times.

```
timer = Timer {
    Event {action = function(event)
        tube.water = tube.water - 5
        tube:notify(event)
    end}
}

timer:execute(8)
```

Figure 3 – Declaring the flow and running the simulation.

Figure 4 shows the output of a simulation. In the final time the amount of water in the tube is zero as five gallons of water are removed from the tube each time step. Note that, if

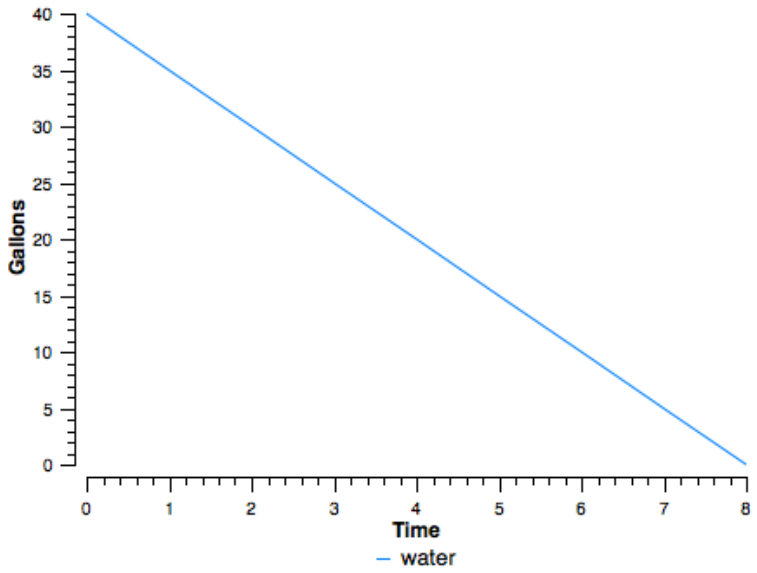



Figure 4 – Amount of water in the tube along the simulation, in gallons.

```
timer = Timer {
    Event {action = function()
        tube.water = tube.water - 5
    end},
    Event {period = 0.1, action = function(event)
        tube:notify(event)
    end}
}
```

Figure 5 – New timer defined for the water in the tube model.

we run the model until a time greater than eight, the simulation would produce a physically inconsistent output. As it keeps removing water out of the tube, the simulation would have negative amounts of water. Therefore, this simulation would be correct as long as we simulate the model to a time equal or less than the initial amount of water divided by the flow per second. Semantic problems such as this one are always up to the modeler, as the toolkit cannot identify them.

Executing the water in the tube gives the impression that time is continuous, as the observed chart is a line. However, in TerraME events are executed in discrete time steps. The default periodicity of each event is one, therefore it removes five gallons of water from the tube at once in each time step. We can see this clearly if we split the event in two, one for reducing the amount of water from the tube and another to observe the model with a more frequent period of observation. The observation event will now have a *period* 0.1, which means that it will occur ten times before water is taken out from the tube. Figure 5 describes the new timer with two events.

This new implementation has the same simple behavior of the previous one, but it has a different outcome, as shown in Figure 6. Instead of having a straight line that could be thought as a continuous simulation, now we have what really happens in the simulation. As the observation time is more frequent than the behavior of the model, it will observe a couple of times the same amount of water before it flows out from the tube. This is a property of discrete event simulators that needs to be taken into account when developing any model. Each

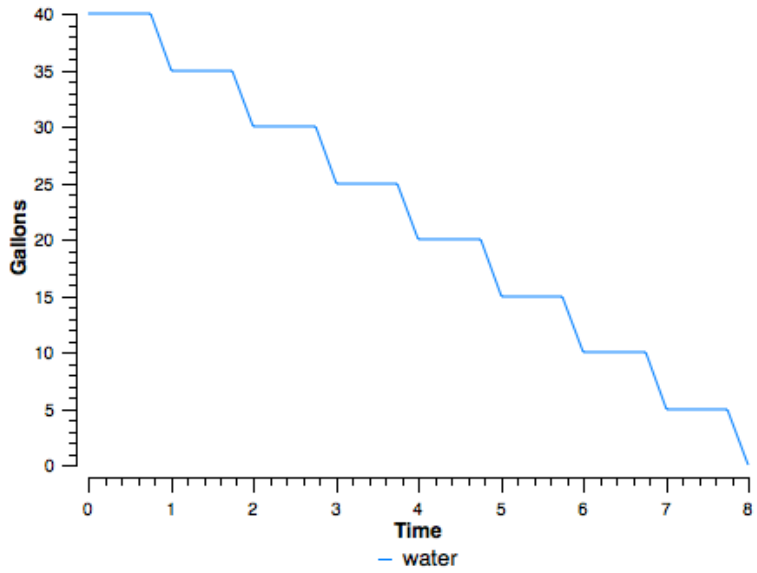



Figure 6 – Real amount of water in the tube along the simulation due to the discrete steps.

model has a temporal scale that should be defined a priori. In theoretical models it is possible to reduce the time scale by reducing the frequency of the events and the amount of flows. Models that use data usually have their temporal scale constrained by the frequency of data acquisition. In any case, one should never observe the model more frequently than it changes because it will not produce any new output.

By using the idea of a system composed by stocks and flows, it is possible to develop models that can produce complex behaviors. Flows can connect different stocks as well as a stock to itself to generate feedback loops, which is the main source of complexity in systems. TerraME also has other functionalities to deal with system dynamics such as events that have different initial times and observation of different attributes of a cell at the same time.

### 1.4.2 Fire in the forest

The second example simulates fire in a forest using the Cellular automata paradigm. The simulation starts with a small fire in one random cell of a forest completely filled with homogeneous trees. The fire then propagates to the neighboring forest cells on and on until all the forest is completely burned.

To represent this process with a Cellular automata we need to have three states: *forest*, *burning*, and *burned*. A forest cell will remain in this state until it finds a burning neighbor. A burning cell will become burned in the next time step, while a burned cell will not change its state anymore.

The implementation of the model starts with a definition of a cell, as presented in Figure 7. The cell will initially have an attribute called *cover*, which will be forest. It also has an update function to describe the behavior of the cell in each time step, implementing the transition rules of the automaton. In this function, the cell verifies its internal state. In the case of being forest, it needs to check whether any neighbor is burning. It uses a *second order* function to transverse its neighborhood. A second order function is a function that takes another function as argument. This function takes the cell itself and a neighbor cell and is executed for each of its neighbors. If any of the neighbor cells has a burning state then the cell also becomes burning. Otherwise, if the state of the cell is burning then it becomes burned. If the state is burned it will not change, therefore this situation is not described in the source code.

An important point in the description of the update function is the attribute *past*. In a Cellular automata, the process takes place in parallel in space. In this sense, the changes in the state of the a cell cannot affect the state of another cell in the same time step. Because of this,

```
cell = Cell{
    cover = "forest",
    update = function(self)
        if self.past.cover == "forest" then
            forEachNeighbor(self, function(self, neighbor)
                if neighbor.past.cover == "burning" then
                    self.cover = "burning"
                end
            end)
        elseif self.past.cover == "burning" then
            self.cover = "burned"
        end
    end
}
```

Figure 7 – Creating a cell of the forest.

```
world = CellularSpace{
    xdim = 50,
    ydim = 50,
    instance = cell
}

world:createNeighborhood{
    strategy = "vonneumann",
    self = false
}

world:sample().cover = "burning"
```

Figure 8 – Creating the forest with a random burning cell.

we need to work with two copies of the cells, one storing the present state and the other storing the past state. TerraME allows cells to have the attribute named past with the attributes of the cell in the previous time step. The methodology to implement a Cellular automata is then to read only from the past and write into the present. Therefore, every line that has an *if* uses *past*, while the lines that update states do not.

The second step to implement the model is to define the whole space. In TerraME, we can use a **CellularSpace** to represent the forest. It is a set of cells representing a spatial region under study, created from a cell and a description of its x and y dimensions, as shown in Figure 8. In the code, *world* is a cellular space with 2,500 cells initially with state forest. Any cellular space can have a **Neighborhood**, a set of cells defining the proximity relations of a given cell. World will have a von Neumann neighborhood, connecting each cell to its four touching neighbors. The final step to build the initial state of the forest is to initialize one random cell with burning state to start the fire, as shown in the last line.

To visualize the spatial distribution of cells, we need to define a legend to configure how states will be colored. The concept of **Legend** associates colors to attribute values in TerraME. We will paint forest as green, burning as red, and burned as brown. Using this legend, we can create an observer over the forest by selecting the attribute to be observed, as shown in Figure 9. Cellular spaces can also notify observers, as shown in the last line. Note that observers of

```
legend = Legend{
    grouping = "uniquevalue",
    type = "string",
    colorBar = {
        {value = "forest",  color = "green"},
        {value = "burning", color = "red"},
        {value = "burned",  color = "brown"},
    }
}

Observer{
    subject = world,
    attributes = {"cover"},
    legends = {legend}
}

world:notify()
```

Figure 9 – Defining a legend and an observer to visualize the states of the Cellular automata.

```
t = Timer{
    Event{action = function()
        world:synchronize()
        world:update()
        world:notify()
    end}
}

t:execute(50)
```

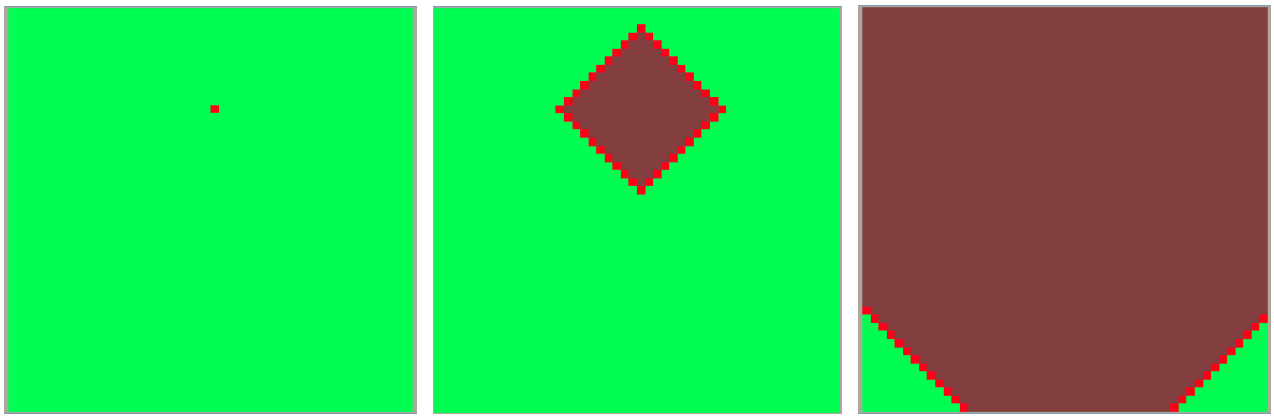
Figure 10 – A timer with an event for the fire in the forest model.

cellular spaces do not need the current simulation time as parameter, as they draw maps with the current distribution of states.

Finally, we need to define a timer with an event to simulate the model. The event executes three functions. First, it synchronizes the cellular space, copying the current attribute values to the past. Second, it updates the cellular space. It calls the function update for each cell of the cellular space, producing a new state based on the new past states, spreading the fire. Third, it notifies the observer to update the screen with the new states. Figure 10 shows the source code.

Figure 11 shows the output of the simulation in three different times. On the left, the image describes the beginning of the simulation, where there is only one burning cell. This cell is chosen randomly, which makes different simulations to start burning in a different cell. In the middle, we have the simulation after ten time steps, where the burning cells generate a square with burned cells inside. The square grows symmetrically in the four directions based on the initial burning cell. In the right we have the final state of the simulation. Executing the simulation until time 50 will not be enough to burn all the cells of the cellular space. If we simulate a couple more steps, the forest will still burn, but as we asked the simulation to stop in time 50 this was the final state. As the cellular space has $50 \times 50$ cells, the simulation would never burn all the cellular space in 50 steps, even if the fire starts in one of the four cells in the middle of the cellular space. To guarantee that all the cells are burned we would need to run the simulation with at least 99 steps.

Cellular automata is useful when one wants to model continuous processes that take place in a given region under study. It is used to study environmental processes such as hydrology,



(a) Initial state (b) After ten time steps (c) End of the simulation, after 50 time steps

Figure 11 – Simulation of forest fire in a $50 \times 50$ lattice. The forest is painted as green, burning as red, and burned as brown.

vegetation dynamics, as well as human processes, such as deforestation or urban growth. TerraME also has functionalities to model complex Cellular automata. Some of them are database interface, creating cellular spaces stored in external sources, visualization of different attributes at the same time, and strategies to create complex neighborhoods.

### 1.4.3 Population growth

In the last model of this tutorial we investigate a model of population growth that explicitly represents individuals through an agent-based model. In this model, agents can move freely in space and reproduce. The idea is to see how one can describe an agent-based model and investigate whether a very simple model can produce an emergent behavior.

The central concept in TerraME to work with Agent-based modeling is the idea of **Agent**. It is an autonomous individual that can have a set of attributes, relations with other agents as well as with spatial partitions, and autonomy to take decisions. The agent in the population growth model has a single function *execute* that describes its behavior, as shown in Figure 12. First, it selects a random cell from the neighborhood of its current cell. If such cell is empty, then with 30% of probability it reproduces and puts its child there. Then it gets another random cell and moves to there if it is empty, leaving the previous cell empty. Note that if the agent is surrounded by other agents it cannot take any action.

The next step of the population growth model is to define a **Society** of agents. It is a set of agents with the same general behavior and attributes. This society will have ten agents and an *instance* describing the overall behavior of its agents, which is the agent that we have just defined. Every time an agent inside a society reproduces, the newborn will be added to the same society of its parent automatically.

The ten agents will be placed in a $100 \times 100$ cellular space with a Moore neighborhood, connecting a cell to its eight surrounding cells. To connect the agents to the cells we need an **Environment**, which aggregates space and behavior. In this model, we will use an environment to create a random placement of agents with at most one agent per cell. Figure 13 shows the source code.

Before running the simulation we need to define two observers, one to see the spatial distribution of agents and the other to see the amount of agents. The first observer is a map and the other is a chart. Then a timer is declared to execute the society. When the action of an event is a society, and not a function as in the other examples, the event first executes the agents of the society and then notifies its observers automatically. Figure 14 describes this part of the source code.

```
singleFooAgent = Agent{
    execute = function(self)
        cell = self:getCell():getNeighborhood():sample()
        if cell:isEmpty() and math.random() < 0.3 then
            child = self:reproduce()
            child:move(cell)
        end
        cell = self:getCell():getNeighborhood():sample()
        if cell:isEmpty() then
            self:move(cell)
        end
    end
}
```

Figure 12 – An agent that reproduces and moves.

```
soc = Society{
    instance = singleFooAgent,
    quantity = 10
}

cs = CellularSpace{
    xdim = 100,
    ydim = 100
}

cs:createNeighborhood()

e = Environment{cs, soc}

e:createPlacement{
    strategy = "random",
    max = 1
}
```

Figure 13 – Putting agents in space randomly.

The evolution of the simulation is quite simple, with the spatial distribution of agents along the simulation shown in Figure 15. Empty cells are drawn as white and occupied ones as black. As the agents can move randomly, there is always a heterogeneous distribution of agents around the areas where the first agents occupied in the beginning of the simulation. If the agents did not move, the simulation would produce a continuous patch of agents. We can see that if we simulate for more a couple of time steps the population will fill all available cells, reaching a stable state where nobody can reproduce or move.

Although very simple, this model generates an emergent behavior. Figure 16 summarizes the amount of agents along the simulation. We can see that the population grows according to a logistic curve. In the beginning, it has an exponential growth until around time 40 as there are no spatial constraints. As the model evolves, it becomes more and more difficult to reproduce because the neighborhood is full or because the border of the cellular space was reached, until the population reach the stable state. Note that this constraint is not explicitly represented in the model, but it emerges from the individual decisions of the agents along the simulation.

This model is non-deterministic, which means that every time it is simulated it will produce

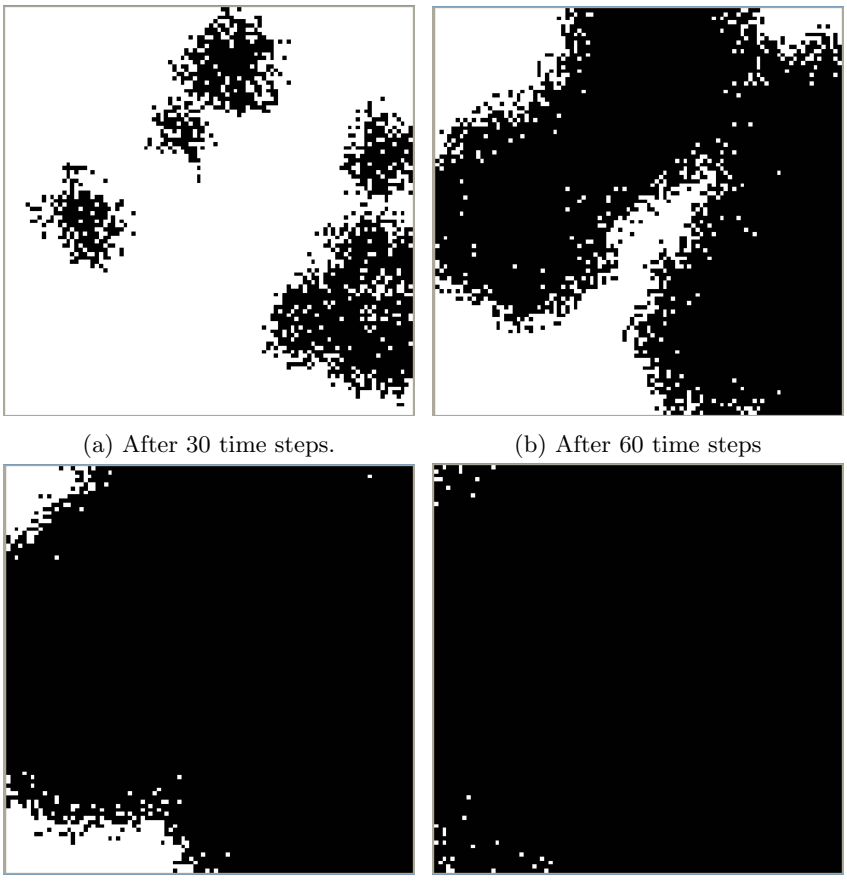
```
Observer{
    subject = soc,
    type = "chart"
}

Observer{
    subject = soc,
    type = "map"
}

t = Timer{
    Event{action = soc}
}

t:execute(120)
```

Figure 14 – Defining the observers and the timer for the population growth model.



(a) After 30 time steps. (b) After 60 time steps

(c) After 90 time steps (d) After 120 time steps

Figure 15 – Population growth in a $100 \times 100$ lattice. Black cells contain one agent, while white cells are empty.

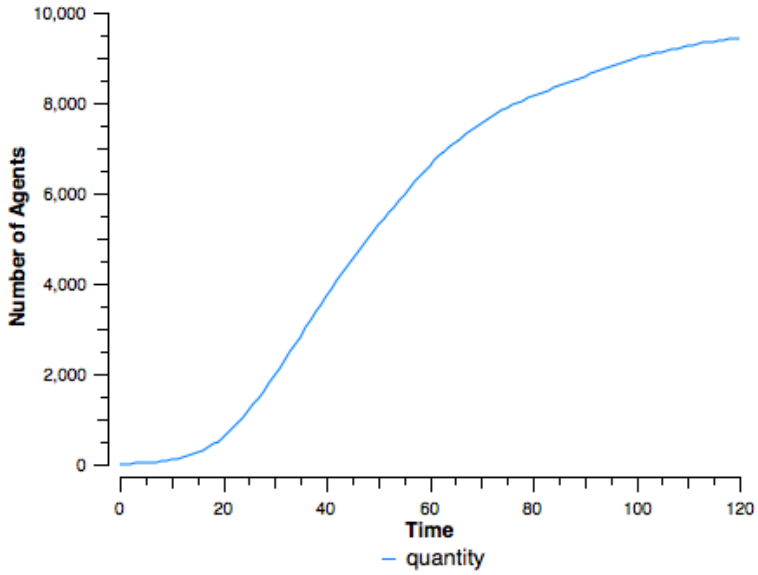



Figure 16 – Evolution of the population size along the simulation, using 30% of probability of reproduction.

different outcomes. In this situation, it is always interesting to repeat the simulation and perform statistical analysis about the stability of the results. Despite its randomness, running this model different times will produce similar outcomes. Agent-based modeling has no limits

in terms of what can be represented. Agents can represent any autonomous decision making entity. They can have a complex internal representation, message passing strategies, belong to social networks, and have multiple placements, among other capabilities. TerraME has functionalities to deal with all these challenges.

## 1.5 Final remarks

This course is taught yearly since 2010 in the Doctorate program of the Earth System Science Center (CCST) at the National Institute for Space Research (INPE), in Brazil. The course is also currently taught in the Graduate Program in Geoinformatics at the University of Münster, Germany and at the Erasmus Mundus graduate program in Münster. It introduces modeling to students with different backgrounds, including meteorology, mathematics, computer science, ecology, geography, and psychology.

The main barrier in the course is to teach the students how to build computer programs. Many of the students come into our courses with no programming background. This means they have to learn the basic concepts of programming (such as types, variables, conditional expressions, and loops) before they can start building models. The fact that Lua is the base language for TerraME is quite convenient for these kind of students. Lua's syntax is simple, direct, and yet powerful. Lua has only one data structure (table) and does not require explicit type declarations. Novice students can master the essential parts of Lua in two to four weeks.

We have designed the course material so as to allow students to learn modelling at the same time they are learning to program. As we discuss in this paper, the TerraME models presented in the course start from the more simple System Dynamics models, then we move into Cellular Automata, and then move on to Agent-Based Modelling. At each step, students learn concepts and have new problems to solve. Our experience has shown that students without programming experience adjust well to problem-based learning approaches.

At the end of the courses, it was possible to observe that all the students could understand the modelling properties and limitations and most of them were able to implement a model published in a peer-reviewed paper as final project. Nevertheless, there were students that were not able to develop their models since they were not interested in learning programming, considering it as a hard and painful process (Smith, Cypher, & Spohrer, 1994). In fact, despite the recognition of modelling as a powerful tool to subsidize the development of public policies, most decision-makers responsible to develop these policies fall in this last group. To turn models into attractive tools for these users, we are currently studying new mechanisms to publish models and to generate graphical interfaces that could be used to configure the parameters of the model. Through these mechanisms we believe to be possible to improve the use of the models by users that are interested in studying the dynamics of some processes, but do not want to learn computer programming.

The original course taught in 2010 was very different from the current version. Teaching the course along several years created a reinforcement loop that helped us to improve the course as well as the toolkit. It required four years of improvement to make the course simpler enough to be considered stable. Since the course is annual, new feedbacks will help us in the process of evaluating the improvements made and establishing new challenges to be addressed.

## References

Angerhofer, B. J., & Angelides, M. C. (2000). System dynamics modelling in supply chain management: research review. In *Simulation conference, 2000. proceedings. winter* (Vol. 1, pp. 342–351).

Axelrod, R. (2003). Advancing the art of simulation in the social sciences. *Japanese Journal for Management Information System*, *12*(3), 3–16.

Batty, M. (2007). *Cities and complexity: understanding cities with cellular automata, agent-based models, and fractals*. The MIT press.

Batty, M., & Xie, Y. (1994). From cells to cities. *Environment and planning B*, *21*(7), 31–48.

Câmara, G., Vinhas, L., Ferreira, K. R., Queiroz, G. R. D., Souza, R. C. M. D., Monteiro, A. M. V., ... Freitas, U. M. D. (2008). Terralib: An open source gis library for large-scale environmental and socio-economic applications. In G. Hall & M. Leahy (Eds.), *Open source approaches in spatial data handling* (Vol. 2, pp. 247–270). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-540-74831-1_12 doi: 10.1007/978-3-540-74831-1_12

Carneiro, T. G. S., Andrade, P. R., Câmara, G., Monteiro, A. M. V., & Pereira, R. R. (2013). An extensible toolbox for modeling nature-society interactions. *Environmental Modelling & Software*, *46*(0), 104–117. Retrieved from http://www.sciencedirect.com/science/article/pii/S1364815213000534 doi: http://dx.doi.org/10.1016/j.envsoft.2013.03.002

Clarke, K. C., & Gaydos, L. J. (1998). Loose-coupling a cellular automaton model and gis: long-term urban growth prediction for san francisco and washington/baltimore. *International Journal of Geographical Information Science*, *12*(7), 699–714.

Couclelis, H. (2000). Modeling frameworks, paradigms, and approaches. *Geographic information systems and environmental modeling*, 36–50.

Eberlein, R. L., & Peterson, D. W. (1992). Understanding models with vensim. *European journal of operational research*, *59*(1), 216–219.

Feitosa, F. F., & Monteiro, A. M. V. M. (2012). Vulnerabilidade e modelos de simulação como estratégias mediadoras: Contribuição ao debate das mudanças climáticas e ambientais. *Geografia (Rio Claro)*, *37*(2), 289–305.

Field, C., Barros, V., Dokken, D., et al. (2014). Climate change 2014: impacts, adaptation, and vulnerability. *Volume I: Global and Sectoral Aspects. Contribution of Working Group II to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Retrieved from http://ipcc-wg2.gov/AR5/

Ford, F. A. (1999). *Modeling the environment: an introduction to system dynamics models of environmental systems*. Island Press.

Franklin, S., & Graesser, A. (1997). Is it an agent, or just a program?: A taxonomy for autonomous agents. In J. MÃ¼ller, M. Wooldridge, & N. Jennings (Eds.), *Intelligent agents III: agent theories, architectures, and languages* (Vol. 1193, pp. 21–35). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/BFb0013570 doi: 10.1007/BFb0013570

Ghaffarzadegan, N., Lyneis, J., & Richardson, G. P. (2011). How small system dynamics models can help the public policy process. *System Dynamics Review*, *27*(1), 22–44. Retrieved from http://dx.doi.org/10.1002/sdr.442 doi: 10.1002/sdr.442

Gilbert, N. (2004). Agent-based social simulation: dealing with complexity. *The Complex Systems Network of Excellence*, *9*(25), 1–14.

Gilbert, N. (2008). *Agent-based models*. London: Sage Publications.

Glaser, M., Krause, G., Ratter, B. M., & Welp, M. (2012). *Human-nature interactions in the anthropocene: Potentials of social-ecological systems analysis*. New York/London: Routledge. Retrieved from http://isbnplus.org/9780415510004

Hadjerrouit, S. (1998). Java as first programming language: a critical evaluation. *ACM SIGCSE Bulletin*, *30*(2), 43–47.

Hagoort, M., Geertman, S., & Ottens, H. (2008). Spatial externalities, neighbourhood rules and ca land-use modelling. *The Annals of Regional Science*, *42*(1), 39–56.

Hesselbarth, H. W., & Göbel, I. R. (1991). Simulation of recrystallization by cellular automata. *Acta Metallurgica et Materialia*, *39*(9), 2135 – 2143. Retrieved from http://www.sciencedirect.com/science/article/pii/0956715191901832 doi: http://dx.doi.org/10.1016/0956-7151(91)90183-2

Holland, J. H. (1998). *Emergence: from chaos to order*. Oxford University Press.

Ierusalimschy, R., de Figueiredo, L. H., & Celes, W. F. (1996). Lua: an extensible extension language. *Software Practice & Experience*, *26*(6), 635–652. Retrieved from http://dx.doi.org/10.1002/(SICI)1097-024X(199606)26:6<635::AID-SPE26>3.0.CO;2-P doi: 10.1002/(SICI)1097-024X(199606)26:6⟨635::AID-SPE26⟩3.0.CO;2-P

Janssen, M. A., & Ostrom, E. (2006). Governing social-ecological systems. In L. Tesfatsion & K. Judd (Eds.), (Vol. 2, pp. 1465 – 1509). Elsevier. Retrieved from http://www.sciencedirect.com/science/article/pii/S1574002105020307 doi: http://dx.doi.org/10.1016/S1574-0021(05)02030-7

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In *Proceedings of the 10th annual sigcse conference on innovation and technology in computer science education* (pp. 14–18). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1067445.1067453 doi: 10.1145/1067445.1067453

Macy, M. W., & Willer, R. (2002). From factors to actors: Computational sociology and agent-based modeling. *Annual Review of Sociology*, *28*, 143–166. Retrieved from http://www.jstor.org/stable/3069238

Meadows, D. H. (2008). *Thinking in systems: A primer*. Chelsea Green Publishing.

Meadows, D. H., Meadows, D., Randers, J., & Behrens III, W. W. (1972). *The limits to growth: A report for the club of rome's project on the predicament of mankind*. New York: Universe.

Medeiros, L. C. d. C., Castilho, C. A. R., Braga, C., Souza, W. V., Regis, L., & Monteiro, A. M. V. (2011). Modeling the dynamic transmission of dengue fever: investigating disease persistence. *PLOS neglected tropical diseases*, *5*(1), e942.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming-views of students and tutors. *Education and Information Technologies*, *7*(1), 55–66. Retrieved from http://dx.doi.org/10.1023/A%3A1015362608943 doi: 10.1023/A:1015362608943

Muetzelfeldt, R., & Massheder, J. (2003). The simile visual modelling environment. *European Journal of Agronomy*, *18*(3), 345–358.

North, M. J., Collier, N. T., & Vos, J. R. (2006, January). Experiences creating three implementations of the RePast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, *16*(1), 1–25. Retrieved from http://doi.acm.org/10.1145/1122012.1122013 doi: 10.1145/1122012.1122013

Railsback, S., & Grimm, V. (2011). *Agent-based and individual-based modeling: A practical introduction*. Princeton University Press. Retrieved from http://books.google.com.br/books?id=VQOjinaRG9cC

Roberts, N., Andersen, D. F., Deal, R. M., Garet, M. S., & Shaffer, W. A. (1983). *Introduction to computer simulation: the system dynamics approach*. Addison-Wesley Publishing Company.

Slimi, R., El Yacoubi, S., Dumonteil, E., & Gourbière, S. (2009). A cellular automata model for chagas disease. *Applied Mathematical Modelling*, *33*(2), 1072 – 1085. Retrieved from http://www.sciencedirect.com/science/article/pii/S0307904X07003563 doi: http://dx.doi.org/10.1016/j.apm.2007.12.028

Sloman, A. (1971). Interactions between philosophy and artificial intelligence: The role of intuition and non-logical reasoning in intelligence. *Artificial Intelligence*, *2*(3–4), 209–225. Retrieved from http://www.sciencedirect.com/science/article/pii/0004370271900117 doi: http://dx.doi.org/10.1016/0004-3702(71)90011-7

Smith, D. C., Cypher, A., & Spohrer, J. (1994, jul). Kidsim: Programming agents without a programming language. *Commun. ACM*, *37*(7), 54 – 67. Retrieved from http://doi.acm.org/10.1145/176789.176795 doi: 10.1145/176789.176795

Star, S. L., & Griesemer, J. R. (1989). Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in berkeley's museum of vertebrate zoology, 1907-39. *Social Studies of Science*, *19*(3), 387–420. doi: 10.1177/030631289019003001

Starfield, A. M., Smith, K., & Bleloch, A. L. (1993). *How to model it: Problem solving for the computer age*. New York, NY, USA: McGraw-Hill, Inc.

Stave, K. A. (2002). Using system dynamics to improve public participation in environmental decisions. *System Dynamics Review*, *18*(2), 139–167. Retrieved from http://dx.doi.org/10.1002/sdr.237 doi: 10.1002/sdr.237

Stave, K. A. (2003). A system dynamics model to facilitate public understanding of water management options in las vegas, nevada. *Journal of Environmental Management*, *67*(4), 303 – 313. Retrieved from http://www.sciencedirect.com/science/article/pii/S0301479702002050 doi: http://dx.doi.org/10.1016/S0301-4797(02)00205-0

Tisue, S., & Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In *International conference on complex systems* (pp. 16–21).

Von Neumann, J. (1966). Theory of self-reproducing automata. *Burks, A. W. (Ed.)*.

Weyns, D., Schumacher, M., Ricci, A., Viroli, M., & Holvoet, T. (2005). Environments in multiagent systems. *The Knowledge Engineering Review*, *20*(2), 127–141.

Wolfram, S. (1984). Cellular automata as models of complexity. *Nature*, *311*(4), 419–24.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review*, *10*(02), 115–152.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2001). Organisational abstractions for the analysis and design of multi-agent systems. In *Agent-oriented software engineering* (pp. 235–251).