UniTs - University of Trieste

Faculty of Data Science and Artificial Intelligence

Department of mathematics informatics and geosciences

Multi-Agent Systems

Lecturer:
Prof. Fraile Beneyto, Raúl

Author:
Christian Faccio

October 26, 2025

github.com/christianfaccio christianfaccio@outlook.it

Preface

As a student of Data Science and Artificial Intelligence, I've created these notes while attending the **Multi-Agent Systems** course.

The course provides a comprehensive introduction to the field of multi-agent systems, covering both theoretical concepts and practical applications. The notes encompass a variety of topics, including:

- Intelligent Agents and Multi-Agent Systems
- Communication in Multi-Agent Systems
- Interacting by Cooperating
- Competition and Negotiation
- Agent-Based Modeling (ABM)
- Learning in Multi-Agent Systems
- Bio-inspired Systems and Swarm Intelligence
- ABM Analysis

While these notes were primarily created for my personal study, they may serve as a valuable resource for fellow students and professionals interested in multi-agent systems.

Contents

1 Introduction

Complex systems are evertwhere around us:

- Traffic systems;
- Disease spereading (e.g. COVID);
- dynamics of social networks;
- Natural ecosystems (take inspiration from nature evolution);
- Software systems.

Figure 1.1: Traffic.

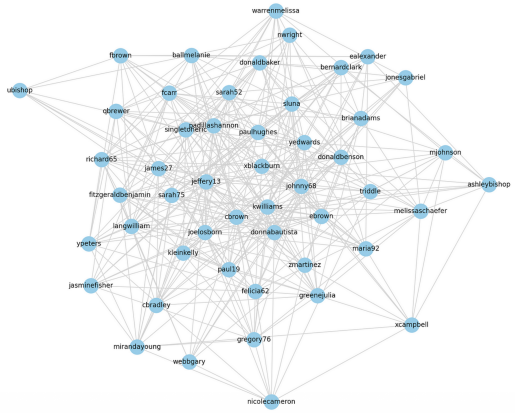

Figure 1.2: Social networks.

Definition: *Complex System*

A **Complex System** is composed by many components interactinv in a non-trivial way. A specific case could be: ”How can we analyze traffic in our city?”

- Traditional analytical methods;
- Simulation-based methods;
- Mathematical optimization;
- Directed graphs;
- ...

There are several approaches to the problem. We need to simplify the model first, otherwise there would be so many variables to consider that it would be impossible to analyze them all together. Then, we can focus on different aspects of the problem, using different models for each to capture the relevant dynamics. Finally, we can integrate these models to get a comprehensive understanding of the system as a whole. At the beginning it is a sort of *black-box*, in the sense that we do not know how the system works internally, but we can observe its inputs and outputs. The goal is to understand the internal mechanisms that drive the system’s behavior.

Definition: *Black-box models*

Black box models focus on modeling the relationships between inputs and outputs on a system, without explicitly considering the internal processes that generate those outputs from the inputs. They are widely used today and are based purely on observed input/output data from the system, without knowledge of the internal mechanism. See for example Neural Networks, statistical models or machine learning algorithms. They lack **explainability**.

We need something complex enough to capture the dynamics of the system, but simple enough to be tractable.

Agents are another way to model complex systems. Intelligent agents excel at modeling individual behavior and the emergence of global phenomena:

- Individual behavior is simple;
- Interactions are complex;
- Macroscopic behavior emerges from microscopic interactions.

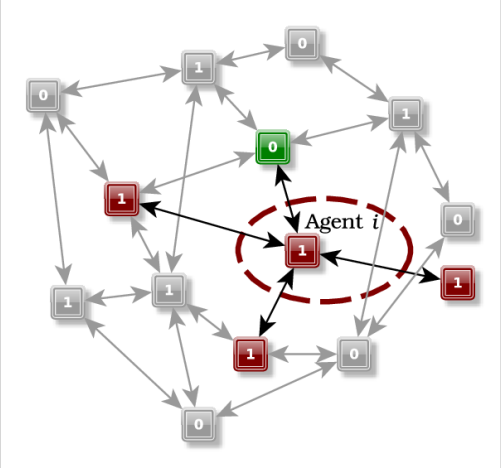

Figure 1.3: The world with agents

But how can we model a problem with agents? In the case of traffic, we can model each vehicle as an agent with individual behavior, represent traffic signals and lights as agents interacting with vehicles, incorporate pedestrians agents with their own behaviors and model the road environment as a graph of interconnected agents representing intersections and road segments.

Interactions emerge from local agent behavior and each agent models an individual component with autonomy.

Tip:

This is useful for simulation and comparing with real world scenarios.

At the end we can simulate different scenarios by modifying parameters, or analyze the emergence of traffic jams from individual behaviors, explore strategies to optimize circulation or calibrate and validate the model with real city data.

Intelligent Agent Systems are systems composed by multiple interacting agents, in which intelligence emerges from interactions among agents and between agents and the environment.

Such intelligence encompasses learning, reasoning, social interaction and other capabilities that are human-like.

Examples of intelligent agents can be found in various domains:

- An automated personal assistant that can make reservations, respond to emails and organize our schedule autonomously;
- An intelligent virtual tutor that can teach us new topics and adapt to our learning style;
- Autonomous cars or delivery drones.

Observation: ***Agents today***

Recent advancements include interactions with the user while maintaining **context**:

- **Memory Integration**: persistent memory to maintain long-term context;
- **Multi-modal Capabilities**: Understanding text and images;
- **Execution of actions**: interacting with external tools and APIs.

Advancements in cooperative AI has been also made, enabling agents to collaborate at solving problems in real-world applications such as in healthcare, logistics and resource management. Moreover, cognitive architectures are being developed to endow agents with human-like reasoning and decision-making capabilities, enhancing their ability to operate in complex and dynamic environments.

- **SOAR Cognitive Architecture**
- **ACT-R Cognitive Architecture**
- **Transformers**

2 How to use Gama Platform

In this chapter we will see how to use Gama Platform to create multi-agent simulations. Gama is an open-source platform for building spatially explicit agent-based simulations. It provides a user-friendly interface and a powerful modeling language that allows users to create complex models with ease. The generality of the agent-based approach advocated by GAMA is accompanied by a high degree of openness, which is manifested, for example, in the development of plugins designed to meet specific needs, or by the possibility of calling GAMA from other software or languages (such as R or Python)

- **Data-driven models**: GAMA offers the possibility to load and manipulate easily GIS (Geographic Information System) data in the models, in order to make them the environment of artificial agents;
- **GAML**: it is possible to create a simulated world, declare agent species, assign behaviors to them and display them and their interactions. GAML also offers all the power needed by advanced modelers: being an agent-oriented language coded in Java, it offers the possibility to build integrated models with several modeling paradigms, to explore their parameter space and calibrate them and to run virtual experiments with powerful visualization capabilities, all without leaving the platform;
- **Declarative User Interface**: The 3D displays are provided with all the necessary support for realistic rendering. A rich set of instructions makes it easy to define graphics for more dashboard-like presentations.

2.1 Platform

For more information and for visual examples, please refer to the official website: https://gama-platform.github.io/

GAMA consists of a single application that is based on the RCP architecture provided by Eclipse. Within this single application software, often referred to as a platform, users can undertake, without the need of additional third-parties softwares, most of the activities related to modeling and simulation.

2.1.1 Installation

1. **Download** → Go to the download page and pick the correct version for your operating system (Windows, MacOS, Linux);
2. **Install** → Follow the installation instructions for your operating system;
3. **Launch Gama**

Observation: *Docker*

Note that a Docker image of GAMA exists to execute GAMA Headless inside a container.

1. Install docker on your system;
2. Pull the GAMA's docker you want to use (e.g. `docker pull gamaplatform/gama:latest`);
3. Run this GAMA's docker with your headless command (e.g. `docker run gamaplatform/gama:latest -help`).

Note that GAMA can also be launched in two different other ways:

- In a so-called headless mode (i.e. without a user interface, from the command line, in order to conduct experiments or to be run remotely);
- From the terminal, using a path to a model file and the name or number of an experiment, in order to allow running this experiment directly.
  - `Gama.app/Contents/MacOS/Gama path_to_a_model_file#experiment_name_or_number` on Mac OS X
  - `Gama path_to_a_model_file#experiment_name_or_number` on Linux
  - `Gama.exe path_to_a_model_file#experiment_name_or_number` on Windows

GAMA will ask you to choose a workspace in which to store your models and their associated data and settings. The workspace can be any folder in your filesystem on which you have read/write privileges. If you want GAMA to remember your choice next time you run it (it can be handy if you run Gama from the command line), simply check the corresponding option.

The main window is then composed of several parts, which can be **views** or **editors**, and are organized in a **perspective**. GAMA proposes 2 main perspectives: *Modeling*, dedicated to the creation of models, and *Simulation*, dedicated to their execution and exploration. Other perspectives are available if you use shared models. The default perspective in which GAMA opens is *Modeling*.

It is composed of a central area where GAML editors are displayed, which is surrounded by a Navigator view on the left-hand side of the window, an Outline view (linked with the open editor), the Problems view, which indicates errors and warnings present in the models stored in the workspace and an interactive console, which allows the modeler to try some expressions and get an immediate result.

2.1.2 Workspace, Projects and Models

The workspace is a directory in which GAMA stores all the current projects on which the user is working, links to other projects, as well as some meta-data like preference settings, the current status of the different projects, error markers, and so on. Except when running in headless mode, GAMA cannot function without a valid workspace. The workspace is organized in 4 categories, which are themselves organized into projects.

The projects present in the workspace can be either directly stored within it (as sub-directories), which is usually the case when the user creates a new project, or linked from it (so the workspace will only contain a link to the directory of the project, supposed to be somewhere in the filesystem or on the network). A same project can be linked from different workspaces.

GAMA models files are stored in these projects, which may contain also other files (called resources) necessary for the models to function.

Observation: *Project and Model Status*

All the projects and models have an icon with a red or green circle on it. This eases to locate models containing compilation errors (red circle) and projects that have been successfully validated (green circle).

n the Navigator, models are organized in four different categories: Models library, Plugin models, Test models, and User models. This organization is purely logical and does not reflect where the models are actually stored in the workspace (or elsewhere). Whatever their actual location, model files need to be stored in projects, which may contain also other files (called resources) needed for the models to function (such as data files). A project may, of course, contain several model files, especially i

- **Library models**: This category represents the models that are shipped with each version of GAMA. They do not reside in the workspace but are considered as linked from it. This link is established every time a new workspace is created. Their actual location is within a plugin (msi.gama.models) of the GAMA application;
- **Plugin models**: This category represents the models that are related to a specific plugin (additional or integrated by default). The corresponding plugin is shown between parenthesis;
- **Test models**: These models are unit tests for the GAML language: they aim at testing each element of the language to check whether they produce the expected result. The aim is to avoid regression after evolutions of the software. They can be run from the validation view;
- **User models**: This category regroups all the projects that have been created or imported in the workspace by the user. Each project can be actually a folder that resides in the folder of the workspace (so they can be easily located from within the filesystem) or a link to a folder located anywhere in the filesystem (in case of a project importation). Any modification (addition, removal of files...) made to them in the file system (or using another application) is immediately reflected in the Navigator and vice-versa.

Tip:

Model files, although it is by no means mandatory, usually reside in a sub-folder of the project called `models`. Similarly, all the test models are located in the `tests` folder.

Warning:

It may happen, on some occasions, that the library of models is not synchronized with the version of GAMA that uses your workspace. This is the case if you use different versions of GAMA to work with the same workspace. In that case, it is required that the library be manually updated. This can be done using the ”Update library” item in the contextual menu.

Each model is presented as a node in the navigation workspace, including Experiment buttons and/or a Contents node and/or a Uses node and/or a Tags node and/or an Imports node.

Although GAML files are just plain text files, and can, therefore, be produced or modified in any text processor, using the dedicated GAML editor offers many advantages, among which the live display of errors and model statuses. A model can actually be in four different states, which are visually accessible above the editing area:

- Functional (grey color);
- Experimentable (green color);

- InError (red color);
- InImportedError (red color).

In its initial state, a model is always in the Functional state (when it is empty), which means it compiles without problems, but cannot be used to launch experiments. If the model is created with a skeleton, it is Experimentable.

2.1.3 Running Experiments

Running an experiment is the only way, in GAMA, to execute simulations on a model. Experiments can be run in different ways.

- The first, and most common way, consists in launching an experiment from the Modeling perspective, using the user interface proposed by the simulation perspective to run simulations;
- The second way, detailed on this page, allows to automatically launch an experiment when opening GAMA, subsequently using the same user interface;
- The last way, known as running headless experiments, does not make use of the user interface and allows to manipulate GAMA entirely from the command line.

All three ways are strictly equivalent in terms of computations (with the exception of the last one omitting all the computations necessary to render simulations on displays or in the UI).

There are two ways to run a GAMA experiment in headless mode: using a dedicated bash wrapper (recommended) or directly from the command line.

Bash Wrapper The file can be found in the headless directory located inside the GAMA's installed folder. It is named gama-headless.sh on macOS and Linux, or gama-headless.bat on Windows.

```
1      bash gama-headless.sh [m/c/t/hpc/v] $1 $2
```

Where:

- $1 input parameter file : an xml file determining experiment parameters and attended outputs;
- $2 output directory path : a directory which contains simulation results (numerical data and simulation snapshot);
- options `[-m/c/t/hpc/v]`
  - `-m` memory : memory allocated to gama
  - `-c` : console mode, the simulation description could be written with the stdin
  - `-t` : tunneling mode, simulation description are read from the stdin, simulation results are printed out in stdout
  - `-hpc` nb_of_cores : allocate a specific number of cores for the experiment plan;
  - `-v` : verbose mode. trace are displayed in the console

Java Command

```
1  java -cp \$GAMA_CLASSPATH -Xms512m -Xmx2048m -Djava.awt.headless=true
       org.eclipse.core.launcher.Main -application msi.gama.headless.id4 \
       $1 \$2
```

Where:

- `$GAMA_CLASSPATH GAMA` classpath: contains the relative or absolute path of jars inside the GAMA plugin directory and jars created by users;
- `$1` input parameter file: an XML file determining experiment parameters and attended outputs;
- `$2` output directory path: a directory which contains simulation results (numerical data and simulation snapshot).

2.2 Model Creation

A GAMA model is composed of three types of sections:

1. `global` : this section, that is unique, defines the ”world” agent, a special agent of a GAMA model. It represents all that is global to the model: dynamics, variables, actions. In addition, it allows to initialize the simulation (init block);
2. `species` and `grid` : these sections define the species of agents composing the model. Grid is defined in the following model step ”vegetation dynamic”;
3. `experiment` : these sections define the execution context of the simulations. In particular, it defines the input (parameters) and output (displays, files...) of a model.

Species A species represents a ”prototype” of agents: it defines their common properties. A species definition requires the definition of three different elements:

- the internal state of its agents (attributes). In addition to the attributes the modeler explicitly defines, species ”inherits” other attributes called ”built-in” variables;
- their behavior;
- how they are displayed (aspects). In order to display our prey agents we define two attributes: `size` and `color` .

Global The `global` section represents a specific agent, called `world` . Defining this agent follows the same principle as any agent and is, thus, defined after a species. The world agent represents everything that is global to the model: dynamics, variables... It allows to initialize simulations (`init` block): the world is always created and initialized first when a simulation is launched (before any other agents). The geometry (`shape`) of the `world` agent is by default a square with 100m for side size, but can be redefined if necessary.

The `init` section of the global block allows initializing the model which is executing certain commands, here we will create `nb_preys_init` number of prey agents. We use the statement `create` to create agents of a specific species: `create species_name +` :

- `number` : number of agents to create (int, 1 by default);
- `from` : GIS file to use to create the agents (optional, string or file);
- `returns` : list of created agents (list)

Experiment An `experiment` block defines how a model can be simulated (executed). Several experiments can be defined for a given model. They are defined using : `experiment exp_name type: gui/batch [input] [output]` :

- `gui` : experiment with a graphical interface, which displays its input parameters and outputs;

- `batch` : Allows to set up a series of simulations (w/o graphical interface).

Experiments can define (input) parameters. A parameter definition allows to make the value of a global variable definable by the user through the graphic interface.

```
1      parameter title var: global_var category: cat;
```

Where:

- `title` : the name of the parameter as it will appear in the interface (string);
- `var` : the name of the global variable associated with this parameter (string);
- `category` : the category in which this parameter will appear in the interface (string, optional).

Output blocks are defined in an experiment and define how to visualize a simulation (with one or more display blocks that define separate windows). Each display can be refreshed independently by defining the facet `refresh` nb (int) (the display will be refreshed every nb steps of the simulation). Each display can include different layers (like in a GIS):

- Agents species: `species my_species aspect: my_aspect` ;
- Agents lists: `agents layer_name value: agents_list aspect: my_aspect` ;
- Images: `image image_file` ;
- Charts.

Bibliography

[1] Michel Alexandre et al. “The financial network channel of monetary policy transmission: an agent-based model”. In: *Journal of Economic Interaction and Coordination* 18.3 (2023), pp. 533–571.

[2] Andrew Crooks et al. “Agent-based modelling and geographical information systems: a practical primer”. In: (2018).

[3] Cliff C Kerr et al. “Covasim: an agent-based model of COVID-19 dynamics and interventions”. In: *PLOS Computational Biology* 17.7 (2021), e1009149.

[4] Steven F Railsback et al. *Agent-based and individual-based modeling: a practical introduction*. Princeton university press, 2019.

[5] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[6] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[7] Jiefan Zhu et al. “An agent-based model of opinion dynamics with attitude-hiding behaviors”. In: *Physica A: Statistical Mechanics and its Applications* 603 (2022), p. 127662.