GAMA

Sommario

[1.INTRODUCTION 2](#_Toc108184747)

[2.ORGANIZATION OF A MODEL 3](#_Toc108184748)

[2.1.Model header(model species) 3](#_Toc108184749)

[2.1.1.Declaration name of the model 3](#_Toc108184750)

[2.1.2.Imports 4](#_Toc108184751)

[2.1.3.Definition of global species 4](#_Toc108184752)

[2.2.Species declaration 4](#_Toc108184753)

[2.3.Experiment declarations 5](#_Toc108184754)

[2.3.1.GUI Experiment 6](#_Toc108184755)

[2.3.2.BATCH Experiment 7](#_Toc108184756)

[2.3.3.Test experiment 7](#_Toc108184757)

[2.3.4.Memorize experiment 7](#_Toc108184758)

[2.4.Basic skeleton of a model 7](#_Toc108184759)

[3.BASIC PROGRAMMING CONCEPTS IN GAML 8](#_Toc108184760)

[4.MANIPULATE BASIC SPECIES 9](#_Toc108184761)

[4.1.GLOBAL SPECIES 9](#_Toc108184762)

[4.1.1.Declaration 9](#_Toc108184763)

[4.1.2.Environment size 10](#_Toc108184764)

[4.1.3.Built-in attributes 10](#_Toc108184765)

[4.1.4.Built-in actions 10](#_Toc108184766)

[4.1.5.The init statement 10](#_Toc108184767)

[4.2.REGULAR SPECIES 11](#_Toc108184768)

[4.2.1.Declaration 11](#_Toc108184769)

[4.2.2.Built-in attributes 11](#_Toc108184770)

[4.2.3.Built-in action 12](#_Toc108184771)

[4.2.4.The init statement 12](#_Toc108184772)

[4.2.5.The aspect statement 12](#_Toc108184773)

[4.2.6.Instantiate an agent 13](#_Toc108184774)

[4.3.DEFINING ACTIONS AND BEHAVIORS 14](#_Toc108184775)

[Declare an action 14](#_Toc108184776)

[Call an action 14](#_Toc108184777)

[Behavior 14](#_Toc108184778)

[Example 15](#_Toc108184779)

[4.4.INTERATION BETWEEN AGENTS 16](#_Toc108184780)

[The ask statement 16](#_Toc108184781)

[4.5.ATTACHING SKILLS 16](#_Toc108184782)

[4.6.INHERITANCE 16](#_Toc108184783)

[5.DEFINING ADVANCED SPECIES 17](#_Toc108184784)

[5.1.GRID SPECIES 17](#_Toc108184785)

[5.2.GRAPH SPECIES 17](#_Toc108184786)

[5.3.MIRROR SPECIES 17](#_Toc108184787)

[5.4.MULTI-LEVEL ARCHITECTURE 17](#_Toc108184788)

[CONTROL ARCHITECTURE 18](#_Toc108184789)

[Finite state machine 18](#_Toc108184790)

[Task-Based 18](#_Toc108184791)

[Rules-based architecture 18](#_Toc108184792)

[User control architecture 18](#_Toc108184793)

[USING EQUATIONS(molto interessante…da integrare quando ho tempo) 18](#_Toc108184794)

[TERMINOLOGIA 19](#_Toc108184795)

[Agent-oriented programming 19](#_Toc108184796)

[Software agent 19](#_Toc108184797)

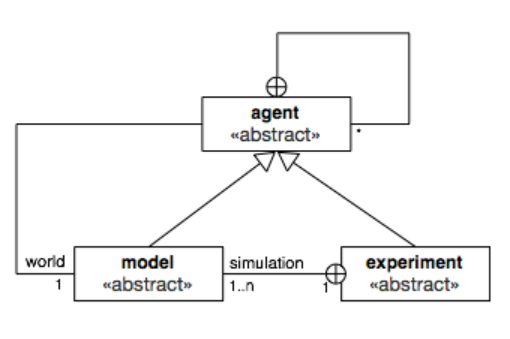
[RIFERIMENTI 20](#_Toc108184798)

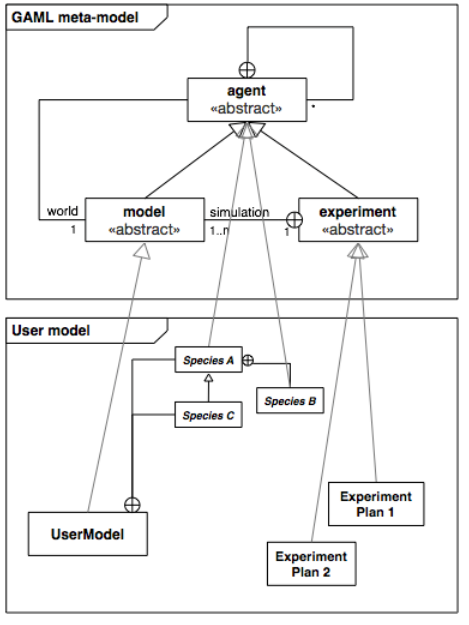
# 1.INTRODUCTION

GAML is an **agent-oriented language** dedicated to the definition of agent-based simulations. It takes its roots in object-oriented languages like Java or Smalltalk, but extends the object-oriented programming approach with powerful concepts (like skills, declarative definitions or agent migration) to allow for a better expressivity in models.

The role of GAML is to support modelers to write their **models**, which are specifications of **simulations** that can be executed and controlled during **experiments,** themselves specified by **experiment plans.**

Like in the object-oriented paradigm, where the notion of class is used to supply a specification of objects, agents in GAML are specified by their **species** which provides them with a set of **attributes (what they know), actions (what they can do), behaviors (what they actually do)** and also specifies properties of their **population,** for istance its **topology (how they are connected)** or **schedule (in which order and when they should execute).**

****



# 2.ORGANIZATION OF A MODEL

## 2.1.Model header(model species)

### 2.1.1.Declaration name of the model



Importing a model can take 2 forms. The first one, called **inheritance import,** in which all the declarations of the model imported will be merged with those of the current model.The second one, called **usage import,** reserved for **using models as micro-models of the current model.**

### 2.1.2.Imports

Inheritance import

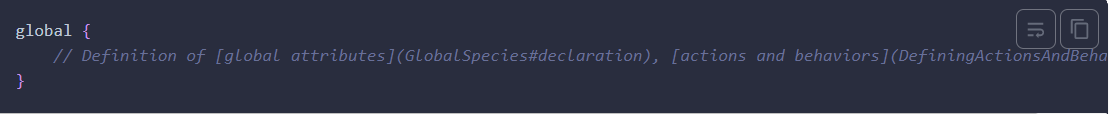


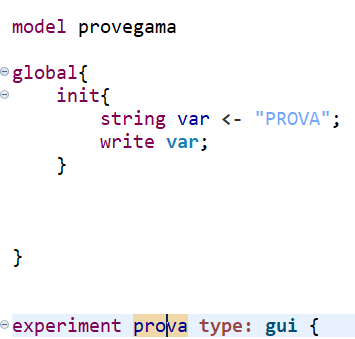
Usage import



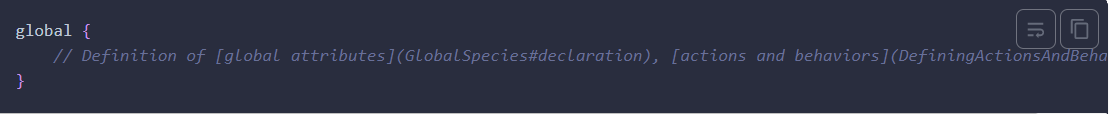
### 2.1.3.Definition of global species

The last part of the **header** is the definition of the global species:



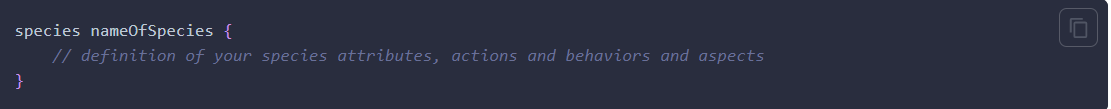


## 2.2.Species declaration

The **special species global** is the world species. You will declare here all the global attributes/actions/behaviors.

**Regular species** can be declared with the keyword **species.** You can declare several regular species, and they all have to be named.

A species defines its **attributes, actions and behaviors** and **aspects.**

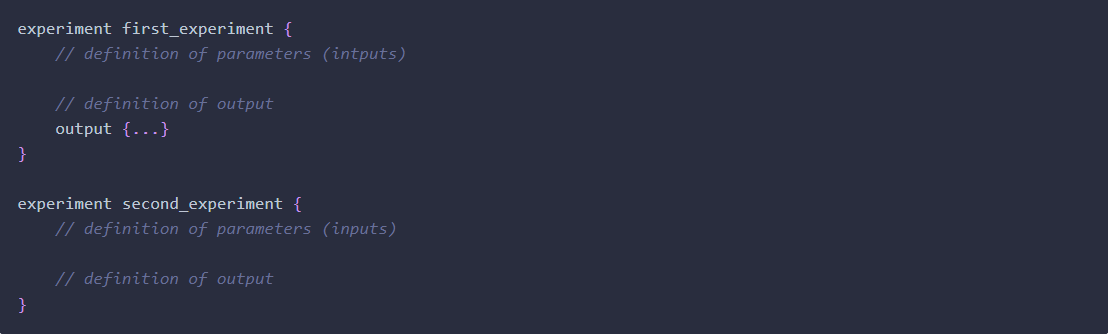
****

Species position:

**Immagine che contiene testo

Descrizione generata automaticamente**

## 2.3.Experiment declarations



### 2.3.1.GUI Experiment

GUI experiment allows you to display a **graphical interface**

****

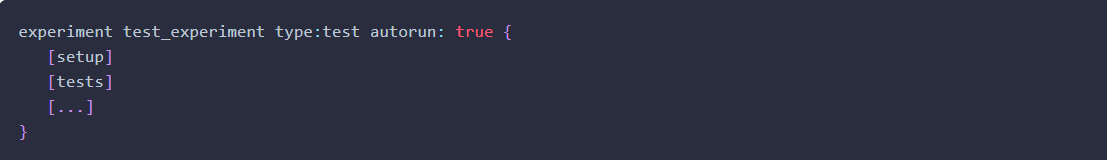
### 2.3.2.BATCH Experiment

**Batch experiment** allows you to execute numerous successive simulation runs (often used for model exploration). It is declared with the following structure:



### 2.3.3.Test experiment

**Test experiment** allows you to write unit tests on a model (used to ensure its quality). It is declared with the following structure:

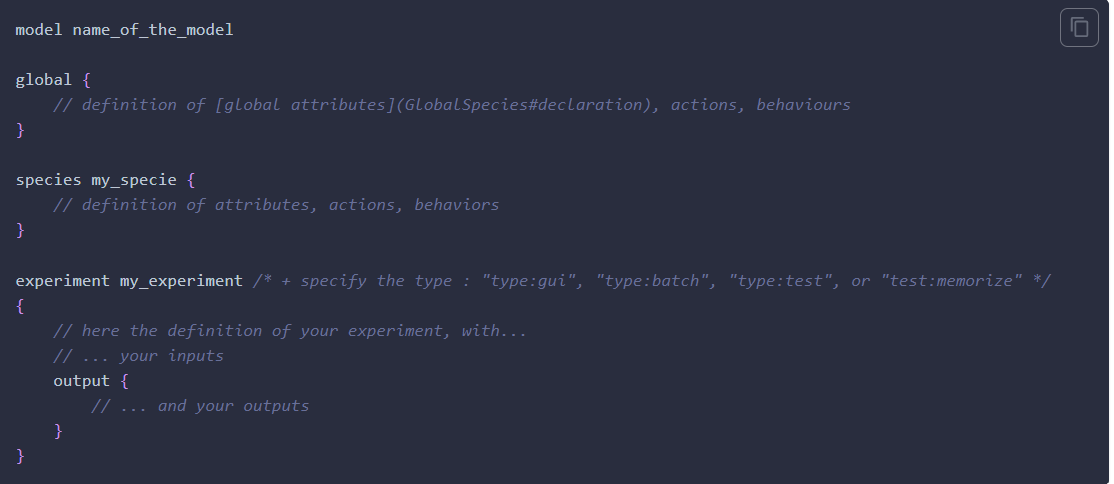


### 2.3.4.Memorize experiment

**Memorize experiment** allows you to store each step of simulation in memory and to backtrack to previous steps. Is is declared with the following structure:



## 2.4.Basic skeleton of a model



# 3.BASIC PROGRAMMING CONCEPTS IN GAML

Mettere un immagine con del codice ben commentato che illustra tutti questi concetti……

3.1.Variables

3.1.1.Basic types

3.1.2.The point type

3.1.3.A word about dimensions

3.2.Declare variables using facet

3.3.Operators in GAMA  
3.3.1.Matematical operators

3.3.2.Logical operators

3.3.3.Comparison operators

3.3.4.Type casting operators

3.3.5.Other operators

3.4.Loop

3.5.Manipulate containers

3.6.Random values

# 4.MANIPULATE BASIC SPECIES

In this chapter, we will learn how to manipulate some basic species.As you already know, a species can be seen as the definition of a type of **agent** (we call agent the **instance of a species**). In OOP (Object-Oriented Programming), a **species** can be seen as the class.Each species is then defined by some **attributes**(member in OOP), **actions** (method in OOP) and **behavior** (method in OOP).

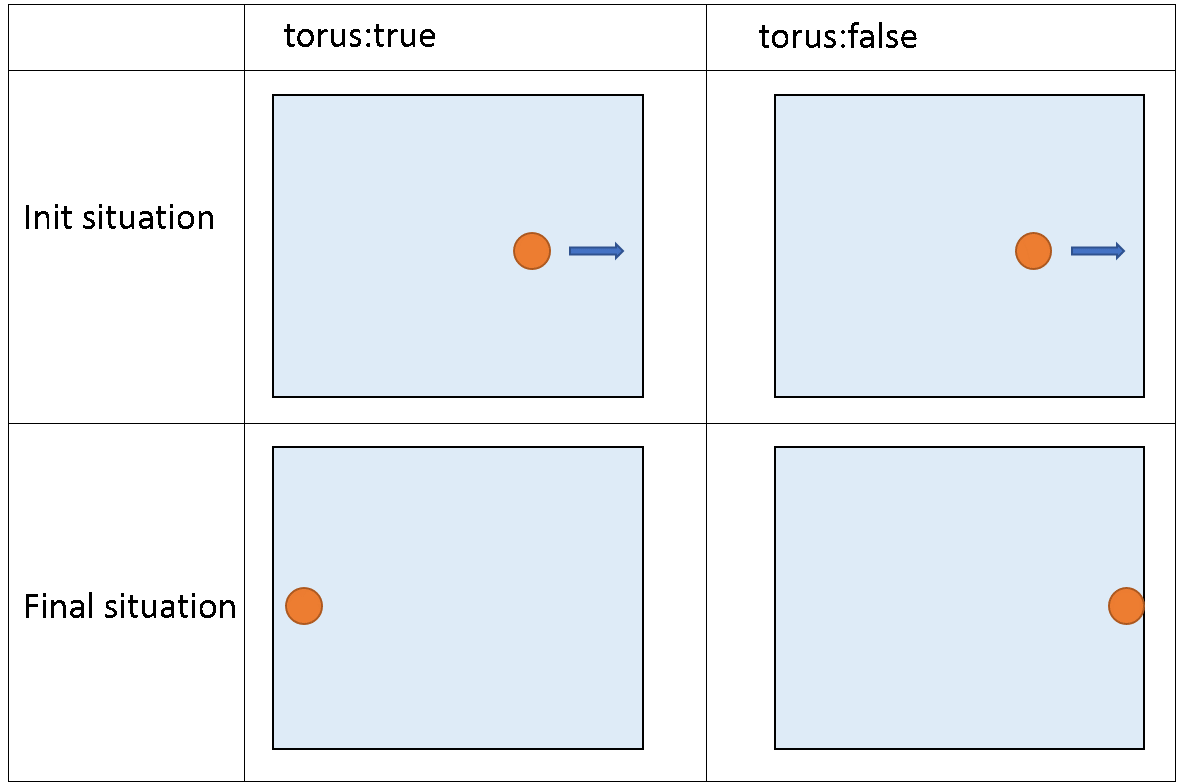
## 4.1.GLOBAL SPECIES

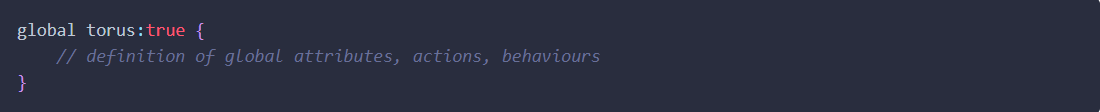
In the **global species** you can define the attributes, actions and behaviors that describe the world agent.

There is one **unique world agent per simulation:** it is this agent that is created when a user runs an experiment and that initializes the simulation through its **init scope**.

### 4.1.1.Declaration

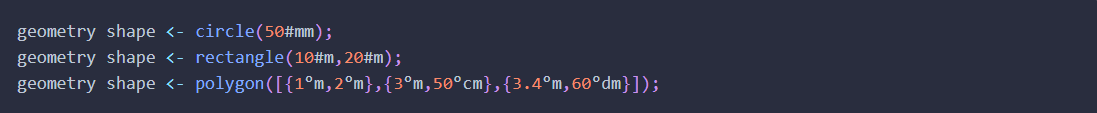
**Global** can use **facets**, such as the **torus** facet, to make the environment s **torus** or not (if it is a torus, all the agents **going out of the environment will appear on the other side**. If it’s not, the agent **won’t be able to go out of the environment**).





### 4.1.2.Environment size

In the **global context,** you have to define a **size** and a **shape** for your environment. In fact, an attribute already exists for the global species: it is called **shape,** and its type is a **geometry**. **By default**, **shape** is equal to a 100m\*100m square. You can change the geometry by affecting another value:



### 4.1.3.Built-in attributes

#### World

#### Experiment

#### Cycle

#### Step

#### Time

#### Starting\_date and current\_date

#### Duration

#### Total\_duration

#### Average\_duration

#### Machine\_time

#### Seed

#### agents

### 4.1.4.Built-in actions

#### Pause

#### die

### 4.1.5.The init statement

You can define an initial state, before launching the simulation.

Here, you normally initialize your global variables, and you instantiate your species.

## 4.2.REGULAR SPECIES

**Regular species** are composed of attributes, actions, reflex, aspect, etc…

They describe the behavior of our agents.You **can instantiate as much as you want agents** from a regular species, and you can instantiate as much as you want agents from a regular species, and you can define as much as you want different regular species.You can see a species as a “class” in OOP.

### 4.2.1.Declaration

Species with attribute



### 4.2.2.Built-in attributes

Immagine che contiene testo

Descrizione generata automaticamente

#### Name

#### Location

#### Shape

#### Host

#### Members

### 4.2.3.Built-in action

Some actions are defined by default for a minimal agent. We already saw quickly the action **write**, used to display a message in the console. Another very useful built-in action is the action **die,** used to destroy an agent.

Immagine che contiene testo

Descrizione generata automaticamente

### 4.2.4.The init statement

After declaring all the attributes of your species, you can define an initial state(before launching the simulation). It can be seen as the “**constructor of the class**” in OOP



### 4.2.5.The aspect statement

This block allows you to define how you want your species to be represented in the simulation.



In the experiment block, you have to tell the program to display a particular species with a particular aspect:

Immagine che contiene testo

Descrizione generata automaticamente

### 4.2.6.Instantiate an agent

As already said quickly in the last session, the instantiation of the agents is **most often** in the **init** scope of the **global** species (**this is not mandatory** of course.You can instantiate your agents from an action/behavior of any species).

Use the statement **create** to instantiate an agent.

The facet **with** is used to specify some some default values for some attributes of your instance.

Immagine che contiene testo

Descrizione generata automaticamente

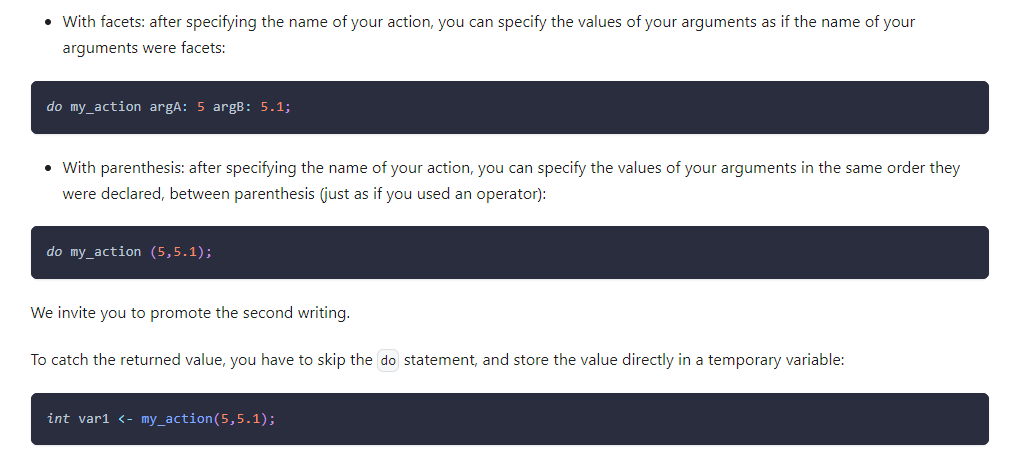
## 4.3.DEFINING ACTIONS AND BEHAVIORS

## Declare an action

An **action** is a **function or procedure** run by an instance of species. An action can return a value (in that case, the type of return has to be specified just before the name of the action), or not (in that case, you just have to put the keyword action before the name of the action).



## Call an action

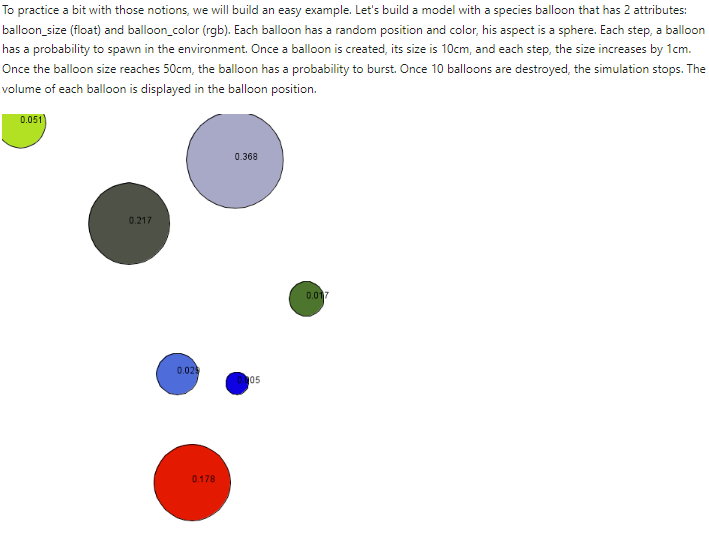


## Behavior

A **behavior** or **reflex** is a set of statements which is called **automatically** at each time step by an agent.Note that, a behavior is linked to an **architecture;** the reflex based architecture is the default one, others can be used with the controls facet of the species.



## Example

Immagine che contiene testo

Descrizione generata automaticamente

## 4.4.INTERATION BETWEEN AGENTS

### The ask statement

The **ask statement** can be used in any **reflex** or **action** scope.It is used to specify the interaction between the instances of your species and the other agents.You only have to specify the species of the agents you want interact with.

## 4.5.ATTACHING SKILLS

**Skills** are built in modules that provide a set of related **built in attributes and built in actions** (in addition to those already proposed by GAMA) to the species that declare them.

## 4.6.INHERITANCE

# 5.DEFINING ADVANCED SPECIES

## 5.1.GRID SPECIES

## 5.2.GRAPH SPECIES

## 5.3.MIRROR SPECIES

## 5.4.MULTI-LEVEL ARCHITECTURE

# CONTROL ARCHITECTURE

GAMA allows the modeller to attach built in control architecture **to agents**.

## Finite state machine

**FSM (Finite State Machine)** is a finite state machine-based behavior model. During its life cycle, the agent can be in several states. At any given time step, it is one single state. Such an agent needs to have one single state (the state in which it will be at its initialization).

## Task-Based

**Task-based**

## Rules-based architecture

## User control architecture

# USING EQUATIONS(molto interessante…da integrare quando ho tempo)

# TERMINOLOGIA

## Agent-oriented programming

**Agent oriented programming** is a programming paradigm where the costruction of the software is centered on the concept of **software agents**.

**Agent oriented programming** can be viewed as a specialization of **object oriented programming.**

The **state of an agent** consists of components such as beliefs, decisions, capabilities, and obligations, for this reason the state of an agent is called its **mental state.**

## Software agent

In computer science a **software agent** is a computer program that acts for a user or other program in a relationship of agency, which derives from the Latin agere(to do). **Agents** are colloquially known as **bots** or **robots.**

They may be embodied, as when execution is paired with a robot body, or as software such as chatbot executing on a phone (Siri) or other computer device. Software agents may be **autonomous** or **work together** with other agent or people.

# RIFERIMENTI

[1] [Documentazione ufficiale di GAMA](https://gama-platform.org/)

[2] [Wikipedia: agent oriented programming](https://en.wikipedia.org/wiki/Agent-oriented_programming)

[3] [Wikipedia: software agent](https://en.wikipedia.org/wiki/Software_agent)

[4] [sciencedirect: agent oriented programming](https://www.sciencedirect.com/science/article/abs/pii/0004370293900349)