# Bacterial Agent - Final Report

Enrico Gnagnarella
enrico.gnagnarella@studio.unibo.it

Matteo Scucchia
matteo.scucchia@studio.unibo.it

July 2021

Bacterial Agent is a multi-agent system in which we will apply the arguments studied in the Autonomous Systems course. Our project turns out to be a reproduction of a bacterial colony, in which each bacterium will be a living being with a main goal to play, that is to survive and reproduce itself. These activities will be carried out within an environment shared with other bacteria. Each of them can coordinates with the other entities through the release of molecules in the environment. This method of communications allow them to cooperate in order to help each other in the surviving tasks.

The purpose of the simulator, therefore, is to show the possible scenarios of the life of a bacterial colony, being a set of collaborative autonomous entities.

# Contents

## List of Figures

# 1 Goal and Requirements

## 1.1 Goal

The main goal of our project is to develop a bacterial colony system. The focus of this project is the creation of smart bacteria with the main objective of surviving by feeding and reproducing. The environment hosts each entity of the system and it affects the life of bacteria indirectly, as a result of variations on nutrients. This field allows us to study and understand better many salient aspects of autonomous systems. The system will be a multi-agent system, where each bacterium is a living and intelligent entity, which interacts with the other bacteria, so bacteria are social and situated entities.

JaCaMo will be the platform that will accompany the development of the project. This framework will in fact help us in the realization of the artifacts of the system and the environment in which they lead their life.

## 1.2 System details

In order to clearify the requirements of the project, we want to expose a detailed explanation of the system.

A bacterium is a living being entity which has a certain quantity of life in the beginning. The level of the life decreases when the bacteria don't eat, and increases when the bacteria eat.

We identified two types of bacteria:

- herbivores, that eat the food produced by the environment
- carnivores, that eat the herbivores bacteria

Initially a set of food, herbivores bacteria and carnivores bacteria spawn randomly in the environment. Then only food will spawn randomly, bacteria must reproduce themselves. Bacteria reproduction takes place for separation, so a bacterium divides itself into two identical children bacteria. A bacterium, to divide itself, needs a certain quantity of life, so not always a bacterium can reproduce itself, and not always it is convenient.

Herbivores bacteria can communicate each other by releasing molecules to advise other herbivores that they found food. So, if an herbivore perceives a molecula coming from a direction, it starts moving in that direction in order to find food.

Carnivores aren't social entities, they cannot communicate with each other through molecules, but they can pick up herbivores molecules, in order to find a group of herbivores and go in that direction, to catch and eat them.

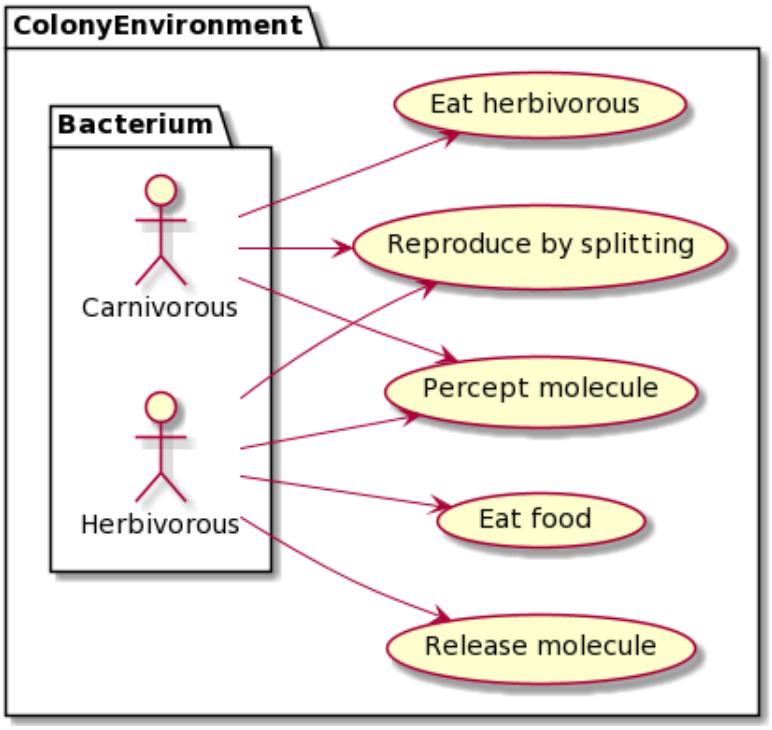


Figure 1: Use case of agent.

### 1.3 Requirements

Functional requirements of the system are:

- Creation of the environment.
- Creation of two types of bacteria (*herbivores, carnivores*)
- Realization of the surviving policy for each bacteria
  - policy of movement
  - policy of reproduction
  - policy of communication
- Realization of system artifacts
  - food produced by the environment
  - molecules, that dissolves themselves over time

The non-functional requirements, but still necessary to the goals achievement are:

- Study the JaCaMo framework, with which we could realize all entities of the system.

- Provide an interface to the user, with which he can see evolution and movement of bacteria in the colony.
- Develop of a build-automation system and CI pipelines for the project, through the Gradle platform and GitLab pipeline tool.

## 1.4 Expected outcomes

The expected result is to have an application that allows the vision of a bacterial colony during its life cycle. A simple and intuitive graphical interface will be available to monitor the status of the colony and its statistics. The group aims to create a system with the possibility of analyzing different scenarios as the number of food and the number herbivorous and carnivorous bacteria varies.

## 1.5 Scenarios

In our system, users can interact with the software only in one way. This way is to start the multi-agent system and see what happens inside it. Our project aims to create a system to analyze and study the behaviours of the JaCaMo agents in the context of a bacterial colony. So there aren't interaction between the system and users. However it's very interesting observe and analyze the behaviour of bacteria, how they communicate with each other and how the colony grows or dies in time.

## 1.6 Self-assessment policy

Our project was developed using the high-level programming language Java[3] and the languages defined by the JaCaMo[5] framework. Having never come into contact with this last platform before this course, we designed and developed the system only after becoming familiar with the tools with which we would go to work.

The project complies with the standards and best practices regarding the programming language and we have explored the little documentation available on JaCaMo in the best possible way. This have been done to create an agent system which exploits all the potentiality that the framework offers.

The effectiveness of the project can be verified by analyzing it from two different points of view: compliance with functional requirements and non-functional requirements.

The project is functioning and fully reflects the functional requirements that we set ourselves. The system was created through the use of different technologies and behaves correctly in its entirety. The agents actually interact in most cases as they should and the behaviors have been implemented in such a way as to guarantee the achievement of the goals established for each agent.

As for the non-functional requirements, it is clear that the development of this project forced us to carry out a thorough study of JaCaMo, with which we have reached a good level of knowledge.

# 2 Requirements Analysis

During this phase we interrogate ourselves about the possible technology to use to realize our system. We started from the requirements of the system to find the best solution.

As explained, we identified two types of bacteria, that we recognized as two possible agents. They must have their own autonomy, they must have their own life cycle, their own goals and plans to reach these goals.

In a generic approach to the problem, without relying only on the technologies seen during the Autonomous Systems course, the possible platforms available to be exploited for our project were many. Below is the list of the main ones analyzed:

- Common frameworks for agents, such as JADE or SPADE.
- Jason, a specific framework which acts as an interpreter for an extended version of AgentSpeak.
- JaCaMo, a more complete framework. It brings together the following three frameworks
  - Jason, the one described above,
  - Cartago, for the definition of an artifact-based environment,
  - Moise, a framework that allows to define an organization inside MAS.

## 2.1 Choice of technology

The SPADE and JADE frameworks are simple to understand and easy to implement. During our academic career, however, we have already worked with both. For this project, we preferred to explore new technologies to increase our knowledge base. Furthermore, in the course of Autonomous Systems we faced BDI agents, and we thought to them as the correct paradigm to realize our project and for this reason we obviously needed a framework that supports this paradigm.

Jason turns out to be an interesting platform in our eyes because it allows to create BDI agents, but not only. It's also possible to define the MAS and therefore it turns out to be complete.

Our final choice, however, fell on JaCaMo, as it includes additional frameworks (Cartago and Moise). The combination of these three technologies convinced us as it would allow us to develop a complete system with precise and detailed specifications for each area, such as the environment and the organization of entities. By comparing Cartago and Moise with Jason's internal management of environment and organization, we can say that both are valid alternatives but the idea of a dedicated framework for each context has guided our final choice.

## 2.2 JaCaMo framework

JaCaMo is a framework for Multi-Agent Programming that combines three separate technologies, each of them being well-known on its own and developed for a number of

years so they are fairly robust and fully-fledged. JaCaMo is a combination of:

- Jason – for programming autonomous agents
- Cartago – for programming environment artifacts
- Moise – for programming multi-agent organisations

thus covering all the levels of abstractions that are required for the development of sophisticated multi-agent systems.

### 2.2.1 Jason

Jason is an interpreter for an extended version of AgentSpeak. It implements the operational semantics of that language, and provides a platform for the development of multi-agent systems, with many user-customisable features. Jason is developed by Jomi F. Hübner and Rafael H. Bordini.

### 2.2.2 Cartago

CArtAgO (*Common ARTifact infrastructure for AGents Open environments*) is a general purpose framework that makes it possible to program and execute virtual environments for multi-agent systems.

CArtAgO is based on the Agents & Artifacts (A&A) meta-model for modelling and designing multi-agent systems. A&A introduces high-level metaphors taken from human cooperative working environments: agents as computational entities performing some kind of task, and artifacts as resources and tools dynamically constructed, used, manipulated by agents to support and realise their individual and collective activities.

CArtAgO makes it possible to develop and execute artifact-based environments, structured in open workspaces, that agents of different platforms can join so as to work together inside such environments.

### 2.2.3 Moise

Moise is an organisational model for Multi-Agent Systems based on notions like roles, groups, and missions. It enables a MAS to have an explicit specification of its organisation. This specification is to be used both by the agents to reason about their organisation and by an organisation platform that enforces that the agents follow the specification.

## 2.3 Agents

The BDI model(*beliefs, desires, intentions*) allows you to represent the characteristics and methods of achieving a goal in a system built according to the agent paradigm. Goal achievement is what the agent works for and it happens by determining the actions the agent is capable of taking.

In our project we have two types of bacteria, herbivores and carnivores, that are considered autonomous entities. They must encapsulate their own control flow, because they need to reach their own goals. For this reason we find in the BDI agent paradigm, that JaCaMo offers, the perfect solution.

## 2.4 Environment

The environment is the part of the system in which agents act and live. The bacteria can perceive the environment, whose tasks are to produce the food, hold molecules and bacteria information. Cartago is therefore a good solution, because the artifact-based environment can satisfy our requirements. This method of defining the environment allows its realization through independent elements, which respect the *Single Responsibility Principle*, as it avoids the creation of a single "monolithic" environment.

## 2.5 Organization

As for the organization, in our project there are no hierarchies or social structures among the various components of the domain. In our specific case there are two families of bacteria, but these are different from each other, so we will not use any paradigm or functionality that Moise provides.

# 3 Design

In the following section we describe the design of the system by three foundamental points of view:

- **Structure** in Section 3.1,
- **Behaviours** in Section 3.2,
- **Interactions** in Section 3.3.

## 3.1 Structure

In our system domain we identify some entities. Each of them needs to interoperate with each other, so we have to modeled correctly. A first division of the entities of the domain into groups, can be made by classifying them into passive and active entities.

### 3.1.1 Active entities

These entities are the active elements of the system, namely herbivorous bacteria and carnivorous bacteria. Both must try to survive, carrying out tasks such as eating and reproducing. The JaCaMo framework, which we have adopted, provides some design models. Active entities are therefore modeled as agents, with an internal status and goals to be accomplished. The state of the agents has been designed through the beliefs construct, while the goals of the agents have been modeled precisely with the concept of goals and plans, which allow the agent to act in the environment based on what they perceive.

### 3.1.2 Passive entities

The passive entities are instead food, molecules and their respective collections. These elements of the domain do not have specific behaviors. They are part of the domain and the active entities, mentioned in the previous paragraph, operate on them.

As far as passive entities are concerned, we studied and analyzed various approaches before choosing the one we considered the best. The options analyzed will be explained in more detail below.

**Possible structural solutions for passive entities** During the analysis process we read paper and study examples online, but no one of these has a similar aspect with our system. So we were in a situation in which we thought several different possible design and we needed to evaluate them. Below all the solutions are reported, that have been identified during the design process, and the pros and cons of each of them.

**The whole environment as a single artifact** This design was conceived with the idea of bringing together in one place all the aspects necessary for the environment and which represent it.

**Pros** It will be easy to implement in practice.

**Cons** Initially we thought there were no defects in this type of design, but following further investigations we verified how the environment should be *artifact-based*, therefore defined by the artifacts that compose it. Group all the concepts of the system into a single artifact is therefore logically wrong.

**Individual entities as artifacts** Another design choice we thought about was to model every single object in the system as artifacts, such as food, molecules and also the respective collections that would contain and manage these entities.

**Pros** These design choices would have the advantage of modeling each entity in a very realistic way.

**Cons** The disadvantage would be the management of every single entity (*food, molecule*), that needs a particular management within the JaCaMo framework and it's not easy at all.

**The collections of objects as artifacts** The last architectural choice on which we have reflected, which is also the decision pursued for the development of our system, is to model single entities (food, molecules) as objects. While the collections of these entities are modeled as artifacts.

**Pros** The advantage lies in the fact that the complexity of the system is reduced enough and in addition it's possible to define a simple set of operations with which agents can interact with the artifacts.

**Cons** The only disadvantage that we have managed to identify is the lower realism of this solution compared to the one previously discussed.

### 3.1.3 Final design of the system

After the above considerations, we have choosen the third solution, that is to model collection of elements as artifacts.

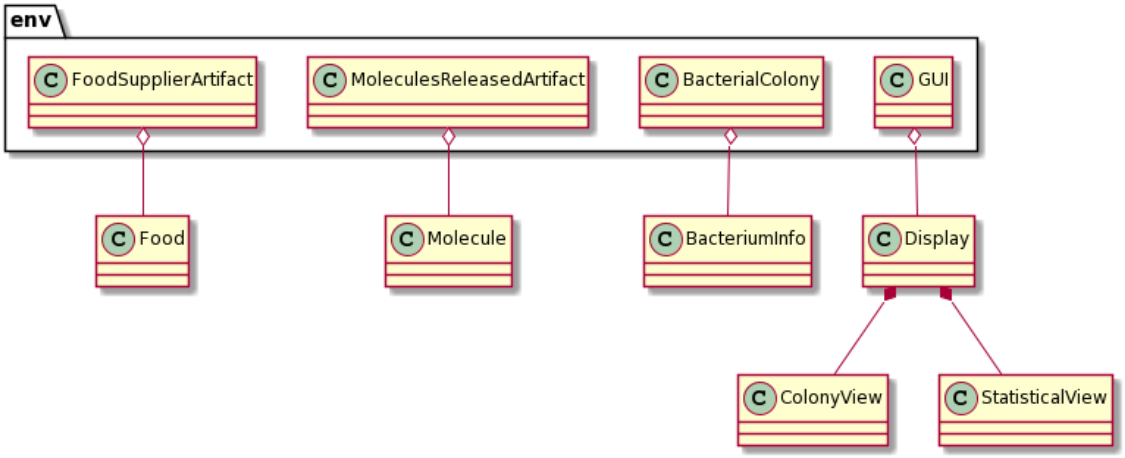


Figure 2: Class diagram of passive entities.

### 3.1.4 Entities modularization

We have modularized the software in such a way as to respect the structures recommended by the JaCaMo framework, dividing the agents from the artifact-based environment.

Within the system there are further components, external to JaCaMo's models, which have been designed by applying known patterns, to guarantee the realization of a high quality project. In our project we implemented a GUI to show the bacteria behaviours. To do this, we used the MVC pattern. It was not trivial to integrate an object oriented pattern as the MVC with an agent oriented framework as JaCaMo. We thought a lot about the possible solutions, that have already been explained in the section 3.1.2. Each of those solutions obviously would have been reflected in different implementative choices of the MVC pattern.

We did several different experiments and finally we implemented a solution that we think it's a good trade off between the JaCaMo philosophy and the clarity and the simplicity.

**Model** The model contains the informations of the entities of the application. For example, a bacterium has a position, a dimesion, an energy level and since it's implemented as an agent, it also has an ID.

**View** The view is the graphic part of the application. For each model object there is a correspective view object with only the information that is important for the visualization.

**Controller** The controller is a bridge between the model and the view. It does one main task: it converts entities from model objects to view objects, scaling them with

respect to the ratio between the model world dimension and the view world dimension, that is the screen size.

**MVC in JaCaMo** In JaCaMo agents act on artifacts. All the entities that are part of the environment are modelled as artifacts, and the agents can act with them. Our idea was to create an artifact that can be seen as an entry point for the GUI. In this way, an agent that we called *system manager* can observe the collections of entities of the enviroment and then it can also call operations on the GUI artifact, to update the GUI.

As said before it's a trade off. It's a centralized updating operation, that is repeated each 200 milliseconds. The alternative was to update the GUI each time a bacterium moves, or a food is created. You can easily understand that it's very computationally expensive, especially if the number of entities in the simulation increases.

It's not properly the JaCaMo phylosophy, in which each entity is responsible of its own stuff, but we think it's the best solution for our project.

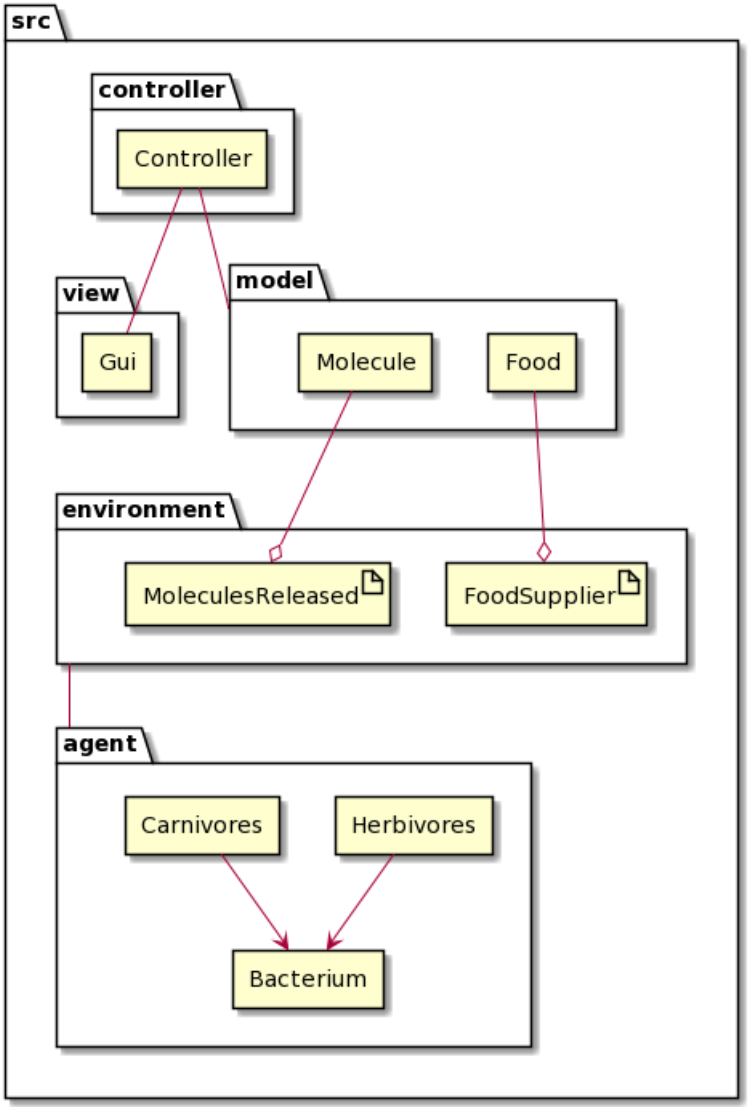


Figure 3: Deployment diagram of whole system.

## 3.2 Behaviour

Each active entity of the system has its own specific goals to execute, that characterize its behaviours in the colony.

### 3.2.1 Herbivorous Bacterium

This type of bacterium, during his life, has five main goal to perform:

- Move itself

- Eat food
- Release molecules
- Percept molecules
- Reproduce by splitting

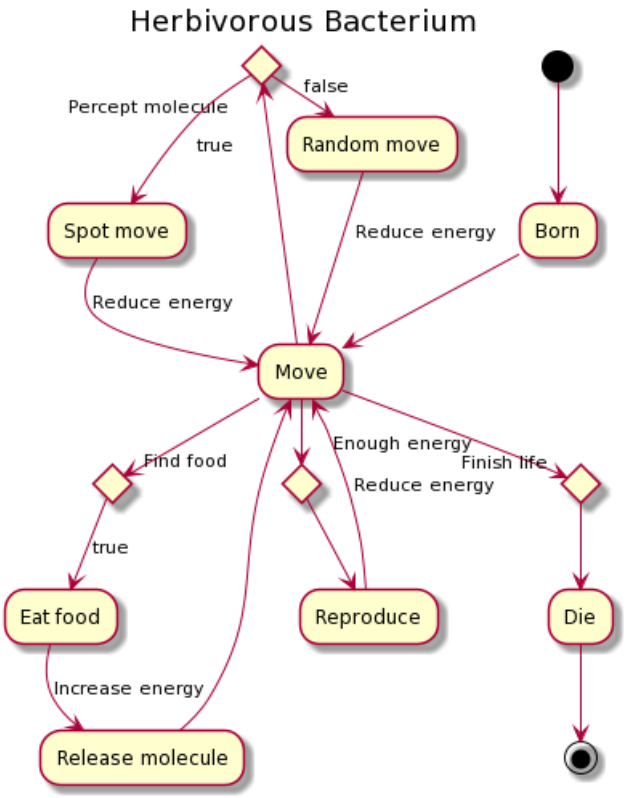


Figure 4: Activity diagram of herbivorous bacterium.

### 3.2.2 Carnivorous Bacterium

This type of bacterium, during his life, has four main goal to perform:

- Move itself
- Eat herbivorous bacteria
- Percept molecules
- Reproduce by splitting

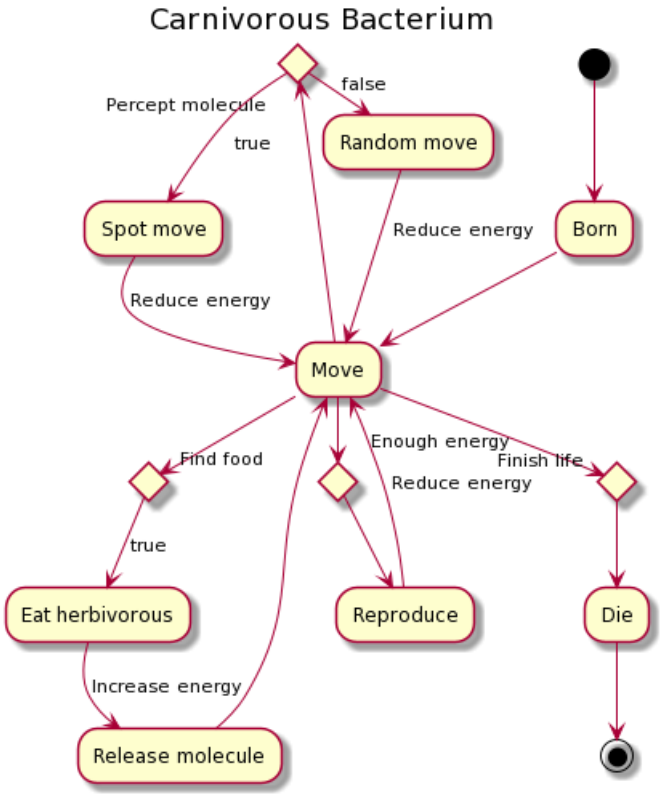


Figure 5: Activity diagram of carnivorous bacterium.

### 3.2.3 Food and Molecules

These entities are passive. They don't have a flow of internal activities to perform but are simply used from bacteria during their life to feed and identify food-rich areas.

## 3.3 Interaction

In our system agents can interact with each other or with artifacts, in the ways that JaCaMo framework allows.

### 3.3.1 Interactions between agents

Bacteria don't communicate with each other directly, except for a special case that is reproduction. They communicate with each other with a low level mechanism that is very common in nature: by the release of particular molecules in the environment.

Herbivorous bacteria are social entities that can communicate with each other with the molecules mechanism, to inform other herbivorous that food is present nearby. Carnivorous bacteria aren't social entities. They can't interact with each other, but they can percepts herbivorous molecules to move in an area where herbivores are present.

The only direct communication between bacteria takes place during the reproduction. Since JaCaMo doesn't allow to pass information to a new agent at creation time, because it conflicts with the autonomy of the agents, the parent bacterium must inform its son about its energy and its position. This is achieved by synchronous message passing between the parent bacterium and its son at creation time.

### 3.3.2 Interactions between agents and artifacts

Obviously agents interact with artifacts. To percept the food or the molecules, herbivorous bacteria interrogate the respective artifact to check if they can eat or start the spot move policy.

In the following, we reported some important sequence diagrams, to clarify what have been explained above.

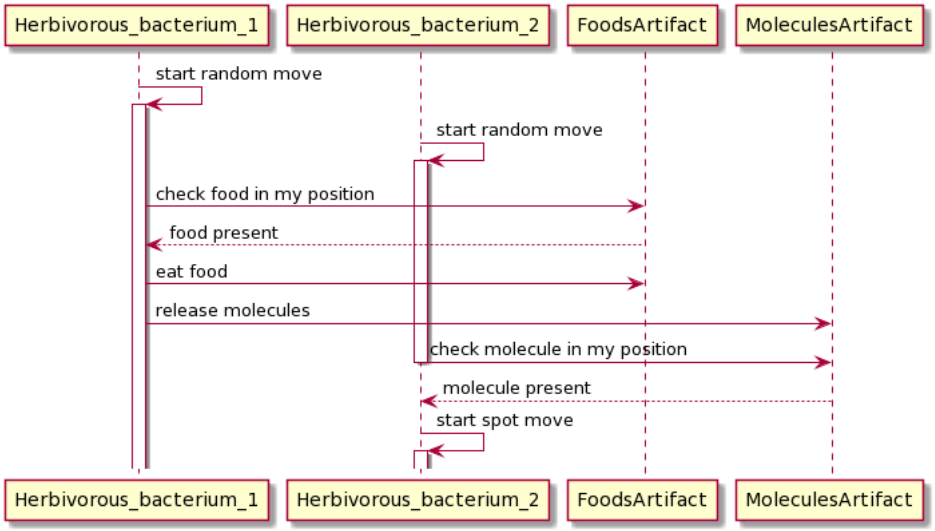


Figure 6: Sequence diagram of herbivores interactions with artifacts.

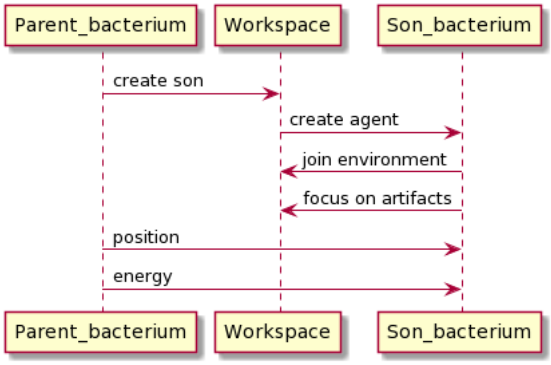


Figure 7: Sequence diagram of bacteria reproduction.

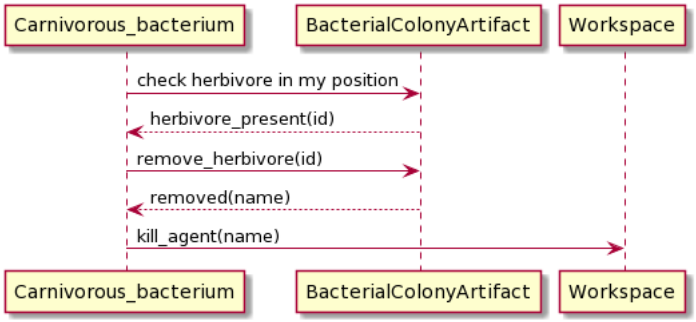


Figure 8: Sequence diagram of the carnivore eating.

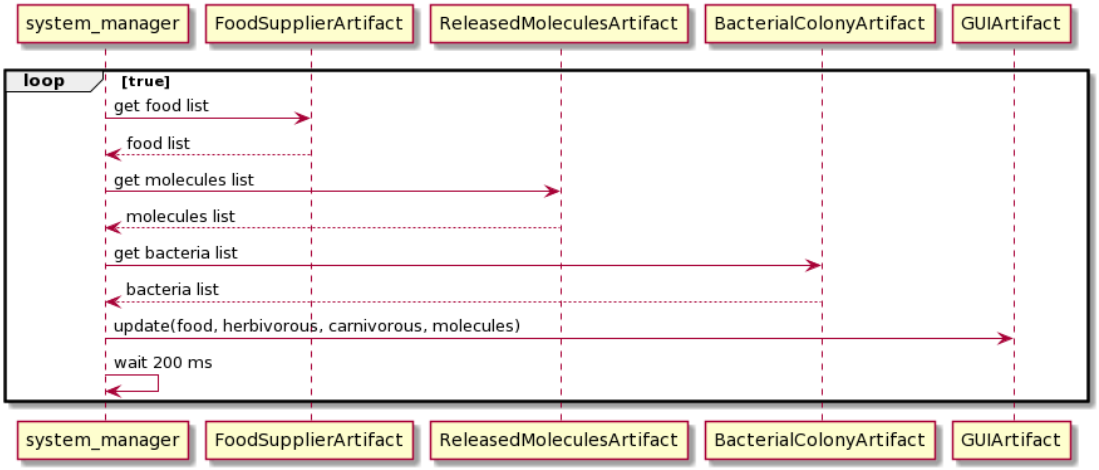


Figure 9: Sequence diagram of system_manager interactions with artifacts.

# 4 Implementation Details

In this section we want to explain some particular implementations of salient parts of the system.

## 4.1 Movement implementation

Bacteria can follow two different types of movement policy:

- random move, where a bacterium moves randomly in the enviroment
- spot move, when a bacterium percepts a molecule, it moves in order to reach other molecules and therefore to find the food.

Each bacterium moves with a particular velocity and they can move as the King in the chess game. Imagining to have a velocity = 1, a bacteria can move only in the eight cells around it. Each direction corresponds to a number between 0 and 7, as reported in the figure below.

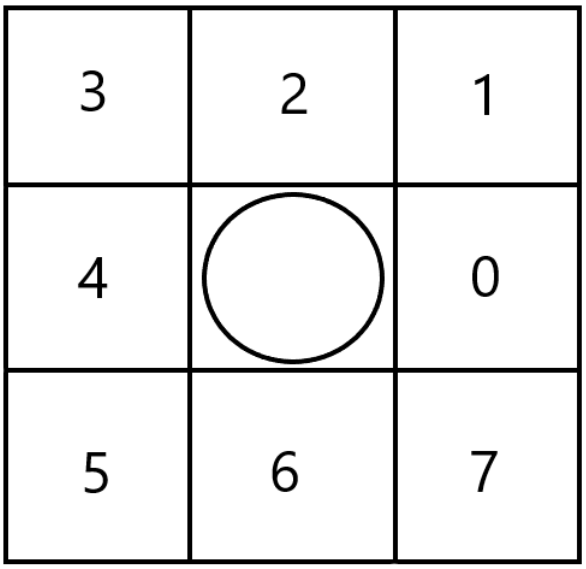


Figure 10: Possible moves of a bacterium.

At the beginning of the life, a bacterium choose a direction as a random number between 0 and 7, and start moving in that direction.

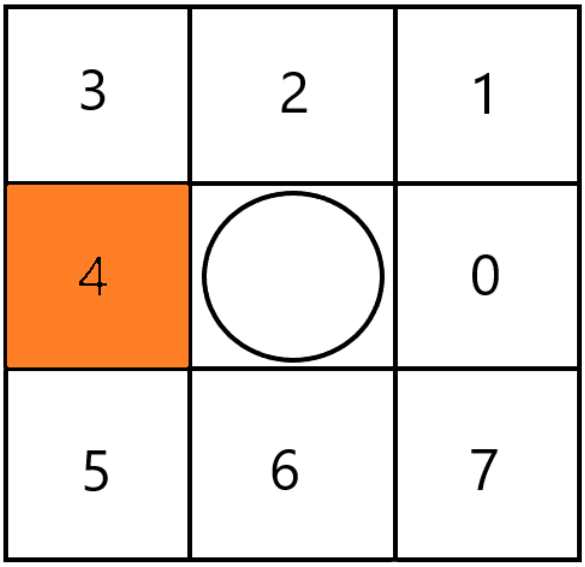


Figure 11: Initial choice of bacterium direction. The orange color denotes the initial direction.

To maintain the movement of the bacteria smoother, we decided that a bacterium can't turn itself too abruptly, but it can only moves in the two adjacent directions with respect to the previous one. So if the direction 4 in the example above is choosen, then the bacterium moves in that direction, and in the next step it can choose between only three directions, as a random number between 0 and 2.

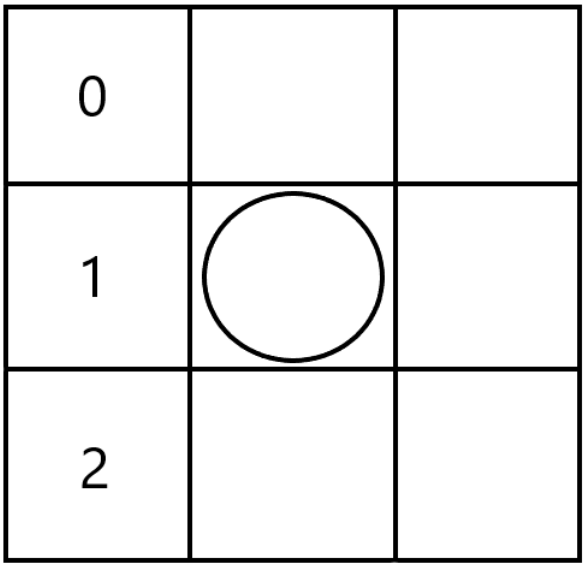


Figure 12: Next choice of bacterium direction.

This simple policy guarantees that it's really hard that a bacterium remains always in the same place in the environment, because with this policy it tends to navigate more and to explore new places.

### 4.1.1 Spot movement

When an herbivore, during its life cycle, finds a molecule then the movement mode, defined as **spot move**, is initiated. This modality follows a certain circumcentric path, starting from the point where the molecule was found, with the aim of staying around and finding other food more easily. The path was modeled through an internal operation, with the directions to follow stored in a list. The spot move in addition also has a degree of tolerance to molecules, defined as sensibility. This aspect was introduced as the bacteria release more molecules for each food source they find. Through the sensitivity it's avoided that the herbivore in question remains blocked as in the same area there is the risk of finding many molecules close together and for each could reactivate the spot mode.

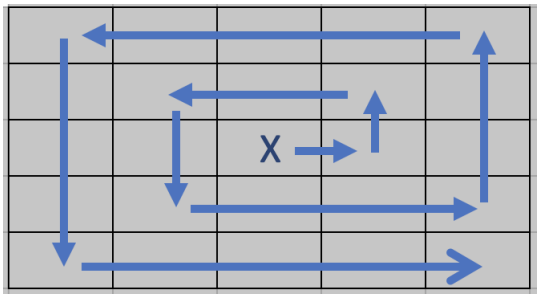


Figure 13: Path carried out in spot move mode.

At the end of this mode, if the bacterium does not find further molecules, the movement mode will return to the default one.

## 4.2 Molecule release implementation

The release of molecules have been implemented as a shedding of molecules in the environment. When an herbivorous bacterium eats, it istantaneusly releases molecules in the environment cells nearby. These molecules disappear after some time. Other bacteria can only perceive the molecules, they don't consume them, so molecules disappear only because a certain quantity of time is passed since their release.

## 4.3 Reproduction implementation

Reproduction takes place for separation: a bacterium generates a clone of itself. To execute this behaviour the bacterium must have an energy higher than a threshold because this energy is splitted in two equal parts between the parent and the son. The son is born in the same position of the parent. To do so, we implemented a synchronous

message exchange between the bacteria. The parent one, at the beginning of the son's life, send him the information about the position and the energy, and then the son can start his life cycle.

## 4.4 Death implementation

A generic bacterium dies when its energy is 0. Its energy increases when a bacterium eats some food, and decreases constantly at each time instant the bacterium doesn't eat. An herbivorous bacterium can die also because a carnivorous eats it. To achieve this result carnivores execute inside it the *.kill_agent()* function. This internal action of Jason allows to kill a specific agent inside the colony MAS.

## 4.5 Implementation details for the remaining parts

The remaining objects developed within the system, although not characterized by specific implementation choices, have been documented through the Javadoc, in order to make the code easy to understand for an external user.

Documentation is available *here*.

# 5 Self-assessment / Validation

Specific formal criteria were not used to validate the project, but rather a set of automated tests produced by us and a careful analysis of the software itself to verify the correctness of the required behaviors. The JaCaMo framework does not offer a test suite to verify the behaviors of the developed agents or artifacts. These entities have therefore been verified in some modality:

- through a meticulous modeling of each individual behavior to be implemented,
- through the visualization of the behaviours of agents and artifacts during the execution of the developed software,
- through a more hand written part which aims to test the basic functionalities of the agents, such as the movement and the energy decay.

Furthermore, Cartago documentation specifies that operations on artifacts are atomic, so the correcteness of the interactions between agents and artifacts in guaranteed by the framework itself. We found some trouble with the atomicity of JaCaMo, that is discussed in section 8.1.

As for the objects defined by us, using the JUnit testing suite, tests have been developed to verify the correct functioning of these objects within our system.

To test the agents, we simply create in the test package a specific .asl file that prints on the JaCaMo console important information about the bacterium. To run it you need to run the Gradle task **runColony_testJcm**.

# 6 Deployment Instructions

The realized software is distributed as a Gradle project, so the only requirement is the presence of the Gradle 6.0 or latest, installed on the machine. The steps to take to run the software are:

1. Download the project from the respective *repository*.
2. Open the project with a development IDE, like IntelliJ IDEA or Eclipse.
3. Execute the Gradle task: **runColonyJcm**.
4. Analyze the behaviour of bacteria colony.

# 7 Usage Examples

In the following, a screenshot of the running application is reported. The user has to launch the application and then it can observe how the bacteria move and act in the environment. On the left side there is an area in which the colony is shown. The herbivorous bacteria are the white circle entities, while the carnivorous bacteria are the red circle entities. The green rectangles are the food and the small blue circles are the molecules released by the herbivorous. By observing it, it's easily recognizable the behaviours of the bacteria. On the right side there are a few statistics reported, to help the observer understand how the colony is evolving:

- the number of food present in the environment
- the number of herbivorous bacteria present in the environment
- the number of carnivorous bacteria present in the environment

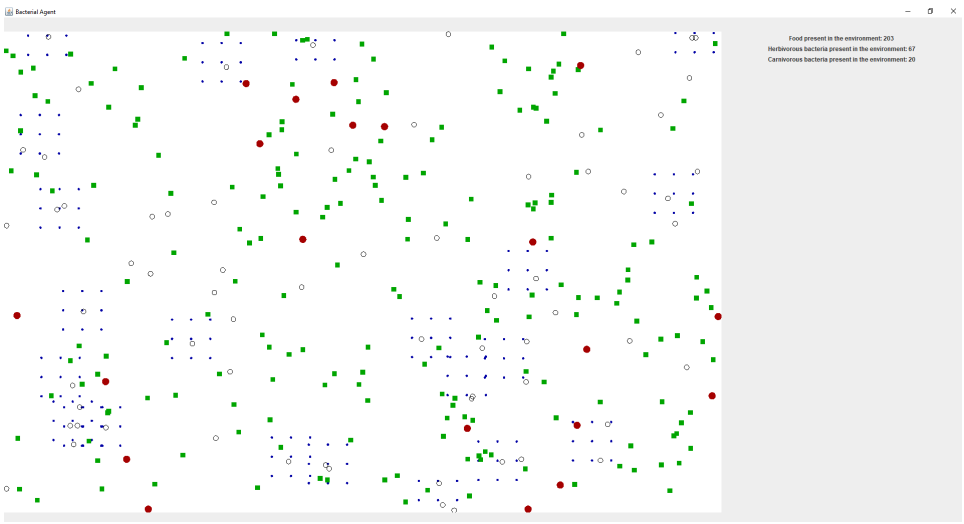


Figure 14: A screenshot of the application.

# 8 Conclusions

In this project we have deepened some important arguments explored in Autonomous System course. We implemented a MAS that is a bacterial colony, in which two different types of bacteria lives. For this purpose we used JaCaMo framework, because bacteria are autonomous entities, that can percept the environment and they have their own goals and plans to reach the goals. As living entities, they need to eat, they can communicate with each other, they move in the environment, they reproduce themsevles and they can obviously die. In general, we are really satisfied of our work, we realized all the requirements defined in the initial report, that are reported in the section 1.3.

## 8.1 Critical issues

In our solution there is a little problem we wasn't able to resolve. It relies on the concurrency model of JaCaMo. As said before, in official documentation methods that are labeled as Cartago operation are defined as atomic. Probably, with a lot of entities in the system, JaCaMo can't handle as atomic all the operations performed. A first problem involves the *kill_agent* operation on another agent, that have been already killed. In the *eat* plan of carnivorous bacterium, if it finds an herbivore this one has to be killed. Probably this operation is not atomic because an error is raised in some particular cases. We think that if the herbivore is dying in the same moment in which a carnivore tries to eat it, there is a concurrency error. We don't understand why, because, JaCaMo operation should be atomic. We also tried to separate this part in a new plan with the JaCaMo *atomic* annotation, but the error still raises. A second problem involves the list of molecules or food. We solved partially the problem, by re-creating every time the list instead of modify it, otherwise a concurrency error would raise, even if all the methods are defined as Cartago operation, and therefore should be atomic.

## 8.2 Future Works

There are some features and functionalities we would to implement in the future. First of all, we want to add new living entities, such as new form of bacteria, but also different microorganisms such as amoebas, with different behaviours. An interesting evolution of the system could be give a form of social communication also to carnivorous bacteria in order to coordinate their attack against herbivore populations.

A nice feature for the user interface is try to pass to JavaFX in order to have modern GUI aspect. Furthermore, we want to try to implement a little menu that get opened when the mouse is hover a bacterium, that recap its energy and characteristics.

## 8.3 What did we learned

Before submitting the project proposal, we studied deeper the argument of the course, in order to have a better understanding of BDI agents. We learned a lot about this topic, that we already seen in other courses but with different facets. This project allow us to research and to understand better how to integrate two different paradigms:

agent-based programming and object-oriented programming. At the beginning, during the analysis phase, we have interrogated ourselves a lot about which framework we can use and how to adapt the visualization part, that is properly object-oriented, to the agent-based part. Our reflections led us to choose JaCaMo, so we learned deeply this framework. In particular we are really satisfied about how we handle the artifacts and the GUI part. We obviously learned better the AgentSpeak syntax and also the Artifacts functionalities.

## References


[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. San Val, 1995.

[2] Giovanni Ciatto, Alfredo Maffi, Stefano Mariani, and Andrea Omicini. Smart contracts are more than objects: Pro-activeness on the blockchain. In Javier Prieto, Ashok Das Kumar, Stefano Ferretti, António Pinto, and Juan Manuel Corchado, editors, *Blockchain and Applications*, volume 1010 of *Advances in Intelligent Systems and Computing*, pages 45–53. Springer, 2020.

[3] Oracle. Java language.

[4] Olivier Boissier Rafael H. Bordini Jomi F. Hübner Alessandro Ricci Andrea Santi. Cartago.

[5] Olivier Boissier Rafael H. Bordini Jomi F. Hübner Alessandro Ricci Andrea Santi. Jacamo: Framework for multi-agent programming.

[6] Olivier Boissier Rafael H. Bordini Jomi F. Hübner Alessandro Ricci Andrea Santi. Jason.

[7] Olivier Boissier Rafael H. Bordini Jomi F. Hübner Alessandro Ricci Andrea Santi. Moise.