

Exploring new worlds using MAS

Nicolò Guainazzo (S4486891)

nico@guainazzo.it

Introduction

This work aims to experiment how a multi-agent system fits into the space field, specifically the exploration of a generic planet by developing a software prototype for the simulation of such a mission. Inspired by what is currently happening with the exploration of Mars through NASA's Perseverance rover, launched in 2020 as part of the Mars 2020 mission, and with what was written in a 2010 paper [1] by Jorge Ocon on possible applications of multi-agent systems in space applications. Moreover, the choice of this field over many others [2] [3] [4] in which the multi-agent paradigm is used has been influenced by the extraordinary growth forecast for the space economy, which recent predictions estimate will be worth \$1.8 trillion globally by 2035¹.

Finally, let us add that this project is not intended to be, nor could it be, a complete version of a multi-agent system for planetary exploration. Instead, it should be understood as a simplified model to understand practically whether and how a multi-agent system could be a useful approach as suggested in Ocon's article cited above.

JADE

Before starting to discuss my work, it is necessary to talk about the tool I decided to use for its development. JADE, which stands for Java Agent DEvelopment framework [5]. It is considered one of the most well-known agent-oriented middleware used today and is also mentioned in Ocon's paper as one of the best choices for MAS in the spatial domain. Entirely developed in Java, it not only allows the development of multi-agent systems but also provides access to many third-party libraries. JADE was initially developed by the Italian company Telecom under the open-source LGPL (Lesser General Public License Version 2) license. In this chapter, I will try to explain why I chose JADE for this project, highlighting its best features for this context.

2.1: Compliance with standards

JADE is designed to be compliant with the FIPA (Foundation for Intelligent Physical Agents) specifications, which ensures interoperability between agents developed using different frameworks that adhere to the same standards. This feature is very important for a large project such as a multi-agent system, especially in a highly complex field like

¹As indicated in this recent [insight report](#) from the World Economic Forum.

space exploration. In fact, it allows for the development of a robust system that is easily adaptable in its components.

2.2: Agent communication

JADE allows the development of rich and complex communications between agents through the use of the FIPA ACL (Agent Communication Language) standard, which descends from being FIPA-compliant as explained in the previous point, using different message types and protocols. This effectively supports a crucial element for exploration and multi-agent systems in general.

2.3: Distributed architecture

JADE supports fully distributed systems, potentially even in different remote machines. This feature is actually essential in the case of a development of this project to be really used in our context, because so far the project does not exploit this possibility but simply each agent exists in a separate thread on the same machine.

2.4: Ease of Development

JADE simplifies the development of multi-agent systems through its comprehensive library of classes and built-in tools. This ease of use allows developers to focus on the specific behaviours and tasks of their agents without worrying about the underlying infrastructure. Additionally, JADE provides a GUI to simplify the visualization of agents and their behaviours, along with other tools such as DummyAgent, which allows for testing by sending simple messages to other agents in development, and Sniffer Agent, which, as the name suggests, enables the documentation of communication between agents.

Project

Now that I have discussed why I chose JADE as my development framework, it is time to explain what my project consists of. As mentioned in the introduction, I was inspired by NASA's Perseverance mission which aims to explore the Jezero crater in search of traces of ancient microbiological life and collect Martian rocks and soil and then with a future mission bring them back to earth. So hypothetically, the agents' environment would be Mars. In general, the environment of the agents can be extended to any similar planet that, in a future - though unfortunately not near - could be initially explored with a rover, as Perseverance does today. Another important thing to add about the Perseverance mission and which I have tried to implement in my project is the existence of Ingenuity, the first drone (helicopter) to fly to another planet.

Therefore, to simulate a planetary exploration mission as a multi-agent system, I created an environment and planned to use four different types of agents: rover, drone, ground control and alien. We will look at them in detail in the following paragraphs, focusing on their behaviour and leaving out, for reasons of time, auxiliary functions, classes for GUIs and in general for secondary implementation choices.

3.1: Environment (Grid)

All agents must be situated in an environment [6], in our case, a planet to explore. To represent a portion of the planet where the agents move, I decided to use a graphical representation structured as a 10x10 grid. Agents must stay within a cell of the grid, moving from one cell to another until reaching their final position. In this grid, there are no obstacles, which would certainly exist in a real situation. I made this choice for two main reasons: First, overcoming an obstacle is a problem strictly related to programming, which does not pertain to Agent-oriented programming or the purpose of the course for which this project was created. Second, given the limited time to complete the project, I preferred to leave this element for a possible future implementation and focus on other aspects, such as implementing a GUI to make the final result more understandable, for example, by effectively showing the grid.

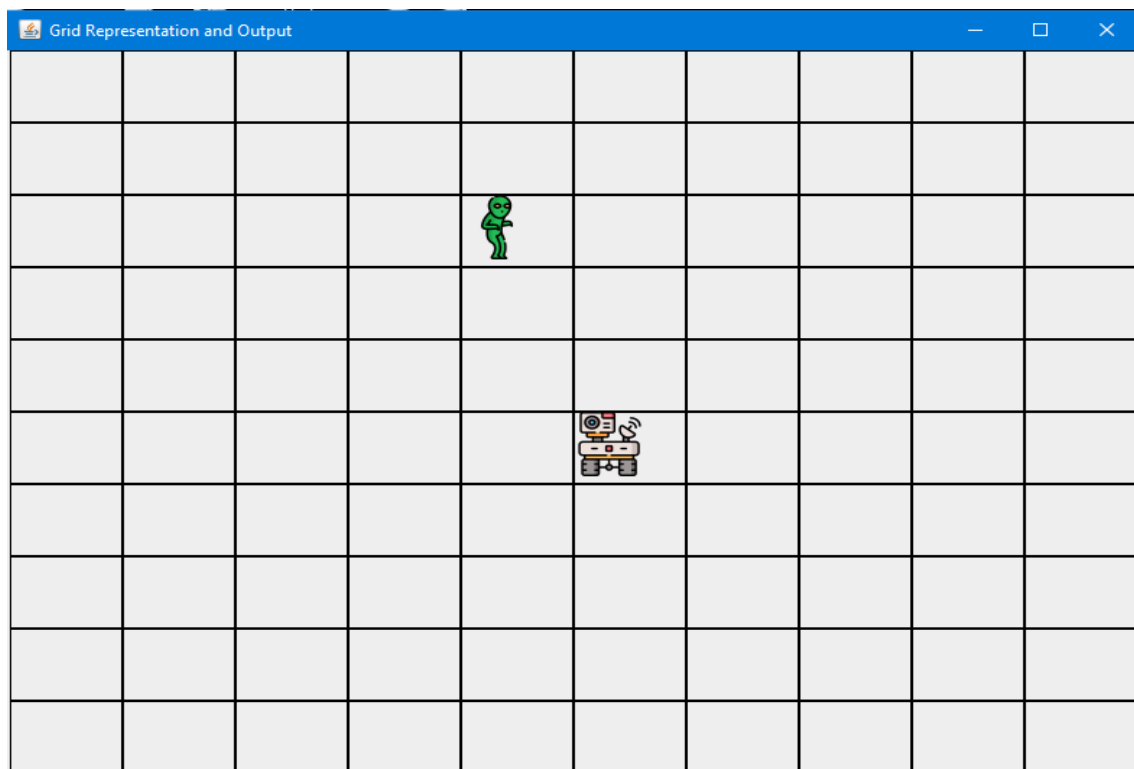


Figure 1. Grid representation using the GUI

3.2: Agents

3.2.1: Rover

The Rover is the agent that physically performs the mission of exploration and sample collection from the planetary surface. As stated in Ocon's paper, the interest in having agents and develop a multi-agent system for space missions stems from the difficulties encountered in directing a rover step-by-step from Earth (Pathfinder Mission), including long communication times and the resulting latency between a command and the rover's actual action.

Therefore, we could say that the rover is the main agent of the system and we will describe its various behaviours in detail later. For now, we will just explain what it does to put it in general terms.

First, the rover listens, waiting for Ground Control to send the coordinates for the target location where the rover will then dig the ground to collect a soil sample. Initially, I had considered having the rover move autonomously, simply thinking of random movement on the map while receiving new coordinates from Ground Control. I abandoned this idea because it didn't make much sense. I tried, as much as possible, to maintain a minimal adherence to reality. Rovers on Mars do not move randomly; the places they need to visit are carefully chosen in advance. Therefore, I adopted a balanced approach between sending every command from Earth to move the rover, which would have eliminated the very possibility of considering the rover an agent according to the known definitions and which also would not have taken into account what Ocon's paper mentioned earlier, and a fully autonomous rover, which in our context, also due to the simplicity of the implementation of its behaviour and of the project in general, would not have been possible. So the rover receives the coordinates to be reached from the ground control but alone makes the movement to this target.

Once the coordinates are received, the movement begins. The rover then autonomously decides whether to launch the helicopter drone based on the distance between itself and the target indicated by Ground Control. The drone will fly for a short period around the rover, ideally to map and survey the terrain while it moves and to assist the rover. The rover can also launch the drone at the request of Ground Control when new coordinates are sent. Once the target is reached, the rover collects a soil sample, performs an initial analysis, communicates with Ground Control, and then waits for new coordinates to continue the activity.

3.2.2: Drone

The drone agent depends on the rover agent from which it is launched (in reality, Perseverance released Ingenuity from its compartment in the belly). The rover decides whether to launch the drone, as we have already mentioned, or it can be launched at the request of Ground Control. The task of the drone in our system is to simulate a flight to survey the terrain from above where the main rover is moving.

The flight is meant to be short; Ingenuity flew a total of 128 minutes in 72 different flights, far exceeding the most optimistic expectations. So I thought about not allowing the heli-drone to stray too far from the rover's position as it moves within the grid. After a few seconds the heli-drone asks the rover where it is and lands at its coordinates.

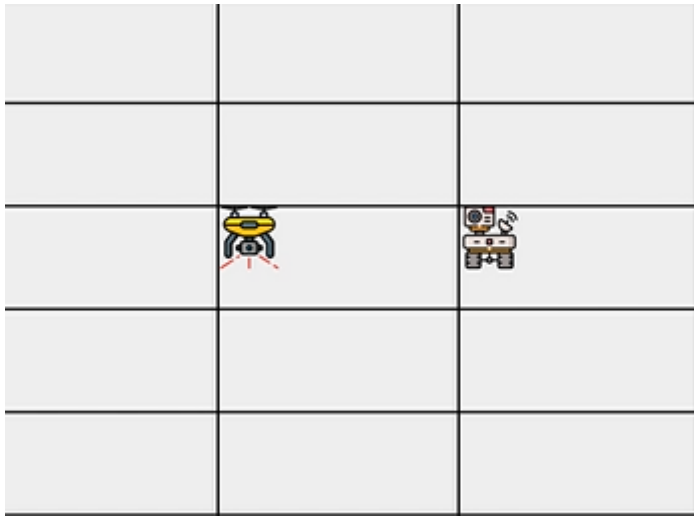
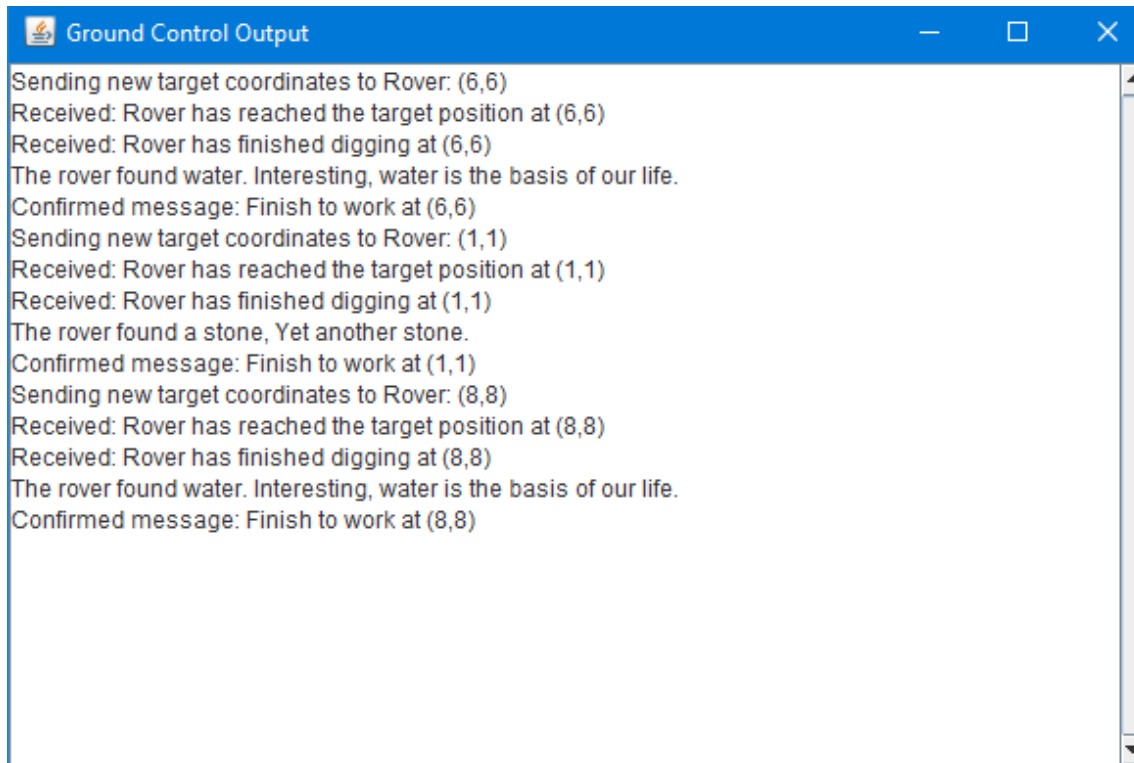


Figure 2. Drone icon in the Grid

3.2.3: Ground Control

The third agent is the Ground Control (GC), which acts as the mission control body. Its main purpose is to communicate with the other agents and assist the ground staff in making decisions and conducting initial experiments. The first activity upon human input is to send the coordinates to be reached to the rover and then tell the rover whether to use the drone. Then the GC starts waiting to hear from the rover if it has accomplished its tasks. When the rover collects a soil sample, it sends the information extracted from the position back to Earth where the GC autonomously starts a more in-depth analysis.

Ground Control is the only agent not present on the grid, as it is obviously not on the planet being explored but rather on Earth. It does, however, have a panel (*Figure 3*) in which to print out received messages and other outputs such as the analysis result.



```
Ground Control Output
Sending new target coordinates to Rover: (6,6)
Received: Rover has reached the target position at (6,6)
Received: Rover has finished digging at (6,6)
The rover found water. Interesting, water is the basis of our life.
Confirmed message: Finish to work at (6,6)
Sending new target coordinates to Rover: (1,1)
Received: Rover has reached the target position at (1,1)
Received: Rover has finished digging at (1,1)
The rover found a stone, Yet another stone.
Confirmed message: Finish to work at (1,1)
Sending new target coordinates to Rover: (8,8)
Received: Rover has reached the target position at (8,8)
Received: Rover has finished digging at (8,8)
The rover found water. Interesting, water is the basis of our life.
Confirmed message: Finish to work at (8,8)
```

Figure 3. Ground control output

3.2.4: Alien

The last agent represents an alien, added to increase the complexity of the system. The alien roams the grid randomly, and thus the hypothetical planet, without a known destination, attempting to communicate with someone, sending message in the FIPA-ACL standards. In our case, it tries to communicate with the rover, which, however, rejects any communication from unknown entities, staying true to the idea of communicating only with other agents deemed reliable [7].

3.3: Behaviours

In this section, I will attempt to thoroughly document each behaviour of the four agents in the system. I have decided to focus on the behaviours because they represent the heart of the agents and thus of the entire system. For each behaviour, I will include the name, type, purpose and provide a brief explanation². Of course, all the code is available on my GitHub repository³

² In listing the behaviour for each section concerning an agent type, I decided to follow the order in which they appear in my code.

³ Here is the [link](#)

3.3.1: Rover

1. "handleAlienProposal"

Type: CyclicBehaviour

Purpose: This behaviour is designed to receive messages from the alien agent if it sends one to the rover, and subsequently to respond by rejecting the proposal from an unknown agent.

Explanation: This behaviour, due to its cyclic nature, continuously checks for incoming messages with the performative *PROPOSE* and the conversation-ID *Alien-Rover-Proposal*. Once such a message has been received, the rover checks whether it has come from a known agent such as ground control or the heli-drone and accepts it in this case by replying with *ACCEPT_PROPOSAL*, otherwise the proposal is rejected and a rejection message is sent to the sender. So the rover rejects all proposals sent by different agents.

2. "handleHeliDroneRequest"

Type: CyclicBehaviour

Purpose: The purpose of this behaviour is to handle the heli-drone's request for new coordinates.

Explanation: Such behaviour waits for the heli-drone to send a request, with performative *REQUEST* and conversation-ID *Drone-Rover-Coordinates*, for the rover's current co-ordinates and then sends them to it using *SendCoordToHeliDrone()* obviously implemented by me. All this is needed for the heli-drone to know where the rover is and not to stray too far away, but we will see this in more detail in the section on the heli-drone.

3. "mainRoverLogic"

Type: TickerBehaviour

Purpose: This behaviour manages the main logic for the rover's movement on the grid and tasks.

Explanation: This behaviour occurs at regular intervals (every second) to simulate movement. If the rover moves towards a target, it manages the movement using the auxiliary function *moveToTarget*, checks if the target is far away using the Manhattan distance (≥ 8) with the *distanceToTarget* function (to launch the drone if necessary), updates the GridGUI with the position of the rover, and uses the *waker behavior* to simulate digging and analysis tasks, as well as communication with the GC using methods like *notifyDigCompletion()*, *notifyGroundControlOfAnalysisResult()*, and *notifyGroundControlOfWorkCompletion()* whose names effectively explain their purpose.

4. "receiveDroneLandingMessage"

Type: CyclicBehaviour

Purpose: It serves to handle the message from the helidrone that it has landed.

Explanation: This cyclic behaviour continuously checks for incoming messages with the `INFORM` performative and the `Drone-Landed` conversation-ID. When it receives such a message, it prints a message on the GUI indicating that the drone has landed and clears its position on the grid.

5. "updateDroneInGridGUI"

Type: CyclicBehaviour

Purpose: This behaviour updates the position of the drone in the GUI according to the request sent by the drone.

Explanation: This behaviour waits for the arrival of messages with the performative `REQUEST` and the `Drone-Grid-Update` conversation ID to update the position of the heli-drone in the Grid-GUI according to the coordinates contained in the message.

6. "handleLaunchHelidrone"

Type: CyclicBehaviour

Purpose: This behaviour is used to launch the heli-drone if the GC requests it.

Explanation: This behaviour waits for the arrival of a message with the performative `REQUEST` and the `GroundControl-Launch-Helidrone` conversation-ID. When it receives such a message, it checks if the drone has already been launched and, if not, creates a new helidrone agent by taking control of the JADE container using the `createNewAgent()` method and finally "launches" it using the `start()` method.

7. "handleTargetCoordinatesMessage"

Type: CyclicBehaviour

Purpose: This behaviour serves to receive the target co-ordinates sent by the GC.

Explanation: This behaviour waits for a message to arrive with the new target coordinates from the GC. Once received, it updates the coordinates for the rover and starts the movement by setting the variable `isMovingToTarget = true`.

8. "ReceivedShutdown"

Type: CyclicBehaviour

Purpose: This behaviour serves to eliminate the rover agent at the request of the GC is only valid on an implementation level.

Explanation: This behaviour continuously checks for incoming messages with the performative `INFORM` and conversation ID `GroundControl-Shutdown`. When such a message is received, the rover agent deletes itself, effectively shutting down and also dispose the Gird-GUI.

3.3.2: Drone

1. `"startDroneOperations"`

Type: `OneShotBehaviour`

Purpose: This is the first behaviour, encapsulating the main behaviours to be executed when the heli-drone agent is created.

Explanation: This one-shot behaviour is executed once when the drone agent starts. It calls the `takeOff` method to initiate the drone's take-off process, and finally starts the drone's movement by calling `moveAround`. We will see these behaviours in details later.

2. `"moveAround"`

Type: `ParallelBehaviour`

Purpose: In this method are embedded the parallelBehaviours to perform the actual helidrone's movement on the grid.

Explanation: This method sets up a `ParallelBehaviour` that runs both a `TickerBehaviour` called `ticker` and a `WakerBehaviour` called `waker`:

- **Ticker:** This behaviour is used to move the drone to random positions on the grid at regular intervals of 400 ms in an attempt to simulate flight. It ensures that the drone moves within the grid boundaries and avoids revisiting positions during flight. It also updates the graphical interface.
- **Waker:** It stops the movement of the drone after 2 seconds, removing the `TickerBehaviour` from the `ParallelBehaviour`. It then requests new co-ordinates from the rover (`requestNewCoordinates`), calls the behaviour to accept the new co-ordinates (`acceptNewCoordinates`) and finally calls the `land` method to 'land' on the rover's current co-ordinates received earlier.

3. `"acceptNewCoordinates"`

Type: `CyclicBehaviour`

Purpose: This behaviour is designed to listen for the position of the rover.

Explanation: Once the message is received from the rover with the performative *INFORM* and the conversation-ID *Rover-Drone-Coordinates*, the behaviour updates the coordinates where the heli-drone maintains the last position of the parent rover.

4. *"land"*

Type: WakerBehaviour

Purpose: This behaviour simulate the landing process.

Explanation: This behaviour waits using the *wakerbehaviour* 2 seconds to simulate the landing phase. Then prints in output that the drone has landed and at what coordinates, informs the rover of the successful landing and finally use the *droneShutDown* to delete the agent.

5. *"droneShutDown"*

Type: OneShotBehaviour

Purpose: This behaviour deletes the drone agent.

Explanation: This one-shot behaviour is executed once when the drone needs to be shut down. It calls the *doDelete()* method to terminate the drone agent which in turn calls *takeDown()* which in this case only prints out a string informing that the heli-drone has been deleted.

3.3.3: Ground Control

1. *"MessageReceiver"*

Type: CyclicBehaviour

Purpose: Wait for the message informing the GC that the rover has arrived at its destination.

Explanation: This cyclic behaviour once received prints the message sent by the rover onto the agent's GUI.

2. *"finishedDigging"*

Type: CyclicBehaviour

Purpose: This behaviour listens for messages from the rover indicating it has finished digging.

Explanation: This behaviour, thanks to the properties of cyclic behaviour, waits to be told by the rover that it has finished excavating a new sample. Then print out on the GC's GUI the message received.

3. `"receiveTheAnalysis"`

Type: CyclicBehaviour

Purpose: This behaviour listens for analysis results from the rover.

Explanation: As in the previous behaviour, this one also waits for the arrival of the message from the rover containing information on what it has found while digging, then carries out a more in-depth analysis, which is simplified here by a switch that prints a different sentence in the GUI depending on the rover's message.

4. `"receiveWorkDone"`

Type: CyclicBehaviour

Purpose: This behaviour is used to know when the rover has finished its work and to start the ground control GUI

Explanation: Once again, thanks to the cyclic behaviour, it waits for the rover to send the message that it has finished its work and then prints the received message in the GUI.

3.3.4: Alien

1. `"moveToTargetBehaviour"`

Type: Composite method (*WakerBehaviour* and *TickerBehavior*)

Purpose: Implement the agent's movement logic on the previously set random co-ordinates to simulate the alien's walking around and send the proposal message to the rover.

Explanation: The method consists of two behaviours: the first is a *WakerBehaviour* that allows the alien to start moving after 10 seconds that the application has been launched. Into this we insert the second behaviour a *TickerBehaviour* that every second moves towards the co-ordinates and if it reaches its destination it sends a message with the behaviour *sendProposal* and then starts moving around the grid again.

2. `"sendProposal"`

Type: OneShotBehaviour

Purpose: As the name implies, this behaviour is intended to send a proposal message in this case to the rover.

Explanation: This behaviour has been designed as a one-shot because it only has to occur once when called in the ticker behaviour of the *moveToTargetBehaviour()* method, in its body a message is created and sent to the rover with a *PROPOSAL* performative not used anywhere else in the project suggesting that a collaboration with the rover be initiated if the rover reply with a message with *ACCEPT_PROPOSAL* performative.

3. *"handleProposalRejection"*

Type: CyclicBehaviour

Purpose: Wait for the rover to respond to the proposal message in case of rejection by the receiver.

Explanation: With the CyclicBehaviour the agent listens and when the response from the rover arrives it prints the rejection and then calls the method to set new coordinates to be reached and then the alien starts moving again.

4. *"handleProposalAcceptance"*

Type: CyclicBehaviour

Purpose: Like the previous behaviour, with the difference that this one awaits an acceptance response.

Explanation: The behaviour listens for a positive response to the proposal, when a positive response is received an *Oneshotbehaviour* is called with the *performCollaborationTasks()* method which sends a message, this behaviour is not explained separately like the others because it is only a toy-behaviour, meaning that the message will never be received by anyone. In fact, due to the implementation of the rover agent the alien will never receive a positive reply, because as already explained in the section on rover behaviours it will only accept messages from known agents. This behaviour has only been implemented for completeness in the reply of a message with performative *PROPOSAL* which, unlike those with performative *INFORM* which assume an affirmative or negative reply. As it does not serve to notify what an agent has done but to ask for the initiation of collaboration between agents.

Conclusion

In conclusion, I can say that the development of a prototype like the one presented in the project seems to me to confirm the actual applicability of a multi-agent system for the field of planetary exploration. Furthermore, the development turned out to be very natural thanks to the agent-oriented programming approach [8], both at the level of problem and solution abstraction and obviously also at the implementation level. This made it clear to me the practical utility of the agent-hood concept [6] for modelling

complex systems, even if the project can only be considered a toy system with its simplifications.

Furthermore, I believe it is interesting trying to think of possible future improvements for the project, here I will just mention those I consider the most important. The implementation of more complex behaviours that are closer to reality, especially in the difficult environment in which theoretically the agents would be called to operate; in this sense I am thinking of introducing navigation systems, both for the rover and for the heli-drone that exploit both classical [9] [10] [11] and generative artificial intelligence [12]. It would also be fascinating to explore drone and agent cooperation to use the former to improve the latter's decisions, for example in choosing a path by providing a view from above, or even add decision support systems that the ground control agent could provide to human controllers.

Another area I think it is important to improve would be communications, increasing its complexity but also its effectiveness by including an ontology that provides a common knowledge base for agents to agree on the meaning of words and commands for our specific domain, as also suggested in the conclusion of Ocon's article mentioned above.

Finally, the environment can be improved. Now represented by the grid in which the agents move, adding complexity such as more impervious and less impervious areas, obstacles and so on.

References

- [1] Ocon, J. (2010). Multi-agent frameworks for space applications. In SpaceOps 2010 Conference Delivering on the Dream Hosted by NASA Marshall Space Flight Center and Organized by AIAA (p. 2069).
- [2] Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-agent systems: A survey. *Ieee Access*, 6, 28573-28593.
- [3] Chakraborty, S., & Gupta, S. (2014). Medical application using multi agent system-a literature survey. *International Journal of Engineering Research and Applications*, 4(2), 528-546.
- [4] Hanga, K. M., & Kovalchuk, Y. (2019). Machine learning and multi-agent systems in oil and gas industry applications: A survey. *Computer Science Review*, 34, 100191.
- [5] Bellifemine, F. L., Caire, G., and Greenwood, D. 2007. *Developing multi-agent systems with JADE*. John Wiley & Sons
- [6] Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1, 7-38.
- [7] Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM*, 37(7), 122-125.
- [8] Shoham, Y. (1993). Agent-oriented programming. *Artificial intelligence*, 60(1), 51-92.
- [9] Rekleitis, I., Bedwani, J. L., Dupuis, E., & Allard, P. (2008, May). Path planning for planetary exploration. In 2008 Canadian Conference on Computer and Robot Vision (pp. 61-68). IEEE.
- [10] Oche, P. A., Ewa, G. A., & Ibekwe, N. (2021). Applications and challenges of artificial intelligence in space missions. *IEEE Access*, 12, 44481-44509.
- [11] Panov, A. I., Yakovlev, K. S., & Suvorov, R. (2018). Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science*, 123, 347-353.
- [12] Zhang, G., & Li, Q. (2024, March). MarsDrone: A Next-Generation Simulation System for Mars Unmanned Aerial Vehicles. In 2024 IEEE Aerospace Conference (pp. 1-10). IEEE.

