Intelligent Agents

Report

David Carter

Contents

List of figures

1. Introduction

In this assignment, I have designed and implemented a multi-agent system for the Mars Mission Scenario through Python. This report gives are descriptive summary of how the technical elements of the solution work, particularly giving consideration to any logic and learning that the agents use.

The simulation can be viewed using my code by running the MarSimRunner.py file.

2. Summary of Solution

In this section, I am going to outline the basic workings of the code, particularly giving attention to any logic used by agents in their actions.

The model can be viewed by running the MarsSimRunner python file.

2.1. Running the simulation (MarsSim class)

MarsSim is a concrete implementation of the Simulator and Observer Classes. The prepare method of the MarsSim class, creates the model, first populating a single spaceship in a random location, and then calls on the spaceship to deploy rovers based on the lower of the number of spaces available around the spaceship and the number set in the Mars Configuration file. Then the prepare method fills all remaining spaces with rocks. Once the grid has been populated this method creates the Gui adding itself as an observer of the GUI, such that it can take actions when the interaction state of the Gui changes.

The render method updates the GUI.

When the GUI state changes, the observer state of the MarsSim is updated and the render and update methods are called.

The update method is used to have the agents act and refresh the GUI with the new information. The update method in the MarsSim Class uses the following logic to update the simulation.

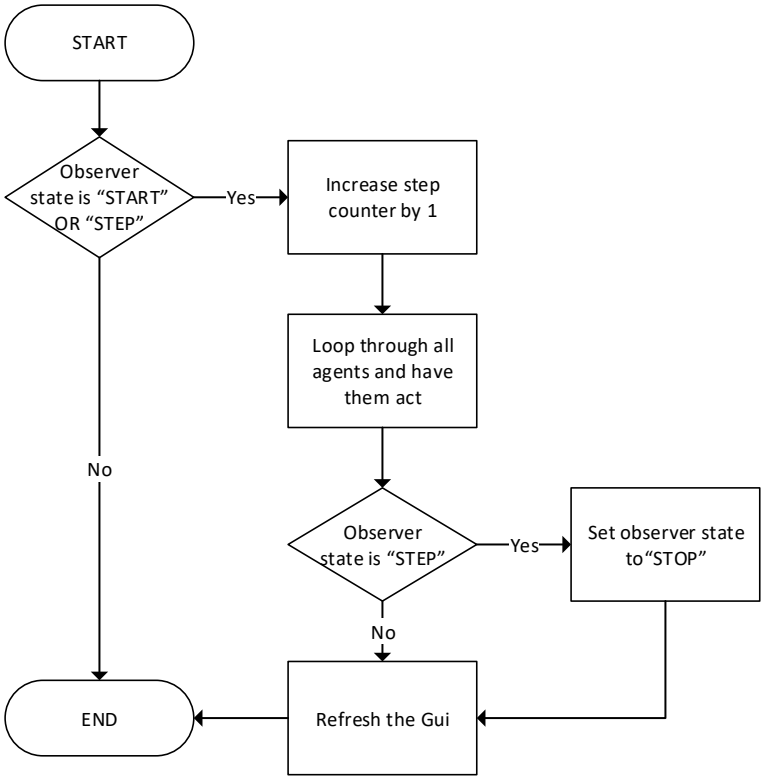

Figure 1: Flowchart to show the update method of the MarsSim class for calling on Agents to act.

This flow allows the user to control the start and stop of the simulation and ensures that agents only act when requested to do so.

2.2. Spaceship actions (Spaceship Class)

Spaceship is a concrete implementation of the Agent Class. When the spaceship is requested to act. The spaceship will follow the steps set out in the following flowchart.

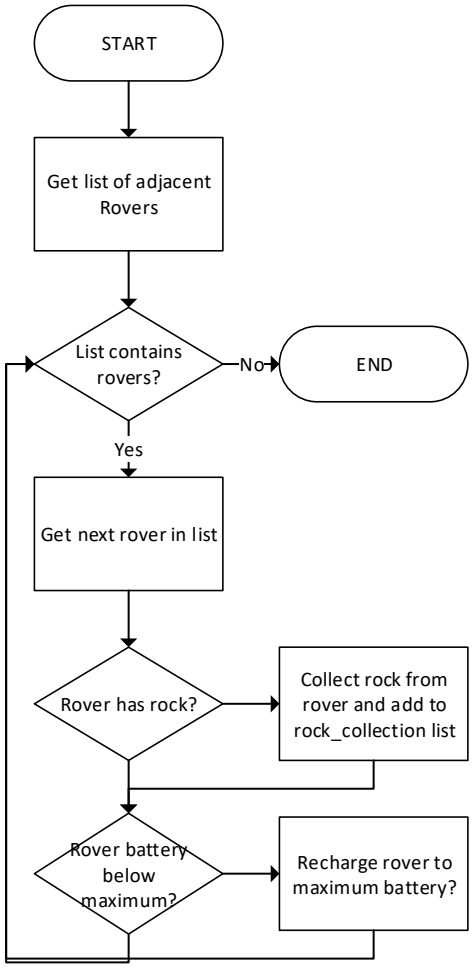

Figure 2: Flowchart showing the actions a spaceship takes when the act method is called

The spaceship primary objectives are to collect rocks from adjacent rovers and to also charge adjacent rovers back to full battery.

2.3. Rock actions (Rock actions)

Rock is a concrete implementation of the Agent Class. The rock is fairly simplistic in its actions. The following flowchart gives an overview of the actions that it takes when the act method is called.

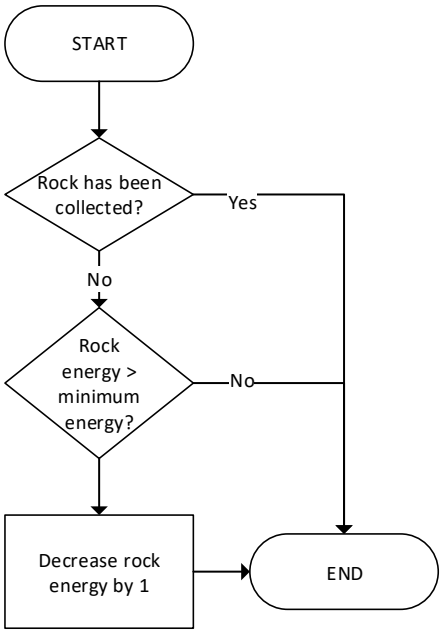

Figure 3: Flowchart to show the actions taken by a Rock using the act method.

Once a rock has been collected, it's energy levels will stop dropping. Otherwise until collected the rock will continue to drop its energy levels by one each time it acts until either it is collected or its energy level has reached the minimum level set out in the MarsConfig class.

2.4. Rover actions (Rover Class)

Rover is a concrete implementation of the Agent Class. The Rover is the most complex agent in this model, as it has many different options that it can take based on the information around it. The top-level logic behind the act method is as follows:

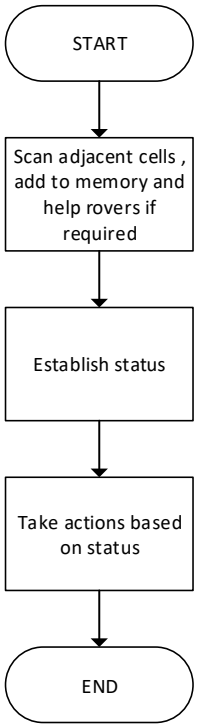

Figure 4: High-level details for the Rover act method

When scanning adjacent cells, the rover will firstly update its memory with the new information from the scan. This way the rover builds up a knowledge of the environment, such that it can take intelligent actions. When updating its memory the rover will also remove any previous entry for the same location, or if an adjacent entry was recorded at another location (as it has now moved, so the information that the entity is at another location is now incorrect. The rover will prioritise the new information over the old information.

The rover's memory is an AllScannedEntities Object, which contains a list of ScannedEntities. A Scanned Entity comprises of an Entity and a location.

As well as updating its memory the rover's scan will also log all adjacent rovers. For each rover in this list, the rover will obtain information on the rover's memory, target goals and status, possibly donating battery power to an adjacent, if that rover is deadlocked. The sharing of information is further explained in section 3.

The rover can have the following statuses:

- Deadlocked – this status is set when the rover has been unable to move for 3 consecutive steps.
- Waiting for contract partner – If the rover has set up a contract, but does not yet have a partner. (further explained in section 3 and 4)
- Under Contract – if the rover has a contract with an agreed partner (further explained in section 3 and 4).

- Inbound – If the rover either has a rock or has reached its battery limit and needs to be recharged before it runs out of battery.
- Outbound - Default status if no other status has been fulfilled.

2.4.1. Deadlock status

If a rover is deadlocked, this means that its 3 previous attempts to move have failed. The rover will then instead attempt to move towards a free spaceship location from its memory if one is available within its remaining movement. Otherwise, it will try to move towards the spaceship and will hope that a path becomes available.

2.4.2. Waiting for contract partner status

When a rover has set up a contract but another rover has not yet accepted a contract then the rover will search for a contract partner by moving randomly close to the spaceship (1 or 2 spaces from the spaceship). If however, it finds another rover with an open contract, it will discard its contract and accept that rover's contract instead.

2.4.3. Under contract status

When the rover's status is under contract it will take actions based on its role within the contract. The rover that sets up the contract is the master of the contract and the agreeing rover becomes the supply.

Master Rover Actions

The master rover will first move towards the recharge point that was set up when creating the contract. The recharge point is always the maximum movement away from the spaceship such that the supply can move adjacent to the recharge point and donate two movements worth of battery from the supply rover to the master. Once the master rover has reached the recharge point, it will then calculate the amount of battery required to reach a rock that it knows about. If it has enough battery to reach the rock and return to the recharge point then it will go and collect that rock, otherwise, it will wait until it has more battery.

If the master rover has a full battery but cannot reach any more rocks, it will mark the contract has complete and then move back towards the spaceship.

Supply Rover Actions

The supply rover checks if it's contract partner is adjacent to it, if the master rover is adjacent then if the master rover has a rock and the supply rover has no rock, the rock will be taken off the master and returned to the spaceship. Additionally, the supply rover will donate any spare battery it has to the master rover up to the master rover's maximum battery.

The Supply rover will continually travel between the recharge point and the spaceship to deliver rocks and recharge its battery to be able to donate battery to the master rover.

2.4.4. Inbound Status

When the rover has the inbound status, it will attempt to move towards the closest empty location adjacent to the spaceship known from its memory, otherwise, it will attempt to move towards the spaceship in the quickest route.

2.4.5. Outbound Status

When a rover has an outbound status. It will prioritise the following actions:

1. It will check for adjacent rocks and collect one if it has sufficient battery to be able to safely collect it.
2. If there are one or more rovers adjacent with an open contract, it will accept one rover's contract, setting itself as the supply.
3. It will move towards the closest known rock from its memory (the selected rock may have already been collected, but the rover is unaware of this).
4. If it cannot reach any rocks from its current location it will move towards an adjacent spaceship location where it could then reach the rock.
5. If its battery is not at full then it will return towards the spaceship
6. If there is a deadlocked rover adjacent to it, it will move to another location (in case it is blocking the rover's route back to the spaceship).
7. If it is adjacent to the spaceship and a rock can be reached with another rover's help, then it will set up and contract with a recharge point, such that it can work together to reach further away.
8. If no other actions are possible it will move towards the spaceship (this is evident once all possible rocks have been collected and all rovers, start to circle the spaceship).

2.5. Mars Class

Mars is a concrete implementation of the Environment Class. Mars contains several methods that enable agents to interact with their environment.

The methods within the Mars Class primarily focus on returning Agents and/or Locations that are in the Environment, move Agents around the environment, or to calculate and return the distance between 2 locations in the environment.

The MarsGui shows the environment and the current locations of agents in the Environment.

2.6. Configuration Files

The class Config and MarsConfig have the default values for the Model Simulation.

3. Rovers Sharing Information (Part B implementation)

In this model, rovers can share information in the following ways.

- Their memory (their knowledge of agents and the location of those locations)
- Their status, such that they can help other rovers that are deadlocked.
- Their battery power – this allows a rover to be able to effectively donate battery such that the battery is not overloaded to go past the maximum value.
- Their current contract and any information related to it (e.g. recharge point, status, master rover and supply rover)
- Their current target rock location that they are moving towards.

3.1. Sharing Memory

Each rover has a memory which is built up over time based on its observations when scanning its environment. Through this, a rover can remember where rocks are or if a location contains a rover or is empty. Through this, each rover has an internal working model of the environment such that it can make decisions on rocks to collect. As the memory is based on their observations the reality can be different from reality. This is sometimes obvious when a rover heads to a location which doesn't contain a rock has it has already been collected, but the rover is not aware of that fact.

To further improve each rover's efficiency, rovers can share their memory with adjacent rovers.

When sharing memory, a rover will get the memory of an adjacent rover, it will then compare each entry in that rover's memory with each entry its memory. This is completed in the __process_info method.

When it finds a matching location, but with different Entities recorded between the two rovers memory. It will work through the following logic in order

1. If the rover's memory says that the entry is empty, then it will ignore the new information
2. If the new info says that the location is empty, then the rover will discard its current entry for that location and add the new information for that location.
3. If the rover's memory says that the entry is a Rock, then it will ignore the new information.
4. If the new info is a Rock then it will discard the current entry for that location in its current memory and add the new information to its memory for that location.

If the information is any other kind of Agent (e.g. another Rover) then the Rover will ignore the information (this is because rover movements change frequently so information is less reliable than own memory).

3.2. Helping deadlocked rovers and viewing battery power

When an adjacent rover has a status of "Deadlocked" then the rover will share any spare battery power that it has so that the deadlocked rover can have more options to return to the spaceship.

To calculate the battery power to donate the deadlocked rover's battery level is checked to ensure that not too much battery is donated. (It is also possible that the rover has already had battery power donated from another rover).

3.3. Contract information

When an adjacent rover has an open contract, the rover may choose to accept the open contract and become the supply to that rover. If the rover is the master of its contract and find another rover has a different contract, but with the same recharge point, the rover may choose to change its recharge point, based on where it has already reached its recharge point and whether the other rover has already reached its recharge point. Contracts are further explained in Section 4.

3.4. Target Rock Location Sharing

When Rovers are adjacent they will share information, on what rock they are currently moving towards. If both rovers are moving towards the same rock, then the rover that has checked will set a new target rock location, to avoid gridlocks in the environment, by having multiple rovers heading to the same location.

4. Rovers co-operating to cover a large distance (Part B implementation)

Rovers can cooperate to cover larger distances by setting up contracts.

The Contract class holds information about the roles each rover holds (master or supply), the recharge point that the master will work from and the supply rover will travel to supply the master rover with battery and collect rocks to bring back to the spaceship. Additionally, the contract records when the master rover has reached the recharge point and has a status ("Needs Supply" when waiting for a contract partner, "Agreed" when two rovers are actively working together or "Complete" when no more rocks can be collected from the recharge point).

A contract is set up once a rover can no longer reach any rocks through its movement alone. It will then setup up a contract with a recharge point on the limit of one rover's movement to allow supply to reach adjacent to this point and still be able to donate some battery power to the master rover.

The supply rover's job is to ferry rocks from the master rover to the spaceship and to ferry battery power from the spaceship to the master rover.

The master rover will move to the recharge point, then once it has enough battery to collect a rock it will collect that rock and return to the recharge point and wait for the rock to be collected and to be recharged.

Once all rocks that can be reached (according to the master rover's memory) have been collected and its battery is full it will change the status of the contract to "Complete". The master rover and supply rover will then both return to the spaceship.

A rover may then set up another Contract if another recharge point could collect more rocks.

Once all rocks have been collected over two rover's movement, no further contracts are setup.

If I was to develop this model this further, I would gradually increase the number of rovers in a contract, such that the distance covered would be greater, using n-1 recharge points for a contract involving n rovers, creating a chain of rovers. However, the more rovers in a contract the slower the collection would be as each rover could need to be fully charged before it could continue with its job. This is due to the exponential increase in steps that would be required to charge each additional rover in a contract.

5. Implemented Classes

The following figures give the class diagrams for this solution

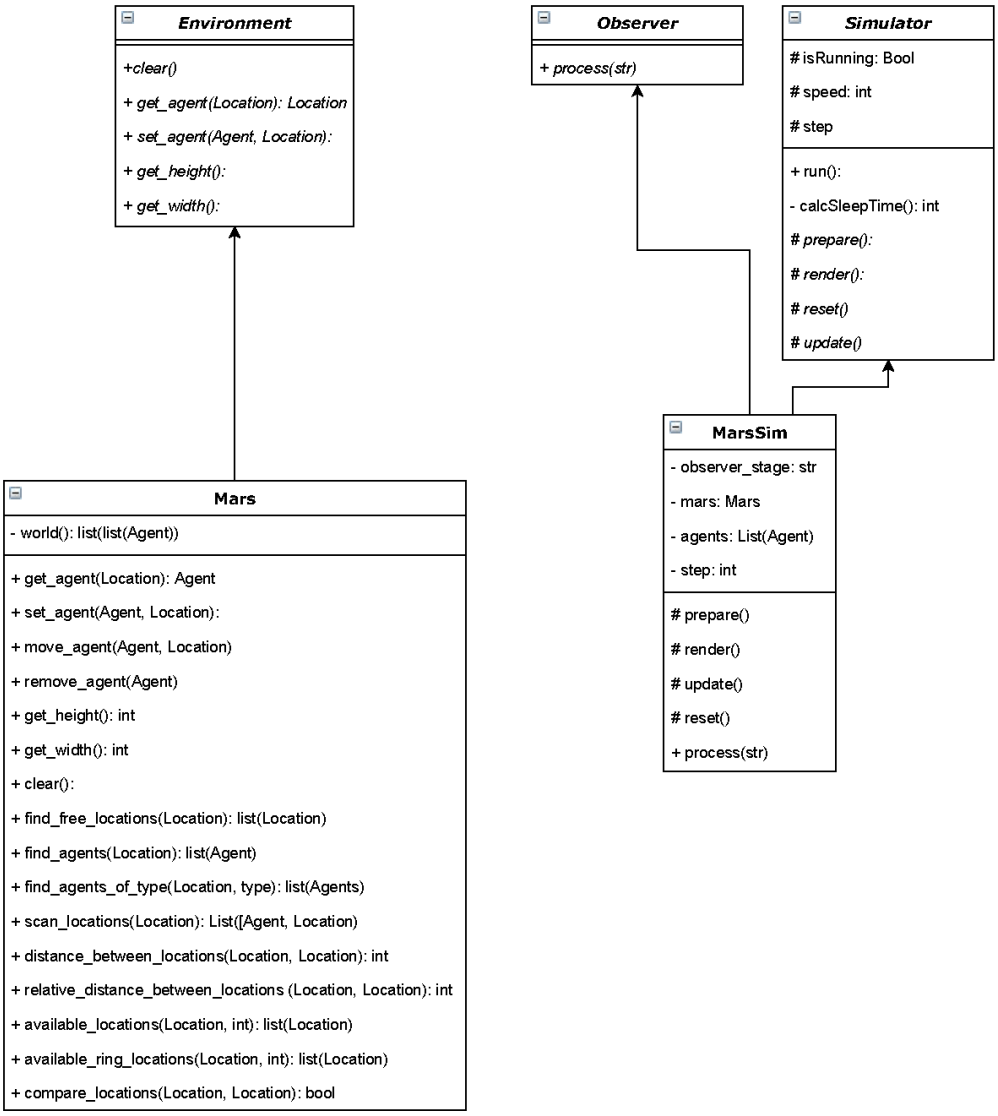

Figure 5: Class Diagram – Page 1

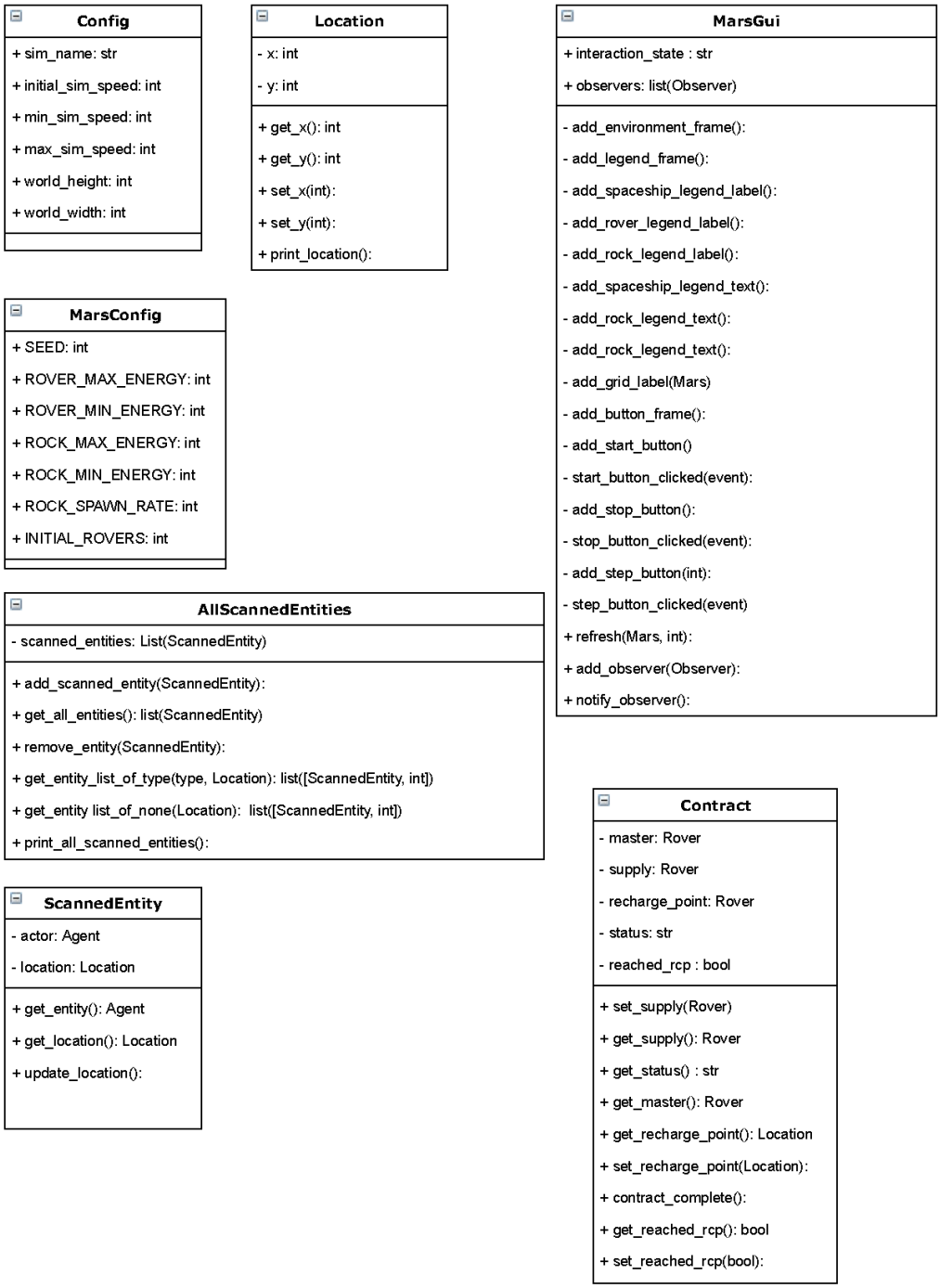

Figure 6: Class Diagram - Page 2

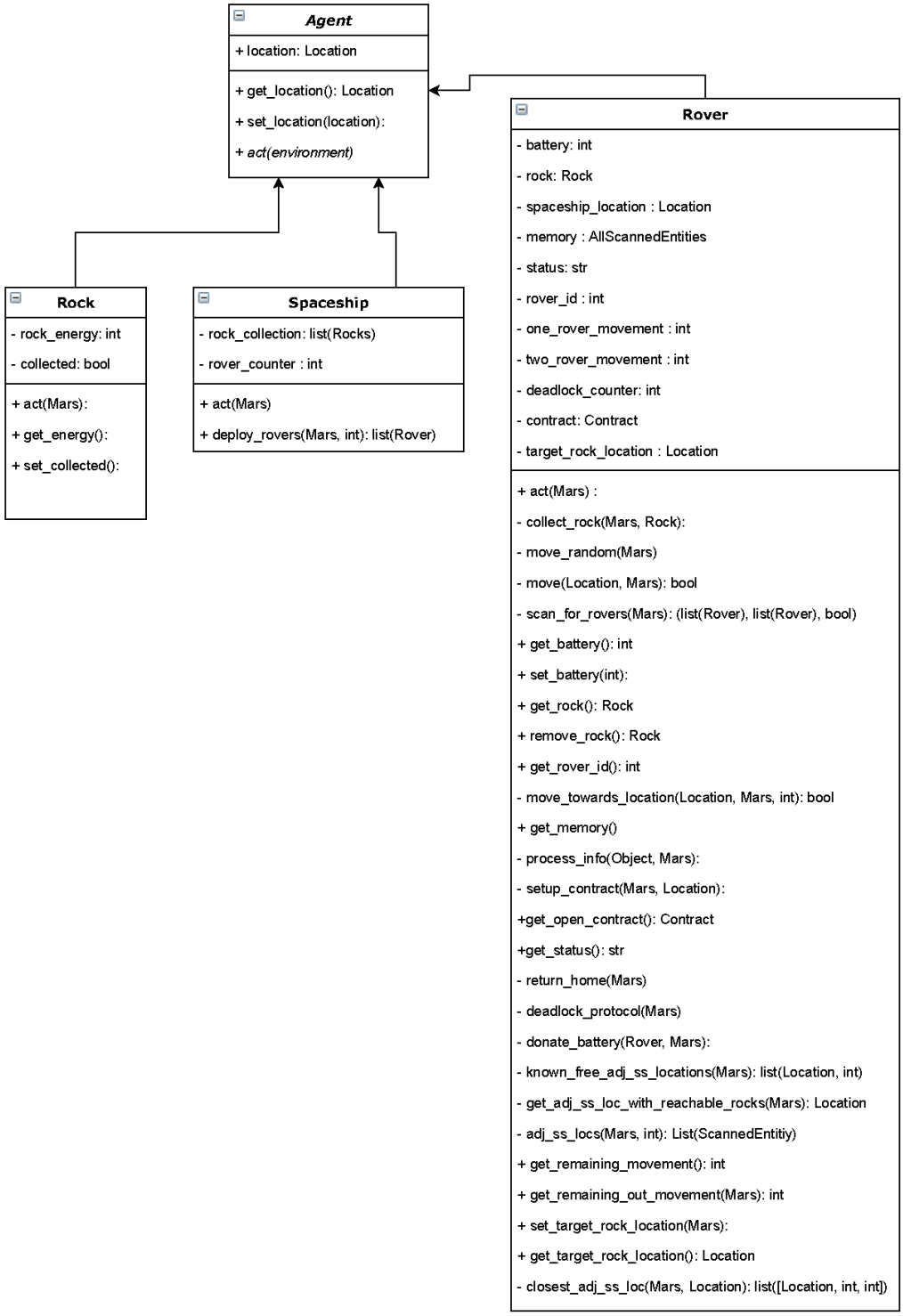

Figure 7: Class Diagram - Page 3 (Agents)