6. AgentSpeak – custom agents, internal actions and competition

Today, we will use one of the more complex examples from Jason. First, we will demonstrate, how to create own world representation for the agent and how to add custom internal actions. Finally, we will start the final competition.

The source codes for today are again in Jason (directory `examples/gold-miners-II`). It is an older scenario from an agent competition. The agents move in an environment with the goal to mine gold. Each agent can take at most three pieces of gold at the same time. The collected gold is transported to a specified location. Each team can contain at most six mining robots and one leader.

Exercise 26 Have a look at the source, try to run them and find out how the competition works.

Custom agent architecture

Last time, we used the default functions for the processing of messages and precepts and the environment was represented with the information in the belief base. It is simple, but not very convenient in more complicated cases. Therefore, Jason allows us to create a custom agent architecture. We can provide methods for the processing of percepts and incoming messages; we can write custom internal actions (because some things are easier in Java than in AgentSpeak) and we can create our own representation of the environment (so that we can search paths easier, for example).

The crate a custom agent architecture, we need to implement the `AgArch` class. In this class, we can redefine the `java.util.List<Literal> perceive()` and `void checkMail()` methods (and others).

We can redefine the `checkMail()` method if we want to process some messages in a specific way. In today example, the messages about obstacles and agent positions are removed and the model is updated according to them. Thanks to this, the agent is able to process more than one message in each loop instead of only one. At the same time, it correctly updates its internal representation of the environment (`arch/MinerArch.java`).

There are similar reasons to re-define the `perceive()` method. In the example, again, the method processes all the percepts and updates the model of the environment. It keeps only those percepts the agents needs to act upon in the queue (`arch/LocalMinerArch.java`, and methods from z `arch/MinerArch.java`).

Agents use a copy of the environment as their internal representation. They use it to not what they know about the environment, either from messages or from their own percepts (`arch/LocalWorldModel`).

The architecture used by the agent is specified in the `.mas2j` file as `agentArchClass` (see the example).

Exercise 27 Have a look at the architecture of the agents in the example.

Custom BeliefBase

Sometimes it is convenient to have a function which takes care of the belief base in a specific way. This can be done by implementing a new offspring of the `DefaultBeliefBase` class with a custom `add` method. You can see two such classes in the example – one of them ensures that a belief with the same functor is presented only once in the belief base (it is automatically updated when a new belief with the same functor is added), the other ensures that some beliefs are discarded before they are even added to the belief base (the agent does not care about these as they are already represented in the model of the environment). For the implementation details see `agent/UniqueBelsBB.java` and `agent/DiscardBelsBB.java`.

The belief base used by the agent is again specified in the `.mas2j` file – as `beliefBaseClass`.

Exercise 28 Have a look at the custom belief bases.

Custom event selection function

If we want to create a custom event selection function (or any other function from the Jason interpreter loop) we can implement a custom offspring of the `Agent` class. The example agent implements a custom event selection function in such a way that it first reacts to any newly found gold (this event has the highest priority). The new class can be again specified in the `.mas2j` file and you can find its example in `agent/SelectEvent.java`.

Exercise 29 Have a look at the custom agent class

Custom internal actions

Internal actions of agents can be used to change its internal representation, communicate with other agents, or do anything that is easier in Java than AgentSpeak (e.g. pathfinding). In order to create a custom internal action, you need to implement a `DefaultInternalAction` class and its `execute()` method. The methods has three parameters, one of them is `TransitionSystem`, which contains the information of the agents environment representation and its internal state. Another parameter is a unifier, which use can use to unify variables with variables in AgentSpeak and the last parameter is a list of `Terms` – the parameters of the action. Again, it is easier to have a look at the examples, you can find them in the `jia` directory. One of the simpler actions is obstacle, which indicates whether there is an obstacle on a given position.

Exercise 30 Prohlédněte si interní akce, které používají týmy v příkladu.

The mining competition

Your goal in the last competition in this term is to write a strategy for a team of six agents, which mine goal in a grid. Additionally, you can have one more agent which manages the whole team.

Environment description

The environment in this competition is a square grid, where some locations contain gold, other locations can contain obstacles. One of the locations is a depot, where the agents take all the gold. It is in the same location for both teams. The agent mines the gold by executing the pick action, it can drop it by executing the drop action (if it drops the gold at the depot, its team gets a point for every piece of gold dropped). Each agent can carry at most three pieces of gold at the same time.

The environment provides the following actions:

- `pick` – pick the gold from the position where the agent stands
- `drop` – drop the gold to the position of the agent
- `up,down,right,left` – move one square in the respective direction

The actions are called as `do(<action>)`, e.g. `do(drop)`.

In the beginning of simulations, the agents receive the following percepts:

- `gsize(sID, W, H)` – the environment for simulation with ID `sID` has width `W` and height `H`
- `depot(sID, x, y)` – the depot in simulation with ID `sID` is at `x`, `y`
- `steps(sID, n)` – simulation with ID `sID` runs for `n` steps

During the simulations, the environment provides the following percepts:

- `pos(x,y,s)` – in step `s` the agent is at position `(x,y)`
- `cell(x,y,t)` – at position `(x,y)` there is thing `t` (`obstacle`, `gold`, `enemy`, `ally`, `empty`) for all locations around the agent (up to distance 1)
- `container_has_space` – agent can carry more gold
- `carrying_gold(n)` – agent carries `n` pieces of gold

The agents also have fatigue, which determines the probability of the failing of percepts (i.e. agent does not get any new information in a given step) and of the failing of actions (the action is not executed). The fatigue of an agent depends on the amount of gold it carries (linearly between 0.1 and 0.5 for 0-3 pieces of gold).

Hints for the competition

Feel free to use the agents prepared in the examples. Use anything you want from them (they have A* for pathfinding). Name your agents in a unique way, so that we can easily run them. The tournament will take place on the last seminar on my computer. Please, send me your source codes and your `.mas2j` file.

You can use the `.send` internal action which sends a message to another agent, or the `.broadcast` action which sends the message to all agents in your team (it is augmented for the competition, otherwise it would send the message to everyone). The `.send` action has three parameters (the receiver, the performative – `tell` is `INFORM` from Jade, `achieve` is `REQUEST` from Jade, there is also `untell` –

and the message). Typically, the beliefs are send directly. The `.broadcast` action has only the last two parameters.

Exercise 31 Prepare the agent for the competiotion ☺.

What have we learned today

1. Change the Jason behavior
    a. Change the processing of messages
    b. Change the processing of percepts
    c. Create a custom function for the event selection
2. Create custom internal action
3. Use a more sophisticated representation of the environment.