Assessment1: Agent programming

Maixent ASSI

28 April 2019

Introduction

Throughout this report, we describe our design for agent programming when we find it relevant for a better understanding. To make it easy to check the results and run the agents, our feedback is organised this way:

- The folder **Task1__Task2** has 5 files in it:
  - "incinerator.asl": the incinerator agent
  - "move.asl": the code for task1, the basis for sweep
  - "move.py": the python code provided to us. It is initialized with "sweep.asl". So running ***python move.py*** will run the sweep agent
  - "move_to.asl": an intermediate file whose name should be put in the "move.py" file if instead of "sweep" one wants to run the move agent
  - "sweep.asl": the sweep agent as explained in this report.
- The folder **Task3** has 4 files in it. We mention the important ones ("move_v1.asl" and "singlecleaner.asl"):
  - "move_v1.asl": the basis file which holds the bulk of the agentspeak(L) code
  - "singlecleaner.asl": the implementation of the single cleaner. To run it type ***python move.py*** in a terminal
- The **Task4__non__collaboration**, **Task4__collaboration** and **Task4__partitioned** folders contain respectively the implementation of 4 non communicating agents, 4 communicating agents and 4 non-communicating agents in partitioned grid. To execute the agent, please get inside each of them and type in a terminal ***python move_2.py***. The **"move_2.py"** python file is a modified version of the *"move.py"*. In particular, it initialise the agents with the necessary initial beliefs in line with our implementation choices.

NB: all the above assumes that **pyson** is installed and running properly.

Task1

Though explicitly not required, we chose to include in the **"move.asl"** the possibility to locate garbage and the moves count simply because this file will be the foundation of our **"sweep.asl"** agent.

Task2

Our sweep agent is looping through the *10x10* grid. We achieve that behaviour by using two nested for loops, one for the horizontal axis and another one for the vertical axis. The two intelligent design choices we made were first to move like a snake and second to mark spots the agent has always been in as **"visited(X,Y)"** and add them to the belief base so that the agent heavily impacted by the randomness in the moves.

Task3

We called our agent **"singlecleaner.asl"**. It is based on **"move_v1.asl"** which is an augmented version of the previous *"move.asl"*. On top of what was done in the tasks 1 & 2, it adds the following:

- Due to the fact that an agent can sense garbage presence in the square surrounding its position, what we did is that each time the agent checks for garbage, we wait until its belief base is updated with garbage information. Then, instead of visiting the surrounding cells, if no garbage has been detected in them **not garbage(X,Y)**, then we mark this cell as **visited(X,Y)**.

- An agent picks a garbage if it is in a dirty cell and its hands are not busy. When both hands are full of garbage, it goes to location *(0,0)*, drops the garbage and ask the *incinerator* to incinerate via an **achieve** performative.
- To avoid situations, where the agent might have finished sweeping the grid, but end up with garbage in its hand, we make a final check to see it there is at least one garbage in hand. It is the case, the agent goes to *(0,0)* do drop it and call the incineration process.

Task4

At this stage, we think that simple thinking can be on par with the message passing solution to multi agent systems because the message checking functionality is clearly happening locally. This is shown by how simply giving each agent its own *"land"* appears to overperform communication.

In a first design, each agent loops or sweeps following the same implementation in *Task3*. In a sense, the agents follow exactly the same steps, which is very unproductive. For illustration, the 4 agents combined made an average of a little more than **26,300** moves (evaluated with 10 runs). What we observe is that some inefficiency occurs when some false garbage is collected. In addition, they visit the same cells, which is not optimised. By communicating with one another, agents can be more efficient. For example, the garbage belief that exists is an single agent setting is removed when this garbage is collected. Indeed, instead of updating their own beliefs, agents can share their beliefs with others, which will make other agents drop any intention about the same belief. With those modifications, agents collectively had around **14,800** moves (evaluated with 10 runs), which is a **70% efficiency gain** over the previous design.

In another design, we used the simple trick to give each agent a range on the x-axis, which is equivalent to partitioning the grid. In this setting, no communication was involved. As the results, the total number of moves of all four agents combined averaged **7,800** or put it differently more than a **3-fold improvement in efficiency** compared to when they all roam in the 50x50 grid, and **100% improvement** over the non-partitioned case with communication.

Conclusion

The assessment gave us knowledge and practical experience using AgentSpeak(L). The python framework used is still in development, thus lacks the full power of the *Jason* implementation. Apart from seemingly shared belief problem (which occurs when the .asl has some belief in it instead of the python code), we saw that the asynchronous nature of the interactions had some issues. When a belief is updated, we were able to retrieve the newly added belief if we waited some time, otherwise, the agent could not detect the change in its belief base. More importantly, in a situation like the one of multiple coordinating agents, the *pyson* framework lacked replanning features which would have given us the tool to totally drop some intention that were no longer valid. Finally, the partitioning trick shows that some simple common sense non-AI can provide immense improvement in efficiency.