Agents & Multi-Agent Systems

Yeshaswini Jekkamshetty

I. Scenario

This simulated Multi-Agent System is inspired from a real world scenario that is currently very relevant. With the on-going global crisis in place due to the Covid-19 pandemic, there are concerned individuals all over the world trying to get answers from authentic sources. One such implementation is the helpline services that is offered in most countries that enables the country's health corporation/services to support its citizens on first hand basis. The service offered by the online NHS 111 website is one such implementation and this scenario is loosely based on its template for the detection algorithm that determines whether a person has the Coronavirus and recommendation system that suggests what the person should do based on result of the detection algorithm.

The scenario is adapted to be a telephonic conversation between the patient and the doctor on a helpline for determination and diagnosis of the Coronavirus by assessing the symptoms of the patient. It can be thought of as an agent-simulation of an agent-human interaction with the doctor being the AI agent and patient being a human agent or as an agent-simulation of human-human interaction between a doctor and a patient. The scenario begins with the doctor agent answering the call and greets formally and then goes on to ask the patient how he may be of help. Upon listening to the patient's query about their symptoms, concerned whether they might have the virus, the doctor decides to ask a set of related questions about the symptoms to determine whether or not the person has the virus based on his expert belief base. When the doctor decides on his diagnosis of the patient's condition, he communicates it and along with it offers a recommendation on what they should do. After the patient is diagnosed with their condition, and has a satisfactory suggestion on what to do next, the patient thanks the doctor and ends the call.

The crafted scenario consisted of a doctor named Emma and three patients named Anna, John and Elsa. Three patients have different symptoms and so they all have a different diagnosis. According to the programmed belief systems, it is expected that Anna is diagnosed to have the virus, John's diagnosis should report that he doesn't have the virus and Elsa's diagnosis comes to show that she might have the virus. If the patient is diagnosed to have the virus, in our case Anna, they are recommended to get immediate help by calling the emergency line on 999. If they might have the virus, but do not show all the symptoms, in this case Elsa, they are advised to stay home and self-isolate. They are also advised to contact their local GP for help with managing the symptoms from home. Likewise, if the patient has no signs of the virus, in our case John, they are advised to continue with their life as usual and to follow precautionary measures and government advisory.

II. Implmentation

There are two AgentSpeak programs that were constructed for the implementation of this scenario. One being the doctor agent (doctol.asl) and the other being for the patient (patient.asl). Using these two AgentSpeak programs, a total of 4 agents were created in the main MAS file (.mas2j file). As discussed in the scenario, the 4 agents are Emma-The Doctor and Anna, Elsa and John – the patients. Simulation is implemented with the use of a virtual environment (HelplineEnvironment.java). It can be operated with the help of a window 'Add Percept' to change the patient agents to observe different versions of the implementation.

A. *Doctor AgentSpeak program*

The AgentSpeak program for Doctor file starts with **initial belief** that he's communicating with a patient and **rules** that the doctor's expert knowledge comprises. There are 3 rules to decide when a patient has the virus, may have the virus and when they don't have the virus. The example of how these rules are devised in relation to the patient's symptoms, is shown below with one instance.

```
yes_virus(X) :- yes_cough(X) & yes_high_fever(X) & yes_diarrea(X).
```

It goes on to having an **initial goal** to have to answer the call. This initiates the simulation and this is a triggering **achievement goal** that is core and characteristic to the doctor agent. This plan of this goal is to answer the call and to greet the patient while also **changing the belief** of that it is now on the call. This plan is only executable in the context that it is off the call, the implementation of which used a **context with weak negation.** This course of action is executed as follows_

```
+!answer(X)[source(self)]
: not in_call(X)
```

```
<- .print("Thank you for calling Coronavirus helpline. I have received a ", X, ". How may I
assist you today?"); +in_call(X);.
   +!answer(X)[source(self)] :  in_call(X) <- .print("this line is busy. call later").
```

Further, there is a **belief change triggering event** that is triggered when it is communicated internally through **Jason interaction** by the patient agent. This event is responsible to ask and communicate internally via **internal actions** to the patient agent about their current state of their symptoms that'll help in the diagnostic decision. The internal communication action that is used is 'askOne'. One instance of this internal communication used to understand the patient agents symptoms is as follows.

```
.send(Y, askOne, cough(Y));
```

Finally, more belief changing trigger events, also triggered when communicated from the patient agent by using the **Jason annotations**, are used to report the final diagnosis along with the recommendations. The context of which were based in the rules of the doctor's knowledge base. After the communication is over with the patient, as they are leaving, in the belief change event triggered when the patient is leaving, goodbyes are said and the belief of being 'in call' changes back to not 'in call'.

B. *Patient AgentSpeak program*

The initial belief base is a compilation of multiple patient agent's beliefs about the symptoms they are experiencing and not experiencing. The different agent's beliefs are included in the section so as to be able to access different versions of this implementations through the control window. As the agent's name and the belief triggering action of the query in symptoms is passed through the window, it triggers the primary **achievement (sub)goal** of the patient agent – to get a diagnosis.

This achievement (sub)goal is initiated in the **context with conjunction** that the patient has a query and is communicating with a doctor. The belief that the patient is communicating with the doctor is instilled in the environment. This achievement goal, is when the patient agent is able to express the issue and ask what they want help with and communicate internally to the doctor agent to trigger the inspection of symptoms from the doctor's side. This is implemented as shown.

```
+!diagnose(X)
       : doctor(Y) & query(X)
                    <- .print(Y, ", I would like a diagnosis with covid ", X);
                    .send(Y, tell, want_questioniare(X)).
+!diagnose(X)
       : true <- .print("it looks like help regarding your query about ", X, " is currently
unavailable").
```

As the inspection is going on regarding the symptoms, some **test (sub)goals** are used to check with the corresponding initial beliefs to respond and communicate accordingly to the doctor agent. An instance of such an implementation is as follows.

```
+?cough(X)[source(Y)]
       : has(X,cough) <- .print("Yes, I have cough");.send(Y, tell, yes_cough(X)).

+?cough(X)[source(Y)]
       : not has(X,cough) <- .print("No, I do not have cough");.send(Y, tell, no_cough(X)).
```

With the receiving of the diagnosis and the recommendation from the doctor agent, a belief triggers that the achievement is accomplished, therefore executes the plan to thank and say good bye to the doctor before leaving. 'Leave' is an action that is incorporated in the environment, that issues a belief that an agent is leaving to all. Upon declaring to leave, with this belief that it's self is leaving, the agent kills itself.