UG3/4 Agent-Based Systems Second Assessed Practical

Part 2 — Group organisation

Use the `submit` *program to submit your solution by executing the command*

```
submit abs 2 student.asl professor.asl
```

on any DICE machine. Note that the professor referred to is the professor in the Infospeak2 environment. The deadline for submission is

Thursday 17th March 2016 at 4 p.m.

Late submission policy: Normally, you will not be allowed to submit coursework late. If you have a good reason to need to submit late, you should submit an ITO support form as soon as possible. This can be found here: http://www.inf.ed.ac.uk/teaching/contact/index.html Only in exceptional circumstances, e.g. illness that stopped you getting to email, would an extension be granted after a deadline has passed. You can find a description of what constitutes ”good reason” to submit late at

http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests

Introduction

In this practical you are asked to design code for a student agent of which 10 instances will be generated in the environment. These students must collectively decide an appropriate time and date to hold a tutorial. Instead of averaging all their preferences out, they decide to resolve this issue by attempting to form a majority, which will then decide amongst the members of that group only.

Setting up the Environment

The steps below will take you through loading and running the assignment code:

1. Download the code and decompress it, e.g. with `tar xzf abs2.tgz`.
2. Start *Jason*, and load `infoSpeak.mas2j`.
3. Edit your code as appropriate, and run or debug with the buttons in the bottom right area of the IDE.

Note that you are supplied with additional predefined functions which are very useful for solving this assignment. The first two functions are useful for the first part of the questions and the third function should be used in the second part. These functions have the following form and function:

- ia.set_favorite(F) - should be called upon agent initialization, as it will unify F with a string representing this agents favourite day and hour for holding the tutorial.
- ia.get_best(L,F,B) - is used to get the best possible match between two agents' favourite times and dates for the tutorial. The three arguments represent:

1. a list of `favorite(A,P)` beliefs, each representing the favoured time string P of agent A;
2. the current agents' favoured time string
3. B will contain the name of the agent closest to the agent calling the ia.get_best function.

For example, ia.get_best([d3h20,d4h18,d2h16],d2h17,B) will have B unifying with d2h16.

- ia.get_rank(L, P, From, R) - is used to get the rank R of the agent's From preference, relative to all other preferences in the list L, with respect to preference P. After calling the function, R will contain an integer: 1 if From's preference is the closest to P than all others in list L; 2 if there is only one preference in list L closest to P etc. This will be useful for deciding whether or not to accept an invitation from an agent, based on his preferences and all your other choices (list L).

These functions should encapsulate any functionality pertaining to agent preferences. This means that you should not have any reason to interpret the preference string, or have any reason to generate such a string yourself.

Agent preferences

There are two ways to specify agent preferences:

- Using the ia.set_favorite(F) function described above. This approach will generate the favourite time values randomly and it is useful in cases you do not want to come up with favourite time values for every agent.
- Using the configuration file. This approach is useful if you want to test the behaviour of your agents in particular cases. The format for this is: `!ap <agent name>:<day>,<hour>`. Note that if a preference is specified for the professor agent, this will be interpreted as a choice.

The Questions

Your task is to implement the student agent so that it exhibits the behaviour described below. The total marks that can be obtained for each part are shown. It is important to comment your code fully. Comments should clearly and concisely annotate all plans, or groups of plans, and identify which rules are part of answers to which part of the question. If the purpose of a function of a plan is unclear, then you may not gain credit for it.

Forming the coalition (20 marks)

In this section you have to implement the student agent within the **InfoSpeak** environment, designing the mechanism for forming the coalition. Note that you can use visitor.asl as a template for this part. The file implements a lot of the functionality but not everything — choices and acceptance probabilities are not implemented.

The agents have to do the following:

1. Every day, each agent will try to convince one of their friends to join them. They need to find the agent closest to them in terms of time preferences not yet in their team and offer them to join their team.
   (2 marks)

2. Every day, each agent will respond to at most one such offer. It will accept the offer with probability $p = 1/R$, where R is the rank of the sender's preference relative to the receiving agent's preference. Use the ia.get_rank(L, From, P, R) function for obtaining the rank — L should be a list of all preferences of people not in your time. Use .findall(favorite(A, V), favorite(A,V) & not teammember(A), L) to create list L.

   Whenever an agent accepts an offer, their choice (i.e. vote) will change to the choice of the team (or agent) which they join. Tip: It is a good idea to broadcast these accepts as information about building the team might be inferred from them.
   (10 marks)

3. Every day, each agent (in this case exemplified with `agent1` for clarity) will then add the following agents to its team:

   - An agent who responded to `agent1`'s offer.
   - An agent to whom `agent1` responded.
   - An agent who responded to an offer of one of `agent1`'s coalition members.
   - An agent who already was in a coalition with an agent that joined `agent1`'s coalition (in other words, coalitions merge when a connection between them is formed).

   (3 marks)

4. When an agent detects that its team has sufficient agents to form a majority, i.e. at least half of the total number of agents are in its team, it must send a message containing the success belief to the professor. This will stop the execution of the system.
   (3 marks)

5. The agents must repeat this until a majority is formed.
   (2 marks)

Please note that the graphical interface will not be of much use to you in this assignment, unless you decide to implement some sort of behaviour for testing purposes. We recommend that you focus on the console and the mind inspector for debugging and testing.

Voting game - Tally Count (35 marks)

In this section you have to modify the professor agent, to count and report the votes for each choice formed by the coalitions in the previous part.

The professor agent has to do the following:

1. The professor must compute the total number of points obtained by every student agent's choice. To do this, all agents must send their choices to the professor upon form-

ing a majority. The professor must then count how many students vote for each choice. (25 marks)

2. The professor must print out the winning choice and the points it has achieved. (10 marks)

Voting game - Borda Count (45 marks)

In this section you have to modify the professor agent, to perform a different voting method: Borda count. This assigns points to all options, which are considered here to be students' favorite days and hours for holding the tutorial, through ranked ballots.

The agents must be modified to do the following:

1. Each student agent must rank all options (unique favourites of all students) and return a ranked list to the professor when asked by the professor (when it receives a message). (10 marks)

2. The professor must request a ballot (ranked list) from each student agent. Upon receiving all ballots, it must perform a Borda count for all unique favourite options of the students. This is achieved by adding points to each option according to rank, as in the example below.

Ranking	Option	Points
1st	d2h16	n
2nd	d3h12	n-1
3rd	d6h13	n-2
...	...	...

   where n is the total number of options (unique favourites of the students). (25 marks)

3. The professor must then print out the winning choice and the points it has achieved. (10 marks)

Plagiarism

Good Scholarly Practice: Please remember the University requirement as regards all assessed work for credit. Details about this can be found at:

http://www.ed.ac.uk/academic-services/students/undergraduate/discipline/academic-misconduct

and at:

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).