ID2209 Distributed Artificial Intelligence and Intelligent Agents

AssignmentY – Agents & stuff

Group 15

Zainab Alsaadi

2021-11-24

This assignment is about creating a Dutch auction using the FIPA communication protocol. It builds upon the simulation of the previous assignment.

How to run

Run Gama 1.8 and import FestivalAuctionModel.gaml as new project. Press main to run the simulation. This task was mainly focused on printouts in the console to show the communication.

Species

In addition to the species from the previous task, the following species were added:

Auctioneer: takes care of the auction and the interested guests. Decides when to start the auction, end it, announce the winner and prices. Communicates ONLY via FIPA.

Implementation

The protocol used in this task is *fipa_contract_net*. First, the guests sign up for the auction using the *inform* performative, adding themselves as content. Afterwards, the auction receives the inform messages from all agents and add each one to the list of participants. It then uses inform performative to announce the start of an auction. *Cfp* performative is then used to send messages containing the price at each round. Then auctioneer decides if the price is reduced after each round in case of no one wants to buy. If the price reaches a minimum value, the auction is cancelled.
If a guest is interested in buying, they use the *propose* performative. The auctioneer then accepts proposal and announce the end of the auction.

Result

Below is a sample printout from running the simulation.

```
[guests(0),guests(1),guests(2),guests(3),guests(4)]
Information: ['Auction starts!']
Information: ['Auction starts!']
Information: ['Auction starts!']
Information: ['Auction starts!']
Information: ['Auction starts!']
sending message
guests(0) received ['Selling for: 500']
guests(1) received ['Selling for: 500']
guests(2) received ['Selling for: 500']
guests(3) received ['Selling for: 500']
guests(4) received ['Selling for: 500']
(...)
(...)
guests(0) received ['Selling for: 500']
guests(1) received ['Selling for: 500']
```

```
guests(2) received ['Selling for: 500']
guests(3) received ['Selling for: 500']
guests(4) received ['Selling for: 500']
sending message
guests(0) received ['Selling for: 450']
guests(1) received ['Selling for: 450']
guests(2) received ['Selling for: 450']
guests(3) received ['Selling for: 450']
guests(4) received ['Selling for: 450']
We got a bid from ['guests(3)','[\'Selling for: 450\']']
Auction ended at ['guests(3)','[\'Selling for: 450\']']
```

As we see, first it shows the list of participants and then each guest prints the message of the auction start. The auctioneer then keeps sending messages and reduces the price until there is a buyer, or the auction is cancelled.

Challenge1

For this challenge, we create 2 auctioneers, one for “clothes” and the other for “art”. The guests have 2 Booleans that are set to true if they are interested in the corresponding auction. This time the guests only “sign up” for an auction only if they are interested in its genre. This way, both auctions keep a list of their own participants without interfering with each other. Moreover, the guest now loops over all cfps messages, since now they can be several messages from different auctions.

Discussion/ Conclusion

Solving the base assignment was not a problem, but the challenges are too challenging if you don’t have enough experience with gama, and it was not easy to ‘google’ the encountered problems. Reading the gama documentation and trying things out was fun but too time consuming which makes it hard to make it to the deadline.
However, the simulation could have been improved but the focus was mainly on the printouts in the console to track the communication between the auctioneer and the participants. The implementation could also have been improved by refactoring or trying better approaches to solve the task.