ID2209 – Distributed Artificial Intelligence and Intelligent Agents

Assignment 1 – Agents & Gama

Group 25

Yunfan Yang

Pei Lun Hsu

13.11.2019

In this assignment, we were asked to simulate the behavior of different agents in a festival, as well as the interaction between different agents. The purpose of the assignment is to let us be familiar with the Gama simulation platform and be able to implement simple multi-agents model by using Gama programming language.

How to run

Run GAMA 1.8 and import Model 02.gaml as a new project. Note that changing parameters nb_guest, nb_info, int nb_shop, and nb_guard in *global* will affect the initial number of the agents guest, information center, shop and guard, respectively.

1. Species

Agent Info

This agent is responsible to obtain the location of shops and guards, and pass the information of location to every guest agents when they enter the information center according to their request, i.e. reflex: ask from guest agents.

Agent shop

This agent is responsible to provide a location when guest can reset their hungry/thirsty state to full when they enter the store.

Agent guest

This agent is responsible for most of the actions. They conduct action **wander**, i.e., Idle state, when they are not hungry, thirsty or trying to report a bad guest. Once one of the above tasks appears, they went to agent info centre to obtain the location of either a store or a guard, and go to the corresponding agent to fulfil the task, i.e., eat, drink or report. After that, return to the idle state until the next task appears.

Agent Guard

This agent is responsible for go to the location of the bad agent, which are obtained from the reporting agent, remove the bad agent, and then go back to its original location.

2. Implementation

We first defined all the global variables including the number and location of agents. Next, we started developing agent guest and implement its behaviours including checking self-state, searching bad agents, going to the info centre, go eating/drinking, go reporting to guard, and backing to idle. Then we developed agent info and shop as agents with the only function is to store the location information. Finally, we created the agent guard. The guards can conduct the task of going to and removing the bad guest agent, once a normal guest agent reporting the location to it.

3. Results

As you can see in the following log, the guests can either go to shop via the info centre or go to the shop directly when they are hungry/thirsty.

Guest4 I'm hungry, I am going to foodshop 1

Guest4 I'm hungry, I am going to foodshop directly

As you can see in the following log, the guest can also report the location of the bad guest to the info centre, go to the guard and go to the bad guest with the guard.

Guest3 I'm going to info to report BAD guest

Guest3 I am close to the Guard

Guest3 I'm going to the BAD guest with the Guard

4. Challenge 1

To complete challenge 1, we add two local variables in agent *Guest* - point variable *visitied_shop_loc* and string variable *visited_shop_type*. Those two variables will store the shop type and location that was visited by the guest last time. A new reflex *directtoshop* is added before reflex *movetoinfo* and contains the same execution conditions with *movetoinfo*. Thus, this reflex, *directtoshop*, will always be executed before *movetoinfo*. Inside *directtoshop*, if the current guest state (thirsty or hungry) equals to the last visited shop type, there is a possibility (50%) that the location of last visited shop will be assigned to the current gonna-go shop position. Then, that guest may jump over the *movetoinfo* reflex and directly go to the shop visited last time without going to the information centre first.
Inside the *movetoinfo* reflex, we add code to collect the state and gonna-go shop location from other nearby guests within a distance of 2. The neighbouring guests will be attributed to a list and the present guest will randomly ask one of the neighbouring guests in the list to check if they have the same state and then get the shop location from it. Thus, while the guest is going to the information centre to ask for the shop location, it will also continuously ask its neighbours for help.

5. Challenge 2

We add three bool variables to address the state of the guest: *is_bad* - whether the guest is bad; *find_guard* - whether the guest finds the guard; is_reported - whether a band guest has been reported. Two point variables have also be added to store the guard location and the bad guest location. Several reflexes have been added to complete this bad-guest-guard function. Every guest will be possible to become bad in each iteration if it is not on the way to report another bad guest. Once it becomes bad, it will stop moving and restore the initial state. If normal guests currently haven’t found a bad guest yet, they will keep searching for all the neighbouring guests at a distance of 5 to check whether any of them is a bad guest. Once a bad guest is found, the normal guest who find it will store its location and go to the information centre to get the location of the guard then go to the guard to give the location of the bad guest to the guard. Finally, it will go to the bad guest again with the guard. On the guard side, the guard agent will reach the bad guest location and search for any guests on that location. If a bad one is found, the guard will ask it to die and go back to its initial location. When the guard is occupied, other guests who want to report the bad guest will need to wait at the initial position of the guard.

Result

Overall the assignment is great. The logic to implement is straightforward and we like the way how the agents are constructed. The debug for the code is really interesting - to change the execution condition of different reflexes and see it working properly is great.