ID2209 – Distributed Artificial Intelligence and Intelligent Agents

Assignment 1 - GAMA and agents

Group 18

Paul Hübner
Samuel Horacek
13.11.2023

In this assignment, we were tasked to become familiar with GAMA and derive a simple simulation of a festival. When festival guests get hungry or thirsty, they can go to information desks to find the location of food or drink stalls. They will most likely remember the locations for the next time they need to replenish. Sometimes, there are entitled festival goers, who are assassinated by security when they make a scene at the information desk.

How to Run

Run GAMA and import *Exercise1.gaml*, *Exercise1_Bonus1.gaml* and *Exercise1_Bonus2.gaml* as new models. Press main to run the simulation.

The parameters in *global* can be tweaked to change the number of guests, information centres, stalls, security guards, etc.

Species

InformationCenter

This is an information centre. Its job is to guide visiting FestivalGuests who are thirsty and/or hungry to the closest desired store. Interactions with it are done through other agents, who ask for its list.

FestivalGuest

FestivalGuest is the most common, always-moving agent. It is enjoying the festival, which tires it out and increases its hunger and thirst it monitors. Once it gets too hungry/thirsty, it goes to the closest InformationCenter to find a store that could help with this problem. Once they arrive at the required store, based on what it offers, their need is fulfilled (i.e. they stop being hungry/thirsty), thus they can go back to enjoying the party and move around randomly as people at festivals do.

In Bonus 1, this behaviour was slightly adjusted. Instead of always needing to go to the InformationCenter, to visit a store, they store information about the last store they visited. For more information, see the corresponding section.

Store

This is a store. A static agent that can sell drinks, food, or both. We differentiate between the three, so an agent who is just thirsty can only go to a drink-only store. We also have a store carrying both to make it easier for guests to enjoy the festival.

SecurityGuard (Bonus 2)

SecurityGuard is an agent kicking out (“killing”) misbehaving FestivalGuests. Its job is to chase them around the festival and if none have misbehaved, he can get a breather and

rest. He would start chasing FestivalGuests who misbehaved when visiting the InformationCenter, which reported them to the guard.

Implementation

Before our implementation, we assumed that festival guests who were either hungry or thirsty had to go to a food or drink stall, respectively. Our simplification makes it such that it is not possible to go to a store that sells both if only hungry or thirsty.

First, the agents were created: we implemented the information centre and then created the festival guest. Once these were working, we added the stores.

After that, we focused on behaviour. The first thing we did was detect if an agent was in a store, this was decently simple. After that, we had to figure out how to communicate the location of the stores to the agents. At first, we had the agent just take it from the local state. Then, we moved it to an *ask* as shown in Figure 1.

```
ask (InformationCenter closest_to location) {
  list<Store> potentialStores <- stores where (each.hasDrink = myself.isThirsty and each.hasFood = myself.isHungry);
  if (length(potentialStores) > 0) {
    myself.targetPoint <- potentialStores closest_to myself.location;
  } else {
    write "ERROR: No suitable store found, check world creation.";
    write "TRACE: Conditions are " + myself.isThirsty + ", " + myself.isHungry;
  }
}
```

Figure 1: Asking the information centre for store locations

Next, we implemented the interactions with stores. This was quite easy since the radius logic from the information centre could be used.

The bonus sections were also quite straightforward, we just had to extend our codebase with more business logic. In the case of the second bonus, we required a new agent.

Results

Our GUI can be seen in Figure 2. The triangle is the information centre, circles are agents. The squares are the shops: blue for drink, magenta for food. Black for both. When agents are hungry or thirsty, they turn magenta or blue as shown in Figure 3.

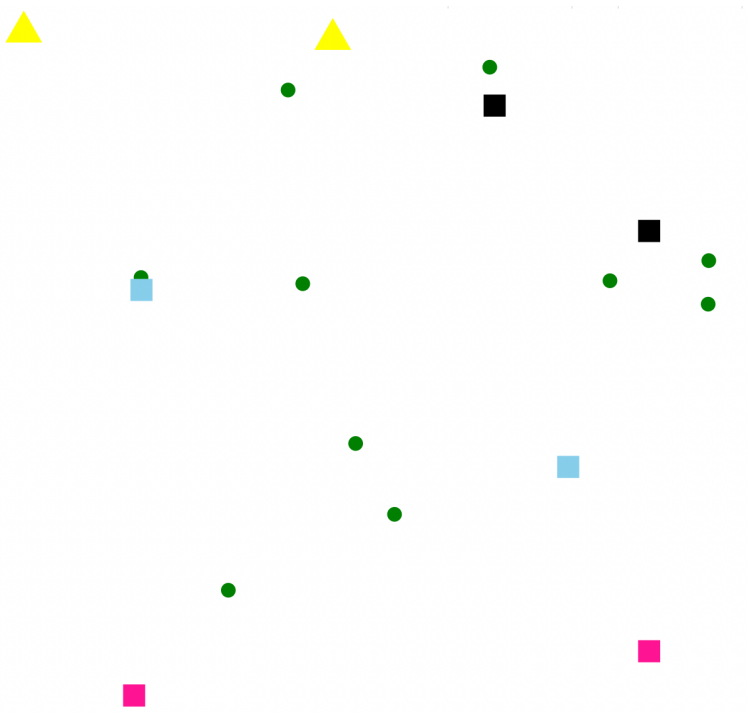

Figure 2: The GUI

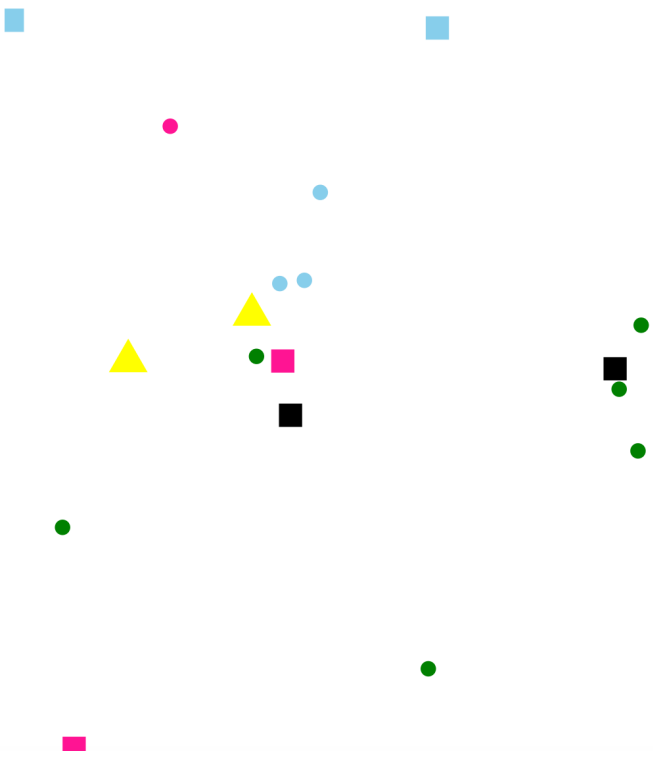

Figure 3: The GUI with hungry and thirsty agents

For the bonus, please refer to the next two sections. The benchmarks and descriptions are located there.

Challenge 1

There is a chance to remember a specific store. A chance of 100% means the agent will remember every store it has visited. If an agent does not remember a store, it will go to the information centre and ask for the nearest store.

We added a memory for each type of store to the agent. This means if the agent is thirsty, and it has been to a drink store, it may remember and go to it. If it is thirsty but not hungry, and only remembers a food and drink store, it must still visit the information centre.

To benchmark this, the number of cycles was limited to 10000, with 1 information centre. Consequently, we recorded the average number of steps taken. The steps start when the agent determines it is thirsty and/or hungry, and stop being recorded once it enters the store.

The average number of steps depending on the memory change is shown in Figure 4. As expected, the higher the chance of memory, the fewer steps the agents take on average. It could be that different worlds impact this, hence we run 5 worlds per trial and take the average.

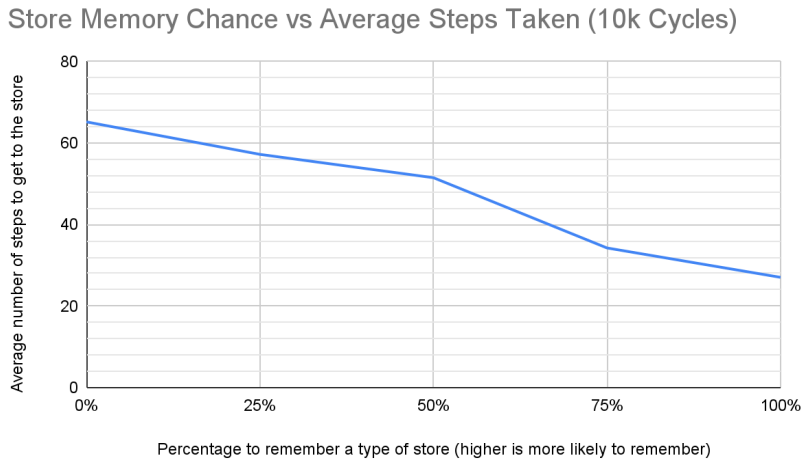

Figure 4: Bonus 1 benchmarking.

Challenge 2

When implementing Bonus 2, the biggest challenge was for the SecurityGuard to keep track of all, even previous, misbehaving FestivalGuests. This was needed to guard against a scenario, where if two guests misbehave in a row, he would not stop after catching the first one.

Instead of just storing the current target agent, we had to store a list of all previously misbehaving agents, which complicated the “chasing” logic. Now, after catching a guest, the guard either decides to follow the next one (in a FIFO order), or if there are no guests to catch it rests.

Creative Implementation

We added the possibility for multiple information centres and/or security guards. That way, it could be that festival-goers have even less to walk to specific stores.

Qualitative/Quantitative questions	*Answer*
Time spent on finding and developing the creative part	An extra 30 minutes.
In what area is your idea mostly related to...	The main assignment.
On the scale of 1-5, how much did the extra feature add to the assignment?	3.
On the scale of 1-5, how much did you learn from implementing your feature?	5, the logic of choosing the closest agent/store added a twist to our computation logic and challenged us to get better at GAMA.

Discussion/Conclusion

Creating the base assignment, with a single information centre and a few festival guests and stores was not difficult. We misread some of the requirements and had to later iterate on this a few times, which was unfortunate. But we enjoyed implementing both of the bonuses, as they were challenging at times. They gave us insight into Gama (and GAML) and how to better structure code for managing our agents and how they communicate.