ID2209–Distributed Artificial Intelligence and Intelligent Agents

# Assignment 1 - GAMA and agents

Group 11
Alexander Carlsson - alecarls@kth.se
Abyel Tesfay - abyel@kth.se

2021-11-12

## Introduction

In this assignment, we were tasked with creating a simple festival simulation in GAMA where guests can get thirsty or hungry and in turn visit stores to quench these needs. Store locations are acquired by asking at an information center and in a later challenge part, store locations can also be acquired from memory. In the final challenge we added a security guard and a chance of guests becoming bad. Bad agents are hunted down by the guard after they have been spotted at the information center.

## How to run

Run GAMA 1.7 (or newer). Create a new project and import the files from the zip. Select one of the files *hw1_basic, hw1_bonus1* or *hw1_bonus2* to view the implementations of the base task, challenges 1 and 2.

# Species

### Guest

Guest agents can be hungry, thirsty or none of the two. When hungry or thirsty agents seek out the InfoCenter agent in order to get a store location. When agents visit the store they go back to not being hungry or thirsty. When not hungry or thirsty the guests wander around the festival grounds.

### Store

Stores are static agents with booleans representing if they contain drinks or food but never both. These stores only serve as locations guests visit when they have needs.

### InfoCenter

The InfoCenter is also a static agent. The InfoCenter serves guests as they arrive with needs. If agents are hungry or thirsty they are given the location of the nearest store with food or drinks accordingly.

# Implementation

We made ourselves familiar with the example code provided by the tutorial and expanded upon it by adding behaviour to the Guests species so that hunger or thirst would remain until a store was visited. We added the InfoCenter species and made it linearly provide store locations to visiting Guests.

When this was done we expanded by making sure that Guests went back to an idling state after visiting a store with the correct facilities, thus a basic version was implemented. Finally we made the InfoCenter give the closest store locations based on their distance to the InfoCenter.

# Results

A short snapshot of a basic simulation if shown in Figure 1. From the figure we can see Guests(circles) in idling states (green) and hungry or thirsty guests (orange, purple). They are in clear traveling lines between stores(squares) and the InfoCenter(yellow triangle).

This shows that guests are idling around the location where they went into an idling state, at a store, and are travelling between the InfoCenter and Stores when they are in need of any of their services. We see that guests only travel to the two nearest stores, one for drink and one for food.

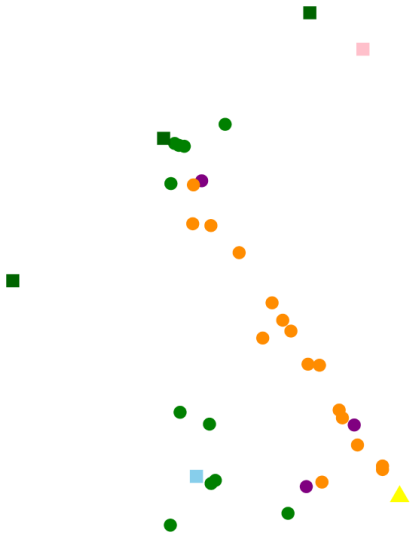

Figure 1: Snapshot of basic simulation

# Challenge 1

In order to complete challenge 1 we added a list to the Guest species which represents remembered stores. We also added a couple of counters to keep track of how many stores that had food or drinks, and counters to keep track of how many drink or food stores a Guest had visited. This was made in order to ensure that the InfoCenter only provided Guests with never-before visited Stores.

With only the memory in place we added the functionality for the InfoCenter to check the Guest memory and only recommend stores not already present in its memory. If the Guest is thirsty and has already visited all *n* drink Stores, the InfoCenter would tell the Guest to refer to its memory instead. Otherwise a new drink store would be recommended.

Lastly, we implemented a function for Guests to pick a store with correct facilities from its memory, and a flip which made Guests usually only refer to memory but sometimes going to the InfoCenter to look for something new. A snapshot of this version is represented in Figure 2.

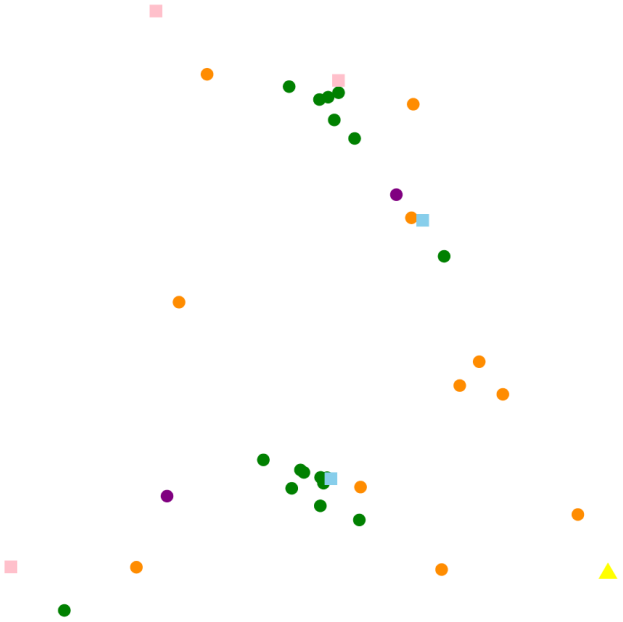

Figure 2: Snapshot of challenge 2 simulation

As we see in Figure 2 travel patterns are blurred as Guests now can visit stores. Some Guests are still on their way to/from the InfoCenter as they look for something new and Guests are still idling around stores.

# Challenge 2

To complete challenge 2, we added a small chance of Guests to start with a true *bad* boolean, represented in red. They behave the same as normal Guests but never change color. We added a check to the InfoCenter to control if a visiting Guest is bad or not. Bad guests are added to a list of bad guests and a new Species *SecurityGuard* is called to the InfoCenter.

The newly created SecurityGuard idles until it is called to the InfoCenter, in which the guard will go to the location. Once at the InfoCenter the guard takes the first bad guest from the list, hunts it down and kills it. If more bad guests have visited the InfoCenter during the hunt or if more bad guests remained in the list when the guard left the InfoCenter the guard will go back for a new description and start a new hunt. A snapshot of this behaviour is shown in Figure 3 where we see the same behaviour from Challenge 2 but with the addition of bad Guests in red and a SecurityGuard in blue on his/her way to hunt down a bad Guest.

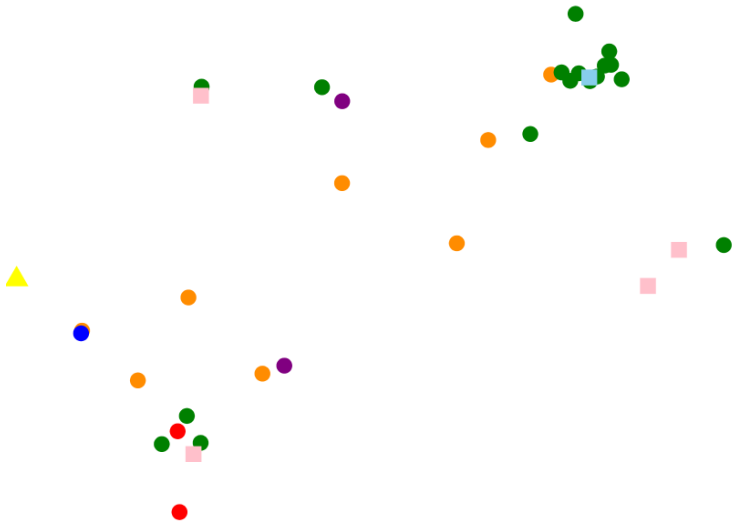

Figure 3: Snapshot of a security guard in action

# Creative implementation

Too much alcohol can change people and induce bad behaviour. In order to display the functionality of our SecurityGuard further and in order to explore GAML further we implemented the functionality for Guests to become bad when thirsty and visiting a store serving drinks. As all guests will eventually become bad we need a new way of adding guests to the simulation, so we created an EntryExit species where new guests are created or killed. With an entry and exit point in place we also allow guests to leave the festival.

We think this deserves an extra bonus as we have further fulfilled the goal of the assignment, to explore GAMA and gaml, while also adding further realism and life to the festival simulation.

# Discussion/Conclusion

Most of the assignment was completed without major issues with some syntactic details needing some time to grasp and figure out. Some larger issues encountered were initialization of lists where our list of bad guests was filled with all guests, this ended up with a rampaging security guard that killed every guest in the festival. We also encountered some outOfBounds exceptions as we lacked *when:* statements on some stores and InfoCenters causing issues when guests just bumped into them.

Through these problems and the assignment in general we believe we have gotten a good foundation to stand on for future assignments.