ID2209 – Distributed Artificial Intelligence and Intelligent Agents

Lab Assignment 3 Report

Group 28

Enrique Perez Soler — Daniel Conde Ortiz

December 15, 2019

1 Introduction

In this assignment we have 2 different tasks. For the first one we created a completely new scenario (chess board) and we used only one agent: Queen. For the second assignment we improved the simulation developed in *Lab 1*, adding FIPA communication between the guests and the stages.

2 How to run

Run GAMA 1.8 and import folder Lab3 as a new project. There are 2 tasks in this assignment so there are 2 files, each with one experiment. Press *Lab3chess* or *Lab3stages* to run each simulation. For the chess simulation we recommend using the step button and for the stage simulation we recommend to reduce the speed in the upper-left corner of the simulation near the "play" button in order to follow better the steps. At the beginning of the files you can change how many agents of each species there are or N for the chess part.

3 Species

3.1 Task 1

3.1.1 Queen

Represented as a red sphere. It has an order inside its same species (0,1,...,N). Queens start in order to look for a place. The search is sequential, trying cell after cell and checking if it collides with all the other already placed queens. As the task says, if a queen can't find a place, they ask the previous queen to change. 2 reflexes implement this in a similar way *find_cell* and *receive_request*, then only difference is that this last reflex is called when the following queen sends us a request to move.

When a queen has found a place, if a queen has asked before to move, it answers and tells them that they can look for a new place now. For this, in the receiving end, another method is implemented *read_agree*.

A few improvements have been added to make the solution finding quicker, such as only checking a row per queen and only checking for queens that are before us in the list.

3.2 Task 2

3.2.1 Stage

It's represented as a blue box. There are 4 of them and they have 6 attributes: lights, speakers, band, accessibility, people and field. They first initialize them to random values between 0 and 1. They also have a variable that represents the remaining concert time.

The main reflex is *reply*, that receives messages from the Guests and replies with it's own attributes. The other two reflexes are in charge of decreasing the concert time each step and generating a new concert

and its random time and attributes.

3.2.2 FestivalGuest

It's represented as a sphere and changes colours depending on the stage it has chosen. Each guest generates random values for preferences at the start of the simulation. It also has a float variable for the punctuation of the current best stage, a list of all stages and the selected one.

When a festival guest has not selected a stage or the current one has ended its concert, it asks (with CFP) all the stages what their values are (reflex *asking_for_info*) and, based on its preferences, chooses one and moves to that stage, once it receives a reply (reflex *read_proposes*).

4 Implementation

For the first task we started with a blank template and created the chess board, using a grid. Then we created the species Queen and learned how to move and interact with the grid. Then we added N Queens and implemented the searching algorithm, leaving the communication part for the end. After this we improved the performance of the algorithm in several parts and let it run to test that it worked with the highest N's.

For the second task we started with last lab's code as template. We changed the Auctioneer species for the Stage one and added the parameters and preferences to both it and the Festivalguest one. After this we implemented the concerts and their time and renewing them. Then we added the communication between both parts and the stage choosing and moving. Finally we added some aesthetic information so we can see better what's happening.

5 Results and Conclusion

For task 1 we have encountered a few difficulties when figuring out an algorithm to search for the right positions/cells in the board. We came up with several ways to approach it taking into account GAMA's programming language scope and functionalities that we have learned in previous works. Nevertheless, due to the fact that we needed for the queens to communicate between each other it turned out it was somewhat a more difficult or, better said, costly task. Meaning, it was costly computationally as should a queen find herself without an acceptable cell it'd communicate with her predecessor asking for a change and that'd mean searching from the first cell for a new available cell moving the rest of the queens as least as possible. But it was really interesting to figure out the sort of back and forward communication that took part when finding the solution. Down bellow we leave some screenshots of the simulation environment and how it's presented (figure 1) and the example solution of N=12 (figure 2)

As for task 2, we proceeded to evaluate the previous labs related to the festival and combined our recently acquired knowledge to find a way to adapt our knowledge in solving the intricacies of this task. Bellow we show a screenshot of the initial state and attributes shown in the simulation (figure 3) and a screenshot of an iteration of the simulation where we can see the guests choosing different concerts or events that best suit their preferences (figure 4)

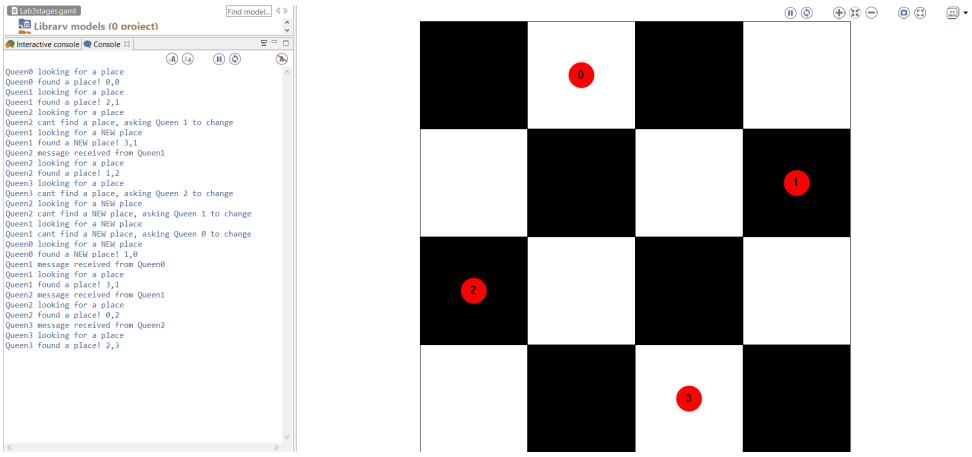

Figure 1: Screenshot of the execution of the Queen's problem in a N=4

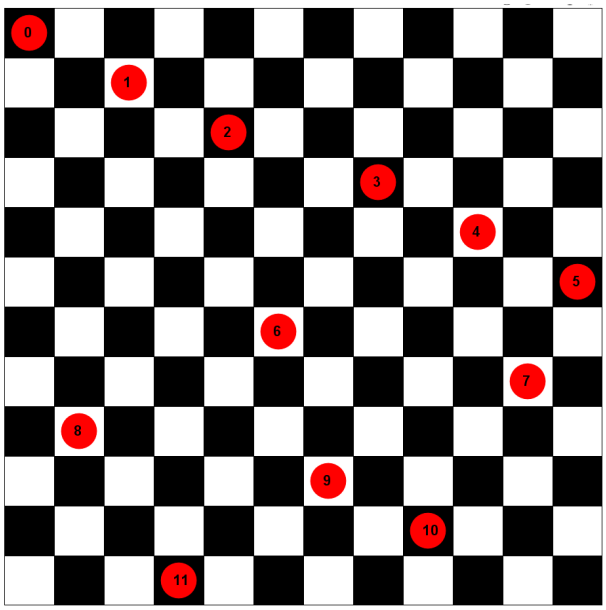

Figure 2: We can see the solution for N=12

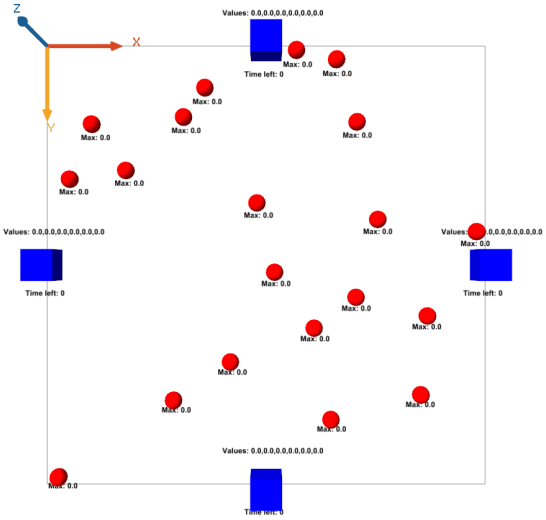

Figure 3: Screenshot of the initial state of the simulation

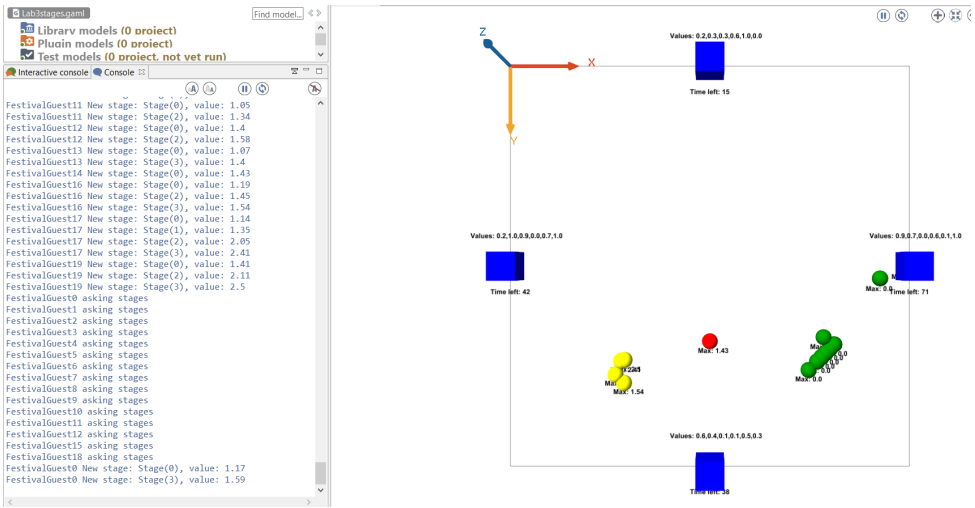

Figure 4: Screenshot of intermediate state of the simulation