Assignment 3 - Coordination and Utility

Group 28

Sara Moazez Gharebagh saramg@kth.se

Sebastian Lihammer lihammer@kth.se

5 December 2021

In this assignment, we were given two separate tasks: Task 1 was the *N Queens Problem*, i.e. finding a way to place *N* chess queens on a chessboard of the size *N*x*N* in such a way that they cannot attack each other (no queens sharing a column, row or diagonal with another queen). Task 2 was more closely related to the previous assignments in that it dealt with simulating an aspect of a festival. This time, the task was to simulate stages putting in different shows and guests choosing which stage to go to based on their preferences and calculated utility for the stages.

1. *How to run*

Run Gama 1.8.1 and import the **namn** folder as a new project. To run the simulation for Task 1, use **namn**.*gaml*. Press main to run the simulation. We recommend changing the speed to be slightly lower than maximum to better see what is happening. To run the simulation for Task 2, use **namn**.*gaml* instead.

2. Species

2.1. Task 1

2.1.1. Queen

Queen agents are initially placed at the side of the chessboard and added to a list. This list is used to keep track of each Queen's predecessor and successor. When all Queens have been initialized, the first Queen in the list starts the position finding process, when it is placed, it will tell its successor to move via the FIPA protocol. If the successor finds a safe position to stand on, it will tell its successor to move and so on. If it does not find a safe position to stand on, it will tell its predecessor to move until it finds a safe position to stand on. Each Queen will search for a position by going through all cells from the first row to the last row until it finds a safe position. This is done by making sure that there are no queens standing on the same row, column or diagonal. A matrix, with the same size as the chessboard, is used to keep track of the positions of all the queens; the occupied cells on the chessboard are marked with *true* in the matrix and the rest are marked with *false*.

2.1.2. Chessboard

The chessboard is a grid species that is created automatically at the beginning of the simulation. The chessboard has the same number of columns and rows as there are queens and has black and white cells.

2.2. Task 2

2.2.1. Guest

As it was decided to start the implementation of Task 2 from scratch, rather than building it on top of the previous festival assignments, the *Guest* species is relatively simple. Each guest is capable of moving and receiving FIPA messages and every guest has six floats that represent the different attributes of a show put on at a stage. Using these floats, which are fixed with a random value at initialization, the guest calculates the utility for a stage based on the method provided in the assignment instructions. If the utility of a new stage that has sent a message to the guest is higher than the stage the guest is currently at, it will move to the new stage.

2.2.2. Stage

Stages have the same six floats as guests, representing the same attributes, and also initially set them to some random value. Unlike for the guests, the values for the floats are not fixed for the stages, and will change at a random time interval (representing the changing of an artist performing on the stage). Every time the values change, the stage sends out a message to all guests containing the new values, which allows guests to determine whether to move to the stage or not.

3. Implementation

3.1. Task 1

The implementation of Task 1 started with creating the chessboard and the appearance and initialization of the queens. The simulation was designed to be in 3D, with the queens coded to appear as golden and chess figure-like. We added the skills *moving* and *fipa* to the Queen agents since they need to be able to move to their position and communicate using the FIPA protocol. Then, we created the reflexes that the queens need to be able to send and receive messages from their successor and predecessor. Afterwards, we created a reflex, *find_safe_position,* and some boolean methods, *validate_row, validate_col* and *validate_diag,* which are used to search for a safe position on the chessboard.

3.2. Task 2

The development of Task 2 was simple and relatively quick. As in Task 1, we chose to implement it so that the simulation would be in 3D: guests are represented as pink figures similar to the queens used in Task 1 and stages are represented as large and flattened black boxes. After deciding the appearance of the agents, we chose which attributes we were going to include, implemented the stage changing attributes at random intervals and sending messages, and then added the ability to read messages and move to the guests.

4. Results

3.1. Task 1

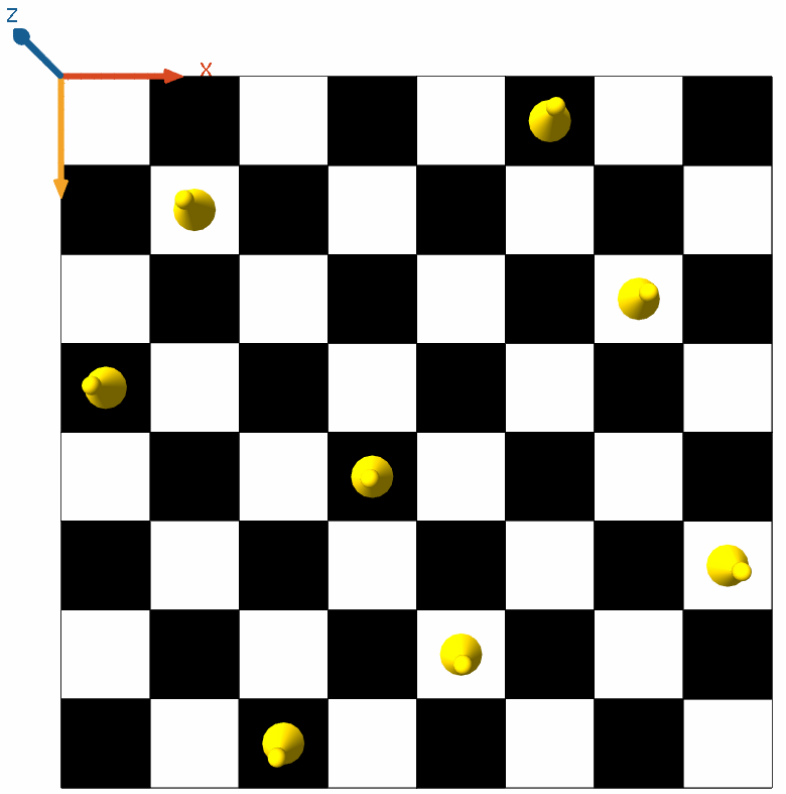

Figure 1: Screenshot of a 8x8 chessboard with 8 queens placed correctly

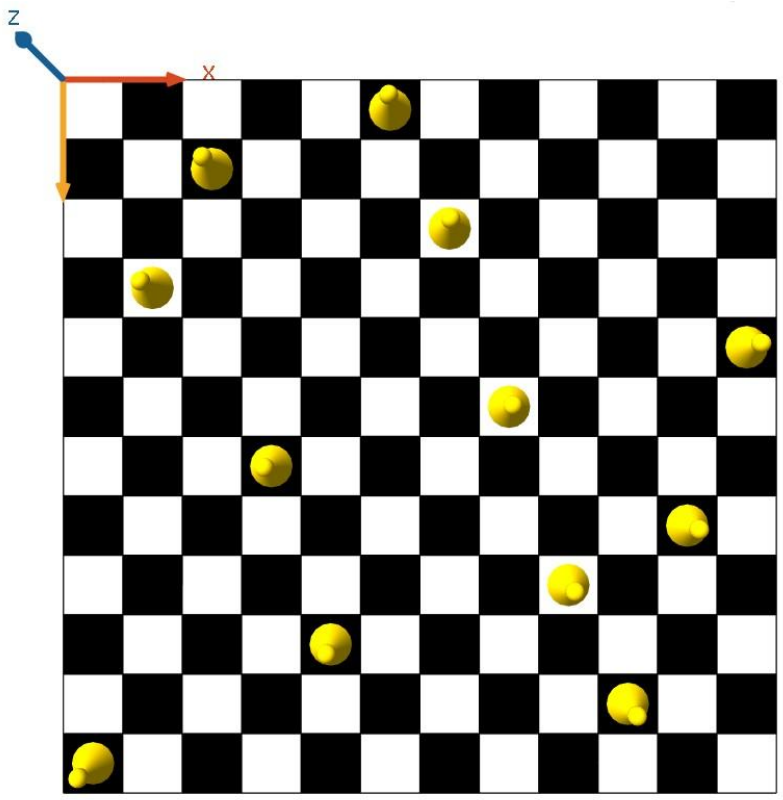

Figure 2: Screenshot of a 12x12 chessboard with 12 queens placed correctly

3.2. Task 2

The text below is an excerpt from the log of the simulation, which shows the preferences of guest 9, the different attributes of the shows put up at each stage and guest 9 choosing to go to stage 2. Because choosing the stage involves going through all received messages and this is at the beginning of the simulation (the only time when all stages put on a show at the exact same time), guest 9 first goes through and calculates the utility for stages 0 and 1. In the experiment run here, Guest 9's set of preferences gives the following utility values: Stage(0): 1.3, Stage(1): 1.63, Stage(2): 1.85 and Stage(3): 1.33, which is why Stage(2) is chosen.

```
Guest(9) has preferences: [0.9,0.6,0.0,0.3,0.5,0.2]
Stage(0) putting on a show: [0.8,0.2,0.6,0.1,0.7,0.4]
Stage(1) putting on a show: [1.0,0.1,0.3,1.0,0.7,0.1]
Stage(2) putting on a show: [1.0,0.7,0.2,0.5,0.6,0.4]
Stage(3) putting on a show: [0.5,0.1,0.5,0.9,0.9,0.5]
Guest(9) is changing stage to Stage(0). Previous utility value: 0.0 New utility value: 1.3000000000000003
Guest(9) is changing stage to Stage(1). Previous utility value: 1.3000000000000003 New utility value: 1.63
Guest(9) is changing stage to Stage(2). Previous utility value: 1.63 New utility value: 1.85
```

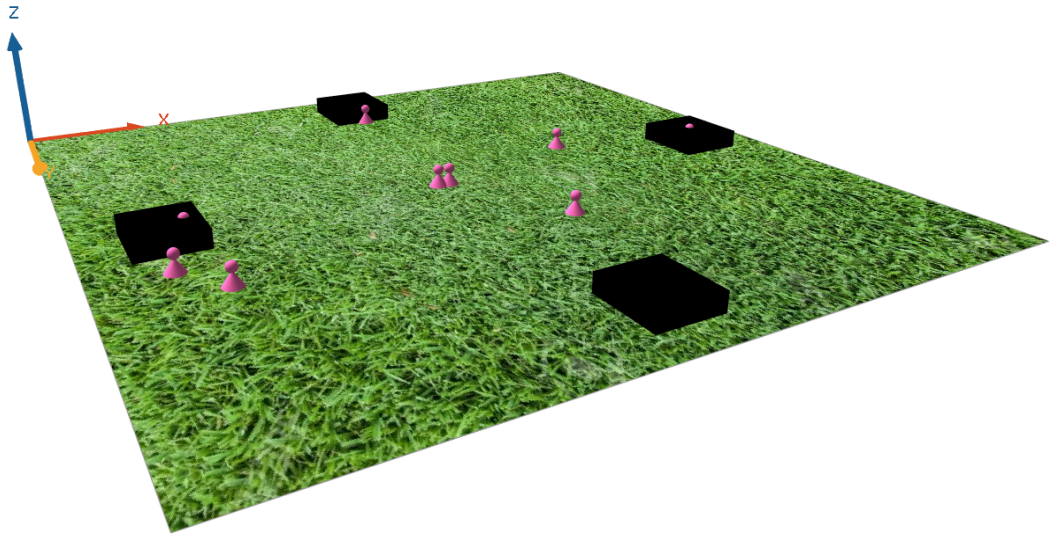

Figure 3: Screenshot of a simulation of Task 2

6. Discussion/Conclusion

The implementation of Task 1 was very difficult and time-consuming and we could not solve the assignment before the deadline. It was, however, very educational and challenging. The final solution of Task 1 is a bit slow when there are many queens, and thus need some improvements. Otherwise, it works fine. We had no problems solving Task 2 since it was a bit similar to assignment 1 and 2.