KTH H16P02 Distributed Artificial Intelligence and Intelligent Agents: Homework 3

Group 25: Fannar Magnusson (fannar@kth.se)
Thorsteinn Thorri Sigurdsson (ttsi@kth.se)

November 29, 2016

1 Introduction

For this third homework assignment we were given two tasks. The first one was to solve the N queens problem with agents (being queens) that can communicate and cooperate to achieve a solution for the problem. The second task was to extend the dutch auction we implemented in homework two, now for intra-platform mobility.

2 Tasks

2.1 N Queens

In our solution of the N Queens problem we only have one type of agent: a `QueenAgent`. To find a solution for the queens problem, for example N=4, we create four QueenAgents from command line arguments where each queen only takes in two arguments, *id* and *N*. When a QueenAgent is setup, she registers her `QueenService` with the DF, stating her *id* and *queen* service name. After that she goes into a waker behaviour. After 3 seconds, when all queens should have registered with the DF, the queen looks up it's predecessor and successor, if any, using the incremental ID for each queen. Finally when all setup is done, `queen0` can initiate the backtracking algorithm to find a solution for the problem. The solution works as follow: If we are finding a solution for e.g. N=4, we have four queens with IDs 0,1,2 and 3. Queen0 starts by placing itself in row 0, but in a random column (to support finding different solutions between runs). The queen then sends a `REQUEST` message to its successor, with her position, telling her to find a position as well. Queen1 tries to find a safe position on her row ID (row 1). If she does find a spot, she adds the position to her list of tried positions and to the list of filled positions on the table. Then she send a message to her successor, with the list of the filled positions on the table. If a queen is unable to find a spot she sends a request to her predecessor, who then

tries to find a new position, other than the positions she has already tried. Before sending a request back to her predecessor, a queen also clears her list of tried positions. If the predecessor is unable to find a new safe position, she sends a request back to her predecessor, asking her to find a new safe position, and so on, until some predecessor queen is able to find a new safe position. If a queen finds a new safe position, she sends a request to her successor asking her to find a safe position, and so on, until the last queen is reached. If the last queen finds a safe position, then all queens have found a safe position, and we print out the solution.

2.1.1 Implementation

The QueenAgent uses a `WakerBehaviour` to find predecessor, successor and start the algorithm. The agent uses a `CyclicBehaviour` to listen to messages from other queens that might send her a message to find a postition. Finally the agent uses a `OneShotBehaviour` to send a `SET_POSITION_REQUEST` to it's predecessor or successor.

When we register the queen service in the DF, we include a property `(ID, QueenId)` in the service description. This allows a queen to search for only her predecessor and successor (by searching for IDs that are one lower and one higher than her own ID, respectively).

2.2 Dutch Auction for intra-platform mobility

We took the example code that was referenced in the homework description, available on this page: `https://www.iro.umontreal.ca/~vaucher/Agents/Jade/Mobility.html` and used it as a base for our solution. We have a `ControllerAgent` that manages all the agents. He can create/kill agents and move them between containers. We also have a MobileAgent which we use as a base for the Curator and Artist Manager agents. They extend the MobileAgent and their GUI extends the MobileAgentGui.

To create the scenario described in the homework, we start by running up the `ControllerAgent`. We can use the ControllerAgent to create the `ArtistManager` that manages the auction. When the artist manager is setup he chooses a random painting to be auctioned. When the setup is done the artist manager is presented in a new GUI window. There we have the possibility to start auction, or setup other agents (which will create all agents required to perform the museo galileo/heritage malta scenario). So if we select to setup other agents, the ControllerAgent will start by cloning the ArtistManager two times and move the clones to Container-1 and Container-2. Note that the clones will have all the same properties as the initial ArtistManager and will therefore know what paining is to be auctioned. When the artist manager clones are ready, the ControllerAgent will create two CuratorAgents and place them in Container-1 and Container-2. When

the curator agents are setup they will get random painting interests and strategies. After the setup for them is done, they will be presented in new GUI windows, which will show their name, interests and strategies. Finally the ControllerAgent will make two clones of each of the two ControllerAgents and place them in the same container as their parent. Note that after a CuratorAgent is cloned, he will get new random painting interests and strategies - we did this to make the auctions more exciting, otherwise all the curators inside the container would just always bid the same amount. Now that we have all the agents setup and ready, we can let the original ArtistManager start the auction. We need to manually press the "Start auction in clones" button on the ArtistManagerAgentGui to do that. The ArtistManager will send a "Start auction in clones" message to all of his clones. When the artist manager clones get that message they will initiate a dutch auction for all the curator agents that have a bidding service running inside the same container. When the artist manager clones have received a result of the their auctions, they will move to their parent's container and report the results. Finally the original artist manager will select the best auction result and send a `ACCEPT_PROPOSAL` to the highest bidder from the two auctions (if there is a highest bidder).

2.2.1 Implementation

The `ArtistManagerAgent` and `CuratorAgent` extend the `MobileAgent` class given in the project's example code. We also created GUIs for them that extend the given `MobileAgentGui` and add appropriate buttons and fields.

When a clone is created, it starts a `CyclicBehaviour` in its `afterClone()` method to listen for start-auction messages from its parent, informing the clone that it should start an auction within its container.

The original ArtistManagerAgent sends the start-auction message to its clones using a `OneShotBehaviour`. Once it has done that, it starts a `CyclicBehaviour` that listens for the auction results from all its clones. Once it has received auction results from all its clones, it selects the best offer that its clones got, and sends an `ACCEPT_PROPOSAL` message to the winner.

The artist manager agents (auctioneers) run auctions only within their current container. They find a list of the curator agents (bidders) in their container by asking the DF for all curator agents (in the platform), and then querying the AMS to find which container each bidder is in, filtering their original list of bidders so it only includes bidders in the same container as themselves. This isn't very efficient, but we found no general way to query the DF or AMS for agents within a specific container.

We implemented a function in the ArtistManagerAgent, `setupOtherAgents()`, that creates the other agents and clones needed in the scenario automatically and moves them to the appropriate containers, so they are ready to

start their auctions. This function is run when the "Set up other agents" button in the original ArtistManagerAgent's GUI is pressed.

We adapted code from the `ControllerAgent` to do this. We also needed a way to update the main GUI window, which the ControllerAgent owns, with the new agents. Instead of creating a whole new main GUI, we implemented a workaround where we send a `new-agents-notification` inform message from the ArtistManagerAgent to the ControllerAgent to get him to update his GUI.