Homework 3 **Kim Hammar**
ID2209 Due Date: 29 November 2016
Distributed Artifical Intelligence and Intelligent Agents

Problem Statement

Implement the following using the JAVA Agent Development Framework (JADE) [1]

Task #1 M.A.S for solving the N-Queens problem where agents coordinate with each other to choose the right positions.

Task #2 M.A.S with mobile agents that can perform dutch auction in different places. Mobility means in this context that agents have the possibility to move between containers.

N-Queens

Agent Design

The M.A.S for task #1 consists of only one agent-type, the `QueenAgent`. The `QueenAgent` is designed to be reactive and receiving messages from other agents, upon receipt of a message from another queen, the agent will compute a "safe" position on the board according to the following algorithm (randomness is used to be able to find different solutions to the same puzzle):

Algorithm 1 QueenAgent algorithm for selecting a slot on the board

Require:
B ▷ Board
id ▷ Id of the agent
T ▷ Safe positions on the current board that have already been tried
Ensure:
p is the selected safe position which have not previously been tried. If no safe position is found, $p = -1$.
procedure SELECT_SAFES_SLOT(B,id,T)

```
R ← Shuffle(B[id])          ▷ random shuffle the row of possible positions
p ← −1
for r ∈ R do
    i ←indexOf(r)
    if safeDiagonally(i) ∧ safeVertically(i) ∧¬i ∈ T then
        p ← i
        break
    end if
end for
end procedure
```

Society Design

The `QueenAgent`'s will take turn selecting slot on the board, the first queen will initialize the puzzle by selecting a slot on the board and then notifying the *next* queen to indicate that it is its turn to select a slot. If a queen finds a safe slot and there is no queen left that have'nt selected a slot, the puzzle is solved. Further more if a `QueenAgent` fails to find a safe slot on the board it will notify the *preceeding* agent about this and ask it to change its slot on the board. If an agent fails to find a safe slot on the board that have'nt already been tried and there is no preceeding queen to notify, the puzzle is considered unsolvable. In order for the queens to find each other they register at the `DirectoryFacilitator` (DF).

Mobile Agents

Agent Design

The `CuratorAgent` and `ArtistManagerAgent` in this scenario uses the same behaviours as for Homework2 with a few modifications:

- Both agents runs in parallel to their other behaviours, a cyclic behaviour `ReceiveCommands` that receives commands from a controller agent to do one of the following: (*i*) move to a container (*ii*) clone itself (*iii*) kill itself
- Both agents use GUI's for interacting with the user and start/stopping auctions rather than the command-line as used in Homework2.
- The `ArtistManagerAgent` will run two cyclic behaviours `ClonesServer` and `AuctionResultServer` in parallel with other behaviours for receiving auction-results from clones and for receiving the final winner-bid from parents, respectively. Consequently, after finnishing an auction, if the `ArtistManagerAgent` is a clone and has a "parent"-agent, it will send the result to that agent and if a `ArtistManagerAgent` is a parent, it will collect results from clones, choose a winner, and notify the clones about the winner.
- Auctions are ran locally on containers, not globally on the platform.

In addition to the `CuratorAgent` and `ArtistManagerAgent`, a third agent called `ControllerAgent` is used. The `ControllerAgent` and associated `ControllerGUI` is inspired from the example code at the tutorial [2] that was recommended as a guide for this assignment. The `ControllerAgent` creates containers as well as agents and issues commands to existing agents to move, clone or die. The `ControllerAgent` communicates with the user through the `ControllerGUI`.

Society Design

The dutch auction interactions from Homework2 have been extended for intra-platform mobility where auctions are performed locally on containers instead of globally on platforms. Further more the results of auctions are forwarded by clones to parent-agents which will synthesize the results and present the best price from its point of view, the result is then forwarded to the clones who will notify the bidders about the result. The interactions between bidders and auctioneers in the actual auction is identical to the interactions presented in Homework2.

ControllerAgent communicates with the `AgentManagementSystem` (AMS) to retrieve a list of all containers on the platform. The ControllerAgent also communicates with agents that it have

created by sending simple one-to-one commands to agents, where no reply is expected. The commands that might be sent from the ControllerAgent to created agents are: clone, move, kill.

Conclusions

Extending simple agents to be intra-platform mobile allows for designing agents with further capabilities for acting autonomously. The agents for this homework were limited to mobility within a single platform but the same design principles could be applied for inter-platform mobility if the underlying agent-architecture allows for it.

Attachments

Documented source code can be found in the attached zipfile. See README.MD in the root directory for instruction on how to execute and build the program.

References

[1] Telecom Italia. Java agent development framework. `http://jade.tilab.com/`, 2016. [Online; accessed 11-Nov-2016].

[2] Jean Vaucher and Ambroise Ncho. Jade tutorial and primer. `http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html`, 2003. [Online; accessed 26-Nov-2016].