Distributed Artificial Intelligence and Intelligent Agents Homework 1

KTH Royal Institute of Technology
School of Information and Communication Technology
Student:Fanti Machmount Al Samisti (fmas@kth.se)
Student:August Bonds (bonds@kth.se)

December 6, 2015

1 Introduction

The purpose of this lab was to learn mobility in the Java Agent DEvelopment framework, the concept of cloning and moving agents between containers.

We also had to implement a multi-agent solution to the N-Queens problem.

This report serves primarily as an explanation of our solution, and some reasoning behind. Attached is all source code with descriptional comments.

2 Task 1 - The N-Queens Problem

The task was to implement a solution to the N-Queens problem using agents. This problem is normally solved with either a BFS or DFS over potential placement configurations. Since we did not want to rely on a centralized knowledge-repository (a stack or queue of configurations to try) we had to think differently.

Our implementation found solutions traversing the solution space in a DFS-like manner, but instead of operating on a shared datastructure, each agent would pass the current positioning of all predecessors plus its own position forward, and keep track of every tried position for itself given that predecessor configuration.

The agents thus communicated using only two kinds of messages. A "Go" message containing current positions sent to the successor, and a "Backtrack" message for telling a predecessor that the current branch is impossible.

When agent N found a possible placement given the N-1 predecessor positions, it would print the configuration and send a backtrack message for the algorithm to continue to find other configurations.

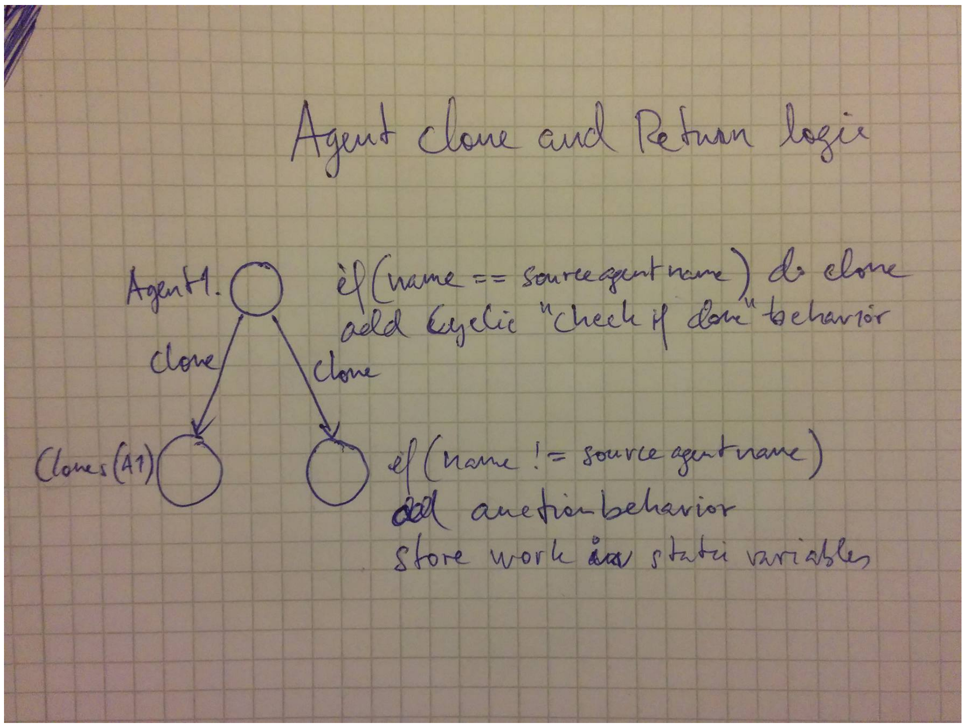
Agent clone and Return logic
Agent1.
if(name == sourceagentname) do clone
add cyclic "check if done" behavior
clone
clone
Clones(A1)
if(name != sourceagentname)
add auction behavior
store work as static variables

Figure 1: Overview of agent and clone logic

3 Task 2 - Hands-on with Agent Mobility

The task was to run at least two auctions, each one running in a different container, but with both artist manager agents and curator agents actually sending clones to do the auctioning/bidding and afterwards collecting the results in the original agent.

Our non-clone solution was using the ContractNetInitiator/Responder behaviors for auctioning, and this could neatly be expanded to cover clones.

To handle different behaviors of clones vs. originals we simply added/removed behaviors dependent on the name of the agent. We kept a copy of the source agent name for comparison, and let the clone agents add behaviors if they didn't have that name.

Any work done by clones was stored as static variables.

We then kept a counter for "returned agents", so that when all agents had returned we could execute an operation on the sum of the clones' work. An overview of this logic can be found in 1.

Timing when dealing with the creating and cloning of multiple agents turned out to be important. We had to add a delay before running any auctioneer code to make sure that all the curator clones were properly set up.

4 Running the code

Import the project into IntelliJ as a Maven project and run Main.java with one argument for board size for N-Queens.

Import the project and run MainClone.java for task 2.

5 Evaluation

In this lab we implemented a distributed solution to an algorithmic problem which we would normally be solving with non-distributed means. Additionally, by extending the previous homeworks we learned about agent mobility and cloning between containers.