NetLogo - Dining Philosophers

Tristan Claverie
950418P612
trcl16@student.bth.se

Damien Duvacher
941206P633
dadu16@student.bth.se

***Abstract*—The dining philosophers is a very common synchronization problem. It is often used as a learning material to teach students synchronization issues such as deadlocks and techniques to solve them. There have been many variants and solutions of this problem. The purpose of the studied program is to solve this problem using a Multi-Agent System**

I. Introduction

The problem is the following : N philosophers are sitting on a round table. Each one has a spaghetti plate in front of him, and two forks. He shares his right fork with his right neighbor, and the left fork with his left neighbor. Thus, there are N forks in total. One philosopher can eat only if he holds the two forks, otherwise he can't.

Each philosopher is in one of the three following state of mind: either *THINKING*, *STARVING* or *EATING*. Philosophers have a certain probability to get from *THINKING* to *STARVING*, in which case they have to eat (thus they need the two forks). While in state *EATING*, they have a certain probability to get to a *THINKING* state : they put down the two forks. The aim of the problem is to maximize the time philosophers spent doing their job, that is *THINKING*.

It may not seem so, but the problem can be tricky to solve, as many deadlock can happen. For example, if each philosopher ends up with only his left fork, then there are no more forks available and the simulation reaches one of those deadlock : the simulation cannot go further.

The studied program intends to solve the dining philosophers problem in two ways using a different Multi-Agent System for each. Each philosopher is represented using an agent. The first solution involves autonomous agents, that is they do not communicate with one another. The second solution involves smarter agents : each agent communicates with his neighbors.

II. Agent Architecture

An architecture for an agent define the global work methods of the agent.
It exists many architectures but 2 are more famous:

- Reactive Agent
- Reasoning Agent

The first one is when there are many behaviours developped for and agent and then when an event occurs, the agent react to this event by chosing one of his behaviour. So the agent just react when events occurs.
Whereas the reasoning agent will look at his environement and his goal then he will deliberate to find a solution to reach his goal regarding his starting state.
The second one is more intelligent but it's not very effective for simple tasks which need quick reaction.
These 2 architectures are different but sometimes you can mix both to create an hybrid agent. So you can adapt to many situations.
In this part we will describe the architecture for the 2 kind of agent in the philosophy dinner problem, for the naive agent (whithout cooperation) and for the smart agent(with cooperation).

A. Naive Agent

This kind is self fish and will act to satisfy his needs.
Basically, the goal is to eat when the philosopher is hungry. Then when the philosopher become hungry he will just try to get the 2 forks to eat and if he has the 2 forks, he starts to eat until he becomes full.
This strategy is really simple because the agent (or the philosopher) just look at the forks, if they are available or not and then take it to eat wihtout consider other agent needs. So for this kind of philosopher it's a reactive agent that is used because this behaviour is just a series of if. So the agent just react to different situation, there is no deliberation to reach his goal.

B. Smart Agent

This philosopher is a little bit more smart because he will look at the marks on the forks and will mark the forks that he need. The goal is the same: eat when hungry. But this timethe agent be careful of other agents needs.
Despite this difference the architecture for this smart philospoher is the same than the naive: a reactive agent.
Because the difference is about the marks on the forks, and the marks are managed by a "if series" too, indeed, if a philospoh has a fork marked and he doesn't have the other fork he will put down the fork because the mark mean that an other philosoph need it.
As you can see before it's just manage by many "if", he doesn't think about his environment and how he can reach his goal.

III. Agent Methodology

We apply the GAIA methodology to the cooperative Multi-Agent System. It allows us to get on organizational view of the

system being modeled. We let the naive Multi-Agent System aside as there are no interactions between agents.

A. Role model

This Agent-based System contains only one role : the Philosopher.

- Role schema: Philosopher
- Description: This role involves thinking as much as possible and helping its neighbors to eat if possible by releasing a fork if it allows the neighbor to eat.
- Protocols and Activities: Think, Starve, Eat, TakeLeftFork, TakeRightFork, ReleaseLeftFork, ReleaseRightFork, TellNeighbors.
- Permissions:
  - reads leftForkStatus // Is the left fork available or wanted by another Philosopher
  - reads rightForkStatus // Is the right for available or wanted by another Philosopher
  - changes leftForkMarked // Mark the left fork as wanted
  - changes rightForkMarked // Mark the right fork as wanted
  - changes leftForkTaken // Take or release the left fork
  - changesrightForkTaken // Take or release the right fork
- Liveness: PHILOSOPHER: (Think, Starve, Take, Eat, Release $)^{\omega}$
  TAKE: (TakeLeftFork, TakeRightFork)
  RELEASE: (ReleaseLeftFork, ReleaseRightFork, TellNeighbors)
- Safety:
  - true

B. Interaction model

There are two protocols in the interaction model : *Take* and *Release*

Take

- Purpose: Take a fork
- Initiator: Philosopher
- Responder: Philosopher
- Inputs: state = STARVING
- Outputs: Fork Status
- Processing: The initiator is hungry and ask his neighbors if their fork are available for taking, if yes take them, otherwise wait and try again.

Release

- Purpose: Take a fork
- Initiator: Philosopher
- Responder: —
- Inputs: —
- Outputs: —
- Processing: When the initiator is done eating, release the forks and notify the neighbors.

C. Behavior rules

- If an agent is starving, try to take the left fork and the right fork.
- If an agent holds only one fork which is aready marked, release it.
- When an agent is done eating, he should release immediately his forks.

D. Agent model

- A Philosopher is responsible for his state as well as his neighbors's.
- A Philosopher divides the shared resources between himself and his neighbors, he always seeks to do the best use of the shared resources, even if it means waiting before taking a resource.
- A Philosopher must update the state of the resources accordingly to his objective.

IV. Agent Communication and Interaction

In this part we wiil talk about the interaction and the communication between the different agents in this multi-agent system. We will first see the intercation between naive agents and then for the smart agent.

A. Naive agent

For the naive version of the agent, the communication is not really advanced because the philospohers don't communicate between them. One philospoh will just communicate with the fork, he will just look if they are available or not. And the interaction is not really complicated neither because a philosopher can just take or release forks. If we want to find a little part of interaction between philosopher it's that if a philosopher have 2 forks another one can't have his 2 forks.
So for the naive version there is no lot of interaction and communication.

B. Smart agent

This kind of philosopher is more interesting because smart philosopher are supposed to cooperate to help each other to reach their goal: eat and think.
Here there is a communication between philosopher because they mark a fork, that means they want this fork and if a philosopher have a marked fork but doesn't have the other one he will release this fork because someone else need it.
It's a kind of communication not directly but by the forks, they send informations to each other marking forks they need. The interaction between philosopher is the same that the naive agent.

V. Conclusion

This multi-agent system developed for solving the dining philosophers problem aims at maximizing the time the philosophers spent thinking. To do so, each agent communicates with his neighbors in order to cooperate and make the best use of the shared resources. Based on the observations, the naive agents reach a deadlock very fast because they do not

take their environment into account. The smart agent use the environment informations in order to get to their goals, thus reaching a solution of the problem.