Multi-Agent System

Concurrent Systems

Mohamad Ammar Alsherfawi Aljazaerly
ID:********

7/Aug/2019

Prof. Hiroyuki Nakagawa

Swarm Simulation Using JADE[3]

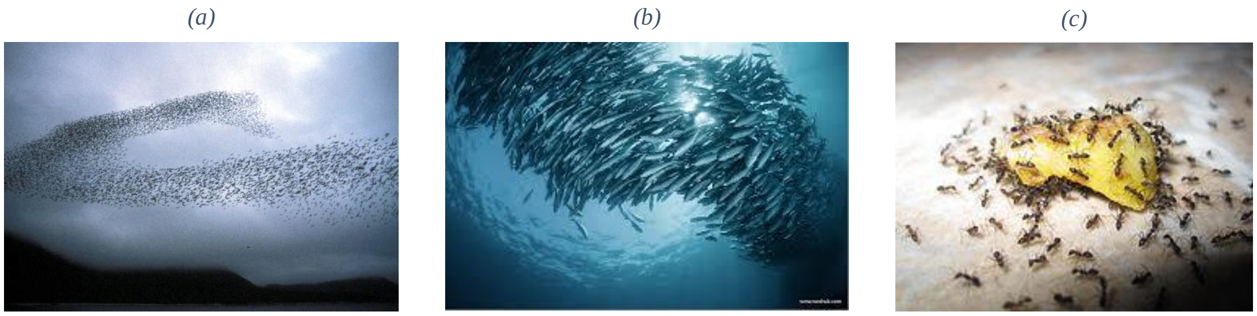

Figure 1: examples of different swarms. (a) is a flock of birds. (b) is a school of fish. (c) is an army of ants

A flock of birds flying in unison or school of fish swimming together has always mesmerized the human curiosity. These swarms have inspired and enabled us to do lots of applications like supply chain optimization [1], modeling of human behavior [2], and many others. Mesmerized by this phenomenon, we present in this report a simple simulation of a pack of agents looking for one another.

System introduction:

The simulation takes place in a rectangular-shaped environment with no obstacles inside. However, this rectangular-shaped environment has its edges connected to form a torus, as shown in the figure below. So when an agent goes through on the side of the environment, it will pop out of the other side.

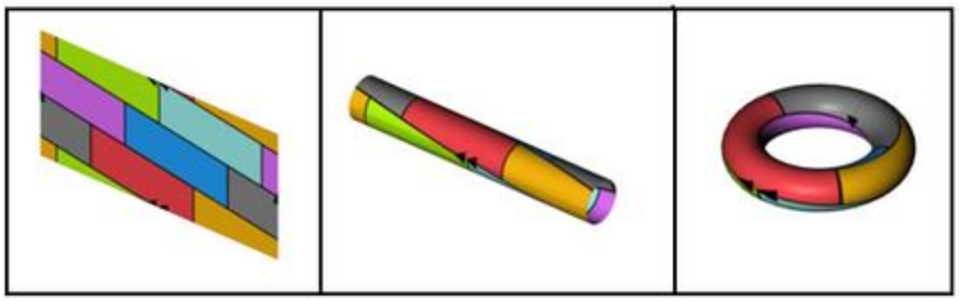

Figure 2: How to create a torus from a rectangle

In our simulation, the environment itself is an agent that tracks other agents' location in it. The environment retains information about the current location and speed of the agents in it, and the relative position of agents to one another. So an agent can request to know the relative position of neighboring agents, and the environment reply with the information of the other agents under a certain maximum distance. This simulates an agent looking around itself to find other agents, and it can only look far off to a limited distance. The agent then can request to change its course by informing the environment to change its speed vector. In the case of no neighbors in sight, the agent would choose to randomly change its course based on a random walk probabilistic distribution. We chose Rayleigh distribution as a base of our random walk generator, which is defined as follow: $f(x;\theta) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)}$. By fine-tuning the sigma parameter, we can control how often the agent will change its course, in the hope of finding another agent.

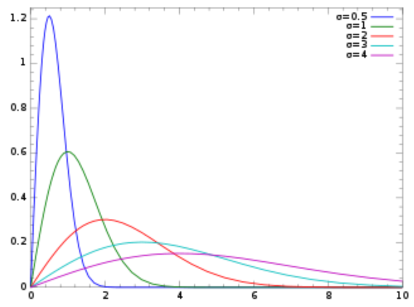

Figure 3: Rayleigh distribution according to the sigma parameter

However, when the agent has a neighbor or more, it will try to change its direction to point to the middle of them. In order not to have all the agents accumulate at one point, we set a relaxation, that only head toward an agent if he is further than a certain distance.

$$dv_a = \frac{1}{|N_D|} \sum_{n \in N_D} p_a - p_n$$

$$N_D = \{n_i : d(a, n_i) < Dmax, n_i \in Neighbors\}$$

Where a is the agent, dv_a is the speed vector target change, p_a is the position of agent a, n is a neighbor agent whose distance from a, "$d(a, n)$", is lower than threshold $Dmax$.
Doing so helps the swarm to spread and gives a bigger chance to find other agents.
On each cycle, the Environment agent will update the system as if Δt time unite has passed, and update pthe osition of all agents according to their speed. $p_a(t + \Delta t) = p_a(t) + v_a \Delta t$.

Simulation execution:

First, set up the "jade_lib" property in the ant's build.xml file. Then lunch the main container "00_RMA" that will set up the JADE server. Then lunch "ENV" which set up the Environment agent and make sure to run this first as the agents expect the environment to be up and running. Finally, lunch "one-particles" to check how one agent behave, or lunch "ten-particles", as the name suggests, it will lunch ten different agents that will start looking for each other.
The system of agents is visualized using java's swing package. Figure 4 shows screenshots of the visualization. Each agent is given a random color to help distinguish them apart when many agents are near each other. At the center of each circle, there is an agent. The direction of the agent is represented by a black line from the center that points in the direction of the agent, and its length is proportional to its speed. The circle itself is the boundary after which the agent cannot detect other agents.
Figure 4.a shows one agent in the system, and figure 4.a2 shows how the environment is a torus as the field of view for the agent extends through one edge of the rectangular-view into the corresponding edge. Figure 4.b1 shows ten agents moving toward each other, and figure 4.b2 shows them after gathering together.

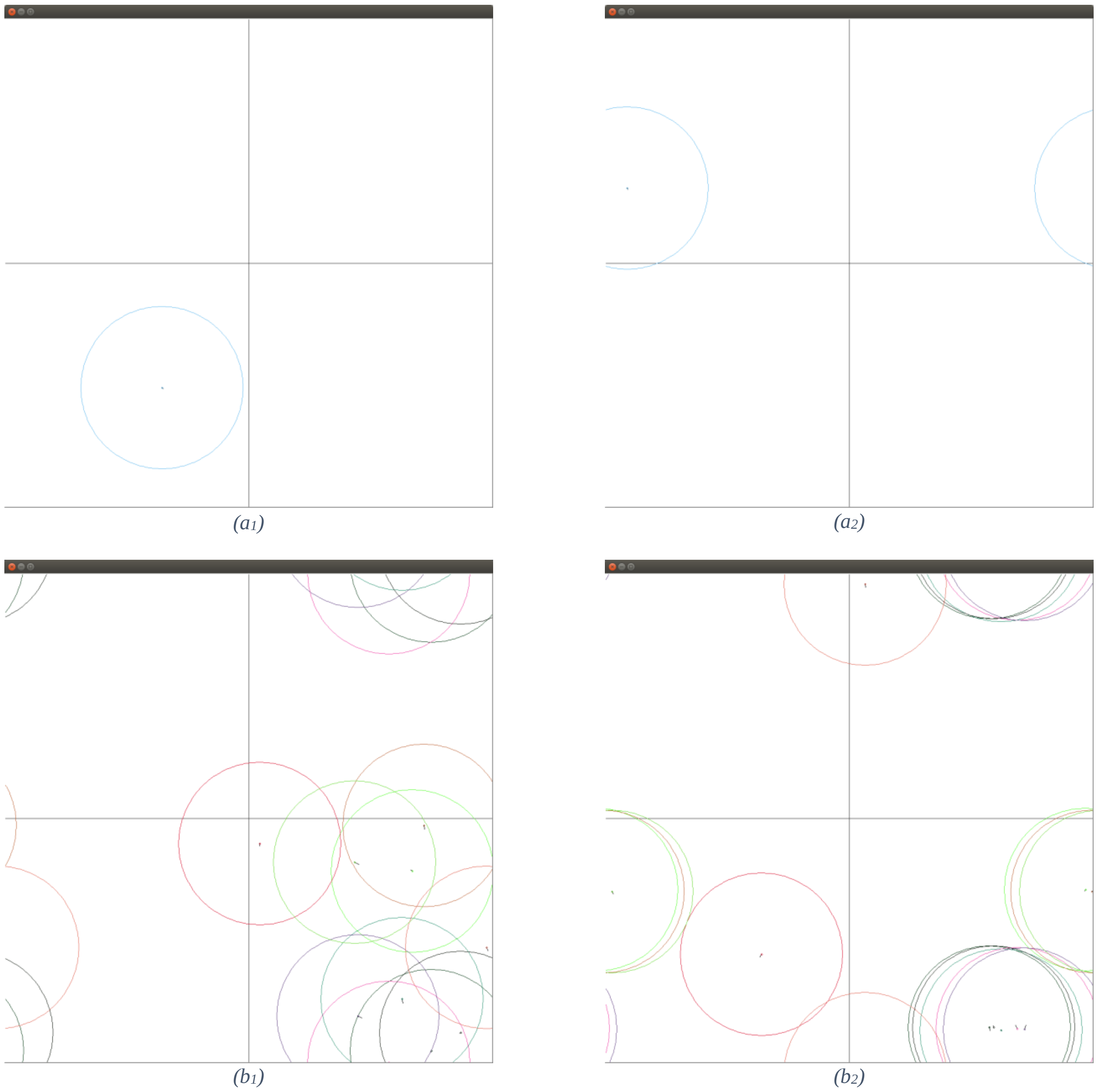

Figure 4:visualization provided by the simulation program. Showing different timeframes from two different simulations.

(a) is a one agent simulation. (b) is a ten agents' simulation

Code Overview:

Main classes are:

- ` *Particle* implements *IParticle*`: is a basic representation of position and velocity
- ` *Environment* extends *Agent*`: is the environment agent that holds the system.
- ` *ParticleAgent* extends *Agent*`: is a basic agent that can join an environment
- ` *PSOAgent* extends *ParticleAgent*`: is an agent that looks for other agents.

- ` *EnvCanvass* extends *JFrame*`: is a visualization handler for the system.

`*ParticleAgent*` set up its identity, ` *conversationId*`, and join the environment by sending a request message. Then it waits for a reply with its particle object.
` *PSOAgent*` have a cyclic behavior, which is `*LookAround*`. In this behavior, the agent will request the relative positions to other agents from the environment and then based on a reply from the environment will send its change of speed to change its direction.
` Environment` holds many behaviors cyclically:

- ` *JoinBehaviour*`: receive join requests from agents, and accept them by replying with a particle object.
- ` *TimePassBehaviour*`: update the system based on the physics model.
- ` LookAroundBehaviour `: receive queries from agents, and reply with relative position object as an answer.
- ` ChangeCourseBehaviour `: receive requests from agents to change their direction.

The environment differentiates between agents by using the conversation id of the message, which is maintained by the agent during the lifetime of the program.

Conclusion:

We developed a simple application for simulating a simple behavior of swarms, to find other members of the swarm. The simulation has a basic mathematical model behind it, and yet the result is encouraging. Using the JADE framework keeps everything modular and straightforward for future improvements or extensions, thanks to facilitating concurrency.

Acknowledgments:

I want to thank professor Nakagawa for giving me one to one support during the development of this project, and for helping us to learn the material step by step. I learned a lot from him.

References:

[1] Perumalsamy, Radhakrishnan and Jeyanthi Natarajan. “Particle Swarm Optimization Model for Dynamic Supply Chain Inventory Optimization involving Lead Time.” (2013).
[2] Janecek, Andreas, Tobias Jordan and Fernando Buarque de Lima Neto. “Swarm/evolutionary intelligence for agent-based social simulation.” 2014 IEEE Congress on Evolutionary Computation (CEC) (2014): 2925-2932.
[3] Bellifemine, Fabio, Federico Bergenti, Giovanni Caire and Agostino Poggi. “JADE - A Java Agent Development Framework.” Multi-Agent Programming (2005).