7-rich-hickey-original-ants-demo - ants-demo

Benefits

In this exercise you'll gain understanding of the following:

- the original ants implementation Rich Hickey used to demonstrate concurrency features in Clojure

Assumptions

- You have Leiningen installed
- You have an internet connection (if you don't have this – then we can copy the maven archive across)
- You have worked through the previous exercises

Code to Read

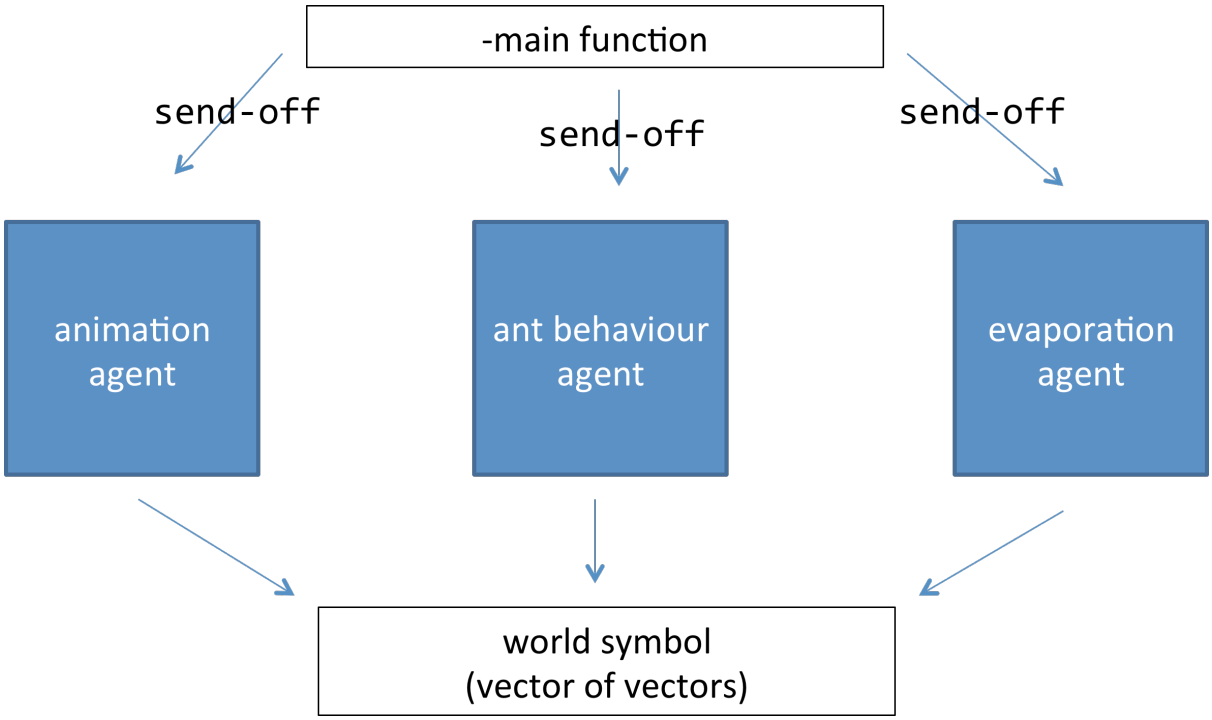
- `lambdajam-2014-core.async-workshop\7-rich-hickey-original-ants-demo\ants-demo\src\ants_demo\core.clj`

Things to Note In the Code

1. This is code originally written by Rich Hickey –best I can work out it was demonstrated in 2009 `http://www.lisptoronto.org/past-meetings/2009-05-clojure-ants-demo`
2. The data structure is created in the symbol `def world` as a `vector` of `vectors`
3. The ants are created in the function `create-ant` and assigned an `agent`
4. The evaporation occurs in the function `evaporate` and `evaporation` and is assigned an `agent`
5. The animation occurs in the function `animation` and is assigned an `agent`
6. The food is placed once in the function `setup`
7. There is no single-threaded 'game loop' – the animation function just reads the shared data structure `world` as it is updated by the other agents
8. The `main` function kicks off three sets of `agents` for animation, ant behaviour and evaporation

Code Model

This is a quick way to understand what is going on in the code:



Activities

1. Kick it off by running `lein run` in `lambdajam-2014-core.async-workshop\7-rich-hickey-original-ants-demo\ants-demo\`
2. Observe the ant movements and the pheromone trails left by the ants
3. Observe the pheromone trails evaporating
4. Observe the ants picking up food and carrying it back to the nest

Questions for Reflection

1. How would you implement 100 threads all concurrently updating the one data structure for a live animation in Java?
2. Is the concurrency essential to the animation – or could it be implemented using a single-threaded loop?animation
3. How would you implement this in ClojureScript?