8-ants-cljs-no-async - ants-cljs

Benefits

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

- converting a Clojure code-base to ClojureScript
- concurrency primitives available in Clojure that are not available in ClojureScript
- converting a JavaScript animation loop to ClojureScript
- using shared data structures in ClojureScript

Note that core.async is out of scope for this section! We're just trying to keep things simple, and do them one at a time.

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

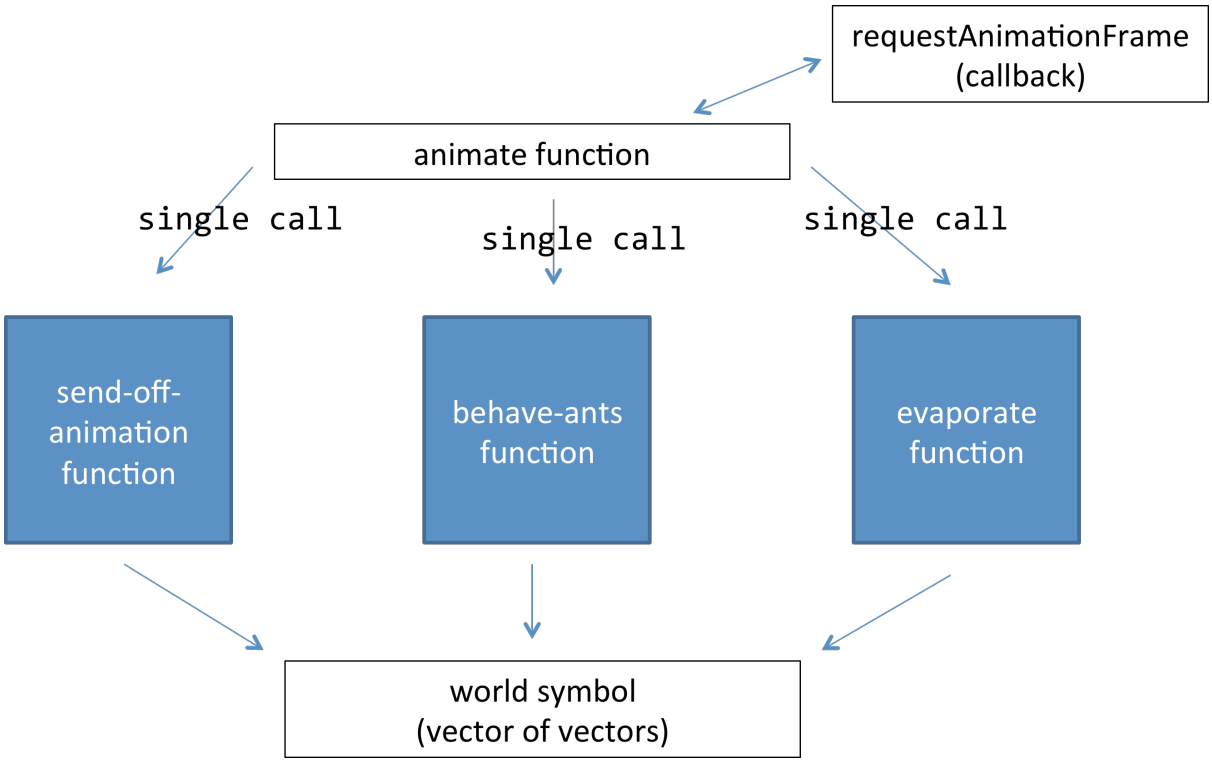
- `lambdajam-2014-core.async-workshop\8-ants-cljs-no-async\ants-cljs\src\ants_cljs\core.cljs`

Things to Note In the Code

1. this is a 'cutover' from the Clojure code to ClojureScript
2. there are a number of things that are different
3. the main loop is the `animate` function
4. `animate` runs in a single-threaded fashion
5. `animate` has a helper callback for `requestAnimationFrame` for the loop
6. `animate` calls the functions `send-off-animation`, `behave-ants` and `evaporate` to each run a single time
7. The functions to draw ants, food, boxes and pheromones have all been updated to draw on the html canvas instead of the Swing panel

Code Model

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



Activities

1. Start the compilation by running `lein cljsbuild auto ants-cljs` in the `lambdajam-2014-core.async-workshop\8-ants-cljs-no-async\ants-cljs` directory
2. View the result by opening `lambdajam-2014-core.async-workshop\8-ants-cljs-no-async\ants-cljs\index.html` in your web browser
3. Click the pause button to 'unpause' it and get it running

Workshop

1. In the namespace declaration at the top add after `[goog.events :as ev]`:
```
[cljs.core.async :as async :refer [<! >! chan put! timeout]])
  (:require-macros [cljs.core.async.macros :refer [go]]))
```

2. In the first line of the `behave-ants` function – add:
```
(go (while true   (if is-running
```

3. In the last line of the `behave-ants` function – add:
```
(<! (timeout 500)))))
```
ensure that the previous line closes off the paren opened by `(if-isrunning`

4. On the first line of the `render` function – add:
```
(go (while true   (if is-running
```

5. On the last line of the `render` function – add:
   `(<! (timeout 500)))))`
   Ensure that the previous line closes off the paren at `(if is-running`

6. On the first line of the `evaporate` function – add:
   `(go (while true`

7. On the last line of the `evaporate` function – add:
   `(<! (timeout (rand-int 1000))))))`
   Ensure that the previous line closes the paren at `(dorun`

8. In the animate function – comment out the lines
   ```
   (when is-running
     (.requestAnimationFrame js/window animate))
   ```

Questions for Reflection

1. How would you modify this so each ant is on its own go block?
2. How would you modify this so that display updates are called on a single cell after a change? (i.e. every process has its own self-contained display function)
3. How would you modify this so that the updates go on a queue of cell changes to the display function?
4. How would you do performance optimisation? (ask for demo)