AMOEBAS – a Game of Clones

Jorn W. Janneck

Table of Contents

1. First off....2
2. Step by step....3
3. Time to grow some bugs....4
4. Competition rules....6
5. Soft rules: on being a good citizen in an amoeba-eats-amoeba world....7
6. Some thoughts on strategy....8

1. First off...

The best way to learn a new programming language is to tinker with it, to try stuff out. The purpose of this programming contest is to get you to tinker with Clojure, the language we will be using in the labs to put some of the mathematical concepts in this course into practice. The idea is that there is no right or wrong or “optimal” solution (at least that I know of), just a myriad different ways of doing things, often with difficult-to-predict consequences, so tinkering, playing around with various options, parameter settings, strategies, seems like the only way to get anywhere.

I have prepared a manual describing the technical bits of the Amoebas framework, which you can download separately from the course Web site. Unless you are already pretty familiar with Clojure, it won’t mean very much to you right now, so leave it for later. This note you are reading is supposed to give a you few basic instructions to get you started.

Please keep in mind that this is the first time we are doing this, so things are bound to blow up here and there in unexpected ways. We’ll sort those things out out as we go along, remember, the entire point of this exercise is to get you started in Clojure in what I hope will be a non-boring way.

This is where you should find all the material pertinent to this programming contest:

http://cs.lth.se/edaa40/amoebas/

Take a look now and then be sure to check for updates now and then, I might expand the documentation or the library, or fix bugs in the code.

2019-03-29

2. Step by step

Before growing your own bugs in your virtual Petri dish, a few things need to be done. Let's go through them one step at a time:

Step 0: This is the most important of them all, and it consists of two things: (a) find/form a group (up to five people) and (b) come up with a battle name. That name shouldn't be too long, and consist only of basic characters without accents, because it will become the name of your top-level package. Let's call it *yourname* in the following.

Step 1: Locate the Clojure bit on the course page:

http://cs.lth.se/edaa40/clojure/

You'll find pointers to Clojure material there, software, tutorial, docs.

Step 2: Install Clojure on whatever machine(s) you plan on using. I use Leiningen, a distribution of Clojure, for the things I do for this course, and I suggest so do you. You can find it here:

https://leiningen.org/

Familiarize yourself with the basic operation of Leinigen: how to start the read-eval-print-loop (repl) (hint: type `lein repl` into a command shell while you are in the main directory of a project), that sort of thing.

Note: If installing or running Leiningen causes any trouble, *please do let me know*. It has happened before, to me as well, so it might be that I have seen the issue you ran into and might have an idea how to fix it.

Step 3: Do the (or a) Clojure tutorial. You'll find pointers to it on the Clojure course page above, try to work at least through the bits indicated there. It can't hurt to know more than that.

3. Time to grow some bugs

Step 4: Download the Amoebas framework and the manual from the Amoebas course page. The framework is a Leiningen project. If you start the `lein repl` in its top-level directory, you should be able to run a few tournaments. Begin by loading the examples:

```
(use 'amoebas.examples :reload)
```

If that worked, you can try a simple pre-canned tournament. The file `examples.clj` (which you will find in the `amoebas/src/amoebas` folder right next to the rest of the code) contains a few of those. Try, for example,

```
(T03 500 {:graphics true})
```

That should produce a window with red and green stuff in it. It will also produce a few warnings on the console – ignore those for now, I haven't figured out how to get around those yet. If you have come this far, it's now time to…

Step 5: … get serious. Take a look at the Amoebas manual – I will likely update it from time to time, so check that you are looking at the latest version. Read it, look at the code, especially the examples in `amoebas.examples` and the library of functions supposed to help you write amoebas in `amoebas.lib`.[1]

Step 6: Create your package. Under the amoebas/src directory, create a directory called *`yourname`*, and in it create a clojure file called `core.clj`. That is where you place definitions of the namespace *`yourname.`*`core`. Therefore, the first lines I that file should be something like

```
(ns yourname.core
    (:use amoebas.defs amoebas.lib amoebas.run)
)
```

Now you can stick your definitions in there: amoeba functions, helper code, tournament definitions to try stuff out (that's why you should use amoebas.run, which is the namespace containing the tournament stuff) etc. Start by ripping off code from `amoebas.examples`, and from there the sky is the limit. Only work inside the *`yourname`* namespace hierarchy. Eventually…

Step 7: … you should produce a definition that binds the variable `Evam` in the namespace *`yourname`*`.core` to an amoeba function: that's gonna be your gladiator, the Eve and Adam of your species.

Note that this should *not* be a factory/creator function, but an actual amoeba function. So, for example, if someone were to send a *mindless divider* (one of the very simple amoebas in

1 As in many programming languages, Clojure definitions are organized into *namespaces* (similar to packages in Java). We will adhere to the convention that the definitions in the namespace `a.b.c.d` are in the file `a/b/c/d.clj` under the main source directory, which in your Leiningen project would be `amoebas/src`. So the stuff in `amoebas.lib`, for example, will be found in `amoebas/src/amoebas/lib.clj`.

`amoebas.examples`) into the fight – not that that would be a very good idea, since it is, well, pretty mindless – they would write something like this into their *`yourname`*`.core` namespace:

```
(def Evam (create-mindless-divider 0.3))
```

Whatever your `Evam` is bound to is what I shall use to produce the genesis specification of the tournaments that will be created.

Step 8: In the end, when tournament time rolls around, zip up the *`yourname`* code tree, i.e. the directory `amoebas/src/`*`yourname`*, and send it to me.

4. Competition rules

Submission: Species are to be submitted to me in the form I described above by the deadline (to be announced). I will provide feedback on any issues I run into (errors and the like, see also the next section), and there will be a grace period for you to fix them. I will then run the tournaments offline, prepare score table(s) and videos of them for the last seminar. You are responsible for the popcorn and the drinks.

Scoring: Since there is an element of chance involved in this game, the competition will consist of three tournaments. In each tournament each species will get a score of the form (n_i, E_i), where n_i is the number of tournament rounds it survived in the i-th tournament (max 2000) and E_i is the final aggregate energy in the i-th tournament (so if it went extinct during that tournament, its score will be $(n_i, 0)$, with $n_i<2000$, otherwise it will be $(2000, E_i)$, with E_i being the sum of the energy of all individuals of that species. The overall score will be

$$(n, E) = (n_1 + n_2 + n_3, E_1 + E_2 + E_3)$$

Scores are ranked lexicographically, i.e.

$$(n_a, E_a) > (n_b, E_b) \text{ iff } n_a > n_b \text{ or } (n_a = n_b \text{ and } E_a > E_b)$$

In the extremely unlikely event of a tie, it will be broken by a single one-on-one tournament, which is repeated until there is no tie.

Maximal number of species per tournament: 10
If there be more groups than ten, there will be multiple competition levels, where the top half (rounded up, so five out of nine would advance) of each competition level advances to the next level. Mixing of species into subgroups will be random.

Number of rounds in a single tournament: 2000

Prize: Each member of the winning group receives 5 points for the first exam, i.e. the one on 3 June 2019. Total points in the exam will be 100.

2019-03-29

5. Soft rules: on being a good citizen in an amoeba-eats-amoeba world

In order to be able to perform the tournaments in a reasonable time (or at all), you will have to observe a few “soft” rules. I hope it’s sufficient to mention this – if one of the submissions seems too far out of bounds, I will contact the authors and ask them to resolve the issue. If that cannot be done before the hard deadline, I might have to pull a submission from the contest, so the rest can finish.

Time. Make sure that you amoeba functions do not become too complex to compute. Unfortunately, I cannot give a precise time budget, but we can estimate it roughly as follows. Suppose we have around 20,000 amoebas in the world (that’s overestimated, but let’s use this as an upper bound, even if especially the first few rounds will include a lot less than that). With 2000 rounds, a tournament might have to call an amoeba function up to 40,000,000 times. If I allow, say, 4h for a single tournament, this means 10,000,000 calls per hour, i.e. per 3600 seconds, which means a single call to an amoeba function, including the admit, should not take more than .36 ms. (I shall run this on an mid-range i7.) Alternatively, if 2000 rounds are to be competed in 4h = 14,400 seconds, a round should not take more than 7.2 seconds. So, rule of thumb, if during your own experiments with your amoebas you run into rounds that cross the 10 second mark, we might have a problem.

Memory. Basically, don’t use too much of it. This is really only a concern if you end up using the `:data` field of the actor state, of if you build amoeba functions that are created with elaborate data structures. I shall use a machine with 16GB RAM, so 20,000 simple amoebas should not be much of a problem. However, beware of memory leaks, i.e. data structures that just keep growing at every step and never stabilize or shrink.

Exceptions. Your amoeba functions should not throw them. If they do, the simulator tries to catch them and executes a `:rest` action, which is probably not what you want. If I detect this happening, I will get in touch and give you the opportunity to sort it out.

6. Some thoughts on strategy

No plan survives contact with the enemy.[2]

Plans are worthless, but planning is everything.
(Dwight D. Eisenhower)

Man plans, God laughs.
(Yiddish proverb)[3]

A successful amoeba population consists of hundreds or thousands of individuals that accrue small advantages over the course of a tournament, rather than striking one decisive blow. There seems to be a lot of experimenting and fine tuning (of strategies and parameters) involved in making a population better, so don't be afraid to tinker to see what works best. That is, after all, the entire point of this contest.

It is of course impossible to cover the entire world with amoebas, because they consume energy and sunshine only adds energy to empty cells. It is easy to see that only less than half of the cells can eventually be occupied by amoebas (with the given constants in `amoebas.defs`), since every amoeba will consume at least one unit of energy each turn, which is the amount of sunshine energy collected by an empty cell. In addition, the amoeba will now and then have to move to pick up the energy collected by an empty cell, which costs three units. As the goal is to gather as much energy as possible, an amoeba species needs to control as much area as it can, and harvest the sunshine energy that is collected by the empty cells in its "domain" with as little movement as possible.

After its initial expansion, an amoeba population at some point develops an inner part, the hinterland, and an outside, the edge or front. The edge either breaks new ground into open territory, or it is in contact with a hostile population. Consequently, there are three distinct roles for amoebas in a sufficiently large group:

- **farmers**, tending to the hinterland, not in contact with the enemy or fuel-rich open territory,
- **explorers**, breaking new ground and venturing into open territory,
- **warriors**, fighting with hostile populations.

The job of farming is to gather energy from surrounding empty cells minimizing one's own energy use, and to push some of it "outside" toward the edge (which means some farmers have to either move themselves toward the edge, or produce offspring that does), so that exploring and combat can occur. Explorers need to conquer as much open space as possible and multiply as quickly as they can to exploit that territory. Warriors have to effectively attack hostile populations and gobble up fuel that is the result of amoebas that got mortally wounded.

2 This is a common English paraphrase of von Moltke (the Elder): "Kein Operationsplan reicht mit einiger Sicherheit über das erste Zusammentreffen mit der feindlichen Hauptmacht hinaus." Helmuth Karl Bernhard von Moltke, "Über Strategie" (1871)

3 "Der mentsh trakht un got lakht"

These different roles need not necessarily mean that one needs to design multiple amoeba functions; some of this happens “organically” even with fairly simple functions. An example of this is `slightlybrainy` – if you observe its populations in action, they farm on the inside (you can tell by the fact that the inside is almost black, although the farmers move a lot, so they seem to waste a lot of the energy), and there is a constant stream of amoebas moving toward the edge to explore or battle. This is presumably driven by its directedness toward food when it moves (that is the part where it sorts the sections according to the amount of food in them): as the center becomes crowded, food gets sparse, and amoebas gravitate outward. But no attempt is made to conserve energy, or to farm in any systematic way, or even to figure out whether an amoeba is close to the edge of the population or far on the inside. So there is much to improve.

However, if you plan on making behavior conditional on the “location” of an amoeba inside its “blob”, you need a way of figuring out the “radius” of the blob (for example, this could be the longest distance of any amoeba inside of it to the edge), as well as the distance of every single amoeba to the edge. Of course, since every amoeba can only see a tiny part of the world (for example, an amoeba at the edge cannot directly know the size of the blob, and one somewhere in the hinterland cannot directly know its distance to the edge), these figures can only be estimates, and they will have to computed iteratively and cooperatively, with the amoebas of a species collaborating to come up with an answer. If you want to try to figure out how to do this, the `:data` field of the amoeba state might be useful here.[4]

It also seems like a good idea to expand quickly, possibly starting several populations that grow together as they expand (this can be done by “sending out” explorers that walk away from the front into open territory for some distance before dividing – they might need to count the steps they took, and keep traveling in roughly the same direction each step, in which case again the `:data` field of the amoeba state would be a place to stick that information). A larger population controls more territory, which means they tend to get more sunshine: them that has, gets. Since every amoeba also gets to do something at every turn, a larger population gets to do more: feed more, multiply faster, or hit more enemies. So building strategies that occupy new open territory quickly is probably not a bad idea, even if most of a 2000 round tournament will probably be spent with farming and fighting. Also, note that while the ratio between the length of the border of an amoeba blob and the size of its territory depends on its shape, generally the area grows faster than the circumference (at least for vaguely circular or convex blobs). As the area is roughly proportional to the amount of sunshine, and therefore new energy, the population receives, a solid, and well-tended, hinterland is crucial to the ability of the amoebas at the front to do their job.

Fighting is a very costly part in the life of an amoeba blob – a single hit is ten units of energy, and it requires at least three of those to dispatch an opponent. It cannot be avoided, but there seems to be a lot of room for improving efficiency by coordination and good tactics. For instance, when deciding whom to hit (there are some rudimentary target selection algorithms in the examples). Each hit costs

4 For example, if you cannot see the edge yourself (leaving aside for now how exactly “edge” would be defined – by the presence of a hostile amoeba, of course, but you also want to detect “open” territory), but you know that in your environment there is an amoeba that knows it is about *n* units away from the edge, and if among those in your environment that is the closest one, and you are *m* units away from it, then you might be around n+m units away from the edge. You could perhaps improve that estimate if you include some information about the direction.

ten units and inflicts a damage of four. In order to make this efficient, targets should be hit quickly by many amoebas so they can’t recover. Also, once killed, a target releases its stored energy, which can be picked up and used – this might be the richest source of energy at an otherwise crowded front that therefore receives little sunshine. However, actually picking it up would imply resting, which might not be the smartest thing to do right at a busy front line full of enemies. So maybe one should try to push back the enemy as hard as possible, and harvest the energy a few lines behind the front. Finally, consider that dividing is a considerable investment, of 20 energy units. Therefore, a frontline warrior might consider retreating slightly into the hinterland when it runs low on energy, instead of waiting around on the front in order to refuel, clogging up the place, preventing fueled-up warriors from getting there, and just getting hit by the enemy.