Amoebas

A Game of Clones.

Jörn W. Janneck

DRAFT

March 29, 2019
Lund University
Computer Science Department

Contents

Chapter 1

Overview

AMOEBAS is a programming game in which players write programs that control the behavior of a species of small, cellular "organisms", *amoebas*. The world amoebas live in is a `WorldSize` by `WorldSize` [1] array of cells that wraps horizontally and vertically.

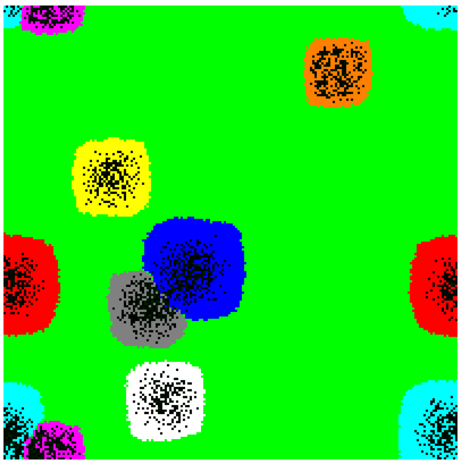

Figure 1.1: Competing amoeba populations during a phase of expansion.

Each amoeba has a certain amount of *energy* and *health*. When their health drops to zero, or their energy falls below zero, they die. One by one, all the amoebas in that world get to take one *action*, which can be one of the following things:

actions

1. *rest*, i.e. regain health and consume fuel in order to gain energy,
2. *move* to an adjacent cell,[2]
3. *hit* an amoeba occupying an adjacent cell, inflicting injury or death on it,
4. *divide*, i.e. spawn another amoeba of the same species as itself onto one of the (free) adjacent cells.

Every action, including resting, consumes some energy, the exact amount of which depends on the action.

Each amoeba must decide on the action it takes based on (1) its own health and energy levels, possibly additional status information, and (2) a view of its *environment*. An amoeba

[1] `WorldSize` is one the constants that define the basic "physics" of the AMOEBAS world. They are defined in the `amoebas.defs` namespace. When writing *amoeba functions*, you should never use literal number values for these constants, but refer to them by their symbolic name instead. So, for example, at the time of this writing, `WorldSize` is 200, but if you use that number directly in your code for the width and height of the AMOEBAS world, your amoeba functions will break if that value ever changes.

[2] Adjacent cells are those either horizontally, vertically, or diagonally directly next to the one the amoeba sits in. Therefore, each cell has exactly eight adjacent cells.

does not know its own absolute location, the environment it can "see" is a small window of cells around it, up to `ViewDistance` cells in each direction. In other words, it "knows" about a square area with side length `2*ViewDistance+1` with its own cell in the center.

amoeba function

Making this decision is the job of an *amoeba function*. The input to this function is the state of the amoeba and its environment, its output is the action it takes. Writing these functions is what this game is about.

sunshine

Once all amoebas have had their turn, the "sun" shines, and it adds `SunEnergy` units of energy to each free cell (i.e. to each cell that is not occupied by an amoeba), up to a maximum of `MaxCellEnergy` . Then the next round begins.

A *tournament* is a specified number of those rounds that start with single amoebas of different *species* (representing different players) being randomly placed in the world. The goal is to collect and bind as much energy as possible, the species with the most aggregate energy content wins.

Chapter 2

Rules

2.1 The world

The entire game happens in a `WorldSize` by `WorldSize` array of *cells*, the *world*. It wraps horizontally and vertically, which means that as one moves off the left or right edge, one re-enters on the right or left edge, respectively, and similarly moving off the upper/lower edge means re-entering on the lower/upper edge.

Each cell holds an amount of fuel between `0` and `MaxCellEnergy` . That fuel can increase (sunshine, an amoeba that gets killed) or decrease (when an amoeba in the cell consumes it by resting). It will, however, never exceed those limits.

Each cell is either free (or empty), or it contains exactly one amoeba. The energy stored in the amoeba is separate from the fuel in the cell and does not count toward the fuel limit of the cell.

2.2 A round

At the beginning of a round every amoeba in the world is located and a sequence established. Then one by one the amoeba function of every amoeba is called and its action executed. The only exception to this is that an amoeba that gets killed before its turn will not get to take an action. After every action, the fuel of each cell, and the health and energy of all amoebas affected by that action are adjusted, before the next amoeba takes an action.

After all amoebas took a turn, the sun shines: the fuel in every unoccupied cell is increased by `SunEnergy` , up to a maximum of `MaxCellEnergy` .

2.3 Amoeba state, environment

Each amoeba contains two immutable pieces of information: its *species*, identifying the player it represents, and its *function*, i.e. the amoeba function that is invoked to determine its next action.

In addition, three pieces of information define its *state*, as they may change over the lifetime of the amoeba: *health*, *energy*, *data*.

The *health* is the injury level of the amoeba. It varies between `0` and `MaxAmoebaHealth` . It is reduced by an amount of `HitLoss` when the amoeba is hit by a neighbor, and it increases by `RecoveryGain` when the amoeba rests. If it reaches zero or less, the amoeba dies. Initially, the health is the same as that of the original amoeba that divided to create it. The very first amoeba of a species starts with `MaxAmoebaHealth` .

The *energy* is the amount of energy the amoeba has stored, which determines the actions it can perform. It varies between `0` and `MaxAmoebaEnergy` . When the amoeba rests, it increases by the smaller number of `MaxFuelingEnergy` and the fuel available in the cell the amoeba is in, minus the `RestEnergy` . Performing actions decreases it. Except resting, no action will ever decrease the energy below zero—if the amoeba functions wants to perform such an action, the amoeba rests instead. If the energy falls below zero, the amoeba dies dues

starvation

to *starvation*.

The *data* is a user-defined piece of information, which can be any Clojure object. Amoeba functions receive the latest value of their data part of their state, and they can return new values together with the action. In this way, an amoeba can keep a record of past events. Also, since the data field of amoebas in the visible environment can be seen, it can be used as a form of communication between amoebas.[1]

2.4 Actions

When the amoeba function is invoked, it returns an *action*, which is represented by a Clojure map containing a number of fields, in the following described under *Format*. In addition to the fields listed there, all commands can also include an optional field `{..., :data` *value*`}` that updates the data part of the amoeba state. If it is missing, the *value* defaults to `nil`.

After the amoeba function has decided on an action, the system first tests whether if can actually perform it. If it cannot, it will execute a *default action*, which is to *rest*. The conditions that are tested before an action is executed are called its *preconditions*.

preconditions, default action

2.4.1 Rest

Format: `{:cmd :rest}`

Preconditions: None.

Effects: Health will increase by `RecoveryGain`, up to `MaxAmoebaHealth`. Energy will increase by the smallest of (a) the fuel available in the cell, (b) the difference between `MaxAmoebaEnergy` and the current energy of the amoeba, and (c) `MaxFuelingEnergy`. From this, `RestEnergy` will be subtracted. If the resulting energy is less than zero, the amoeba will die of starvation, its cell becomes empty.

starvation

2.4.2 Move

Format: `{:cmd :move, :dir` *n*`}`
The parameter *n* must be a number between 0 and 7, identifying the neighboring cell to move to. 0 is the top-left neighbor, then counting clockwise.

Preconditions: The amoeba must at least have `MoveEnergy` units of energy stored, and the neighboring cell in direction *n* must be empty.

Effects: The energy of the amoeba is reduced by `MoveEnergy`, and it moves from its current position to the specified neighboring cell.

2.4.3 Hit

Format: `{:cmd :hit, :dir` *n*`}`
The parameter *n* must be a number between 0 and 7, identifying the neighborng cell to move to. 0 is the top-left neighbor, then counting clockwise.

Preconditions: The amoeba must have at least `HitEnergy` units of energy stored, and the neighboring cell in direction *n* must contain an amoeba. We call it the *target*.

Effects: The energy of this amoeba is reduced by `HitEnergy`. The health of the target if reduced by `HitLoss`. If its health becomes negative as a result, it is killed. Its stored energy is added to the cell it was in (which thus becomes empty), up to `MaxCellEnergy`.

[1]Note that an amoeba can see the data fields of all amoebas in its environment, regardless of species!

2.4.4 Divide

Format: `{:cmd :divide, :dir` *n*`}` or `{:cmd :divide, :dir` *n* `:function` *f*`}` The parameter *n* must a number from 0 to 7, identifying the neighboring cell the newly generated amoeba will be placed into. The parameter *f*, if present, must be a valid amoeba function. If absent, it defaults to the amoeba function of this amoeba. In addition, either format can contain an optional field `..., :child-data` *value*`...` . If present, *value* will be the first data item of the amoeba state of the newly created amoeba. It defaults to `nil` .

Preconditions: The amoeba must have at least `MinDivideEnergy` units of energy stored, and the neighboring cell in direction *n* must be empty.

Effects: A new amoeba is created in the specified neighboring cell. It has the same *species* as this one, and its amoeba function is the one specified, defaulting to the amoeba function of this amoeba. The energy of this and the new amoeba is the same, and it is computed as the original energy of this amoeba, minus `DivideEnergyLoss` , divided by two, and then rounded down.

2.5 Sunshine

At the end of each round, after all amoebas have performed their action, `SunEnergy` units of energy are added to each *empty* cell, up to a total of `MaxCellEnergy` .

Chapter 3

Amoeba functions

An *amoeba function* is a Clojure function that decides which action an amoeba takes in any given situation. It is also what a *player*, the human creating a species, produces and submits to participate in a tournament. It is *not*, however, what defines a species—different amoebas of the same species may be run by different amoeba functions, and in principle the same function could run amoebas of the same species.

The result of calling an amoeba function is a data structure describing an *action*, as described in section 2.4.

3.1 Parameters

An amoeba function is called with the following argument list:

```
[energy health species env data]
```

The parameters `energy` , `health` , and `data` are the corresponding fields in the amoeba state, the first two integers in the appropriate ranges (`0` to `MaxAmoebaEnergy` and `0` to `MaxAmoebaHealth` , respectively), the latter a user-defined value. The parameters `species` is the identifier for the species of this amoeba (could be any data object, is usually a keyword or a string). Finally, `env` is the function used to query and access the cells of the visible environment of the amoeba.

3.2 Environment

The environment function maps (relative) positions (specified either as two separate numbers x, y or as a two element vector containing x and y) to a maps containing information about the cell. (If either x or y are out of range, i.e. their absolute value is greater than `ViewDistance` , the result is `nil` .)

That map contains two entries:

- `:fuel` —the amount of fuel in the cell,
- `:occupant` —the amoeba in the cell, if any, otherwise `nil` .

If there is an amoeba in that cell, the structure bound to `:occupant` is another map, with the following fields:

- `:species` —the species identifier of that amoeba,
- `:energy` —the amount of energy stored in it,
- `:health` —its health,
- `:data` —the data component of its state.

Chapter 4

Tournaments

Tournaments are simulations of the amoeba world where individual amoebas of different species are randomly placed in the world and then the system is run for a number of rounds, or generations.

4.1 Creating tournaments

A *tournament* is created by the function `tournament` in the `amoebas.run` package. It takes a *genesis specification* as its first parameter and an *RGB map* as an optional second. The result is a *tournament function*, that can then be used to run the tournament.

The genesis specification maps species identifiers to amoeba functions. A species identifier is a distinct value that is used to distinguish a particular species from others. We will use Clojure keywords for this purpose, but in principle any value could be used. The genesis specification then maps those to an associated amoeba function, which controls the behavior of the original amoeba representing a species. For each species identifier in the genesis spec, such an amoeba is created and placed in a random location. The RGB map furthermore associates the species identifier with a vector containing an RGB triple, which is used to represent amoebas of that species when displaying the world status or exporting images.

The result of applying `tournament` is a *tournament function*. The tournament is started by invoking it, with two parameters: (a) the number of rounds to execute the tournament, and (b) a map of flags controlling the output produced during and after the tournament. These are the currently defined flags and their possible values:

- `:generational-stats` , values: `true` or `false` , default is `false` . If set, stats will be printed on the console after each round.
- `:report-extinctions` , values: `true` or `false` , default is `false` . If set, a notification will be printed after the round when a species went extinct, i.e. when the last individual of that species died.
- `:last-gen-stats` , values: `true` or `false` , default is `false` . If set, stats about the last generation will be printed on the console after the tournament is finished.
- `:graphics` , values: `true` or `false` , default is `false` . If set, a window will be opened and a graphical rendering of the world state will be produced after each round.
- `:save-graphics` , values: `true` or `false` , default is `false` . If set, a graphical rendering of the world state will be saved as a PNG file in the specified directory.
- `:csv-stats` , values: `true` or `false` , default is `false` . If set, two CSV files will be written in the specified directory after the tournament is finished, containing stats about size of each species population and its the aggregate energy after each round.
- `:directory` , values: a string representing a path, default is ".". The directory for storing information during and after the tournament. It will be created if it does not exist already.

If the `:save-graphics` option is used, the images are saved with the name `g`*ddddd*`.png`, where *ddddd* is a five-digit, leading-zeros representation of the number of the round, starting with 0. It can be used to create movie clips, e.g. using `ffmpeg`:

```
ffmpeg -i g%05d.png video.mp4
```

Chapter 5

Miscellaneous

5.1 Software overview

`amoebas.defs` Contains the basic physics constants of the AMOEBAS world.

`amoebas.lib` Definitions and functions supposed to help write amoeba functions.

`amoebas.examples` A few very, very simple amoeba functions, and some tournaments using them.

`amoebas.run` Stuff related to tournaments, including the `tournament` function to actually create tournament functions.

5.1.1 Internal namespaces

`amoebas.simulation` The simulation engine for the AMOEBAS world.

`amoebas.display` Code that renders the state of the AMOEBAS world into a more or less pretty picture.

`amoebas.util` Code that does not seem to fit anywhere else.

Chapter 6

Glossary

action The thing an amoeba does in a step. The possible actions and how they are represented as Clojure data structures are described in section 2.4.

amoeba Little, virtual creatures that swarm, feed, multiply, and hit each other over their non-existent heads in our tiny, virtual Petri dish of a world.

amoeba function The Clojure function controlling the behavior of an amoeba. It is described in chapter 3.

amoeba state

cell A distinct place in the amoeba world, just big enough to hold a certain amount of fuel and up to one amoeba.

direction One of the eight neighboring cells an amoeba can reach from its own, to either go there, hit another amoeba, or spawn a clone into. Directions are represented by a number from 0 to 7, counting clockwise starting from the top left corner.

distance The distance between two cells is the smallest number of steps needed to go from one to the other. Since we can move diagonally, this boils down to the max norm, or the L^∞ norm, in the coordinate system of our amoeba world.

divide One of the actions an amoeba can take. The result of dividing is another amoeba of the same species.

energy Amoebas require and consume energy to perform actions. Energy can be stored inside an amoeba, and it can be present in cells, which amoebas can transfer into their own store. It gets replenished at a certain rate through sunshine. (Sections 2.4 and 2.5)

environment The part of the world around an amoeba that is "visible" to it, in the sense that its amoeba function receives an environment function as a parameter that it can use to query its surroundings.

friendly A friendly is an amoeba of the same species as ours.

fuel Fuel is energy stored in a cell. An amoeba occupying that cell can transfer that fuel into its own storage.

genesis specification A map from species identifiers to amoeba functions. Used to create tournaments.

health An amoeba's health is a measure of its level of injury. When it is hit, it sustains injury and its health drops. If it becomes negative, the amoeba dies. An amoeba's health increases when it rests.

hit Amoebas fight by hitting each other. Hitting another amoeba causes injury and may kill it.

hostile A hostile is an amoeba of a different species than ours.

location See *position*.

move An amoeba action where the amoebe leaves its current cell and moves into one of its neighbors, which has to be empty.

neighbor One of the eight cells surrounding a cell, either vertically, horizontally, of diagonally adjacent to it. Neighbors are numbered clockwise 0 to 7, starting from the top-left neighbor.

occupant The amoebe residing in a cell, it there is one.

position The coordinate pair identifying a cell. In the context of amoeba functions, these are always relative (usually to the amoeba itself), since an amoeba does not know its absolute position. Within the simulator, both absolute and relative positions are used.

region A region is a collection of positions. It does not have to be contiguous.

rest An amoeba action where the amoebe remains in its cell, heals if injured, and transfers some of the cell's fuel into its own storage (if there is fuel in the cell).

RGB map A map that maps species identifiers to vectors containing RGB triples.

section A region corresponding to one of the eight neighbors identifying a vaguely triangular area in the direction of that neighbor.

species The amoebas bearing a particular species identifier.

species identifier An object that uniquely identifies a species, and thus also a player. It could in principle be any object, we will use Clojure keywords for this purpose.

sunshine At the end of each tournament round, after all amoebas decided on an action, sunshine adds energy to each empty cell.

tournament A simulation of the AMOEBAS world for a number of rounds.

world The world in which amoebas "live". In the narrow sense, this is the cells and the fuel in them, in a wider sense, the term "AMOEBAS world" also includes the amoebas themselves.