Part V

Python Project: evolutionary game of life

48 Generalities

The aim is to create a "Game"[al] centred around a visual and interactive simulation of natural selection.

The serious purpose behind the game is to provide a refresher on evolutionary mechanics, both as a matter of general culture and as a prelude to the study of AI and metaheuristics in general.

In particular, genetic – and, more broadly, evolutionary – algorithms are a vast class of approaches simulating aspects of Darwinian biological evolution for the purpose of solving complex problems for which brute force or analytic approaches are unsuitable.

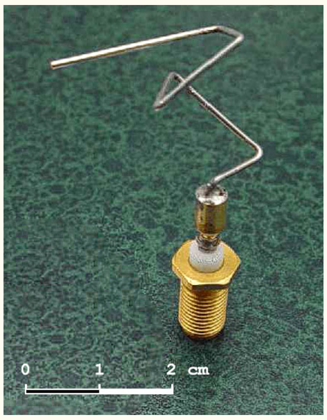

Figure 3: *NASA's ST5 satellite antenna, designed via evolutionary algorithms to meet the stringent and very specific requirements of the mission. It is the first such object to have been sent to space (2006).*

The results thus obtained can be very good, but are often a bit "alien-looking", as evolutionary processes, whether "real" or simulated, often defy human intuition, aesthetics, and engineering principles. (A look at deep-sea creatures should convey that quite well.)

For instance, the NASA's ST5 antenna in Fig. 3 was obtained by evolutionary processes. It performs quite well compared to human-designed antennae. The figure actually shows the second version of the antenna: the first was more tree-like. A minute change in mission

[al]I couldn't come up with a fun name for it, preferably with a pun or a gratuitous insersion of "INSA" in it, so I'm just calling it "the Game", for now.

parameters resulted in two completely different antenna layouts, whereas humans would come up with more incremental changes.

Of course, no human intervention was really needed to adjust the design the second time: the simulation was already set up, so entering the new constraints and pressing a button was all that was needed (plus a few days of computing time on a supercomputer).

A challenge with evolutionary processes (from both the points of view of computer science metaheuristics and biology) is how difficult it is to accurately predict the effects of a change in parameters, and how easy it is to come up with plausible-sounding "just-so" stories that don't pan out in the end.

The Game should provide a fun experimental platform to play with populations by altering several aspects of the creatures and their environment, and seeing how they react and evolve. It should generate high-quality graphs showing the evolution of those characteristics over time.

To be clear, while the process of *programming* the Game is in itself a way to teach evolutionary processes to INSA students, the final product should be a good tool to help present those notions to, say, high school students and such.

Compared to the 2017 and 2018 projects, which are archived on Celene, this one is rather less open-ended and ambitious, as you are not required to actually design, set up, and deploy metaheuristics on a given open problem, but merely to simulate some very specific traits in a very specific simulation.

However, I shall be a bit more demanding when it comes to feature-completeness, stability, and usability of the final product. Furthermore, there is room for the more ambitious groups to considerably expand the scope of the project once they have covered the basics.

I give in this document some design choices that must be respected, so as to make it easier to compare the projects of different groups.

49 Specifications

The Game simulates a *world*. Since our divine powers are quite limited, our worlds shall be quite modest: a grid of size $N \times N$, with $N = 100$ by default (but this should be an easily modifiable parameter; every numerical value I give should be so; the values will be tweaked as your groups progress and experiment with them, so that they yield relatively stable populations.).

The Game should represent this world graphically, at the very least with a view from the top.

I strongly recommend setting up a terminal-only graphical engine *first* — take full advantage of the fact that the terminal can display colours and special characters. This should be fast and easy to make.

For the final product, additional support for a 2.5D isometric view – in the style of "old" games like Diablo I & II, Age of Empires I & II, etc – is also strongly recommended.

It should be reasonably cool-looking, without taking you too much time to make.

It is probably not a good idea to try and support full 3D rendering, unless some members of your group already have strong experience in that domain. I'm not even sure what a good library for that would be. Perhaps Panda3D?

The Game should of course have a GUI allowing the user to play with all the parameters of the simulation. TkInter (simpler to begin with) and PyQt5 (more powerful, more complex, external requirements[am]) are two possibilities for building it.

Note that the execution of **the simulation must be independent from the rendering**, as rendering takes a lot of processing power. Thus not only should the two run in separate threads, but it should be possible to simply activate and deactivate rendering at will while a simulation is running.

In the world, there live creatures; they are all named Bob. There are initially $P = 100$ of them.

Bobs should be graphically represented by sprites that present as clearly as possible their individual attributes (speed, size, memory, etc). The representation should be as user-tweakable as possible to allow emphasis on whatever characteristics they are interested in at the time. For instance, I should be able to set things up so that faster creatures are bluer, and bigger creatures are redder, resulting in various hues of blue, red, and purple. Then I should be able to change it completely. Bobs' size should at least support a representation acting on the size of the sprites.

Each Bob spawns in a random cell in the world grid at the beginning of the simulation.

The simulation proceeds by time increments, or *ticks*: at each tick all Bobs perform an action, for instance walking to the next cell.

How complex their tasks and actions are depends on the number of characteristics which will be simulated.

49.1 Basic level: food hunting

Let's say that a *day* amounts to $D = 100$ ticks. Each day, a quantity $F = 200$ of *food points* spawns randomly in the World, each containing $E_F = 100$ *energy*. There is nothing preventing

[am] `https://pypi.org/project/PyQt5/`; cf. `http://doc.qt.io/qt-5/examples-graphicsview.html` pour de la documentation C++. C'est à adapter à la version Python, car PyQt5 est juste une bibliothèque de liens (bindings) vers Qt5.

several instances of food from spawning in the same cell, in which case the energy values add up.

Any food left uneaten disappears at the end of the day, just before the new food is spawned in.

Each Bob has an *energy* level, starting at $E_{\text{spawn}} = 100$, when they spawn, and capping out at $E_{\text{max}} = 200$.

This energy goes down by 1 each time they move, and they move at random each tick, going to an adjacent cell. Diagonal moves are not allowed.

When a Bob find food (is in the same cell as food), it instantly eats as much of it as it can, gaining all its energy. If its energy caps out in the process, there are leftovers. It can then stay stationary so long as there is food, but each tick spent stationary still consumes 0.5 energy.

If two Bobs access the same food at the same tick, one of them gets all, arbitrarily.

If a Bob's energy level ever falls at or below zero, it dies.

If a Bob's energy level caps out, it reproduces via parthenogenesis, spawning a new Bob in the same cell (several Bobs can occupy the same cell). The new Bob spawns at $E_{\text{birth}} = 50$ energy, while the "mother" Bob loses $E_{\text{mother}} = 150$ energy (putting her at $E_{\text{max}} - E_{\text{mother}} = 50$ energy).

Experiment with this and tweak the values to see what effects the different parameters have, until the values yield a stable population with interesting rates of births and deaths.

49.2 Velocity

At the basic level, all Bobs were identical. Let's change this, and start introducing individual characteristics that can be acted on by natural selection.

Let's start by adding *velocity* to the simulation.

The default Bobs all had a velocity of 1, moving 1 cell in 1 tick, at a cost of 1 energy.

Now, each Bob B has its own velocity, initially $B_v = 1$, but each Bob that is born may mutate a bit in that respect: say that C is born of B. Then C_v should fall uniformly in the range $[B_v - \text{Mut}_v, B_v + \text{Mut}_v]$, where $\text{Mut}_v = 0.1$ is the mutation rate for velocity.

Note that since velocity is not an integer quantity, handling it properly is not trivial. At each tick, the fastest moves first, and eats all it can. If a Bob has a non integer movement velocity, say, 1.5, then on a tick, it will move through an entire cell (eating all), and half a cell, eating 50% of the food energy there. On the next tick, it will eat the remainder (if nobody beat him to the punch), and move through another whole cell.

Speed has a cost, however. We are going to follow physics roughly, here. The kinetic energy of an object is given by

$$\frac{1}{2}mv^2 \ ,$$

so, following that, on each tick, B will consume $B_c = B_v^2$ energy to move. For instance, double the speed means quadruple the energy cost. (It's a bit more complicated in real life, because that only accounts for the cost of acceleration, but this formula is sufficient to introduce clear, intuitive trade-offs in our simulation.)

Run the simulation and see how speed evolves over time.

The impact of each of characteristics such a this should be configurable in real time – for instance if I want to remove speed from the equation I should be able to set $\mathrm{Mut}_v = 0$ whenever I want.

49.3 Mass

Now let's add the possibility for Bobs to evolve some serious muscle mass. Mass will be, like velocity, a genetic characteristic. Our default Bobs had mass $B_m = 1$. Like velocity, mass is genetic, and mutates in the same way. We do not model a cost on birth, however, mass has a cost on mobility. Following the kinetic energy model, mass multiplies the movement cost on each tick: a Bob will therefore consume

$$B_c \ = \ B_m B_v^2$$

energy on each tick for movement.

Mass should be reflected in the size of the sprites in the graphical representations. Note that mass is proportional to volume, and volume is proportional to the cube root of mass, so the size of the sprites should scale like $\sqrt[3]{B_m}$.

What is the benefit of being bigger? You can eat (much) smaller creatures. If Big Bob B and small bob b meet (are in the same cell at the same tick, even if one of them is only "partially" there due to non-integer speeds), and the difference in size is important ($\frac{b_m}{B_m} < \frac{2}{3}$) then, Big Bob can, and will, eat small bob. Small bob dies. Big Bob's energy level B_e is updated according to the equation

$$B_e \ := \ B_e - \frac{1}{2}\frac{b_m}{B_m}b_e + \frac{1}{2}b_e \ = \ B_e + \frac{1}{2}b_e\left(1 - \frac{b_m}{B_m}\right)$$

The idea is that the fight takes some energy out of Big Bob, although less and less the bigger is he in proportion to small Bob, and he gains up to half of small bob's energy level. In this model, he always gains *some* energy from eating another Bob.

It should be easy to change this model within the code.

If $\frac{b_m}{B_m} \geqslant \frac{2}{3}$, they ignore each other.

49.4 Perception

Now that there are predators and prey in the world, it help to be able to avoid the ones and pursue the others.

Each Bob initially had a perception score $B_p = 0$ meaning that they are blind as bats – but without any cool echolocation either. B_p measures of course the distance at which they can determine whether a cell contains noting, food, or Bobs. It is the radius of the circle of detection.

This is a genetic characteristic. Unlike speed and mass, we are going to keep that value integral, and mutate it by 0 or ± 1, equiprobably.

Since we are in a discrete world, some care will have to taken to compute those circles properly. Given that diagonal moves are disallowed, the notion of distance that matters is not actually that of the standard Euclidean geometry (ℓ^2 norm), but the Manhattan distance (ℓ^1 norm); in dimension 2, which is what we shall work in, it is computed as

$$d_1(A, B) \quad = \quad |x_B - x_A| + |y_B - y_A|$$

Compare to the Euclidean distance:

$$d_2(A, B) \quad = \quad \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}.$$

Circles in Manhattan geometry actually look like squares do in Euclidean geometry – with a $\frac{\pi}{4}$ rotation, so, pointy-end up.

At $B_p = 1$, all adjacent cells are detected.

Up till now, The Bobs' decisions have been purely random. Now, whenever they detect food, they make a beeline for it. Unlike in Euclidean geometry, there are many different shortest paths to take. Take the algorithm: so long as you're not there, reduce either your x or x-distance, with same probability.

Of course, if at any time Bob detects a new food source, while on its way to another, it should reevaluate its plan and make a beeline towards the closest one. If it detects several food sources, it should favour the bigger ones.

By food sources, we mean both spawned food and unlucky smaller Bobs. Since how good a food source a prey Bob is depends on its energy level, and that a prey Bob may run away, a predator Bob will always favour stationary spawned food to other Bobs.

When several prey Bobs are detected, the smallest ones will be favoured.

When a Bob detects a larger Bob, it moves to maximise distance between the two. When it detects several larger Bobs, it moves to maximise the distance to the closest one, and, *ceteris paribus*, to the others.

When a Bob detects prey and predators simultaneously, its prey behaviour overrides its predator behaviour: survive first, hunt second.

Perception cost is not affected my either mass or velocity. It does require eyes and a big brain, though, which requires constant energy expenditures. (A human brain consumes about 20% of the body's total energy)

Let's say that for each point of perception radius, there is a flat $\frac{1}{5}$ penalty (again, the GUI must allow this to be modified at will) to energy each tick. The consumption becomes:

$$B_c = B_m B_v^2 + \frac{1}{5} B_p$$

49.5 Spacial memory

A big brain has other uses. An important one is to remember the existence of stuff that's not visible right now.

Currently, unless they are making a beeline for some kind food, or running away, Bobs have a $\frac{1}{4}$ probability of going back where they just came from, which is not optimal to find spawned sources.

Now, let's introduce the inheritable characteristic *memory space*. Each Bob B has currently $B_{mem} = 0$ memory points. Again, this will remain an integral value, and mutate by $0, \pm 1$, equiprobably.

A Bob has several (mutually exclusive) ways to use its memory points:

A Bob can remember the $2 \times B_{mem}$ cells it visited, and hereafter avoid them, unless doing so would lengthen a path to food or escape.

With higher priority than that, a Bob can use its memory points to remember a place where it saw spawned food not currently in its view. This can happen if a Bob detects two food sources at opposite ends of its perception, and goes towards the one, therefore losing sight of the other one. Or if it sees food while running away from a predator.

It uses one memory point to remember one food location (therefore forgetting two of its oldest visited locations). It remembers the respective energy levels of the food.

It only uses this point upon leaving sight of the food. The point is freed the instant a remembered food comes into sight again.

While not pursuing food currently in its view (that has the priority, even if it remembers a larger food source farther away), or running away, a Bob will make a beeline for the largest food source it remembers seeing.

The cost is a again flat penalty per point.

$$B_c = B_m B_v^2 + \frac{1}{5} B_p + \frac{1}{5} B_{mem}$$

49.6 Other characteristics

Each group should invent, define, and implement a few other characteristics. They should be explained in the report.

What I outline explicitly in this document should be construed as the bare minimum as expect from each group. Once that basis is assured, be creative.

I would be delighted to see additions tackling the evolution of altruism, or aggression strategies (hawks vs doves), etc.

It would be advisable to plan the more ambitious "freestyle" features somewhat in advance, and to consult me before investing significant time in them.

49.7 Sexual reproduction

So far, Bobs have reproduced solely by parthenogenesis. Now, let's add sex to the mix – though we will consider that Bobs are hermaphrodites, like snails, and not attempt to distinguish Bobs and Bobettes.

On top of still having the option of parthenogenesis, when two Bobs B and C meet, and don't eat each other, and have high energy levels $B_e, C_e \geqslant 150$, they mate, losing 100 energy each, and creating a new Bob D at initial energy $D_e = 100$.

Note that, compared to parthenogenesis, more total energy is invested into the new Bob, and it starts out with a higher energy level, but less is required from *each* parent. This is meant to model the advantages of shared parenthood.

Of course all those values should be easily modified parameters.

D has all his genetic characteristics set to the average of its parents, and then mutation is applied as usual.

This can be toggled on and off; parthenogenesis as well.

It will be interesting to see the effects of this addition on the speed of adaptation to changing conditions.

50 Groups: size and composition

This project is done in groups of 5 or 6.

This years, groups will be determined "randomly", not chosen by students. The aim is both to save time and avoid reproducing the usual cliques.