Evolution Number 9

or another

Fast'n Crazy Breeder (with a lot of digital fallout)

Everybody is talking of artificial intelligence, artificial life etc. And all that talk is inspired by a big rhetoric boost, nourished by a profound misunderstanding of the actual simplicity of these undertakings. It's just mumbo-jumbo philosophy, tricky and sophisticated.

Genetic algorithms are a part of that – and this is what Evolution Number Nine is all about. It provides the Gamestudio user with a dll – and a very easy interface: Just one function he has to manipulate. [In the current state all that is very rude, just fit enough to demonstrate that it works. In the next releases this will be refined]

I will try – very short - to describe what genetic algorithms can do and why it could be interesting to use them in game development.

One might read that genetic algorithms are a kind of evolution system – and if you want to be a natural philosopher, you might put it that way. There are mechanism like selection, crossover breeding, elite heredity etc. – but all these mechanisms are metaphors, because it's digital evolution and the things that evolve are the one you have foreseen. I would prefer looking at them as a sorting mechanism which allows to go through hundreds of possibilities.

Compare it with chess. Imagine you had a mechanism which allows you to evaluate every position in the game. This evaluation would turn out into a number (some fitness value). Now you have a lot of possible moves – and the genetic algorithm would do nothing else than tracking the various combinations. Most of them will be crap – and if your evaluation system work properly the machine will analyse this is crap.

The bad moves are now sorted out – and only the promising ones are pursued. In a way that's what a genetic algorithm does.

a) it creates a population (possible moves) which consists of

b) various attributes (could be the colours of the model – which is the actual demo)

c) At the beginning the population are filled with random values

d) Now the evaluation begins

e) After evaluation the »good« specimen may reproduce, the bad ones will disappear

f) the process begins once again (new game, new generation)

The play where you will come into play is the step number d). You will have to say what you want – and make some fitness rules.

Let's take the example I provided in the "fitness.wdl" file.

The values which we want to evaluate are stored in the "smallarray" array.
This array consists of three elements: red green and blue

Let's say we want to favorite combinations which are very dark. So we could do the following:

```
function CalculateFitness(voidvar)
{
var temp;
temp = smallyarray[0] + smallarray[1] + smallarray[2];

if (temp > 200)
{
chromosom = 0;  // this is the fitness value for the string – it's zero
}

else {
     // and here we could differentiate
     }
```

If you evaluate something as 0 you can be sure that this version will be omitted in the evolution and selection process. Only the fittest will survive - and you will have to describe what you think is fit. This description is the basic thing – and it is the one and only function you will have to bother about. The whole mechanism itself (with selection, reproduction, mutation and crossover mechanisms) is done in the background.

The values should be positioned in a reasonable range. Myself I concentrated on the range between 0 and 1 (so 555 on the screen is actually 0.55), but there is no objective limit for that.

I have to admit beforehand that my example is not telling that much. I could fancy a whole bunch of more intriguing possibilities.

Instead of using the colour information it would be much more appropriate to evaluate the fitness of an object towards it's environment.

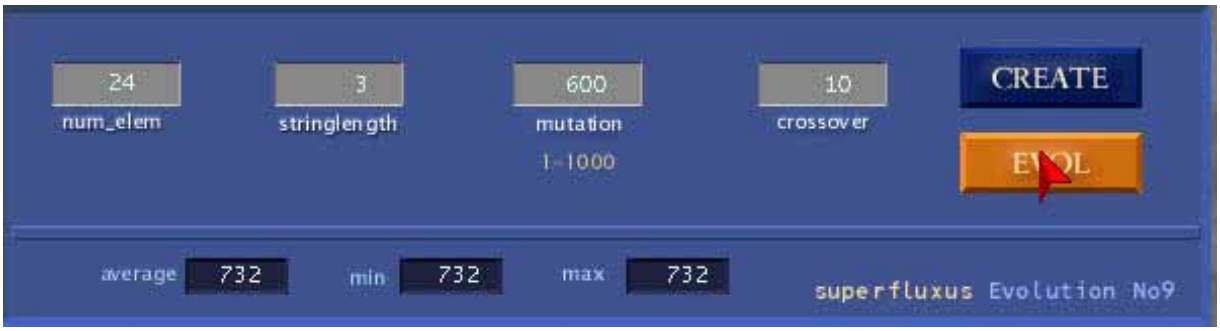
So we cold do some motion studies and investigate the fitness of a moving entity to circumvent others. We needed for that a scan command, an overall velocity, a variable containing the frame rate – and we would ask: how oft can we refresh the scan command without bringing the frame rate down and produce optimal recognition result at the same time.

I hope the example shows that the *Evolution Number 9* dll will work as a general problem solver. (It may optimise a parameter room – where simple trial and error would cost too much time). But in general: It's up to you to define where you want to apply it.

So one of my reasons to publish the program in this early state is to find out what people might use it for.

The interface

As you will see with a fast glance, the interface of *Evolution No 9* is extremely simple.



Let's have a look at the parameters.

num_elem stands for the number of elements the population consists of. It should be a twofold of 2 (in the evolution it will be split into a male and female group - and therefor the number should be equal)

stringlength stands for the attributes eachg element has. In our exampe we have chosen a stringlength of 3 because it codes color information (RGB). The stringlength can be much higher, this depends on the problem you want to solve.

mutation stands for the mutation possibility. A value of 1000 menas that the chance for a mutation equals 1 - so each generation one element is subject of a unforeseen mutation. - This might be a desirable effect because the population usually tends to homgenity - which is a loss of information. Mutation brings new elements into the game - therefor a gain of information.

crossover stands for the number of elements that will couple in each generation. This produces some interesting mixer effects.

The values in the lower row average, min and max are there just for feedback reasons. The first gives the overall fitness of all the elements, min gives back the minimum, max the maximum fitness.

The future of this program (version 0.91 and above)

I will do a refinement of the interface. You may choose between various evolution and reproduction systems (This is implemented in the dll, but not yet visible as an option).
These options will produce a bigger or a smaller deviation in the population – so one may put is surprise factor.

If you happen to know "Tiny Things" (the particle generator I wrote) the concept in general is quite similar. There is a production environment and there is a playback environment. So one could use the dll also as a strange random generator, which do the genetic algorithm calculations on the fly. The other option (the studio) will end up in a printout of desirable values. Maybe it

will be more, and frankly I have to say: I don't know what kind of ideas may enter my mind.

So have fun.

Martin Burckhardt

Ps. if you like to contact me directly, getting into a more profound discussion, please mail to

mb@superfluxus.de