Using Neural Networks and Genetic Algorithms as Building Blocks for Artificial Life Simulations

Gerd Beuster[1]

Abstract

Artificial neural networks and genetic algorithms are used very often in artificial life simulations. In this paper we describe Artificial Life Environment (ALE). ALE uses neural networks and genetic algorithms, among other parts, as building blocks to set up artificial life simulations.

Creating Simulations from Building Blocks

Writing software for artificial life simulations is a complex, time-consuming and error-prone task. With ALE (Artificial Life Environment) we are developing a tool to make the creation of simulations easier. The basic idea of ALE is to set up a simulation from building blocks. Since many artificial life simulations share common characteristics, these common characteristics should be identified and encapsulated in interchangeable building blocks with common interfaces. The advantages of this approach include:

- The process of writing simulations is sped up, because with a common grounding, the researcher can focus on his or her simulation, and has not to bother with user interfaces design and other elements not central to the simulation, because these can be provided by the simulation framework.
- The software contains less bugs if it is used and debugged by a larger group of people.
- Software written for a specific problem is usually highly specialized and can only be used by the people who programmed it. With a common system, sharing simulation components and results between groups of researchers becomes easier.

Artificial Life Environment

ALE consists of a two parts. The main part is a C++-class-hierarchy which provides the building blocks for artificial life simulations. Custom building blocks are constructed by inheritance from the base classes. The second part of ALE is a graphical user interface which allows easy access to the simulations created with ALE.

ALE focuses on (though it is not limited to) simulations with populations of autonomous entities who interact in a spatial environment. For this, ALE provides four kinds of basic building blocks:

- `Body`

  Artificial life simulations are usually driven by autonomous entities. ALE has a building block for these, called `Body`. In each step of the simulation, all `Bodies` are successively given a chance to act. When an entity gets activated, it examines its environment, and decides on how to act. How the `Body` perceives the environment, how it decides about its actions, and which actions it can perform, depends on how the researcher implemented this building block.

  Two other building blocks are provided to help in the definition of the internal structure of the `Body`: `Chromosome` and `NeuralNetwork`.

- `Chromosome`

  The `Chromosome` contains the genetic information of the entity. This is usually a string of integer or float point values, together with crossover and mutation operators. How this genotype information is reflected in the phenotype, for example who it influences the strength or agility of the entity, depends on how the `Body` building block is implemented.

- `NeuralNetwork`

  The neural network is the "brain" of the entity. The entity uses it to decide how to act. It is the task of the `Body` to feed the information about the current state of the entity and its environment into the neural network, and to interpret the result of the network's calculation as an action. For example, the `Body` might have a perception of the surrounding telling it that some food-source is close by. This information is converted into a format that can be fed into the neural network. The

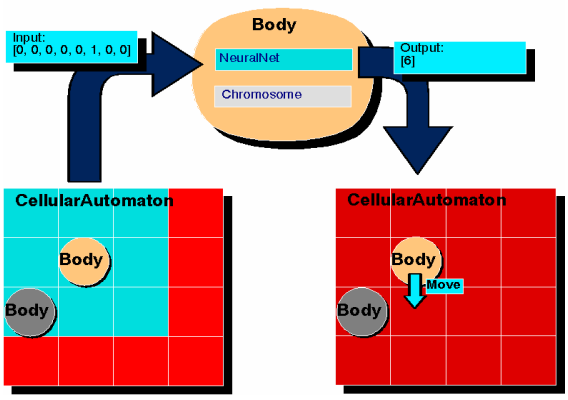

Fig. 1: The steps an entity does when it gets activated: Examining its environment, calculating the next activity, acting.

neural network calculates an output value. The output value is interpreted as an action, for example to move closer to the food source.

By subclassing from the base `NeuralNetwork` class, the researcher can create variations of this building block with different neural network architectures and parameters.

- `CellularAutomaton`

  The fourth and last building block for ALE simulations provides the environment of the entities. For this, we use a `CellularAutomaton`. Our cellular automata have — in difference to the commonly used models — asynchronous update functions instead of synchronous ones, and support inhomogenous cells. Again, if the researchers wants a different type of environment, he or she can reimplement this class and use her class instead as a building block for the simulation.

The interaction of these building blocks is shown in figure 1. First, the entity examines its environment. (In this example its Moore neighborhood.) This information is fed into the neural network. The result of the neural network's calculation is interpretated as an activity. (In this case, to move in direction South.)

An Example Simulation

Simulations are set up by combining these four classes or subclasses of it. ALE has mostly been used to simulate the co-evolution of populations of predators and preys. In these simulations, the predators and preys are special subclasses of `Body`. They interpret their `Chromosomes` as instructions on how to construct their `NeuralNetworks`. The entities have a certain energy level which can raise

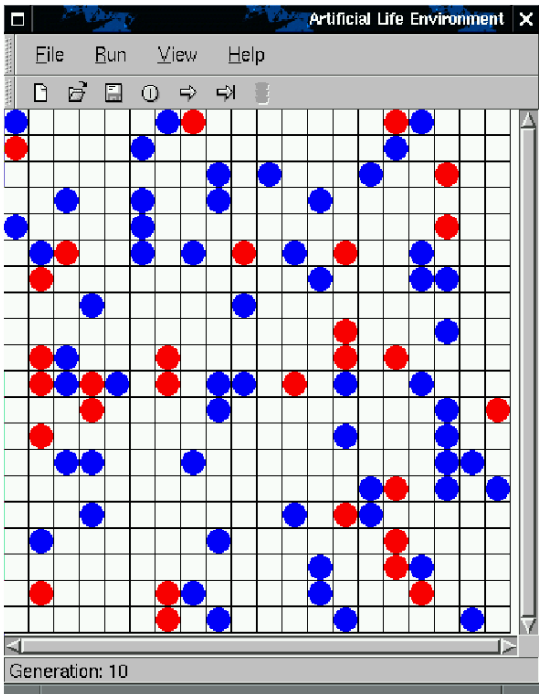

Fig. 2: A screenshot from the predator-prey-simulation. The lighter spots are the prey entities, the darker spots are the predators.

or fall, depending on their actions. When the energy level of an entity drops below zero, it dies. When the energy raises above a threshold, the entity reproduces. In the reproduction process, the `Chromosome` of the child is copied from the parents `Chromosome` with some mutations.

There are four parameters affecting the energy household of an entity:

1. The energy level at birth
2. The energy threshold for reproduction
3. The energy usage per move
4. The energy gain for eating other entities

An entity eats another entity when it moves on the cell occupied by the other entity. The only difference between the `Predator` and the `Prey` subclass of `Body` is that preys can not eat other entities, i.e. they are not able to move on cells occupied by other entities. Therefore, in order to survive, the prey-entities have to have a negative energy consumption. With negative energy consumption, an entity gains energy on every move. We can think about this as the process of taking energy from the environment, like plants are faciliating sunlight. With this setup, it must be the implicit goal of the predators to move onto cells occupied by other entities, and the goal of the preys to move away from the

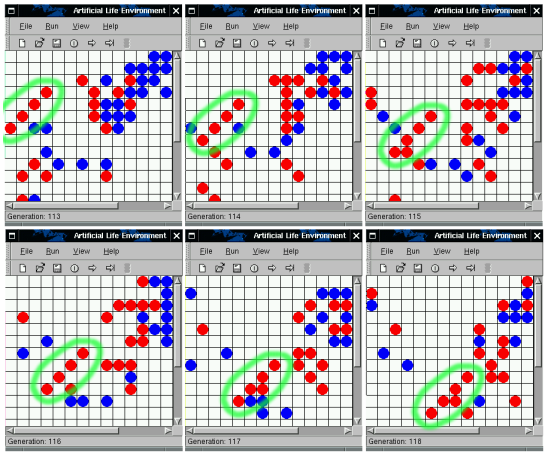

Fig. 3: From upper left to lower right: Six screenshots of predators "fishing" for preys. There is a circle drawn around the predators who are using the fishing-technique.

predators. Figure 2 shows a screenshot of the simulation.

The `NeuralNet` of the entities is defined by the rest of the `Chromosome`. We used a very simple scheme to encode the neural network. The structure of the networks is fixed. We use fully connected feed-forward networks without shortcuts. The single genes in the `Chromosme` define the weights of the neural network's connections. By using this very simple scheme, we can not expect the evolutionary process to generate very sophisticated behaviors. We can observe some improvement in the behavior of the entities, though. One interesting phenomenon to observe is a a rudimentary form of cooperation. The predators start to "fish" for preys by moving diagonally in rows. This form of cooperation is quite efficient, because the predators are systematically wandering over the *Cellular Automaton*, fishing for preys. In evolutionary terms, it is also very easy to develop this kind of behavior. When an entity reproduces, the new entity is placed on a cell adjacent to the parent entity's cell. The only behavior the entities have to show is to always move diagonally in the same direction. Figure 3 shows a sequence of updates in which predators use the fishing-technique to hunt preys.

A second, more interesting phenomenon was observed. When creating eco-system simulations, there are many free variables. In our example simulation, we need sensible values for the original size of the predator- and prey-population, and for the energy-household of the entities. If these parameters are not well chosen, the eco-system breaks down very quickly. Either, the preys vanish, followed by the predators who find no more food, or the predators die-out, and the preys take over the whole `CellularAutomaton`. There are several ap-

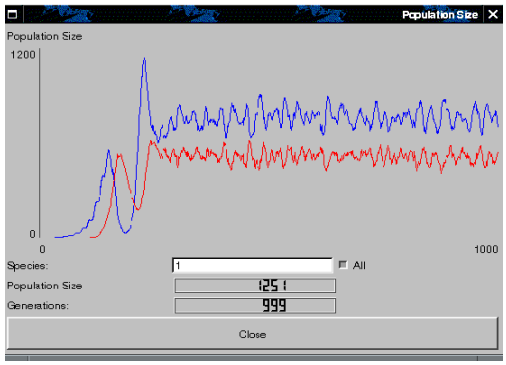

Fig. 4: Development of the population sizes of the predator- (upper line) and the prey-populations (lower line) over 1000 generations.

proaches to solve this problem. One simple yet unsatifsfying solution is to inject new entities whenever one of the populations is about to die out. This keeps the simulation running, but is not a very natural approach to handel the problem. Another approach is to search for good parameter combinations by apropriate search methods, for example by a genetic algorithm.

Surprisingly, there is a very easy and natural solution to this problem. When we make the energy household of the entitites subject to evolution, the eco-system gets stable very quickly. This is done by including the four parameters who govern the energy household of the entities into the *Chromosome*. Figure 4 shows who quickly the population sizes stabilizes when these parameters are subject to evolution: After about 300 generations, the population sizes become quite stable. We see also a phenomenon that can also be observed in natural eco-systems: The size of the prey-population is somewhat larger than the size of the predator-population.

Teaching

ALE has a number of features which make it very well suited as a tool to teach general principles of artificial life simulations. With ALE, a beginner in the field of artificial life does not have to write simulations from scratch. The clear structure of ALE, in combination with a set of predefined classes and a graphical user interface, allow him or her to get a quick start into the field by playing with and manipulating existing building blocks. The system gives a direct, visual feedback.

Conclusions

ALE is still in alpha stage. When ALE has become a more mature product, it will serve as a valuable tool both for research and for teaching in

the field of artificial life. For researches, it provides a set of building block which allow to develop a project without spending time on the programming of low level features. The building block concept also makes it easy to exchange parts of the simulations and to compare simulation results. Additonally, the developer gets tool for the analysis of simulations.

In the realm of teaching artificial life, the graphical user interface and the already existing building blocks allow the beginner in the field of artificial life to get a direct experience for simulations and how they are affected by different parameters, without having to program a whole system.

The current version of ALE is available for download at http://www.uni-koblenz.de/~gb/ale/. This version is not ready for productive use. It is only of interest for programmers who might want to participiate in the development of the system.

References

[1] Gerd Beuster, *Artificial Life Environment*, Master's thesis, University Koblenz-Landau, Koblenz, 1999, http://www.uni-koblenz.de/~gb/ale/studienarbeit_ale.ps.gz

[2] A. K. Dewdney, *Simulated evolution: wherein bugs learn to hunt bacteria*, Scientific American, May 1989.