Res. Lett. Inf. Math. Sci., 2004, Vol.**6**, pp143-155
Available online at http://iims.massey.ac.nz/research/letters/

A framework and simulation engine for studying artificial life

H.A. James, C. Scogings & K.A.Hawick

Institute of Information & Mathematical Sciences
Massey University at Albany, Auckland, New Zealand.

The area of computer-generated artificial life-forms is a relatively recent field of inter-disciplinary study that involves mathematical modelling, physical intuition and ideas from chemistry and biology and computational science. Although the attribution of "life" to non biological systems is still controversial, several groups agree that certain emergent properties can be ascribed to computer simulated systems that can be constructed to "live" in a simulated environment. In this paper we discuss some of the issues and infrastructure necessary to construct a simulation laboratory for the study of computer generated artificial life-forms. We review possible technologies and present some preliminary studies based around simple models.

1 Introduction

Artificial Life, or ALife, is an exciting emerging field that attempts to capture the "rules" of a live system in a computer simulation (6; 2). There have been a few popular projects which have pioneered this Animat (12) field, including Avida (1) and Tierra (10) and a small number of research laboratories (for example (3; 11)). Simulations base their rules on mathematical modelling, ideas of biology and also physical observations. The above approaches have sought to model the complexities of life in a way that would lead to the exhibition of behaviours that we see demonstrated in real life: predation, herds forming and offspring being born. So far researchers have succeeded in qualitatively showing the above types of phenomena in ALife systems. Such phenomena, called "emergent behaviour" has been cited as one of the major breakthroughs for the ALife field (8).

Current systems model the entities in the virtual world as Agents, where agents have some sort of goal they wish to fulfil. This goal could be specific (e.g. mate with as large a number of similar animals as possible in order to guarantee the

survival of the species) or could be specified as something more general (e.g. survive, which necessarily requires one to find food, avoid predators, reproduce and pass on information to offspring), for example projects such as ECHO (5) at the Santa Fe Institute (11).

The major issue facing those who would create an exhaustive ALife model of a realistic live system is that the possible parameter space (e.g. the number of possibly independent variables that would need to be specified to properly describe the system) is far too large for any model to successfully enumerate. In addition, the number of different states that each agent may be in (in a system realistically emulating a live animal) is too large to effectively manipulate on even a modern computer with a population of "interesting" size. As such, existing systems are limited to model only simple systems with simplified goals and states. In addition very few useful statistics are calculated by the systems; while many of the above projects are useful mainly as demonstrator systems, the field has found considerable usefulness in the movie and computer games industries (e.g. (9)).

Taking a slightly different tack, we are constructing frameworks with which we are able to quantitatively investigate the properties of various ALife models. Instead of attempting to completely enumerate the internal state of our models' agents we attempt to simplify our agents down to the bare basic information that is required to maintain state and achieve a goal. We are also investigating the effect that the layout of the "world" and also communications facilities might have on the properties of the systems.

Our frameworks are constructed in such a way as to be able to investigate emergent behaviour, perhaps parameterisable so that below a critical separation size agents exhibit one behaviour, above it they do something else; or predators die if the prey density is below a certain amount. In order to achieve any statistically significant, generalisable results, we must repeat an experiment many times with different starting conditions; our frameworks are designed to both support re-producible experiments and also experiments with different starting conditions. Analysis of the models' statistics can show when behaviours are convergent. Finally our frameworks support the ability to save and restore the state of the complete system, allowing us to start the simulation from a state with known properties.

From a technical perspective it is not necessary (nor perhaps even desirable) that the engine do everything without recompilation - it may be we want it to be very fast or at least directly executable on a cluster system. It is perhaps more important that we have a pattern or style or convention for how things are done in the engine rather than have everything rigidly prescribed. For example it may be enough just to have established data structures and collection of code fragments that are compatible rather than a huge perfect code edifice, with which other programs, perhaps measuring statistical information, can attach.

In this paper we describe the issues and infrastructure that are necessary to construct a simulation laboratory for the study of ALife forms. In section 2 we discuss the various properties of ALife models. In section 3 we discuss various com-

munications strategies that may be supported by the environment or the agents themselves. In section 4 we describe general implementation issues for ALife frameworks. In sections 5 and 6 we describe two of our prototype frameworks, and in section 7 we summarise the paper.

2 Model Laws

The main aim behind building different frameworks for exploring ALife models is to derive quantitative data from the behaviour of the systems. From both the observable and quantitative data produced by our simulations we attempt to derive any "laws" that are applicable to the systems.

For example, we might look for the following (semi-)statistical phenomena and behaviours in the system: Is the population stable? Does it grow/shrink without bounds? Is there some sort of stable (or semi-stable) ratio of predator to prey? Are the changes in population regular or oscillatory? Or random? Do certain parameters affect the periodicity or the amplitude of any cycles? Do the combinations of any parameters have interference effects? Is it possible to express any of these laws as, for example, power laws or scaling laws? From these questions it is obvious that any frameworks must support structured systematic simulations across the parameter space.

It may also help to categorise species according to the way they go about achieving their goals. For example, species may be cooperative, aggressive, passive, symbiotic or solitary, and they may as a secondary effect exhibit other behaviours such as herding.

Through proper framework design it is possible to measure many properties of a system, including those that initially would not seem applicable from a purely computer science perspective. For example reproduction rates and clustering information would seem sensible property to measure in a predator-prey model but perhaps measuring the energy or entropy in the system would not, but may lead to extra understanding of the behaviour of the model.

3 The Role of Communications

One of the more interesting features to emerge from the ALife literature is the phenomenon of collective behaviours, where agents are seen to operate in concert. Most traditional ALife frameworks do not have any notion of inter-agent communications apart from the extent that a neighbouring agent's current state might be able to affect another's state in the next simulation time step. It has been suggested that any emergent collective behaviours are simply artifacts of the instructions that each agent is following rather than any notion of "intelligence".

To further explore this issue, our frameworks attempt to add the facility whereby agents have the ability to communicate some information, however limited in volume

or scope. This feature adds considerable complexity to the frameworks, as discussed in section 5 in not only the range of messages but also more fundamental issues such as when the messages are delivered in the agent update cycle.

4 Framework Technologies

When considering the implementation of a simulation framework it is necessary to first consider the nature of the "world" in which agents will exist and interact. Most existing models use a discrete or cellular space as opposed to a real (x, y, z) space found in real life. In our frameworks we have adopted a discrete space; we have found that varying the size of each discretised point between models can effectively approximate the use of a real space.

The agent to be sited at each point must also be considered. Different types of agents, and their interactions, will of course be dependent on the nature of the model (see section 2). Some models may require the use of many different types of agent, each with a specialised task (or set of goals). The task of actually formalising and expressing goals in an agent-based system is the subject of much research. In the implementations described in sections 5 and 6 we use a simple state-based behaviour table (modelling agents as finite state automata).

The frameworks we have designed and built follow the same pattern of operations:

1. initialise the world and restore the positions of any agents
2. prepare the system for updating (including any reproductive consequences)
3. commit any proposed changes that the agents wish to make
4. perhaps store the new configuration
5. measure the new state of the system, producing any relevant statistics

An initial decision must be made as to the role and responsibilities of the simulation framework. Does the framework arbitrate between two agents that wish to move to the same space or not? This leads to the two major system-level perspectives: local and global. If a framework has a local perspective it allows each cell to essential take care of itself, and if two agents wish to inhabit the same cell then the framework itself does nothing to prohibit this, whether or not it leads a "legal" state. In a global perspective the framework may arbitrate between agents that wish to perform an action that leads to an illegal state. There is a fundamental trade-off: using global state introduces a considerable amount of computational complexity to resolve any apparent conflicts, but it also makes the introduction of long- and short-range communications easier. In addition does the framework provide global control in the form of a systolic 'pulse' update that all points (and hence agents) rely on for timing, or in a local context are al agents able to execute asynchronously?

In our models a global update mechanism has been implemented to simplify agent synchronisation. Furthermore, another question that must be answered is: What support will the framework environment give to agents? Will the framework give the agents information as to their neighbouring sites? Will the environment store variables that are required by all sites, (i.e. global or universal parameters) that may or may not change during execution?

Another consideration is the language that will be used for implementing the framework and the agents. There is often a trade-off between ease of programming and display one one hand and the speed of the system on the other. The two prototype frameworks described in the following sections highlight these tradeoffs, with the first being implemented in the Java language, which lends itself to graphical applications and the second being implemented in C++ for speed.

When the framework has matured to the point of being used for data collection in large or complex parameterised studies it is often beneficial to prevent any graphical output to speed up the simulation. However, we also observe the benefit of spending some time to visualise the state of the system described by the framework, especially in the early phases of development, despite the cost in programming time. A graphical display often aides in the understanding of the system – and especially helps to identify errors! From an academic standpoint it is also very useful to have the graphics display connected to a file output device, such as a postscript generator, that can be used to generate quality screen output for the inclusion in reports.

As one of the aims of our simulation frameworks is the parameterised study of models under different starting conditions, it must be possible both to save/restore configurations and also to manage the parameter spaces that will be searched during repeated program executions. For this reason, it is vitally important that the agent implementations support some way to reliably save and restore their state. It is also vital that the framework support some way of saving the configuration, together with the agents' configurations that are able to be restored. The framework's state may include any geometry information and also the values of any global simulation parameters. In early versions of the framework described in section 5 global parameters were recorded in the filenames of any output configuration files; of course as the number of such parameters increase it is often useful to include such information inside the configuration file.

In retrospect we observe that while it is comparably inefficient to store configuration data in a test-based format inside files, this technique lends itself to far easier searching of files during any post-production analysis of data.

A final consideration is the method that is used to measure any statistics produced by the system. We observe that the collection of statistics is a time-consuming process; it is often efficient to collect statistics only at regular intervals to speed up the system.

5 A Prototype Geometrical Framework

A prototype geometrical framework was devised to test out the effects that 2-D geometry has on the behaviour of well-known cellular automata (or agent) models. Three different geometries were incorporated: square, triangular and hexagonal; these three geometries have the required property that cells are able to neatly abut against each other.

The (traditional) square lattice has four nearest neighbours; it also has four next-nearest neighbours, which can be included in calculations by increasing the *neighbour radius*. The triangular lattice has three nearest neighbours; there are also three other cells with corners that touch each cell. Each element in the hexagonal lattice has six nearest neighbours.

Written in Java, the model is an attempt at a theoretically correct partitioning of the cellular automata's constituent parts despite the relative lack of speed of the Java runtime system. The framework creates a lattice of elements, each of the specified shape. Lattice elements are placed into an (x, y) space with relative coordinates so they are able to calculate their nearest neighbours based on the *neighbour radius*. Automata, or agents, are located inside the lattice elements. Agents are themselves unaware of their environment, they must use the lattice framework to discover the state of their neighbouring agents. Also incorporated into the highest level of the framework is a systolic update mechanism for allowing each element to be updated while maintaining the system's consistency.

When the prototype was first built the rather difficult issue of an efficient update algorithm was considered. The model that was adopted was essentially a two-stage commit model in the hope that it would lend itself to a parallel implementation in the future. In the first step each element in the model would note the neighbouring cells' state. Based on the state, and the internal rule table of the automata a decision would be made, resulting in the transition to a new state (even if the new state was the same as the current state). Once each element in the lattice had completed the first phase the second phase would begin which performed a commit on the new state for each element. After the second phase the framework will calculate some properties of the model's current state, if applicable, including statistics on the neighbour states and histograms of cluster sizes.

Several simple models were coded and tested to ensure the system's correctness, such as the traditional Conway's Game of Life and a version of the Ising statistical mechanics simulation (4). These models were implemented as though there was no actual movement of agents between framework elements. Each agent was already resident on a framework element; when not activated the automata was in a zero state and when active it was in a non-zero state.

The next stage of development was to incorporate the ability for automata to move between framework elements. The addition of this feature caused problems with the rather simple two-stage update procedure due to various threading issues between automata elements. A solution to this problem took the form of a

Monte-Carlo update scheme, whereby random sites in the lattice were chosen to concurrently perform the first phase of the update until all sites have been visited then the second phase was started. This fixed the problem of concurrent updates across the framework's elements.

A waterdrop model, which represents the way that water dripping into a sandy area forms rivulets was also coded. New agents are added to the framework element at the centre of the screen. They randomly choose a nearest neighbour site to move to; barring a water droplet moving backwards there is a far greater chance of the droplet following a path that has already been established than creating a new path. This model, in different geometries is shown in figure 1.

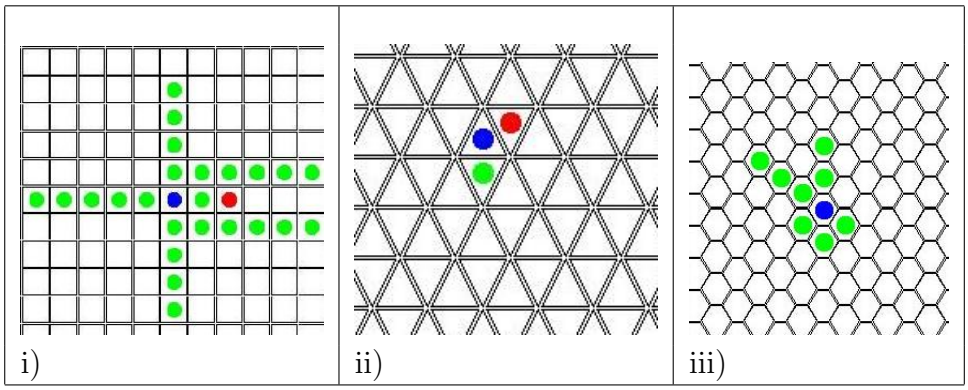

Figure 1: Prototype Geometrical Framework showing the waterdrop model on (i) square, (ii) triangular, (iii) hexagonal lattices. Waterdrops are added at the central site and move to a randomly chosen neighbouring site. Preference is given to neighbouring sites where a waterdrop has already been; there is a small probability of an 'unvisited' site being chosen.

The waterdrop model required no communication between either the framework and the agent or between the agents. As the framework developed the ability to allow automata to communicate in a short-range fashion was desired. This allowed the creation of more sophisticated models, such as the emulation of a worm with a head and a tail. The tail is left as a "tombstone" by the worm head as it moves; the tail remains for a given number of steps thus limiting its length. The worm chooses a random direction in which to travel that is not currently occupied by either its tail or another worm.

Short-range communications allowed small amounts of state (or agent *intention* to be communicated. A message could be left either with the lattice element that was due to be vacated by an agent or placed in a neighbouring element for collection. Short-range communications were to be used in the instance where a worm crossed over the tail of either itself or another worm. Specifically for our worms model this situation would cause the "promotion" of the leading edge of the remaining tail segment to be a new head, and therefore able to make its own movement decisions.

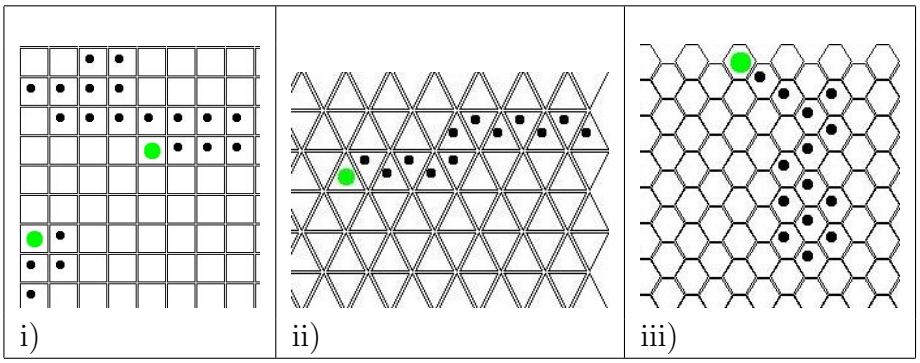

Figure 2: Prototype Geometrical Framework showing the mobile worms model on (i) square, (ii) triangular, (iii) hexagonal lattices. The worm's "head" chooses which direction to move and the "tail" is dragged via short-range communications. The second worm in (i) may cut the first worm's tail causing a new worm "head" to be created.

After the prototype had been developed and tested for correctness, we decided to test the effects of long-range influences on the model. This was done in two ways: long-range communications and also by perturbing the 'normal' framework lattice. We describe the first of these long-range communications only.

Long-range communications allow agents to pass messages to those agents who were not immediately adjacent to each other. For example, in a model of foxes and rabbits this might be used to model the coordination of foxes' attack through the use of an audible mechanism.

6 A Prototype Population Model

A simple prey-predator model has been constructed and is under investigation. This type of model contains two groups of "animals" - the predators and the prey. Animals of both types move, breed and die after living a maximum life span. Predators eat prey if they can catch it. Predators that have not eaten prey within a certain period will die. Prey do not need to eat as the current simplistic model assumes sufficient natural resources to sustain life.

The model is executed as a sequence of cycles. During one cycle, the state of every animal is updated and thus one complete model state is constructed. In a specific model, the cycle might correspond to a year or a month but in this theoretical model it is simply referred to as "a time step".

The state of an animal is updated by applying rules to the current state. The current state of an animal is contained in a number of variables that are used to record information such as: location, age, hunger, number of prey neighbours, number of predator neighbours.

There is a different set of rules for each animal type (in this case, predator and

prey). Typical rules include: predator will move towards prey if hungry; prey will move away from predator if adjacent; and animal will breed if adjacent to another animal of the same type.

Each rule has a priority and they are always applied in priority order, e.g. prey rate moving away from predators higher than breeding. Most rules have a condition, i.e. the rule will only be applied under certain circumstances – see above examples. Thus, for each animal, in each time step the list of rules is checked in priority order and as soon as a rule can be applied, it is and no further rules are checked.

6.1 Implementation

The model has been implemented as a sequential C++ program. As discussed in section 5, there are two ways of updating the state of such models – simultaneous update or sequential update. Simultaneous update lends itself to a parallel implementation but is difficult and time consuming to implement in a sequential program as it requires two states to be maintained for the model at all times – the "current state" and the "future state" where the future state is constructed by applying rules to the current state. At the end of each time step, future state becomes the current state and the process is repeated.

This model uses sequential update because it is faster and requires only a current state which is constantly changing as the state of each animal is changed. One possible drawback of this system is that certain animals are updated prior to other animals. In order to ensure that this does not consistently advantage one particular group, the animals are updated in a random order which is changed at the end of each time step.

6.2 Analysis

A successful model is one in which a stable environment can be created enabling many cycles to pass before the collapse of one, or both, of the predator and prey populations. This success is dependent on 5 key parameters:

- **predator maximum age** number of time steps that elapse before an animal dies;
- **predator hunger level** number of time steps that can elapse before the predator must catch and eat prey;
- **predator birth rate** percentage chance of successful breeding each time breeding is attempted;
- **prey maximum age**;
- **prey birth rate**.

Although each of these parameters is important in its own right, it is the combination of them that creates a successful, or otherwise, model environment. For example, a high prey birth rate will ensure a rapid increase in the number of prey animals, but a low predator birth rate will also increase the number of prey animals – since less predators means less prey are eaten.

In an attempt to test each of the combinations of the above parameters for convergence – to test whether the model enters a stable (or semi-stable) state, the cross product of the model's parameters are being executed across many different random starting configurations. This is an extremely time consuming process that has benefited greatly from use of the Helix supercomputer cluster.

6.3 An example

Figure 3 follows the progress of one execution of the model. The parameters for this run were set to the following values:

- Pred. Max Age = 80
- Pred. Health = 50
- Pred. Birth Rate = 20
- Prey Max Age = 20
- Prey Birth Rate = 50

Figure 3i) shows a small part of the situation at the end of time step 40. At this stage there are 183 predators and 1974 prey in the model. The prey have a fairly high birth rate (50%) and have populated most of the world. But due to the longevity of the predators (80) they are building up steadily and each "pack" of predators occupies its own area in which all prey have been eaten. Each predator is hungry only for a short time and then finds adjacent prey to eat.

Figure 3ii) shows a small part of the situation at the end of time step 80. At this stage there are 510 predators and 2693 prey in the model. Predator increase since step 40 is 179% and prey increase is 36%. Although the predator birth rate (20%) is lower than the prey birth rate (50%), each predator lives much longer (80) than the maximum age of prey (20). This longer life span increases the number of successful breeding attempts for each predator, thus the number of predators are able to increase faster, as long as the prey are available as a source of food.

Figure 3iii) shows a small part of the situation at the end of time step 120. At this stage there are 995 predators and 2617 prey in the model. Predator increase since step 80 is 95% and prey have decreased by 3%. The predators have eaten almost all the prey in the area shown and large numbers of hungry predators are visible. Many of them will be unable to reach the now-distant prey before dying through lack of food.

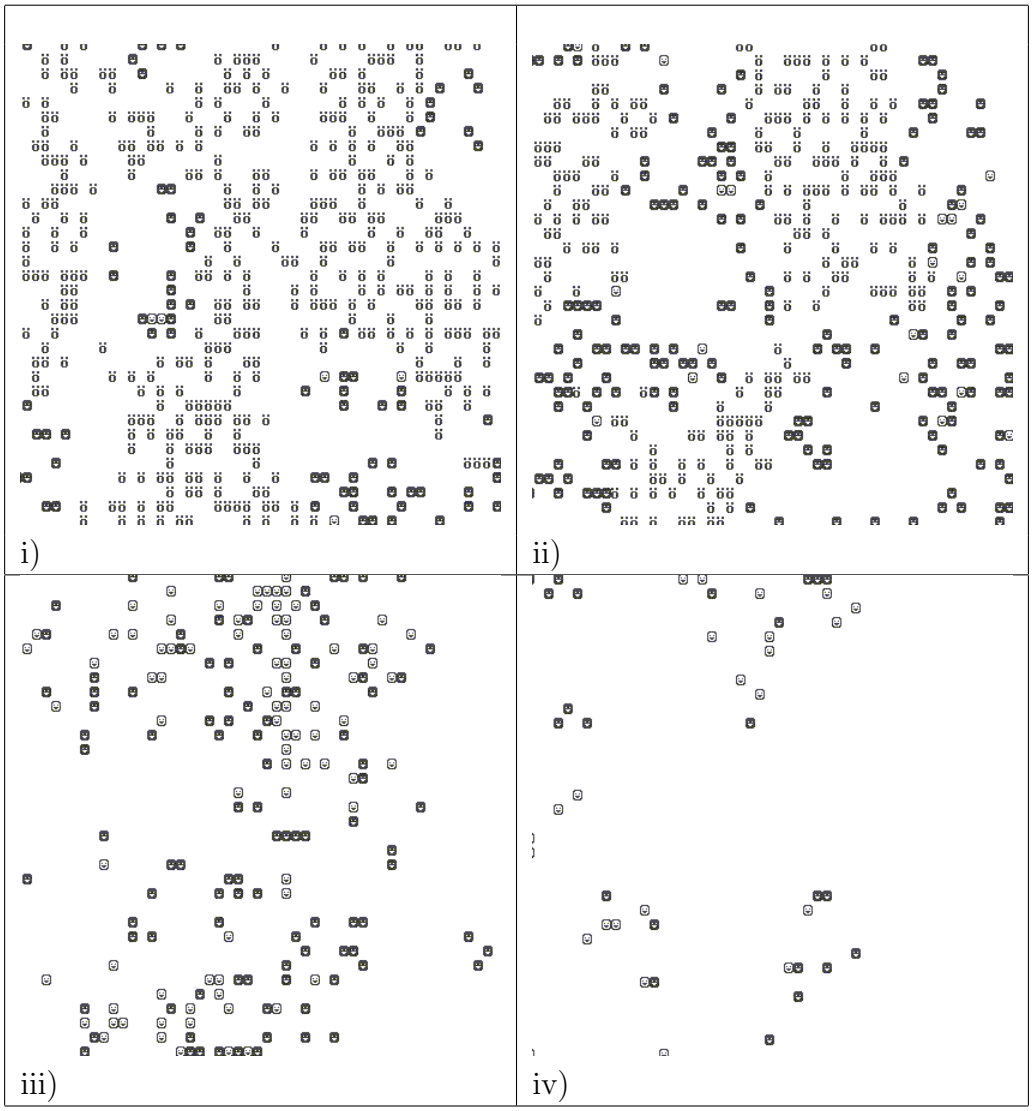

Figure 3: A Predator-Prey simulation at timesteps i) 40, ii) 80, iii) 120 and iv) 160. Key: ☺ = Predator; ☻ = hungrey predator; ö = Prey.

Figure 3iv) shows a small part of the situation at the end of time step 160. At this stage there are 1270 predators and 2414 prey in the model. Predator increase since step 120 is 27% and prey have decreased by 8%. No prey are visible in this area and many predators are hungry. The rate of decrease of prey is increasing and the rate of increase of predators is decreasing.

The parameter setting in this example does not lead to a stable model as the maximum age of predators is too high. This can be compensated for by either lowering the maximum age or raising the maximum age and/or birth rate of the prey.

6.4 Future work

The analysis of the model needs to be completed and this will produce the optimum combination of parameters. A number of other parameters have been identified, such as a variable food source for prey, more complex rules for animal behaviour and the effects of disease on populations. These parameters can then be included in the model. This model will then be used to create an Artificial Life "engine" – software that would allow the rapid creation of similar models that could be used in simulations in a number of areas.

Another area of future work would be the introduction of "genetic breeding". In the current model, each animal has exactly the same set of rules built into it when it is created. It would be interesting to enable animals to select a combination of the parents' rules in order to make a new rule set. This may change the composition of rules and also the priority of rules. In time, given a stable model, new more efficient life forms may emerge (7).

7 Summary and Conclusions

We have considered the requirements for building effective frameworks to support the study of Artificial Life models. This has lead to the understanding of the need for a carefully-designed framework in which the systematic study of models, together with careful control of parameters can be made; such a design can only be made after consideration of many factors including the model laws and the statistics that are desired from the system.

We have also discussed the need for careful choice of implementation language. While execution efficiency is the critical factor, simplicity of formulation is also very important. From the two prototype frameworks we have developed, languages such as Java and C++ seem quite suited to this task, allowing an object-oriented approach while lending themselves to visualisation. While have found that the measuring of various statistical information to be by far the most time-consuming part of running large simulations, it is very useful to have a program that supports averaging or statistical analysis of properties obtained through many differfent trajectories

through the model's phase space. We believe our experimental programs satisfy these needs.

Acknowledgements

The authors gratefully acknowledge Massey University and the Allan Wilson Centre's contribution of Helix supercomputer time in the production of the frameworks and data reported in this paper.

References

[1] Adami, C. Avida (Digital Life Laboratory) Available at http://-dllab.caltech.edu/avida

[2] The International Society of Artficial Life Available at http://www.alife.org

[3] Caltech Digital Life Lab. Web pages available at http://dllab.caltech.edu

[4] Hawick, K.A. Agent Formulation of the Ising Model Technical Note CSTN-004. Available from http://www.massey.ac.nz/~kahawick/cstn November 2003.

[5] Holland, J. ECHO Available from http://www.santafe.edu/projects/echo/-echo.html

[6] Kauffman, S.A. "Investigations" Oxford University Press ISBN 0-19-512104-X

[7] Levy, S. "Artificial Life" Penguin Books, 1992. ISBN 0-14-023105-6

[8] MIT Press The Journal of Artificial Life

[9] NaturalMotion Endorphin, Active Character Technology Available at http://-www.naturalmotion.com

[10] Ray, T. Tierra Available at http://www.isd.adr.co.jp/~ray/tierra

[11] Santa Fe Institute. Available at http://www.santafe.edu

[12] Wilson, S. The Animat Path to AI in *From Animals to Animats 1: Proceedings of The First International Conference on Simulation of Adaptive Behavior*, (pp. 15-21); Meyer, J-A. & Wilson, S. (eds), Cambridge, MA: The MIT Press/Bradford Books (1991).