Floatworld : A Simple Artificial Life Framework for Simulated Evolution

Taliesin Beynon

November 20, 2008

Abstract

1 Introduction

1.1 Aim

In this project, an attempt will be made to produce a simulated environment in which virtual "creatures" compete for space and energy. We will then examine the ability of evolution by natural selection to drive the increase in fitness of the population via differential reproduction.

What choices should be made for the structure of the simulated environment? Two constructs are at the very least necessary: space in which creatures can be located, and a population of creatures. While the spatial extent of organisms in real life is arbitrary, ranging from a scale of 10^{-8} m to 10^{2} m in all 3 spacial dimensions, we will assume for simplicity that all organisms in the model are the same size. Moreover, we will assume space is discretized into individual "cells", and laid out in a 2-dimensional rectangular lattice, called the **grid**.

How will the creatures be constructed? The orthodox view is that the following are necessary but not sufficent conditions for an entity to be considered living:

- Metabolism
- Reproduction
- Adaptation
- Response to Stimuli

The study of computer simulations in which these features emerge is known generally as "Artifical Life". It is a cross-disciplinary field involving scholars from computer science, evolutionary biology and ecology, and applied mathematics.

The virtual creatures we will examine will be designed to satsify all four of these conditions. In Section 3 we will examine more closely the grid and creature constructs, and how various design decisions were made to satisfy these criteria.

1.2 Previous Work

Tierra is a software simulation written by the ecologist Thomas Ray. In Tierra, a virtual machine[1] executes short programs that contain instructions to copy themselves to new memory locations on the virtual machine. Each short program is taken to be a virtual organism.

In essence, different organisms compete for execution time on the virtual machine. Each copying operation has a small chance of introducing an error into the new copy of the original program. By virtue of this fact variation is introduced into the population, and programs that are shorter or use other techniques to acquire more time on the virtual machine will copy themselves faster and come to dominate the population. In this manner natural selection operates on the population to produce programs that are better adapted to the virtual environment.

Experiments with Tierra have revealed patterns of parasitism, in which extremely short programs exploit longer programs to self-replicate. A derivation of the Tierra program known as **Avida** has been used extensively in artifical life research. It has been used to study the problem of "irreducible complexity" in which features requiring several non-adaptive mutations emerge [1]. Other interesting work involving Avida has studied specialization of digital organisms into various ecological niches [2] and how evolution affects genome complexity [3].

Sugarscape is an agent-based simulation written by the M.I.T. scholar Joshua M. Epstein and Robert Axtell [4] in which individual agents in a grid-like lattice make economic decisions about how to trade two seperate resources that are both essential for survival, abitrarily called "sugar" and "spice".

Another example of research conducted in the field of artificial life is [5], in which digital organisms similar to the ones developed in this project compete and/or cooperate with each other for space and energy. Results showed that various familiar game-theoretic cooperative strategies emerged, as well as some that had not been seen before.

References

[1] Richard E. Lenski, et al, "The evolutionary origin of complex features", Nature 423, 2003

[2] Richard E. Lenski, et al, "Ecological specialization and adaptive decay in digital organisms", American Naturalist 169, 2007

[3] Richard E. Lenski, et al, "Genome complexity, robustness and genetic interactions in digital organisms", Nature 400, 1999

[4] Joshua M. Epstein & Robert Axtell, "Growing Artificial Societies: Social Science From the Bottom Up", M.I.T. Press, 1996

[5] Mikhail Burtsev & Peter Turchin, "Evolution of cooperative strategies from first principles", Nature 440, 2006,

[1] A virtual machine is a simulation of a Central Processing Unit operating on one or more programs. In other words, a virtual machine is a simulated computer running on a real computer.

2 Selected Topics

In this section we will review a few basic concepts which will be useful in our discussion of the *Floatworld* virtual environment. First we will discuss *artificial neural networks*, the constructs used to implement a primitive form of intelligence to govern the virtual creatures. We will also discuss the classical concept of *natural selection*, and the application of this idea in the field of *genetic algorithms.*

2.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a generalized, simplified model of the computational activity performed by networks of neural cells in many living organisms. As in biological neural networks, an ANN is divided into a fixed number N of neurons, or nodes, which communicate with each other using a fixed set of connections. Each neuron is labelled with a natural number $n \in \mathbf{N}$ where $\mathbf{N} = \{1, 2, \ldots, N\}$. The **activity** of an individual neuron is represented by a real number α_n for $n \in \mathbf{N}$, where activity is the analogous concept to the firing rate of a biological neuron.

Each connection from neuron m to neuron n has an associated **weight** w_{nm} which determines how the activity of the first neuron will influence the activity of the second neuron. Roughly speaking, a positive weight means that activity of the first neuron will increase the activity of the second neuron, and a negative weight means the activity of the first neuron will decrease the activity of the second neuron. If $w_{mn} \neq 0$, we say that neuron n is connected to neuron m.

Time is discretized into steps $t = 0, 1, 2, \cdots$. The activity of a neuron at any given timestep $\alpha_n(t)$ is determined by the activity of all the other nodes at the previous timestep according to the function

$$\alpha_n(t) = f\left(\sum_{m \in \mathbf{N}} w_{nm}\, \alpha_m(t-1)\right) \tag{1}$$

In other words for each neuron, the weighted sum of the activity of all the neurons is passed through the a function (the so-called **activation function**) $f : \mathbb{R} \to \mathbb{R}$ to determine the future activity of the neuron n. Different neurons exhibit different behaviour because the weighting factors w_{nm} depend explicitly on the neuron n.

In fact the notation w_{mn} suggests a computationally efficient way of computing the activity of an ANN recursively: using matrix multiplication. If we use the column vector $\overrightarrow{\alpha}(t) = (\alpha_1(t), \alpha_2(t), \cdots, \alpha_N(t))$ to represent the activity of all the neurons in the ANN at time t, then the recursation relation in Equation 1 takes the form

$$\overrightarrow{\alpha}(t) = F(W \cdot \overrightarrow{\alpha}(t-1)) \tag{2}$$

where W is the matrix whose elements are given by w_{ij} for $i, j \in \mathbf{N}$, and $F : \mathbb{R}^N \to \mathbb{R}^N$ is the map induced by $f : \mathbb{R} \to \mathbb{R}$.

2.2 Adjacency Matrix

A directed graph consists of a set of verices V and a set of edges E. An edge is an ordered pair (u, v) where $u, v \in V$. A graph can be represented visually whereby each node is a point and each edge is an arrow connecting the corresponding points.

Let a directed graph have n vertices, and label the vertices of the graph by the integers $\{1, \cdots, n\}$. The **adjacency matrix** G of a directed graph is the square matrix of size n in which

$$G_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$$

As an example, Figure 1 shows both a simple graph and the adjacency matrix of the graph. Vertex numbers correspond to matrix columns and rows.

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

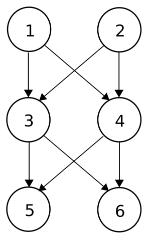

Figure 1: Adjacency matrix and corresponding graph

2.3 Network Topologies

We can consider an ANN to be weighted finite directed graph G, i.e. a finite directed graph in which the adjacency matrix G does not merely take values in $\{0, 1\}$ but in $\mathbb{R}$. Note that in an adjacency matrix, $w_{ij} = 1$ implies vertex i connects to vertex j, whereas in the weight matrix $w_{ij} = 1$ implies neuron j connects to neuron i. Therefore, $G^T = W$. From this perspective the relation described in Equation 1 implements a model in which the graph of the ANN is fully connected, and every neuron has a connection with every other neuron.

Equation 1 models a neural network in which the notions of input and output are not defined. Information is not presented to or extracted from the neural network in any explicit fashion. Usually ANNs are introduced differently, in which the topology of the graph G is constrained in some way.

In so-called **feed-forward** ANNs the graph G is a directed acyclic graph. This implies that the vertices of G can be **topologically sorted**: formally, there exists a map $d : \mathbf{N} \to \mathbb{N}$ such that

$$w_{mn} \neq 0 \implies d(n) < d(m)$$

In simple terms the activity of neuron m in such an ordered or **layered** network is only dependant on the activity of neurons prior to it in the order, i.e. those

n for which $d(n) < d(m)$. Such networks are often depicted graphically in a layered format.

In many kinds of neural network devoted to information processing we speak about the sets of **input neurons** $\mathbf{N}^{\downarrow}$ and **output neurons** $\mathbf{N}^{\uparrow}$. These are precisely the neurons which have no connections to them, and no connections from them, respectively. Formally we can write

$$\mathbf{N}^{\downarrow} = \{n \in \mathbf{N} \mid (\forall m \in \mathbf{N})\ w_{nm} = 0\}$$
$$\mathbf{N}^{\uparrow} = \{n \in \mathbf{N} \mid (\forall m \in \mathbf{N})\ w_{mn} = 0\}$$

The feed-forward ANNs mentioned above are used in many practical applications. In general the activity of the input neurons in a feed-forward ANN is fixed by a time-dependant function, so that

$$(\forall n \in \mathbf{N}^{\downarrow})\ \alpha_n = f(t, n)$$

Intuitively f has the effect of "feeding" information into the ANN. The activity of the neurons in the set $\mathbf{N}^{\uparrow}$ is then understood to be the output of the entire system. Usually the function f is time independant, and in that case the behaviour of the neural net is equivalent to some function $H : \mathbb{R}^{|\mathbf{N}^{\downarrow}|} \to \mathbb{R}^{|\mathbf{N}^{\uparrow}|}$. This is because the ordering on the feed-forward ANN makes it possible to calculate the time-independant activity of the output neurons in a single computational pass, without explicit use of the timesteps $t = 0, 1, 2, \cdots$.

However, in some applications the neural net has to respond to time-varying inputs. In other applications a strict-ordering on the graph G is not possible because it is cyclic, and so the ANN does not necessarily have a fixed output even if f is constant with time. Both of the previous conditions will be true of the neural networks used in this project.

Such networks are called **feed-back** neural networks, and their activity usually has explicit time dependance. However, such a network can still be considered a function $H : \mathbb{R}^{|\mathbf{N}^{\downarrow}|} \times \mathbb{R}^{N_S} \to \mathbb{R}^{|\mathbf{N}^{\uparrow}|} \times \mathbb{R}^{N_S}$, where N_S is the dimension of the **internal state** of the ANN, being just the number of neurons that have connections that "feed back into" the network, i.e. the number of $n \notin \mathbf{N}^{\downarrow} \cup \mathbf{N}^{\uparrow}$ for which there is at least one $m \notin \mathbf{N}^{\uparrow}$ such that $w_{mn} \neq 0$.

2.4 Weight Masks

Under some circumstances, it is advantageous to fix many of the weights in the weight matrix W to be zero. In this way, the graph of the ANN are constrained to a subgraph of the fully connected N-graph. This might be useful to enforce a feed-forward structure, or more generally to ensure that number of degrees of freedom of the ANN is a subset of the full number N^2.

Let M be the transpose of the adjacency matrix of a graph with N vertices. Taking $M.W$ (where . indicates point-wise multiplication of matrix entries) ensures that $M_{ij} = 0 \implies (M.W)_{ij} = 0$. In other words $M.W$ is the weight matrix of an ANN whose graph is a subgraph of the graph of the ANN with weight matrix W.

2.5 Genetic Algorithms

Evolutionary computation describes the paradigm in which principles of Darwinian evolution are employed in a computer code to develop solutions to a given problem. Many different problems can be approached from the evolutionary computation perspective. It is required of the problem that potential solutions can be objectively judged on their suitability or quality by an appropriate function.

Darwinian evolution among living beings is often described colloquially as "survival of the fittest". This, however, is a tautology: fitness is defined precisely to be the tendency of an organism or entity to survive and reproduce. However, the same is *not* true of computer codes using Darwinian evolution. In the context of genetic algorithms, the "fitness" of a solution is imposed from the outside; it is described explicitly using a **fitness function** $f : S \to \mathbb{R}$, which is a function that evaluates the quality of a solution in the solution space S. A higher value of $f(s)$ means a better solution s.

The process by which an adequate solution is produced is simple to describe: 1) random changes are made to the current best solution to produce a population of new solutions 2) these solutions are evaluated by the fitness function 3) the fittest solution or solutions are then mutated (and in some models, combined with each other) to produce the next population. This procedure is iterated hundreds or thousands of times until it has converged on a solution. It is a *general* feature of such algorithms that random variation is introduced into the population at each generation and the fittest solutions are selected to produce the next population.

This process forms a rough analogue of the biological scenario in which Darwinian evolution occurs in the biological realm, in which natural selection acts on random genetic variation to produce an incremental increase in the quality or fitness of the population of organisms as a whole.

We can explicitly describe the situation in the following way. We first assume each solution is described as a vector in $\mathbb{R}^N$. A population of M solutions is given by a sequence $S = (s_n), n \in \{1, \cdots, M\}$.

The population at time $t + 1$ is given by

$$S_{t+1} = (\widehat{s}_t, V(\widehat{s}_t), \cdots, V(\widehat{s}_t))$$

where $\widehat{s}_t \in S_t$ is the "best solution" at time t, i.e. the element of the population S_t such that

$$s \in S_t \implies f(s) \leq f(\widehat{s}_t)$$

and where $f : \mathbb{R}^N \to \mathbb{R}$ is the fitness function and $V : \mathbb{R}^N \to \mathbb{R}^N$ is a stochastic **mutation function** that introduces random changes or "mutations" to the vector, selected from some random distribution.

We easily obtain that $f(\widehat{s}(t))$ is monotonically increasing in t, and so the quality of the solution found by this iterative procedure will continually improve (or for a badly designed genetic algorithm, stay constant).

This is a rough description of how many genetic algorithms work. Actual implementations are usually more complicated and there are many subtleties involved that have not been mentioned here.

3 Design of *Floatworld*

3.1 Introduction

In Section 1.1 we reviewed the four most basic properties required of any living entity, which were:

- Metabolism
- Reproduction
- Adaptation
- Response to Stimuli

We now review the approach to satisfy each of these requirements within the *Floatworld* virtual enviroment.

Metabolism can be defined as the utilisation by an organism of an external supply of energy in order to maintain its structure and reproduce itself. In *Floatworld* , we will formalise the notion of energy: the environment, described in Section 3.2, will contain localised distributions of energy, as described in section 3.10, which virtual creatures can acquire in order to promote their survival. Virtual creatures will require energy in order to perform the basic actions of moving around within the virtual environment and reproducing themselves. The mechanism underlying this form of metabolism will be discussed in Section 3.6.

Reproduction is the ability of an organism to produce an accurate copy of itself. Individual creatures will be capable of reproduction. In the simple *Floatworld* model, all reproduction will be **asexual**, in that creatures will create offspring independantly of one another.

Adaptation is the ability of a lineage of organisms to become better suited to their environment as the generations go by. To ensure that the virtual creatures are capable of adaptation, we will specify that each creature has a fixed *genome*, a body of information that uniquely determines its behaviour in the virtual environment. This genome will be subject to random mutation upon reproduction of the creature, the resulting mutations affecting the genomes of the offspring of the parent creature.

For creatures to **respond to stimuli**, some form of information processing will be required. Creatures will have to use information from their local environment to decide on a course of action. The constructs we will use to achieve this will be feed-back neural networks, as described in Section 2.1. Therefore the *genome* mentioned above will consist of the weight matrix that completely specifies the action of a single ANN.

3.2 The Grid

The virtual environment or **grid** is divided into a finite number of cells arranged in a rectangular array of size $N_r \times N_c$. Formally we denote the grid by E, with elements, or **cells**, written as e_{ij}. Each cell e_{ij} has eight neighbouring cells, illustrated below:

$e_{(i-1)(j-1)}$	$e_{(i-1)j}$	$e_{(i-1)(j+1)}$
$e_{i(j-1)}$	e_{ij}	$e_{i(j+1)}$
$e_{(i+1)(j-1)}$	$e_{(i+1)j}$	$e_{(i+1)(j+1)}$

To avoid edge effects periodic boundary conditions are applied, so that for all i, j, $e_{0j} = e_{N_r j}$ and $e_{i0} = e_{iN_c}$.

Formally, each cell e_{ij} is a pair:

$$e_{ij} = (\xi, \epsilon)$$

where $\epsilon \in \mathbb{R}^+$, and $\xi \in K \cup \{0\}$. ϵ denotes the amount of energy that a cell contains, which is available for consumption by creatures. ξ denotes the creature currently occupying the cell. If there is no creature, $\xi = 0$.

As before, the elements of E are time-dependant, which will be indicated where necessary by $E(t)$ and $e_{ij}(t_k)$. We will often use E to mean merely the matrix of energies ϵ.

3.3 Creatures

The grid is populated by a number of virtual creatures. Each creature is under control of an ANN with a fixed number N of neurons as described in Section 3.13. The population of creatures at a time t is indicated by $K(t)$ or just K. Formally the state of each creature is specified by a (time-dependant) vector k:

$$k = (W, \alpha, \epsilon, \gamma, \rho, \delta)$$

- $W \in \mathbb{R}^{N^2}$ is the genome (or weight matrix) that determines the ANN
- $\gamma \in \mathbb{N}$ is the age in time-steps of the creature (with a maximum of 100)
- $\epsilon \in \mathbb{R}^+$ is the energy of the creature
- $\alpha \in \mathbb{R}^N$ is the activation vector of the ANN
- $\rho \in \mathbb{Z}_{N_r} \times \mathbb{Z}_{N_c}$ is the position of the creature on the grid.
- $\delta \in \mathbb{Z}_4$ is the orientation of the creature on the grid

For a particular creature $k \in K$, we denote its weight matrix by W_k, its age by γ_k, and so on. Creatures are limited to an age of 100, after which they automatically die.

Orientations represent which of the four cardinal directions the creature is facing, and are encoded as follows: if we visualize the grid as a matrix written in the normal fashion, $\delta = 0$ corresponds to up, $\delta = 1$ corresonds to right, $\delta = 2$ corresponds to down, and $\delta = 3$ corresponds to left.

3.4 Input

For a virtual creature to be able to behave in a way approrpriate to its local environment, the ANN controlling the creature must be provided with suitable inputs. What inputs are suitable? Here, a distinction is useful between **external inputs** which provide the ANN with access to information about the local environment and **internal inputs** which provide the creature with information about its current state.

Six external inputs are provided in the current model, of two seperate types. Three of the inputs give the ANN information about the local distribution of energy, the other three provide information about the local distrubition of other creatures. How shall local energy and creature distributions be reduced to single real numbers? To clarify the approach, we define a “kernel matrix” below.

Let M be a matrix of size $m \times n$. If K is a square matrix of size $2t + 1$ (where $t \in \mathbb{N}$), then we call K a **kernel matrix** with a position-dependant kernel product $K \cdot M[u, v]$, defined by

$$K \cdot M[u,v] = \sum_{i=-t}^{t} \sum_{j=-t}^{t} K_{(t+1)+i,(t+1)+j} M_{u+i,v+j}$$

where indices of M are consider modulo the size of the matrix, so $M_{i+m,j+n} = M_{i,j}$.

This idea can be used to naturally define a **vision function** for a creature: it is a weighted sum of the local environment of the creature, where the local environment is specified in matrix form. The weights used are contained in a kernel matrix. Figure 2 illustrates the three kernels K_1, K_2, K_3 used for this purpose in *Floatworld* .

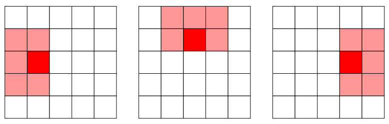

Figure 2: Three kernels used for vision functions. The more and less intense shades represent values of 2.0 and 0.5 respectively. All other entries are zero.

Assume k is a creature with $\delta_k = 0$ and $\rho_k = (u, v)$ (if the creature has a different orientation, the kernel matrices are rotated accordingly). We calculate the three vision functions e_i by taking the kernel product of each of these kernels with the energy matrix at the creature’s position:

$$e_i = K_i \cdot E[u,v] \qquad i \in \{1,2,3\}$$

Then e_1, e_2, e_3 summarise the local distribution of the energy in the left, forward, and right directions from the perspective of the creature. Note that the squares immediately to the left, in front of, and to the right of the creature have the largest contribution to these three summaries, with neighbouring elements playing a smaller role.

To accomplish the same task for the distribution of creatures, we use the same kernels as above but we replace the enery matrix E with another matrix termed the **occupancy matrix** O. O is defined to be

$$O_{ij} = \begin{cases} 1 & \exists k \in K, \text{s. t. } \rho_k = (i,j) \\ 0 & \text{otherwise} \end{cases}$$

In this fashion we obtain six vision functions $e_1, e_2, e_3, o_1, o_2, o_3$ which feed information about the local environment into the input neurons of the creature’s ANN.

3.5 Internal Inputs

The inputs to a creature's ANN are not limited to external factors. The simplest example illustrating such an internal input is the creature's energy: creatures must know when they have enough energy to reproduce, otherwise their attempt at reproduction might result in premature death (reproduction entails an energy cost of 50 units, so creatures must have at least 50 energy to survive reproduction).

It is also important to provide other inputs. To ensure that a creature might implement different strategies at different periods in its lifecycle, we provide the creature's age as another input to its ANN.

Two further internal inputs are provided: the first is a constant input of +1.0. This is important for technical reasons, namely that neurons can be suitably biased regardless of the other inputs. As an example, an ANN might implement a strategy in which the creature always moves forward, unless some other circumstance occurs. These default behaviour can be encoded as a positive weight between the constant input and the output neuron corresponding to the forward action.

Lastly, a random input is provided to allow for unpredictable strategies (to give the creatures "free will", so to speak)

The complete list of internal inputs is:

- creature energy
- creature age
- constant value of +1.0
- random real in range (-1.0, +1.0)

3.6 Actions

Creatures possess energy ϵ, which they must use in order to perform actions in the world. There are $n = |\mathbf{N}^{\uparrow}|$ actions, and each action $1 \le a \le n$ has associated with it an energy cost $\text{cost}(a)$. A creature that performs one of the n available actions has the associated energy cost subtracted from its total energy ϵ. If subtracting this cost from ϵ would result in $\epsilon \le 0$, the creature "dies": it is removed from the grid.

Which action a creature performs in a given timestep is determined soley by the activity of the output neurons in the creature's ANN. Let σ index the set of output neurons $\mathbf{N}^{\uparrow}$, so that the activity of the n output neurons are given by $\alpha_{\sigma(1)}, \alpha_{\sigma(2)}, \cdots, \alpha_{\sigma(n)}$. Then the selected action is the j'th action, where $1 \le j \le n$ is such that

$$\alpha_{\sigma(j)} \ge \left\{ \alpha_{\sigma(i)} \;\middle|\; 1 \le i \le n \right\} \cup \{1\}$$

Essentially, the action corresponding to the neuron with the highest activity is selected, with the condition that it be above a threshold of 1. If there is no such action, i.e. if $(\forall i)\ \alpha_{\sigma(i)} < 1$, then the creature performs no action. Note this still possesses an energy cost $\text{cost}(0)$, which is intended to ensure that creatures have a constant "metabolism" even when not performing any actions.

In the model that was implemented, a creature can execute one of the following four actions, with the associated costs

i	cost(i)	action name
0	0.1	no action
1	1.0	move forward
2	0.5	turn left
3	0.5	turn right
4	50	reproduce

These actions are described in more detail in the following section.

3.7 Movement

When a creature $k \in K$ moves, the cell in front of the creature is checked. Here, "in front" refers to the cell whose relative positon depends on the orientation of the creature and is, as usual, taken modulo the size of the grid. If it is occupied by another creature, nothing occurs. If it is empty, the creature position ρ_k is updated to equal the position of the cell in front of the creature, and the energy in that cell is added to the creature:

$$\begin{aligned} \epsilon_k &\leftarrow \epsilon_k + E_{\rho_k} \\ E_{\rho_k} &\leftarrow 0 \end{aligned}$$

3.8 Reproduction

When a creature $k \in K$ reproduces, the grid cell in front of the creature is checked. If it is occupied, nothing happens. If it is empty, a new creature k' is instantiated and placed at that location, with random orientation.

The new creature has $\epsilon_{k'} = 10$ initially, and $W_{k'} = M(W_k)$, where M is the **mutation function**. The mutation function is described in the next section.

3.9 Mutation

In biological systems, mutation happens randomly when errors in gene copying and recombination result in small changes to the sequence of As, Ts, Gs and Cs in the genetic code of the daughter organism. Mutation in the *Floatworld* model also occurs randomly. The "gene" equivalents of a *Floatworld* creature k are considered to be the weights in the ANN controlling a creature, i.e. the elements of W_k.

In *Floatworld* , there are two types of mutation, and each weight $w_{ij} \in W$ has small probabilities p_1, p_2 of undergoing mutation of each type. The majority of weights of a creature's ANN are zero, due to the **network mask** that is applied to limit the toplogy of the ANN. These weights remain zero for all time, and so only weights that have corresponding mask value of 1.0 are eligible for mutation.

For any given weight w in the weight matrix W, the corresponding weight w' in $M(W)$ is calculated as follows:

$$w' = \begin{cases} \text{Random}(w - m_1, w + m_1) & \text{with probability } p_1 \\ \text{Gauss}(w, m_2 \cdot w) & \text{with probability } p_2 \\ w & \text{otherwise} \end{cases}$$

The parameters m_1 and m_2 control two different types of mutation. The first type is a random change drawn from a uniform distribution that is not dependent on w, and the second type is a random change that is proportional to the current weight value. It has a gaussian distribution around the previous weight value, with standard deviation being some fraction m_2 of the previous value w.

The second factor ensures large weights undergo larger random fluctations via mutation than smaller weights; this is a sensible way to ensure that large weights do not effectively become static due to their modulus. The first factor ensures zero weights are still capable of becoming non-zero.

Experimention yielded sensible values of

$$p_1 = 0.05/N \qquad p_2 = 0.1/N \qquad m_2 = 0.1 \qquad m_1 = 2.0$$

3.10 Energy Input

In the *Floatworld* system, the total energy does not remain constant. Creature actions, such as movement and reproduction, remove energy from the system. To ensure that creatures can continue to survive and reproduce, energy must be continually injected into the system through a device known as a **feeding scheme**. To define feeding schemes we define their effect at each timestep on the energy matrix E. A simple feeding scheme defines a possibly time-dependant **feeding pattern** matrix $E'(t)$ of the same size as E. The feeding pattern is then added to the energy matrix E at each timestep:

$$E(t + 1) = E(t) + E'(t)$$

However, there is a significant problem with this technique. Experiments have shown that placing no limit on the magnitudes to which an element of E can reach results in highly unstable population sizes. This occurs for a simple reason. When the population is small, the total energy $E_{tot} = \sum_{i,j} E_{ij}$ builds up to very high values, resulting in a large increase in the number of creatures. These large populations then rapidly consume all the energy, which precipitates a large drop in the number of creatures when there is not enough energy to sustain them. The eventual result is an unstable population size that can sometimes lead to total extinction.

A simple solution is to modify the feeding scheme, using the rule

$$E_{ij}(t + 1) = \min\left\{E_{ij}(t) + E'_{ij}(t), e_{max}\right\}$$

where e_{max} is a sensible limit on the energy that any particular grid cell can contain. A value of $e_{max} = 20$ yields good results.

3.11 Feeding Patterns

The choice of what type of feeding pattern to use is an important one. The simplest and least interesting feeding scheme is constant, where $E' = c$ for some small positive constant c. However, such a feeding scheme results in only very simple strategies, for the obvious reason that creatures are not forced to evolve behaviour that is sensitive to the local energy distribution, as the local energy distribution is, on average, the same everywhere.

A more interesting feeding pattern, called **spot feeding**, gives better results. In spot feeding, there is a fixed number n of **spots** of fixed radius r that are located at grid positions (u_i, v_i) for $i \in \{1, \cdots, n\}$. The energy pattern is then defined by

$$E'_{jk} = e \times \left| \left\{ 1 \leq i \leq n \mid d((u_i, v_i), (j, k)) < r \right\} \right|$$

where $d : \mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{R}$ is the usual metric on the grid of size $N_r \times N_c$ with periodic boundary conditions, namely the space $(\mathbb{Z}/N_r) \times (\mathbb{Z}/N_c)$.

Thus E' is the matrix on which each cell within a radius r of a spot position has been incremented by e. Here e is chosen to be some small real number, for this project $e = 0.1$ was chosen. An example of the energy distribution resulting from such a feeding pattern is shown in Diagram 3:

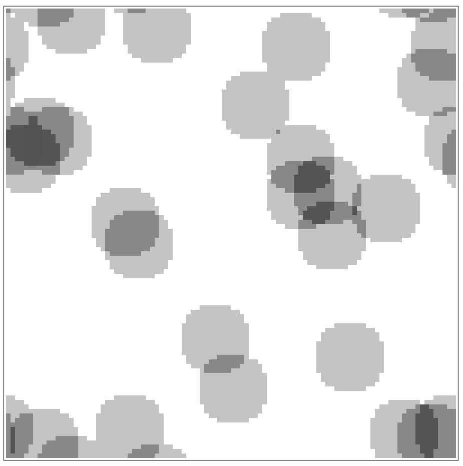

Figure 3: Diagram illustrating results of spot feeding on a grid of size 100×100. There are 25 spots of radius 8 and energy 0.1. Darker cells contain more energy.

An extension to this scheme is to specify a small **jump probability** p_j, the probability that at each timestep a given spot will be assigned a new, random position. This achieves good results because it ensures that creatures are forced to adapt to a gradually changing distribution of energy. A value of $p_j = 0.002$ was used for this project.

3.12 Stochastic Spot Feeding

Another interesting feeding scheme called **stochastic spot feeding** is a variation on spot feeding but does not explicitly use a feeding pattern. It also uses spots of fixed radius, energy, and randomly changing position, and on average injects the same amount of energy into E as does spot feeding. At each timestep, and for each spot of radius r at position (u, v), the stochastic spot feeding algorithm randomly chooses $\lfloor f \times \pi r^2 \rfloor$ cells of E within a distance r of (u, v) and increments each cell by an amount e/f (up to the maximum e_{max}). Another words, some randomly selected fraction f of the cells within each spot are fed. A value of $f = 0.02$ results in both good performance and interesting creature strategies.

Stochastic feeding results in an energy matrix E that has a "noisy" appearance. An example of the energy distribution resulting form stochastic spot feeding is shown in Figure 4, where the positions of the spots are the same as for Figure 3.

There are two major advantages to using stochastic spot feeding:

- computationally, it is significantly faster for large grids
- it produces more interesting creature strategies

Ordinary spot feeding results in energy spots that have a well defined boundary. In experimental runs it is found that creatures very rapidly learn to recognise and stay within this boundary. The noisy spots resulting from stochastic spot feeding do not have clearly defined boundaries, and creatures develop more robust behaviour to take advantage of the spatial distribution of energy.

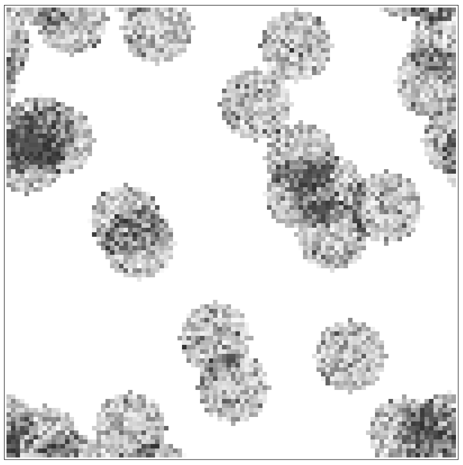

Figure 4: Diagram illustrating results of stochastic spot feeding on a grid of size 100×100. There are 25 spots of radius 8 and energy 0.1. Darker cells contain more energy.

3.13 The Creature "Brain"

In this section we give more information about the ANN and how it is is simulated.

For *Floatworld* the choice of activation function (see Equation 1) was the arctangent function $\text{atan} : \mathbb{R} \to (-1, 1)$, which is standard choice for many ANNs.

What choice of network topology is appropriate? A general principle with artificial neural networks is to choose the minimum possible number of neurons, and also the minimum possible number of non-zero weights. The reason for this is that evolution by natural selection can be understood to be a directed search (or optimization) in the space of all genomes towards "fitter" genomes. The larger the number of non-zero weights that can be adjusted, the higher the dimension of the genome space and the more difficult the optimization becomes. In Sections 3.4-3.6 we discussed the ANN inputs and outputs. We now discuss the network topolgoy as specified by the network mask.

There are two obvious constraints that we can begin with. Firstly, by definition, there should be no connections *to* inputs neurons, and there should be no connections *from* the output neurons.

Our second constraint is that we wish for the creature strategies to have some **persistant state**. That is, we wish for the ANNs to be able to "remember" past situations. In a purely feed-forward network, the behaviour of the output neurons depends only on the behaviour of the input neurons and so has no capacity for memory. We therefore require a feed-back ANN. We can then achieve a feed-back structure by specifying 3 additional neurons connected to both output and input neurons and allowing these 3 neurons to connect to themselves and each other.

Therefore our network mask has the following structure:

- input neurons connect to both the 3 "hidden" neurons and the output neurons
- the 3 "hidden" neurons connect to one another and the output neurons

There are 6 external inputs (consisting of two sets of three vision function inputs) and 4 internal inputs (consisting of energy, age, random and constant inputs). We have also specified 3 hidden layer neurons, and 4 output neurons (consisting of move forward, left, right, and reproduce outputs), so $N = 6 + 4 + 3 + 4 = 17$ neurons. We obtian a network mask of

$$M = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0
\end{pmatrix}$$

We see that M contains a non-zero sub-matrix of size 13×7. Any creature that evolves will only have non-zero weights within this sub-matrix, and so from this point on, any diagrams illustrating creature weight matrices will have these reduced dimensions. Moreover, matrices will be depicted such that positive weights are coloured red, negative weights blue, and zero weights are left blank. Less intense colours will represent smaller weight magnitudes.

We will often refer to the a creature's weight matrix and its genome interchangeably (they mean the same thing: the matrix W that is inherited by child creatures from parent creaturs). We will also refer interchangeably to the creature's genome and the strategy the creature persues, the terminology being that the genome **implements** a given strategy and that a strategy is **embodied** by a given genome.

3.14 The Initial Genome

For natural selection to apply in a simulation, an initial population of creatures is needed that has the capacity to survive and reproduce, even if does so using an inefficient strategy. In all the experiments conducted in this project, one extremely simple initial genome was used to seed this initial population.

The strategy that was used is summarised by the following simple algorithm:

- if $\epsilon > 120$, reproduce
- otherwise, move forward

To implement this strategy via an ANN, we use only three non-zero weights. Let us denote th constant input neuron and energy input neuron by `cons` and `energy` respectively. Also denote the output neurons corresponding to the move forward and reproduce actions as `move` and `rep`.

Then the only non-zero ANN weights are:

$$\begin{aligned} w_{\texttt{move,cons}} &= 1 \\ w_{\texttt{rep,cons}} &= -159 \\ w_{\texttt{rep,energy}} &= 1 \end{aligned}$$

A creature with the above ANN will have a default action of *move forward*, because the $\alpha_{\texttt{move}} \geq 1$. However, if $\epsilon > 120$ then $\alpha_{\texttt{rep}} = 1.0 \times w_{\texttt{cons,rep}} + \epsilon \times w_{\texttt{energy,rep}} = -159 + e > 1$, and so the *reproduce* action will override the *move forward* action and the creature will reproduce.

This strategy results in a population of creatures that move in straight lines and reproduce when they have acquired enough energy from the grid. Figure 5 illustrates an example of several creatures implementing this strategy.

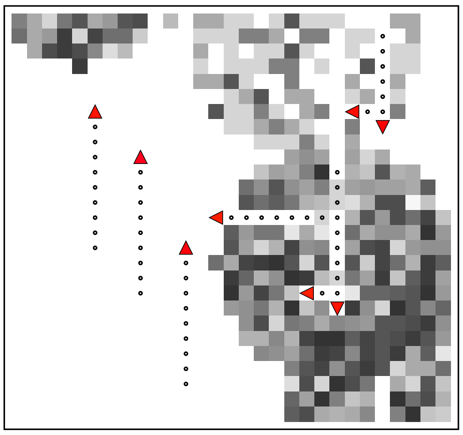

Figure 5: Example simulation illustrating the simple strategy implemented by the ANN described above. Previous creature positions are shown by simple trails. Three recently born creatures can be seen emerging from the trails of their parents.

4 Creature Behaviour

In this section we examine the strategies that result when natural selection acts on the initial genome. In Section 3.14 we saw how the initial genome produces a creature that moves in straight lines and reproduces when its energy exceeds a threshold. We now constrast this behaviour with that of an evolved creature. A single simulation was performed using 200,000 timesteps, and the resulting genome and its behaviour are discussed below.

A panel displaying a sequence of creatures from the evolutionary timeline leading up to the final creature is display in Figure 6. For more information about how this sequence was generated, see Section 6.1.

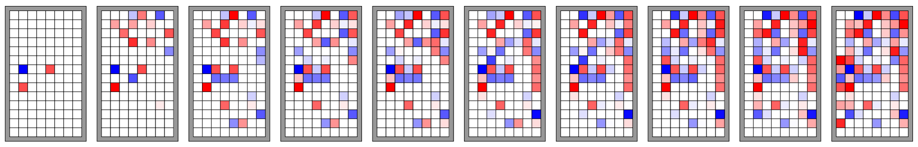

Figure 6: Panel displaying the progression of genomes over 200,000 timesteps of evolution. Positive weights are coloured red, negative weights blue, and zero weights are left blank. Less intense colours represent smaller weight magnitudes.

4.1 Colonisation

The speed at which a single creature can colonise a new feeding spot is an important factor in its fitness. The following diagram illustrates the ability of the evolved creature to rapidly colonise a feeding spot. Individual frames are taken at intervals of 20 timesteps.

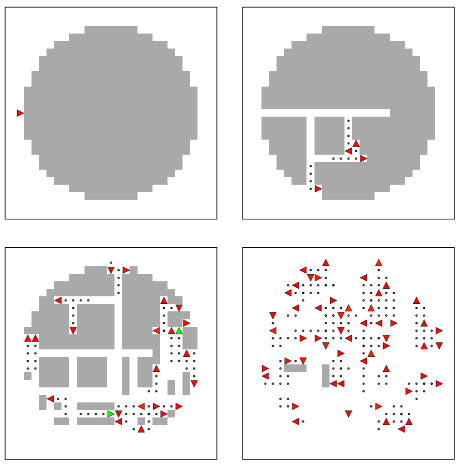

Figure 7: Illustration of colonisation of a new feeding spot. Each frame represents 20 timesteps of the simulation. Creatures in the process of reproduction are coloured red

4.2 Scavenging

Figure 8 illustrates the evolution of the ability to detect which direction a creature should turn to acquire the most energy. A creature is presented with two trials of "breadcrumbs" of energy which it follows accurately in both cases.

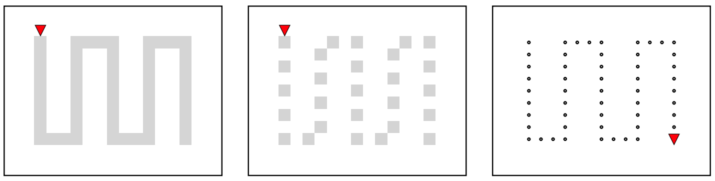

Figure 8: Illustration of scavenging behaviour of an evolved creature. Both initial energy distributions result in the same path-following behaviour.

4.3 Migration

Once the evolved creature has colonised a feeding spot it usually does not leave it. Occasionally a feeding spot will form at a location on the grid and subsequently jump, leaving a deposit of energy that is no longer renewed. One of these deposits can remain undiscovered for many hundreds of timesteps.

A curious behaviour occurs when it is finally discovered, however: after the deposit is consumed, the resulting creatures rapidly disperse away from the deposit. A timeline of this behaviour is displayed in Figure 9. It is possible that creatures have evolved to recognise such deposits as distinct from normal feeding spots and use them to "finance" rapid reproduction to aid expeditions to new territory.

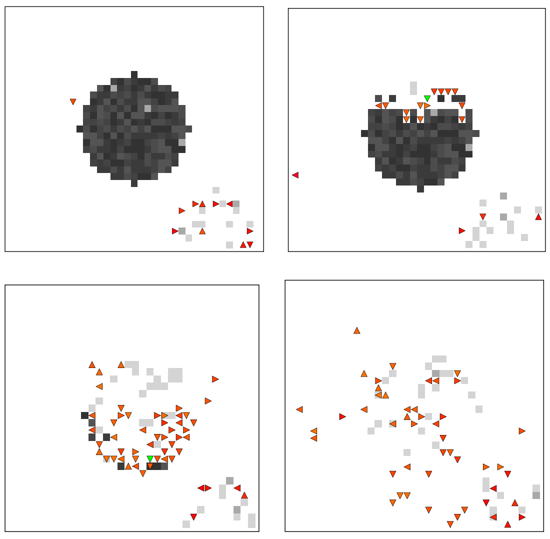

Figure 9: Rapid consumption followed by migration.

5 Species Coloring

5.1 Introduction

At any time t the population $K(t)$ is not homogenous: there is a variety of different strategies being employed by members of the population. How are we to analyze this variety?

For visual purposes it would be ideal to have a simple visual quality of each creature, such as colour, give an indication of the underlying weight matrix. To quantify this idea, we now define similarity of matrices and colours:

Let $d(W, W')$ denote the distance between the two matrices W, W' as given by the l_2 norm: $d(W, W') = ||W - W'||_2$. We define a colour (more precisely, a hue) as a real number c in the range $[0, 1]$. The range of colours in $[0, 1]$ is shown below in Diagram 10. Colour is represented cyclically in the usual fashion, so the colour $c \equiv 1 + c$. Therefore the **colour space** is mathematically represented as $\mathbb{R}/\mathbb{Z}$. Then distance between two colours is simply the normal metric in $\mathbb{R}/\mathbb{Z}$.

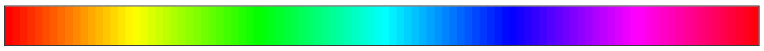

Figure 10: Color spectrum $[0, 1]$, the left end corresponding to 0 and the right end to 1.

Let $T = \{W_1, \cdots, W_n\}$ be a set of matrices of the same size. What we desire is a map $\varphi : T \to \mathbb{R}/\mathbb{Z}$ such that distances are preserved, i.e. such that φ is an isometry. Of course this is not in general possible, but we would like φ to be as "close" to an isometry as possible. How is this to be achieved? An important point is that the maximum distance between two points in $\mathbb{R}/\mathbb{Z}$ is $\frac{1}{2}$, and so we take care to normalize the metric on T so that $d(W_i, W_j) \leq \frac{1}{2}$.

5.2 Approximate Isometries

From an algorithmic perspective, we desire a process that, given a vector $x = (x_1, \cdots, x_n)$ which has defined on it a metric $d(x_i, x_j)$, produces output $y = (y_1, \cdots, y_n)$ with the distances $d(y_i, y_j) \approx d(x_i, x_j)$. To quantify the sense of this approximation, we introduce an energy function as follows:

$$E(y) = \sum_i \sum_j \frac{\big[d(y_i, y_j) - d(x_i, x_j)\big]^2}{2}$$

If we have a solution y that preserves distance, i.e. for which $d(x_i, x_j) = d(y_i, y_j)$, then it is obvious that $E(y) = 0$. Conversely, poor solutions will have large differences $d(x_i, x_j) - d(y_i, y_j)$ and so will have large energy $E(y)$. Imagining a solution as being the state vector of a classical system, we can use the Lagrangian E to obtain a force on each component of the vector x:

$$f_i = \partial_{y_i} E = \sum_{j \neq i} \Big(d(y_i, y_j) - d(x_i, x_j) \Big) d'(y_i, y_j)$$

In $\mathbb{R}/\mathbb{Z}$, the derivative of the metric with respect to the first argument is given by

$$d'(a,b) = \begin{cases} 0 & a \equiv b \\ 0 & a \equiv 1-b \\ -1 & 0 \in (a,b) \\ 1 & \text{otherwise} \end{cases}$$

The approach is then clear: a solution y which minimises $E(y)$ will represent a map that is locally as close to preserving distance as possible. To find such a local minima, we choose a random initial solution vector y and solve the equations of motion numerically. It is important that a friction term is included to ensure the $E(y(t)) \to E_{min}$ as $t \to \infty$. Essentially we are solving an energy minimisation problem using the method of gradient descent.

A code implementing this algorithm was tested on the following data set: the points of a regular polygon in $\mathbb{R}^2$ were calculated along with the corresponding metric $M_{ij} = d(x_i, x_j)$. A solution y was deemed to be correct if it approximated the angles of the corresponding points relative to the origin (up to shifting and reversal). Tests showed that for for all $n \leq 20$, 200 timesteps were suffient to find the correct solution. In other words, the algorithm is both reliable and computationally fast.

5.3 k-Means

Having solved the problem of representing genome difference via colours, we must now tackle a technical detail: the size of the population is often $|K| > 100$. The above algorithm works well on small sets of 5 or 10 elements, but for larger numbers it becomes impractical. A key fact is that the population K actually only contains some number $n << |K|$ of "essentially different" genomes. How is one to recognise when two genomes are essentially different? Without any context, this is an arbitrary question. The context is provided by the other members of the population. We wish to regard two creatures as belonging to the same species if their weight matrixes are similar relative to the variation in weight matrices throughout the whole population.

If we know how many species we want to identify in advance to be k, there is an algorithm known as **k-means** for partitioning a dataset into k clusters. It is an iterative procedure to identify the centres of any clustering that might be present in a high-dimensional data set. Assume our data set $\{x_1, \cdots, x_n\}$ is drawn from a vector space V. From V we choose k initial "centroids" —how they are chosen is unspecified and will be described later —which we denote $\{y_1, \cdots, y_k\}$.

We then assign to each x_i the y_j to which it is closest, which express as a map $f : \{1, \cdots, n\} \to \{1, \cdots, k\}$ with the property that

$$(\forall j \in 1, \ldots, k)\ d(x_i, y_{f(i)}) \leq d(x_i, y_j)$$

For each y_i we now calculate a new value by taking the centroid of all the x_j assigned to y_i:

$$y'_i = \frac{1}{|f^{-1}(i)|} \sum_{f(j)=i} x_j$$

Given suitable initial values and iterating the previous step multiple times, the values y_i converge quite quickly to the centroids of any clusters present in the set $\{x_1, \cdots, x_n\}$.

5.4 Initial Centroids

How do we choose initial values for $y_1, \cdots, y_k$? To evaluate different approaches, an artificial data set $\{x_1, \cdots, x_{100}\}$ was produced:

$$x_i = 10 \times (i \bmod 5) + \mathrm{RandomReal}(-1, 1)$$

By definition this data set is clustered into 5 small regions at values around $0, 10, 20, 30, 40$. The k-Means algorithm was applied to this data set, and the output after a small number of iterations was compared with the true cluster values. It was quickly discovered that random initial values were not suitable: often, two centroids would converge to the same cluster, and another cluster would be excluded. The following approach overcame this problem:

The first centroid y_0 is randomly selected from the data set. We then choose successive y_i's such that $\min \big\{ d(y_i, y_j) \bigm| j < i \big\}$ is maximized. In several thousand trial runs, this algorithm for choosing initial centroids always resulted in the correct partitioning of the data set into clusters.

5.5 Summary and Results

We summarise the k-species colouring problem: we wish to partition a population K of creatures into k species, and we wish to colour those species so that the subjective difference between colours reflects the objective difference between the underlying weight matrices. This is accomplished by performing the k-Means algorithm to partition the population into k species, and then solving an energy minimisation problem using gradient descent to find a good approximation to an isometry between the genome space and colour space.

Figures 11 and 12 show the results of applying the species colouring algorithm to a population after 12,000 timesteps. Figure 11 shows the spatial distribution of the different species on the grid, and Figure 12 shows the underlying weight matrices associated with each colour. Notice that similar colours have largely similar weight matrices. Also notice that in the first figure there is a correlation between spacial proximity and genetic proximity.

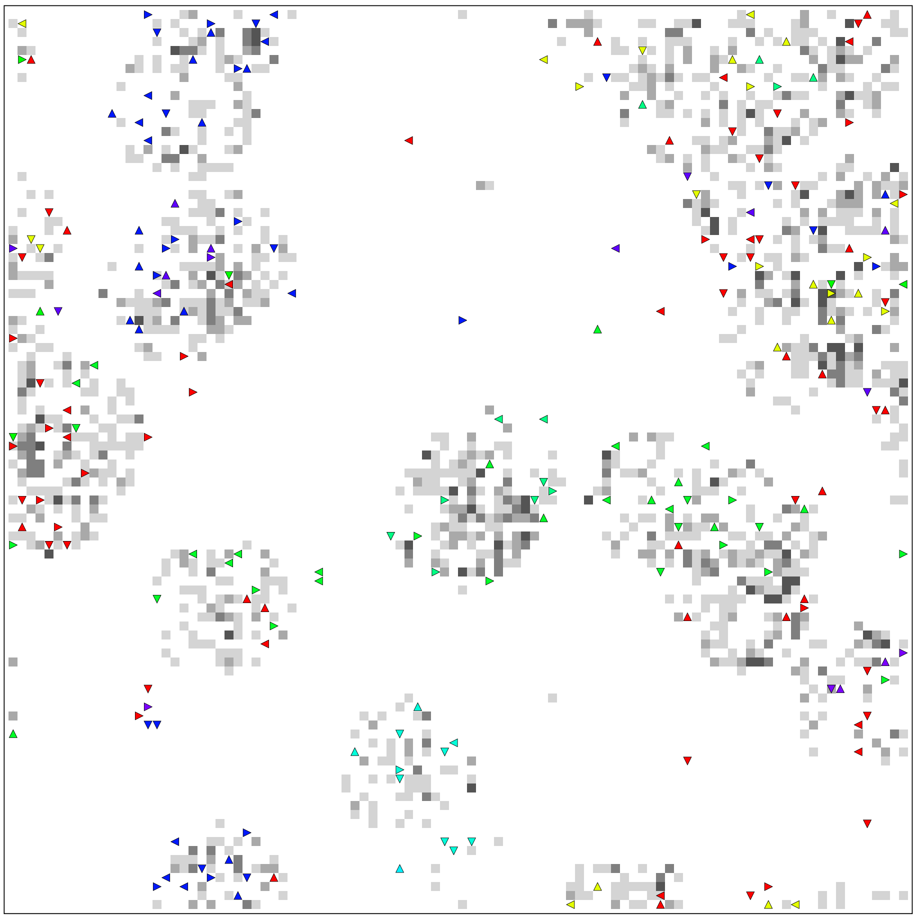
Figure 11: Grid of size 100 × 100 where creatures have been coloured with the species colouring algorithm ($k = 10$). Similar species tend to cluster in the same locations.

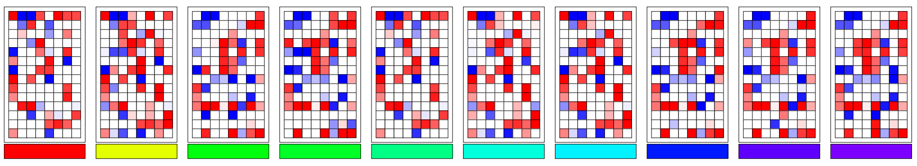
Figure 12: Panel presenting weight matrices representative of each species/colour. Notice that for similar colours, there are similar structures for the corresponding weight matrices.

6 Evaluating Fitness

How does one evaluate the extent to which natural selection results in the adaption of creatures to their environment? How does one even know that adaption has occured, rather than simple "genetic drift", that is, random variation in the genome —the neural network weights —with no consequence in behaviour? In this section we examine various techniques for quantifing adaption, or **fitness**.

Let us limit ourselves to comparing two seperate creatures, whose relative fitness is compared in a reproducible way. To do this, we evaluate the performance of an **ensemble** of different simulations in which the same two genomes are present. This approach leverages a key advantage that simulated systems such as *Floatworld* have over real biological systems: controlled, repeatable experiments can be performed using hundreds or even thousands of individual simulations under identical conditions. This is often impossible or impractical to achieve in the biological sciences.

By a **trial** we refer to the simulation of a fixed size grid for a fixed number of timesteps T, using for the initial population the genome described in Section 3.14. In this section it will be assumed that the population never goes extinct, i.e. that $K(t) \neq \{\}$ for all $t \leq T$. For a single trial we denote by $\mathbb{W}$ the set of all arising genomes:

$$\mathbb{W} = \big\{ W_k \bigm| k \in K(t), 1 \leq t \leq T \big\}$$

For all of the following examples we will compare creatures that have been evolving within a single simulation run. We will consider a sequence of genomes $(W_i)_{i \in \{1, \cdots, n\}}$, where $k_i \in K(t_i)$. We wish to evaluate the performance of this single **representative sequence**, which we hope will give us a representation of the evolution of the population as a whole.

The initial and naive approach is to let $t_i = T \cdot (i/n)$ and randomly select a creature $k_i \in K(t_i)$ to obtain each genome W_{k_i}. In other words, a creature is randomly selected from the population at n successive timesteps. This approach is flawed for two reasons: the random selection might choose a genetically unfit creature that is not representative of the population, and it might select creatures from a subset of the population that later becomes an evolutionary dead end —in familiar terms, we might select a representative of a species that later goes extinct. The representative sequence will therefore not be representative in the normal sense of the word.

How do we select a better sequence?

6.1 Creature Lineages

We first introduce some terminology. Let k, k' be a creature and one of its offspring. We write $W_{k'} > W_k$ if and only if a mutation has been introduced into k'. This partial order induces a tree structure on $\mathbb{W}$ called the **phylogenetic tree**, in which the root of the tree is the initial genome W_0, and the children of a node W_i represent all genomes derived from W_i via mutation. By a **lineage** of length k we simply mean a sequence of genomes that is an strictly ordered chain $W_0 < W_1 < \cdots < W_k$. In other words a lineage is the analogous structure to a "family tree" of ordinary sexually-reproducing creatures.

A creature lineage provides a much better way of selecting a representative sequence, for the reason that a lineage, by definition, represents successive stages of the evolution of a single species. The *Floatworld* code can easily be modified to keep track of such lineages. By choosing an arbitrary creature at the end of a trial and recalling its lineage, we can obtain a "good" representative sequence.

Often, however, the length k of the lineage is too long. If we wish to obtain a sub-sequence of length $n < k$, there are at least two potential methods. The first, called **equal time spacing**, is to let $t_i = T \cdot (i/(n-1))$ for $i = 0, \cdots, n-1$ and select each genome W_i to be the genome in the lineage that was current at timestep t_i. The second, called **equal mutation spacing**, simply selects $W'_i = W_{\lceil k \times i/(n-1) \rceil}$ for $i = 0, \cdots, n-1$, so that it takes genomes at equal intervals from the original lineage.

We will use equal mutation spacing because, unlike equal time spacing, it prevents periods of rapid evolution from being "lost" within a single time-interval.

6.2 2-way Competition

If we have two seperate genomes, how are we to evaluate their fitness in a comparative fashion? A simple method is to involve them in direct competition. To do this, we 'seed' a grid with 50 copies of each genome. We then evaluate the number of each type of creatures after 200 timesteps, during which mutation is disabled. A genome representing a superior strategy will reproduce itself at the expense of the other genome, resulting in a larger number N_a of the former than N_b of the latter, after 200 timesteps.

However, we would prefer a single real number in range $[-1, 1]$ to summarise the results, with -1 indicating the first genome is highly inferior (when $N_a \approx 0$) and 1 that the first genome is highly superior ($N_b \approx 0$). Intermediate values represent less significant results, with 0 indicating no discernable difference, i.e. when $N_a \approx N_b$. Firstly, this real number should be anti-symmetric, because exchanging N_a with N_b should give the opposite conclusion. Also, absolute numbers of creatures depend on energy density and other arbitrary factors, so it makes sense to consider the ratio N_a/N_b.

We now try to identify a function f such that $f(N_a/N_b)$ has the desired properties. It is clear that f should be symmetric under inversion, because if we exchange a and b we want $f(N_b/N_a) = -f(N_a/N_b)$. One simple continuous function with this property is $f = \ln$. However we still obtain $\ln(N_a/N_b) = -\infty$ when $N_a = 0$ and $\ln(N_a/N_b) = +\infty$ when $N_b = 0$. To map the range $[-\infty, \infty]$ to $[1, -1]$ we use arctan. Then all the desired properties are satisfied by the following function:

$$f = \frac{\arctan(\ln N_a/N_b)}{\pi}$$

To get an accurate measure of the relative fitness of the two creatures we repeat the measurement of f on an ensemble of 100 simulations, and take the average result. This we call the **pair-wise finess** of two genomes W_a, W_b, written $f_{\text{pair}}(W_a, W_b)$.

6.3 Competition Matrix

We can summarise their pairwise fitness of n genomes using a **competition matrix** M, whose elements are $M_{ij} = f_{\text{pair}}(W_i, W_j)$. This matrix gives us a convenient summary of the results of competing every single strategy in a representative sequence with every other strategy.

Figure 13 illustrates the competition matrix for a creature lineage obtained over $50,000$ timesteps with $n = 20$ using equal mutation spacing. The antisymmetric nature of the measure f_{pair} is immediately clear. Figure 14 illustrates the genomes contained in the lineage for this trial.

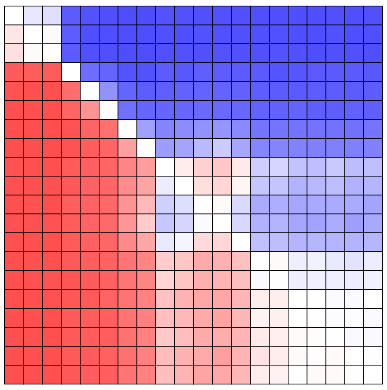

Figure 13: Example of a competition matrix when $T = 100,000$ and $n = 20$.

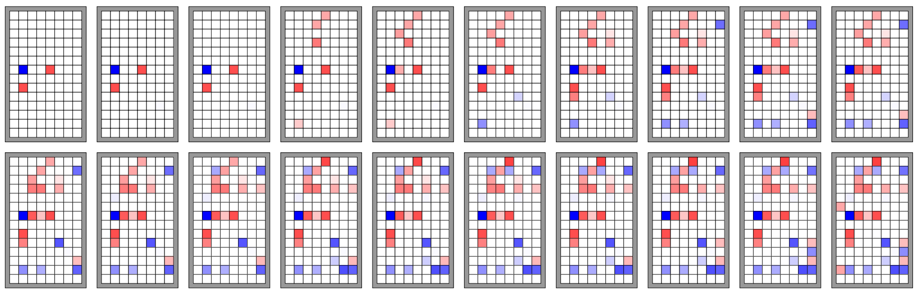

Figure 14: Individual genomes of lineage from trial of length 50k with the first genome on the top left and last on the bottom right.

To give an example of how to interpret such a matrix, take the Figure 13 above. If we look at the first row of the matrix, we see that every non-diagonal element is intense blue. This indicates that the pair-wise fitness of the first genome with every successive genome is close to -1. In other words, the first genome is highly inferior to every succesive genome. This pattern repeats itself with lower rows, in which each creature is, roughly speaking, superior to earlier creatures and inferior to later creatures.

The pattern breaks down by the 9th genome. It is discernibly superior to the 11th and 12th genomes. Apart from this small counterexample, we do not see many early genomes that are superior to later genomes, which agrees with our intuitive understanding of natural selection as resulting in creatures that are always better adapted. Evolution almost never "runs backwards", so to speak. Another feature of this matrix is that improvement ceases by about the 16th genome. This limit in the increase in fitness suggests that, in the language of optimization theory, a local optimum in genome space has been approached by this population after 50,000 timesteps.

6.4 Single-Genome Measures

The competition matrix, as described above, does not itself provide an explanation of the strategies embodied by the ANNs. To understand *how* the particular strategy implemented by one genome is better than the strategy implemented by another genome, we must examine the details of the strategies embodied by both genomes.

Unlike algorithms explained in human terms, it is extremely difficult to understand simply from the ANN weights alone what strategy a particular genome embodies. It is only in the interaction between the genome and the environment that a particular stategy emerges. The next question to ask is what features of a strategy can be measured, as opposed to just subjectively described.

If we desire to discover *objective* measurements of different components of a given strategy, we must once again turn to creature ensembles and their statistical behaviour. One approach would be to break down the overall strategy of a genome into various component parts, and evaluate these directly using ensembles to produce a small set of numbers that summarise these results.

How this is to be done depends sensitively on the environment. With the stochastic and ordinary feeding spot schemes detailed in Section 3.11, their are a few obvious patterns to creature behaviour. The first is how well a creature makes use of the energy resources within one spot. The second is how well a creature discovers new spots, and also how well a creature deals with the situation in which the spot it is occupying "jumps" to a new position.

In the following sections we explore these and other objective features of a single genome. We will use 5 different trials (each depicted in a different colour) for each graph in order to get a representative sample of possible results. The representative sequence for each trial was obtained by sampling the lineage of a genome obtained from 50,000 timesteps at 20 equally spaced mutations.

6.4.1 Population Capacity

We can regard the **population capacity**, the long-term average population size, to be an indirect measurement of the efficiency at which individual creatures use energy to survive and reproduce. A more energy-efficient strategy will result in a larger average population, all other factors being constant, because more energy can be expended by members of the population in order to reproduce.

We can evaluate the average population size by seeding a simulation with a large number of creatures using the relevant genome. We disable mutation, and continue the simulation for 500 timesteps (=5 creature lifespans) in order for the population size to stabilize, and then record the average population size over the next 1000 timesteps. Repeating this process 100 times we obtain an measure of the population size with low standard deviation, measured on several genomes to be $< 5\%$.

Figure 15 shows a graph of average population size for all genomes in the 100k19 lineage, the leftmost being the first genome. It is easy to see a trend from the initial capacity of about 220 creatures to a final value in the range 350-400.

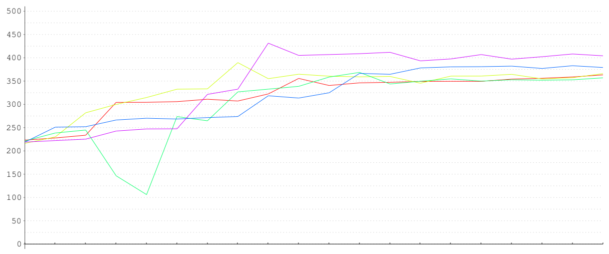

Figure 15: Average population size for the 100k19 lineage.

6.4.2 Birth Rate

Another measure of how competitive a strategy is the number of offspring a creature embodying the strategy will produce in its lifetime. To negate the competition created by the presence of other creatures, we seed a simulation with only one creature and prevent the creature from reproducing. We do however count the number of times the **reproduce** action occurs. The simulation continues for the entire lifespan of the creature.

Repeating this simulation 100 times and taking the average we obtain the an estimate of the **birth rate** —the average number of reproductions a creature will perform in its lifespan, competition being non-existant.

Figure 16 shows a graph of average birth rate size for 5 seperate trials of length 50k timesteps. The birth rate fluctuates strongly although there is a marked general trend towards high birth rates as evolution progresses.

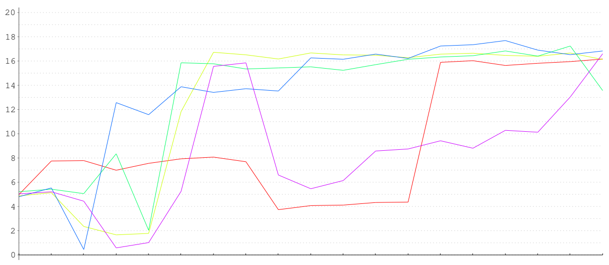

Figure 16: Ensemble birth rate for an individual creature with no competition for 5 seperate trials of 50k timesteps each.

6.4.3 Migration Rate

The final measure discussed here is that of migration. Migration is the tendency of creatures to leave the feeding spot they are currently occupying in search of a new, potentially unoccupied feeding spot. We define this to be the average rate at which creatures leave a feeding spot, measured in creatures per time step.

In all trials only a single feeding spot is used, with radius 12 and energy density 0.2. As with previous measures, mutation is disabled. Moreover we disable creature birth and death, opting instead to "respawn" a creature within the feeding spot when it dies. This prevents the possibility that the population will fluctuate excessively or go extinct.

The number of creatures that leave a disc of radius 12 is counted and averaged over 1000 timesteps. The entire process is repeated 20 times to derive an average migration rate. Figure 17 shows a graph of average migration rate for 5 seperate trials of length 50k timesteps. A small trend towards greater migration rates is evident.

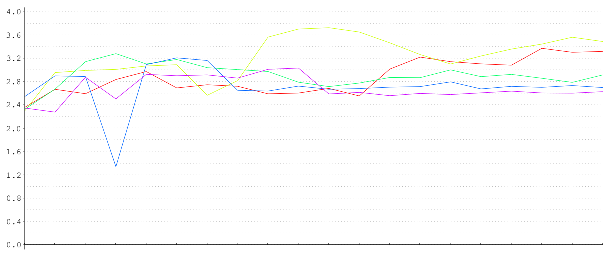

Figure 17: Ensemble migration rate for a population within one feeding spot for 5 seperate trials of 50k timesteps each.

6.5 Conclusion

A variety of methods for evaluating creature fitness have been presented, including two-way competititon, competition matrices, and ensemble measures such as population capacity, birth rate, and migration rate. All of these measures show a general increase in creature efficiency that goes some way to explaining the pair-wise fitness improvements that can be seen in the competition matrix in Figure 13. Potentially many more measures could be usefully defined, but these measures should depend on the details of the strategy one wants to understand.

7 Implementation

In this section various technical details of the implementation of *Floatworld* are discussed.

7.1 Class System

C++ was chosen as the language of implementation. C++ is a robust, object-oriented language with a reputation for high-performance.

`Creat` is the class that embodies all the information specific to one creature, such as age and energy. It sets up internal inputs, calculates the activity of the creature's ANN, and determines which action is necessary. Lastly it keeps track of lineages, including deleting extinct sub-trees of the phylogenetic tree in order to free up memory.

`Matrix` handles low-level matrix operations, including memory management, multiplication of the weight and state vectors of individual creatures, file input/ouput of individual matrices and collections of matrices, and display of these matrices in a format compatible with LaTeX.

A class called `PrimitiveContext` encapsulates all the programming routines that are necessary to draw the grid, individual creatures, and various graphs. The `PrimitiveContext` class is sub-classed by both `AllegroContext` and `EPSContext`, which provide interfaces to the Allegro graphics library and the Encapsulated Post Script vector graphics file format. This allows the same routines to either draw creatures, matrices, grids and graphs to the screen or to a file that can be embedded in LaTeX. It is this object oriented design that made the diagrams in this project possible.

7.2 Performance

Two avenues of exploration become more feasible with an efficient code, and those are:

- long timescale trials of over 100,000 timesteps
- efficient evaluation of the statistics of large ensembles of simulations

Much effort was spent on optimizing the *Floatworld* code. A technique known as profiling was used to identify which parts of the program were preventing high speed evaluation and these parts were aggressively optimized. Initially, 80-90% of the running time of a simulation of 10,000 timesteps was spent on the matrix multiplication of weight matrices with activation vectors. By exploiting the structure of the weight mask and performing more technical optimizations this figure was reduced to 10-20% of the running time.

Another major bottleneck was the calculation of vision functions for each creature. In the original form a kernel matrix was provided and was explicitly calculated to determine the vision function. Due to the large numbers of '0s' in the kernel matrix, this technique was very slow and accounted for 60% of the running time of the program after the previous optimization. This bottleneck was removed by "hard-coding" the structure of the kernel matrix into the vision functions, removing the necessity of calculating the kernel product and reducing the number of memory lookups. Vision functions now account for $\pm 15\%$ of the running time of the program.

The last major bottleneck was the form of the feeding scheme. The original spot feeder required every element of the matrices E and E' to be both written and read every timestep. Two schemes were attempted to reduce this. The first was to encode the feeding pattern E' using run length encoding[2]. This meant that the zero elements of the feeding matrix E' could be ignored, which decreased the number of memory accesses and therefore sped up the code. Run length encoding entailed a significant cost, however, because one had to re-encode E' every time a spot jumped. The overall speed-up from run length encoding was $\pm 10\%$.

A more successful strategy for reducing the compational burden of the feeding scheme was to employ stochastic spot feeding as described in Section 3.12. The first advantage this provided was to avoid calculating the distance from each cell (u, v) of E to each spot location to determine if the cell was inside a spot. Instead, each spot calculated a random fraction f of the cells within it and updated their energy. In other words, if the grid is square and has size n and contains k spots, the computational complexity of spot feeding went from $O(kn^2)$ to $O(fk)$. Secondly, avoiding the use of a pattern matrix removed n^2 memory accesses.

7.3 Performance Measures

We can objectively measure performance by testing how many timesteps the code can perform per second. This is not a good measure, however, because updating a small population involves less computational effort that updating a large population. In fact, the updating of individual creatures comprises the majority of computational work involved in simulating a grid. To take account of this, we employ the number of **creature-steps per second**. Each creature requires one creature-step per timestep.

The current incarnation of *Floatworld* can execute $\pm 600,000$ creature-steps per second on an Athlon 64 2GHz processor with 1 GByte of RAM. This constrasts with the original figure from before optimization was performed of 10,000 creature-steps per second. As an example of how to use this figure, we consider a 100×100 grid with 20 feeding spots, for which a stable population of around 400 creatures is normal. Therefore we can expect to simulate around $600,000/400 = 1500$ timesteps per second under these conditions.

[2]Run length encoding considers a matrix to be a string of numbers by concatenating successive rows. This string is then compressed by removing "runs" of the same number, such as (5,5,5,5,5,5), and replacing them with a simple count, in this case (6,5).

8 Future Work

8.1 Introduction

This project has been about the *Floatworld* framework. It is not in itself a research project, it has been about the construction of a framework for performing research in artificial life, including tools that can be used to evaluate and understand the evolution of artifical organisms within the framework.

The computer code making up *Floatworld* is written in modern object-oriented style. Modifications are easy to make and, where possible, use virtual member functions to achieve easy extensibility. An example is the `Feeder` class, which encapsulates a basic feeding scheme that may or may not use a feeding pattern (both are supported). In addition it supports optional run length encoding and caching to improve performance.

The spot feeding schemes that have been used in this project are simple classes derived from the `Feeder` class, but other schemes would be easy to implement, including some that are detailed below.

8.2 Creature Interactions

The ability for creatures to interact with one another can be introduced to *Floatworld* quite simply. A general principle of neural networks is to make the number of input and output neurons as small as possible. In order not to need an extra output neuron to control the ability for one creature to interact with another, we need only make the simple limitation that two creatures must be adjacent on the grid for interaction to occur. As only one creature can occupy a grid cell at a time, a sensible method to enable interaction is to specify that when one creature attempts to move onto the grid cell occupied by another, an interaction occurs.

While the scope for creature interaction is large, only three concrete possibilites are suggested here:

- Predation
- Sexual Reproduction
- Cooperation

We briefly discuss these possibilites.

Predation refers to the situation in which one creature can "attack" another for its own gain. For such attacks to have any meaning, they must involve the transfer of energy. More specifically, let k, k' denote the attacking creature and its victim. Then the predation rule could be of the form

$$\epsilon_k \leftarrow \epsilon_k - \alpha_1$$

$$\epsilon_{k'} \leftarrow \epsilon_{k'} + \alpha_2$$

where α_1, α_2 are positive constants with $\alpha_1 > \alpha_2$ (otherwise mutually attacking creatures would obtain a net increase in energy).

Sexual reproduction occurs when the genomes of two creatures are combined in some way in their offspring, instead of the asexual reproduction that occurs normally in *Floatworld* . A small complication is that there is no obvious grid position in which to instantiate the potential offspring of two creatures, as they are already adjacent.

One potential solution to this problem, which also happens to solve the question of how both creatures can mutually decide to reproduce, is to merely allow interaction to “select” a mate. Any subsequent reproduction will produce an offspring with a genome that is a combination of both parent genomes. As an example of how this might occur, let W, W' be the two parent genomes. Then the elements of the child genome W^* are:

$$W^*_{ij} = \begin{cases} W_{ij} & \text{with probability } 0.5 \\ W'_{ij} & \text{otherwise} \end{cases}$$

Co-operation occurs when two creatures cooperate for their mutual gain. In the context of *Floatworld* , this could take the form of a small energy reward when two creatures choose to cooperate. A potential assymetric method could be the following: when creature k chooses to cooperate with creature k',

$$\epsilon_k \leftarrow \epsilon_k - \beta_1$$

$$\epsilon'_k \leftarrow \epsilon'_k + \beta_2$$

where β_1, β_2 are small positive constants with $\beta_2 > \beta_1$. In fact, this does not result in mutual gain for both creatures: the first creature is actually harmed. However, it is implicitly enables a form of cooperation because, if both creatures choose to cooperate with each other, they will both experience a net gain in energy.

A simple modification to *Floatworld* was made to enable the above type of co-operation with values of $\beta_1 = 2, \beta_2 = 12$. Within 40,000 timesteps a simulation run yielded a series of small clusters, one of which is shown in Figure 18. These clusters were highly unstable, growing in size over a period of a few hundred timesteps before rapidly disintegrating. This has a simple explanation: every cluster easily falls prey to “cheaters” to neglect to cooperate and obtain energy at the expense of other creatures. Eventually such cheating destroys the community of cooperators.

8.3 Hebbian Learning

A possible extension to *Floatworld* is to introduce ANNs that change over time. A standard technique known as **Hebbian learning** is appropriate. In Hebbian learning, the connection strength (the weight) between two neurons is adjusted positive whenever their activity is correlated and adjusted negative whenever their activity is anti-correlated.

A possible implementation is to store a **time-averaged** activity vector α' for each creature that represents the average activity of the creature’s ANN over the last several timesteps. Each timestep this average activity is used to update weights according to the Hebbian rule.

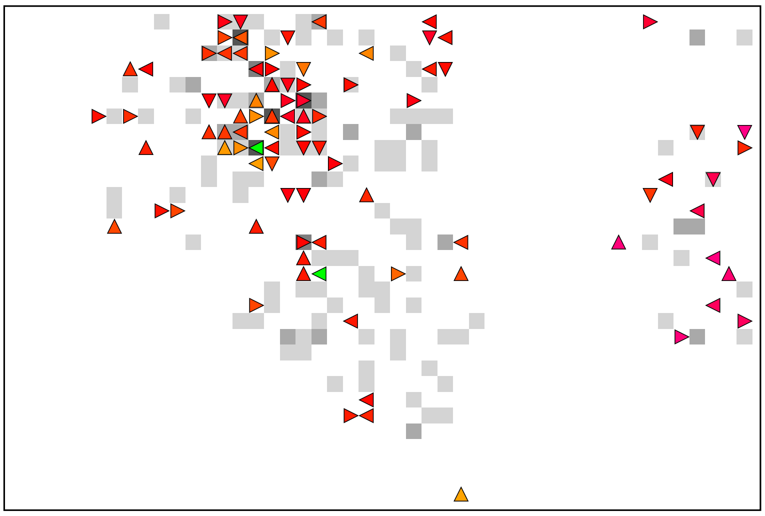

Figure 18: Illustration of clustering behaviour when cooperative interaction is enabled.

Creatures using Hebbian learning could adjust their strategies during their lifetimes. Instead of specifying fixed weights for the ANN in each creature genome, a genome could consist of “learning rates” for each weight that specify how rapidly it should be updated. A genome would thus bias certain types of learning over others.

8.4 Complex Environments

Another potential extension to *Floatworld* is to implement more complex environments. Two ideas are suggested here: environments that consist of a more complex feeding scheme, and environments that are enriched with extra information for creatures to exploit.

As an example of the first idea, consider an environment in which the feeding spots consist of different radii and energy densities. Creatures would have an incentive to discover the best possible feeding spot. This type of feeding scheme might also result in specialisation in which more than several co-existing species arise that exploit different types of energy source.

Another possibility is if there is a type of “day/night” cycle present in the feeding scheme, so that energy injection ceases at regular intervals. A natural question would be whether creatures could adapt to have less energy expenditure during periods of no energy injection —an analogue to the hibernation of certain mammals.

An example of the second idea would be if an extra input to the neural networks is introduced that specifies additional information. This information could be the distance to the closest feeding spot, or other information pertaining to the feeding scheme.

8.5 Application of Supercomputers

One interesting possible area of research is how *Floatworld* can be extended to exploit the vastly increased processing ability of a supercomputer cluster, such as the BlueGene/P that has recently been installed in Cape Town's Center for High Performance Computing. There are two possible ways in which this could be done:

- code to calculate ensemble averages could easily be parallelised to run simultaneously on many machines. This would enable much more accurate estimates, or alternatively much more computationally challenging measures to be made feasible.
- extremely large grids, of size in excess of 1000×1000, could be efficiently simulated by partitioning the grids into subgrids that are each handled by an indiviual machine

8.6 Predicting Pairwise Fitness

In this project, no detailed attempt was made to understand or reverse engineer any of the strategies that were employed by evolved creatures. The three measures introduced in Section 6.4 go some way to representing different aspects of creature strategy, but much more can be done.

An interesting approach would be to derive a large number of single-genome and pair-wise measures of fitness, and to employ a machine learning algorithm in order to predict the pair-wise fitness $f_{\texttt{pair}}$ of a large number of different genomes from these measures. The advantage of this idea is that the machine learning algorithm could detect patterns that are too difficult to perceive directly. It could, for example, determine that some measures are much more important than others in estimating how creatures will behave in a competitive environment. Such information could aid the attempt to understand creature strategies in detail.

8.7 Releasing *Floatworld* as Open Source

Open Source software is free for anyone to view or modify. Releasing software as Open Source often encourages other programmers and academics to get involved in the project. Releasing *Floatworld* as Open Source software could result in a community of users who use and improve it.

9 Conclusion

In this project we have performed a quick overview of the topics of genetic algorithms and artificial neural networks. We have reviewed the *Floatworld* virtual environment in which digital organisms compete for resources and space, and reproduce themselves. We have seen evidence that efficient strategies emerge via natural selection even when starting from an exceedingly simple initial strategy.

We have examined tools for classifying creatures into various species and evaluating their genetic similarity. We have also defined measures and algorithms to compare two species to guage their relative fitness, and have examined these measures in the context of single-genome measures that suggest that more evolved creatures implement more efficient strategies.

Lastly we have summarised some of the technical aspects of the *Floatworld* code. We have also suggested avenues for future work. In conclusion *Floatworld* presents an interesting framework for studying simulated evolution and potentially a powerful tool for new research in the field of Artificial Life.