Mid Sweden University
Institute for Informatics (ITK)
Joel Palmius

2000-02-07

The Design of Self-Organisation

Formal Examiner: Viveca Asproth
Scientific Advisor: Anita Håkansson

ABSTRACT

As computing capacity continues to grow, the complexity of the systems to design does the same. In the end this increased complexity will set a limit to what systems can be developed, since the new systems will not be intellectually graspable. It is possible that self-organisation could be used to address this problem. By having done an implementation of evolution and artificial life as self-organisation, a few general principles for self-organisation, and their consequences for design, have been formulated. These principles describe staticity traps, directionality, the force of increasing complexity in evolutionary systems, and requisite intervention.

Keywords: Self-organisation, Artificial Life, Complexity, Evolution

1. Background

As computing capacity has grown, the applications and uses of computers has exploded. From simple machines that were intellectually graspable into their smallest detail, today's machines has grown to be so complex that it is only through advanced tools we are able to handle them. It is sometimes said that we can do whatever we want with our computers - *only our imagination sets the limit*.

This situation is usually described as something positive. There's that key phrase though, the idea that our imagination, or in practise our minds, set the limit to what can be done. This is something that could be considered obvious, the fact that the mind is limited and cannot grasp infinite complexity (G. A. Miller, 2000). I find that there is a problem though. Programs need programmers and computers need constructing engineers. If the computers and the software grow so complex that it is not possible to understand them anymore, will not the development be significantly impaired?

In part, this problem has been addressed by building 4GL[1] tools, but still, there is human control over the tiniest details, a conscious design from the very top to the very bottom. I believe that this approach will eventually reach the same limit, eventually problems and computerised systems will grow so complex that it will be hard to design the tools used to design the systems.

One solution as I see it, is to let go of most of the control. The idea I have is that it would be possible to formulate principles for self-organisation in general systems. I think this is valid both for computer systems/programs (which, I think, would currently benefit the most from this) and other real-world systems (such as social and biological systems. "Real-world" as an opposite of virtual).

It is my firm belief that this kind of organising principles could be used in many chaotic situations. It is also my belief that this way of organising things could produce pretty interesting results such as complex behaviour (as we have seen in the case of natural evolution).

[1] "Fourth Generation Language". Apart from the definitions in section 4, a short dictionary for acronyms and terms can be found in Appendix E. Readers unfamiliar with the area should probably take the time to browse through the dictionary and the definitions before reading on.

2. Problem Definition and Purpose

As indicated in the previous section, the problem is how to handle growing complexity, to find ways to cope with complexity in an efficient and reasonably economic manner, preferably in such a way that the complexity itself is something that should not have to be tinkered with. I think self-organisation might be such a way.

Clearly, it would be a good example of hubris to try to address the whole area at once in a work of this size. What I can hope for is to help crystallising some general principles and their consequences for design out of the big primordial soup of general systems and self-organisation. This hopefully in such a way that they might be usable.

To summarise, the purpose of this work is *to formulate some general principles for self-organisation and to point at their consequences for design of self-organising (in this text computer based virtual) systems*.

I believe this is interesting and valid for everyone handling complex systems. In the first stage, computerised systems development will benefit the most, but in some distant future, I believe that well-formulated principles for self-organisation could be usable in everything from biology to mechanics[2]. In the same distant future these principles might be collected into a model or method for the implementation of self-organisation in general systems.

The intended target group for this work is researchers, practitioners and students within areas such as AI, AE, AL, GST, Living Systems and Informatics. I shall suppose some superficial acquaintance with theories and models such as Autopoiesis and Artificial Life. This is probably a limitation in readability with most categories of possible readers.

[2] This if a common language or model for self-organisation could be developed. I am especially fond of the idea that state space vectors could be common for all systems, see discussion in section 7.3.

3. METHOD

My initial plan on how to find these general principles, was to study several different examples of self-organisation within a wide range of areas. These, I planned, should include areas such as quantum physics, embryonal development, evolutionary biology, control systems and robotics, none of which I have even been idly tinkering with before. From there I would draw conclusions about general principles for self-organisation, and try to implement these. After discussions with various instances[3] I felt informed that this would A) take about 20 years, and B) would be way outside what is considered an appropriate level of work for a one-semester master thesis. In short, this approach was stupid.

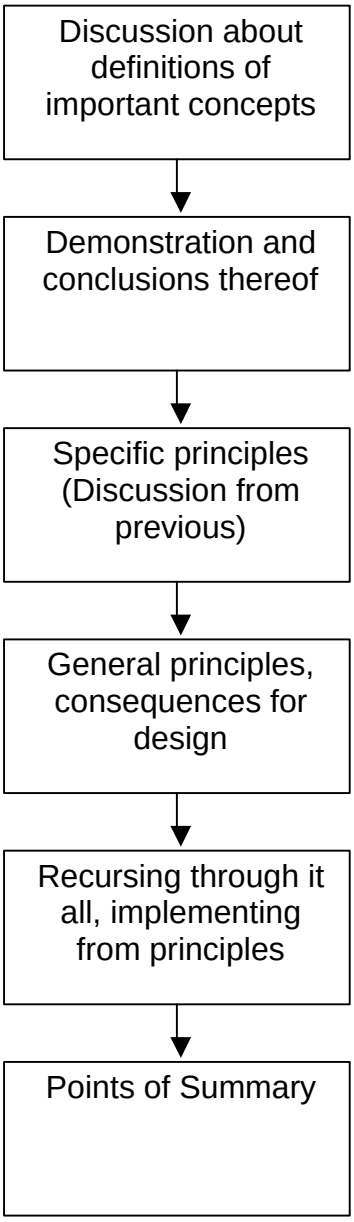

Figure 1: Schematic view of the method. In reality, there has been iterations and parallel working, but as a way of conceptualising the method, this might serve.

Instead I have done the diametrically opposite: I have started with an implementation and from there worked towards general principles. This bottom-up approach has some obvious limitations concerning generality, something I hope might be compensated by its higher transparency. A discussion about the bias this approach brings can be found in section 9.1.1

3.1 Discussions of Definitions

I have begun the work with defining some of the most important concepts used in the text. This is not simply a statement for the sake of clarity, but also a discussion about the meanings and implications of the concepts, and their impact of this work. As such, this discussion serves as a necessary building of a theoretical base for the rest of the text.

3.2 Implementing Self-organisation

Since the area of self-organisation is a wide and not particularly well-defined area[4], I have, as indicated above, started from the bottom by implementing an actual case of self-organisation. The system of choice happened to be an artificial life demonstration. The purpose of this demonstration, for the sake of this work, is to serve as a base for discussion, when discussing specific and general principles. As such it is not a goal in itself, although I hope to be able to use it to indicate some interesting points about evolution in self-organisation. The reason for choosing the artificial life demonstration as object of study

[3] Anita Håkansson (scientific advisor for this work), and a couple of colleagues at ITK.
[4] See, as an example, the discussion in the definitions section.

is mainly that I know that it works since it has been done before (T. Ray, 2000). It is a limited application of reasonably low complexity, and I believe it is a good area for testing classic self-organising concepts such as competition and evolution. AL is an often used tool for investigating self-organisation (C. Lucas, 2000).

The internal purpose of the demonstration (what it is supposed to do) is to produce algorithms more fit for survival than an initial "base" algorithm. It is an example of self-organisation (see definition later in the text), since the later states will have more fit algorithms than earlier states without explicit intervention. To use SSM metaphors, the purpose of the transformation done in the system made up of the virtual habitat is to convert a pile of boring algorithms into a pile of interesting algorithms.

3.3 Formulating Specific Principles

From there I discuss a number of points that have arisen partly when studying the results of the demonstration, and partly when working with the actual implementation. This results in a couple of "principles", or general rules, that must be considered when implementing self-organisations of this kind.

3.4 Formulating General Principles

In this section I bring together the demonstration and the specific principles and try to hold a general discussion on a couple of rules that I think are common for all self-organisation.

3.4.1 Principles

The principles presented are generally on a meta-level, something that is required for generality. I hope that my line of thought from practical detail to general meta-principle will be reasonably easy to follow.

3.4.2 Consequences for design

To bring the discussion a bit closer to earth, I will discuss the principles' implications on the design of self-organisation. Though a bit more real world, this discussion will also necessarily be on a somewhat abstract level.

3.5 ..And Back

When having finished the general principles (which are what this work is all about), I will try to demonstrate their appropriateness by turning back and rethinking the design of my demonstration with the new knowledge gathered. There will be no actual implementation, only a thought experiment, since doing the whole recursion in practise would be outside the stipulated resources for this project.

3.6 Discussion

After the recursion, I will wrap the different pieces together and discuss some possible sources of error in my deductions and inductions, and try to summarise a couple of points of interest from the general principles.

3.7 Some Underlying Assumptions

For the sake of clarity I will state some of my basic assumptions that I think have had an impact on this work.

The first and most important is that I believe in a deterministic universe. In theory, if you had all the background information, it would be possible to calculate exactly what is going to happen in every situation (of course, it might be impossible in practise to have sufficient information). This belief has a few consequences: I do not believe in true randomness or chance. These are just words that denote that we, with the information that we have, do not know what is going to happen. Things simply does not happen without a reason, and exactly the same impulse in exactly the same environment, will produce exactly the same result.

Another consequence is that the calculability of the whole universe apply to living organisms, this including humans. And what is calculable is simulatable. Thus, in theory, it is possible to simulate living organisms, including humans.

A second assumption is a consequence of the law of large numbers. If we try an infinite number of times, everything that is possible will have happened an infinite number of times. This is a prerequisite for evolution. Of course "possibilities" and "probabilities" are also just words denoting that the initial situation is not entirely known and will produce results within a limited range.

A third assumption that might be more of a cultural thing, is that I dislike and distrust the notion of objectivity. The world is a result of observation and is not something we have direct access to. Therefore, two different observations and interpretations initially is of equal validity, since there is no way to judge between them. Later on, objectivity is mainly a question of majority, if nearly all observers believe one interpretation, that observation will be called objective. In this work I will try to convince you, the reader, that my way of viewing the world is valid, but I do not claim that it is the only way to view the world.

4. Definitions

These definitions are discussions about the interpretations of the words as well as the implication of the concepts. As such they serve as a necessary base for the further discussion.

4.1 Complex and Complexity

Although the definition of "complexity" has caused much debate (F. Heylighen, 2000b), I shall, in this text, use the word in a way I believe is as practical as possible. I shall define "Complexity" as "*The quantity of subsystems and interrelations in a system*", and "Complex" as "*Has many subsystems and interrelations*".

The consequences of the above is that the variety of the system increases with its complexity (since it will get a larger state space). Therefore the definitions of complexity and complex can also be "*The level of variation within the system*" and "*Has a lot of variety*". I see these definitions as synonymous with the above.

4.2 Evolution

Darwin defines evolution as "descent with modification" (C. Darwin, 1859), meaning that species change by small inheritable changes in every generation shift. This combined with the theory of natural selection, the theory that an individual with a less efficient constitution (relative to the environment) will have to struggle more to survive than one with a more efficient constitution, leads to the conclusion that species "develop".

A point should be made here for clarity: Darwin himself never talked about selection in the usual meaning of the word in connection with "natural" evolution. He used natural selection as a metaphor, "Almost *as if* selected", referring to the evolution that takes place when breeders choose which animals should produce the next generation. There is no agent choosing among viable alternatives (F. Varela et. al, 1987). The "selection" is a purely mechanical process: An animal running slower than its mates will be caught by predator and produce less offspring. An inheritable characteristic it has (such as running a bit too slow) will then have less of a chance to move on to the next generation. Given billions of these automatic prunings every day, given billions of individuals, and given a large time-scale, these changes accumulate to produce the species we see today (see also discussion about Miller's thesis below).

The Darwinian definition of the word is, as I see it, pretty much what is generally meant when speaking about evolution. Implied in the common use of concept is the view that species steadily get *better* though. This differs slightly from Darwin. From what I have understood of Darwin, he means that "fit" is a concept highly relative to the current environment, and that there is no objective highest "fit" applicable to all environments.

Miller claims that one characteristic of evolution is that it generates organisations (organisms) of higher and higher complexity:

"The general direction of evolution has been to produce systems with more information or greater complexity of organization. In general the higher levels of living systems developed later than the lower levels, and more complex types at a given level later than less complex types." (J. G. Miller, 1978, p. 76)

Miller explains this by claiming that more complex behaviour is likely to be an evolutionary advantage over less complex behaviour, since this complexity might improve the individual system's viability.

That this is a plausible explanation can be verified by applying Ashby's law of requisite variety (W. R. Ashby, 1956) to the situation: An individual possessing more complexity can display a wider range of behaviour and thus be able to handle more variety in its environment.

By combining these definition with Ashby's theorem about self-organisation in dynamic systems (W. R. Ashby, 1962) (see also the discussion about this in the definition of self-organising systems below), we can see that it would have been more strange if evolution had *not* produced complex life forms than the fact that it has (See also further discussion in sections 5.6.1, 7.2 and 9.2)

To sum up, my definition of evolution is *"The principle of small inheritable changes in every generation shift that combined with the law of requisite variety and environmental pressure produces a force that works towards increasing the average complexity in the population".*

4.3 Intelligent

I will use the term "Intelligent" in this text, in such contexts as "Intelligent behaviour" and "Intelligent humptidum". Marvin Minsky has proposed a definition of *artificial* intelligence as:

"trying to get computers to do things that would be considered intelligent if done by people" (M. Minsky, from S. Turkle 1991)

This is pretty much the way I will use the word: Intelligent is what would be called intelligent if done by a human.

4.4 Life and Artificial Life

As far as I can judge, very few authors try to give a functional, collected definition of life as such. In the literature I have read, the usual way of resolving the question of definition has been by giving examples of things that could be said to be alive, and of things that are not. Other methods has been like that of Miller (J. G. Miller, 1978), to give a number of criteria for life (or, in his case, living systems), but not trying a definite collected definition.

Varela et. al defines life as being autopoietic systems:

"The biological evidence available today clearly shows that living systems belong to the class of autopoietic systems." (F. Varela et. al, 1974)

Earlier in the same text they have defined autopoietic systems:

"The autopoietic organization is defined as a unity by a network of productions of components which (i) participate recursively in the same network of productions of components which produced these components, and (ii) realize the network of productions as a unity in the space in which the components exist".

This means, if I understand it right, that any system which main purpose is to uphold itself, which produces itself, and which subsystems participate in this task, should be called life. This is a wide and generous definition, as it does not require "life" to be biological nor non-constructed. It is also a very fitting definition for this text.

Now that we have a grasp of what life means, we can move on to discussing *artificial* life. Here we have two different concepts: one is the formal discipline, and the other is the concept. It is not necessarily so that these two converge.

To start with the easy part, the formal discipline, one definition is by Langton:

"Artificial Life (Alife) is a field of study devoted to understanding life by attempting to abstract the fundamental dynamical principles underlying biological phenomena, and recreating these dynamics in other physical media- such as computers- making them accessible to new kinds of experimental manipulation and testing." (G. Langton, from V. V. Miagkikh, 2000)

This is a straight-forward definition which I do not see much sense in arguing with. It might be the case that the definition is not necessarily a good description of what I will do later in the text, but as a description of the formal discipline it is usable.

Now, to move on to the difficult part: What discerns "life" from "artificial life" (the concept, not the discipline)? Given the previously said, *are* there any differences in practise between "natural" and "artificial" life? Both "natural" and "artificial" fall in under the definition of "life" given earlier. Or as when Turkle is referring to Minsky again:

"Artificial life, to paraphrase Marvin Minsky, is the discipline of building organisms and systems that would be considered alive if found in nature" (S. Turkle, 1991, p. 151)

If we, as an example construct a mouse using a synthesis of ten other mice' DNA, and manage to make it live and function, will that be "life" or will it be "artificial life"? My answer is that with the definition of life I am using, the distinction between natural and artificial is pointless. Thus, I will not use the prefix "artificial" in this text.

4.5 Organisation

In Principia Cybernetica's Web Dictionary (F. Heylighen, 2000a) one can find the following definition:

"The relations, and processes of communication, including co-ordination and co-orientation among the components or variables of a system that (a) determine the dynamics of interaction and transformations it may undergo in a physical space and (b) constitute its unity whether only for an observer or also for itself."

Lucas defines organisation as:

"The arrangement of selected parts so as to promote a specific function" (C. Lucas, 2000)

To summarise, organisation is how everything relates to everything within a system. A system can not be without organisation (since a bunch of objects without organisation would not be called "system").

4.5.1 "Good" and "Bad" Organisation

Ashby discusses the notion of "good" and "bad" organisation and proposes the three following rules:

"1. most organizations are bad ones; 2. the good ones has to be sought for; and 3. what is meant by 'good' must be clearly defined, explicitly if necessary, in every case" (W. R. Ashby, 1962)

And he continues:

"There is no such thing as 'good organization' in any absolute sense. Always it is relative; and an organization that is good in one context or under one criterion may be bad under another"

Clearly, if we are to believe Ashby, there is no point in trying to define what good organisation is in general. For autopoietic units, "good" organisation is that organisation which helps the unit maintain its viability. "Bad" organisation is when something within its constitution threatens its existence.

4.6 Self-organisation and Self-organising systems

Heinz von Foerster makes a distinction between self-organising, mechanical and thermodynamical systems, and claims that a criterion for a self-organising system must be that it decreases its entropy over time (H. Foerester, 1960). An implication of this is that the suprasystem or environment must absorb that entropy, otherwise the second law of thermodynamics ("Energy can not be created nor destroyed") would be violated. He also opens his article by claiming "There are no such things as self-organising systems!".

I do not agree with Foerster in this. I suspect that we have entirely different views of what a self-organising system is. My view of a "system" is that it is a relative concept, and therefore highly dependent on the observer. If we have a group of numbers we choose to call an unorganised system, say 1-5-4-7-2-3-9-6-8-0, and have them organise themselves according to some principles, so that they make up the organised system 0-1-2-3-4-5-6-7-8-9, there will not have been a decrease in thermodynamic entropy. This since both "system" and "self-organisation" are *relative concepts*. It can not be said that one order of numbers has less entropy in any objective sense than the other.

Ross Ashby could be said to take a diametrically opposite viewpoint of the subject than vFoerster. He does discuss whether the term "self-organizing" should be used at all, but ends up in defining it as a system moving from a "bad" organisation to a "good" organisation. He claims that all systems strive towards equilibrium, that they adapt to their environments. Or, as he says (in reference to evolution as self-organisation):

"I say that this looking for special conditions is quite wrong. The truth is the opposite - every dynamic system generates its own form of intelligent life, is self-organizing in this sense. [...] ..we can see the truth of the statement that every isolated determinate system obeying unchanging laws will develop organisms that are adapted to their environment". (W. R. Ashby, 1962)

There seems to be, or to have been, considerable debate about the definition of self-organisation, and whether or not it is possible at all. The way I see it, much of the problem is derived from the different definitions of boundaries and their relation to the system that should be said to be self-organising.

Chris Lucas summarises the characteristics of self-organising systems in an attempt to define self-organisation:

"The essence of self-organization is that system structure often appears without explicit pressure or involvement from outside the system. In other words, the constraints on form (i.e. organization) of interest to us are internal to the system, resulting from interactions among the component and usually independent of the physical nature of those components. The organization can evolve in either time or space, maintain a stable form or show transient phenomena." (C. Lucas, 2000)

I see a number of difficulties with this definition. The first and most important is that there is no discussion about where to draw the limit between the system in question and its environment. Lucas uses a "hard" approach to the definition of system (later in the same text), as he apparently sees this division as something obvious and objectively possible. Another point to be made is the use of the formulation "explicit pressure", without giving a good definition about what that is, and what discerns it from "implicit pressure".

My definition of self-organising system for the purpose of this work is: "*A set of related objects and a boundary which we choose to call 'system', that order itself according to some internal principle in such a way that we deem its later states as more organised or more efficient than its earlier states*".

This is a very broad and somewhat relativistic definition. Note here that especially "boundary" and "related objects" are not definite immediately discernible facts. It is more a question of what we choose to include in our study. I shall later in the text use this definition in a way I think will help making it clear what I mean by this.

4.7 State and State Space

There are a number of synonyms for "state space" (which is what I choose to call the concept). These are, among others, "phase space", "variety" and "variability". Chris Lucas defines "state space" as:

"This is the total number of behavioural combinations available to the system. [...] Generalizing, any system has one dimension of state space for each variable it can change." (C. Lucas, 2000)

In other words, every single co-ordinate in the space represent a unique combination of variable settings. The total space is every possible combination of variable settings.

Penrose defines "phase space" as:

"...the phase space of a system is a space, normally of enormous dimensions, each of whose points represent an entire physical state in all its minute detail." (R. Penrose, 1990)

Here we have the view that the space is the sum of *physical* states. This limits the definition in a way I consider somewhat unnecessary .

I will use "state" in the meaning *"the configuration of a system and its parts in relation to everything relevant to the system",* and "state space" in the sense *"The total number of possible states, no matter how unlikely, the system can take"*.

4.8 System

There has been considerable debate during the years whether to use or not use the word "system", if it should not be called "holon", "integron" or something else instead, and what it in fact means.

To recapitulate this debate is far beyond the scope of this work. I shall be satisfied with discussing a limited number of definitions and then present my own.

Lucas defines system as:

"A system is a group of interacting parts functioning as a whole and distinguishable from its surroundings by recognizable boundaries. [...] The system has properties that are emergent, if they are not intrinsically found in any of the parts, and exist only at a higher level of description." (C. Lucas, 2000)

As far as I know, this is the usual kind of definition of system. It is a practically usable, concrete engineering-type definition. Implied in this definition is the notion that systems actually exist and that it is the observer's task to find them.

There are of course other ways to view this. As an example, Checkland argues in favour of the view that systems are *abstract concepts* (P. Checkland et. al, 1997). Although "systems" have a few defining properties such as boundaries and emergence in common in all definitions, Checkland speaks about systems as something that is not existing per se, but is more a way of viewing the world. Here the researcher's task is not to *find* the systems, but to *define* what he *chooses to view* as a system. Two researchers need not define the same interrelated parts as constituting a system and this is, according to this view, not something that is necessarily wrong.

I shall use the word "system" in the latter sense.

4.8.1 Boundaries

Since the abstraction of "system" from its surrounding is an open question, the same must be true for boundaries. In this work much of the goal is to find a limiting force or principle that, in a way, is defining for the system. This boundary is not a limit in space as much as it is a limit in purpose and function. I suspect that they way I use boundary will not be what might generally be considered the usual meaning of the word.

5. The Demonstration

In order to get a grasp of the area of study (self-organising systems), I have decided to do a small implementation and thought experiment. The implementation of choice is a demonstration of evolution of living algorithms. This mainly because it has already been shown that it is possible to build such a demonstration. Since the demonstration is not a purpose in it self (it is merely a tool to reason from), not much energy has been spent on documenting or structuring the code. The text below is with necessity a mix between "method" and "results", since it is necessary to describe what has been done in this particular case to provide a base for understanding the results.

5.1 The Habitat

The habitat consists of a matrix of 1000x1000 cells. When initialised, these cells contains a random amount of "food" (in this case everything from 0 to 5 units). After each pass through the database, 50 random cells increases their food content by one (to simulate slow growth). At each 750 passes through the database, roughly a third of the cells increases their food content by one (to simulate a seasonal explosion in growth).

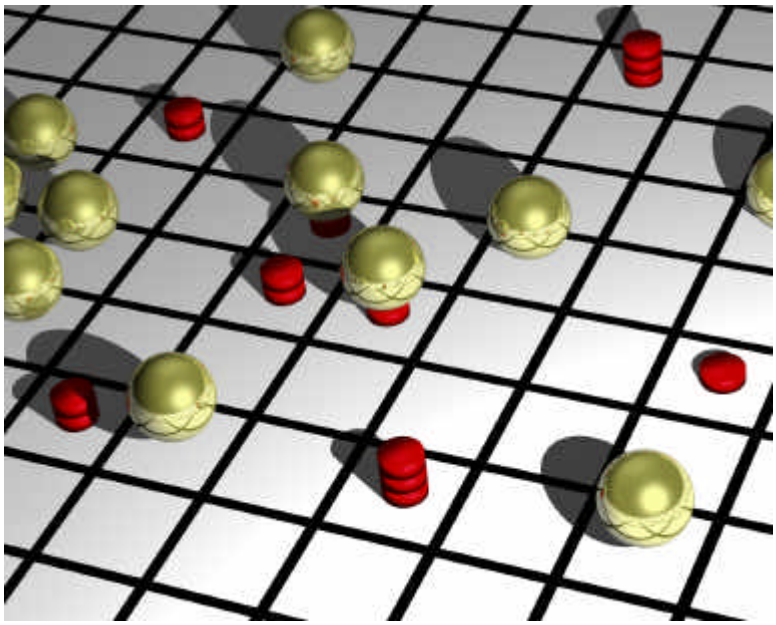

Illustration X: The habitat is a matrix of cells. There are heaps of food lying around (the red blobs), and the creatures (the yellow spheres) has to search for these heaps to avoid starving to death.

The habitat is in practise an object that also keeps a redundant check of all the creatures' positions and controls collisions and directions.

Movement is possible in eight directions, and only one creature can occupy a cell at a time. The movement is not restricted by any borders to the habitat, since the habitat is spherical (if you walk over the edge to the east you appear at the border from the west). The properties of the habitat (its size, its growth of food and its geometry) have been chosen at random. Other settings within a wide range would probably produce different effects during the run, but in principle they would not have an effect on the overall result (producing evolving complexity of any kind). Of course, totally wild settings outside a certain (yet unmeasured) range, would probably disable creatures from surviving and breeding, and so would render the habitat uninhabitable.

5.2 The Creatures and Their Code

In this habitat, "creatures" live and breed. The creatures are small autonomous pieces of "genetic" code that are run within the virtual machine that consists mainly of the habitat.

Cmd	Description	Returns
NOP	No operation	-
MOV	Move	OK/Failed
EAT	Eat	OK/Failed
TUL	Turn Left	-
TUR	Turn Right	-
CDR	Check Direction	Current direction
SRE	Set reg <VALUE>	-
PUT	Put <ADDRESS>	OK/Failed
GET	Get <ADDRESS>	Value of address
SEE	See	Food/Nothing/Creature
CMP	Compare species	None/Same/Different
MEF	Measure forward	Amount of food
MEH	Measure here	Amount of food
ATK	Attack	OK/Failed
ATD	Attacked	Attacked/not attacked
IFE	If equal <VALUE>	<jumps lines>
IFL	If less <VALUE>	<jumps lines>
IFM	If more <VALUE>	<jumps lines>
JMP	Jump <LINES>	<jumps lines>
SAY	Say <NUMBER>	OK/Failed
LSN	Listen	None/messages
GEN	Get energy level	Energy level
GMS	Get Memory size	Memory size
GGS	Get genome size	Genome size
ILL	Illegal operation	-

Figure 2: The genetic code of the creatures. The first column is the name of the command. The second is a description of the command and its parameters. The third is the result of the command. The appearance of the code has been inspired by old C64 assembler, which was one of the first programming languages I learned.

The code that is run is a combination of 25 instruction that are reasonably low-level (see Figure 2). Of course, if my intention had been to simulate biological life, I would have had to figure out an even more low-level language to avoid imposing my own prejudices on the simulation. Now, however, my intention was merely to demonstrate self-organisation, which makes the actual set of instructions less important. Indeed, the instructions has mainly been picked out of the air since it is my belief that about any set of instructions with a reasonable probability of maintaining creature viability would suffice.

The main categories of actions I wanted to be able to implement with the language was: 1, ways of finding and gathering energy. 2, ways of knowing environment. 3. ways of knowing self. And 4, ways of interacting with other creatures. Only the first is a requirement to maintain viability of creatures (they would starve to death if they did not move about enough to find food, or if they did not eat the food when it found it). The other categories I put there in the hope that creatures would evolve into displaying an intelligent behaviour after large series of mutations.

To allow fair competition and interaction, all creatures are run parallel: they have an internal pointer to the next instruction, and are allowed to execute one instruction each database pass.

5.2.1 Creature Parameters in Practise

The creatures are in practise records in a local paradox table. This has during the project been shown to be a rather stupid choice of database (see discussion in sections 9.1.2 and 5.2.2).

Column	**Purpose**	**Initial value**
Serial	Table key	Max(serial)+1
Parent	Foreign key to parent	Serial of parent
Species	Foreign key to species table	Parent or max(species)+1
Age	Current age of the creature	0
Maxage	Maximum age of creature	Gsize * 16
Energy	Current energy of creature	Gsize * 4
Reqnrg	Energy required for mitosis	Gsize * 8
Offspring	Number of offspring so far	0
Gsize	Size of genome	Depends on mutation/parent
Msize	Size of memory	Gsize * 2
Nextpos	Next instruction to be executed	0
Register	Working area for results	0
Xpos	Position in habitat	Square adjacent to parent
Ypos	Position in habitat	Square adjacent to parent
Direction	Current heading	Random
Attacked	Flag denoting if recently attacked	0
Heard	Value of what was last said	0
Addjump	Variable to handle comparisons	0

Figure 3: The various properties of the creatures. Many of these depend on the size of the genome.

The base of most of the parameters is "GSize", the size (in number of instructions) of the genome. This size is the same as that of the creatures parent, unless the creature has mutated during mitosis. The second most important parameter is "Energy". Each instruction (apart from "NOP") draws one energy unit.

5.2.2 Mutation, Evolution and Proliferation

A successful "EAT" adds 20 to "Energy". If "Energy" reaches the "Reqnrg" (Required Energy) level, mitosis takes place. Graphically speaking, the creature builds a copy of all its properties and then splits off that copy into a new creature (see illustration 2). In practise a new record is appended to the creature table, and half of the parent creature's energy is transferred to the child.

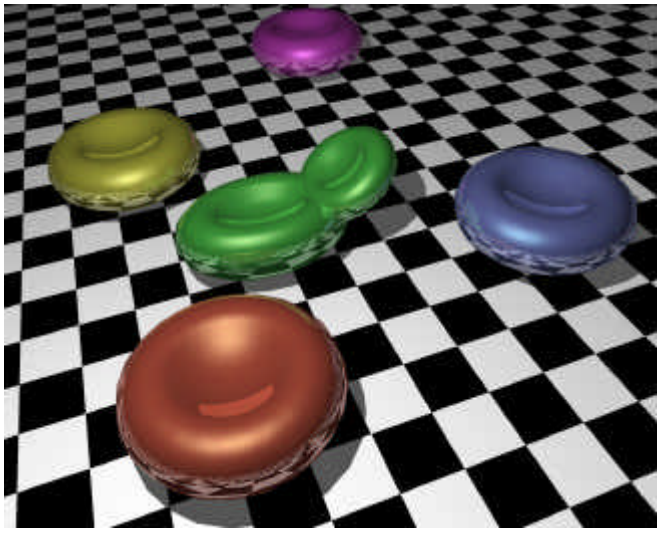

Illustration 2: When the energy level of a creature reaches "Reqnrg", it asexually reproduces by splitting off a child (mitosis, the green blob in the middle).

As each "gene" (instruction) is copied, there is a probability of 3/1000 that a mutation takes place. If it does, the probabilities of a subtract, change or addition in the genome are equal. In other words, if there were no evolutionary pressures at all, the average genome length would remain the same within the population.

Since the amount of energy added to "Energy" is the same for a small and a large creature when executing an EAT command, small creatures will reach their "Reqnrg" level faster than large creatures. On the other hand, large creatures will live longer (see "Creature Parameters in Practise" above).

There are three important evolutionary pressures implicit and explicit in the system. The first is that of food availability (and in the longer run its implication on "Energy") and the competition thereof. The second is creature interactions (as an example the "ATK" command). The third and very unfortunate is an event I call "God's Wrath".

As local Paradox tables in the BDE version has shown to have a most depressing attitude towards high loads, I have been forced to implement a global catastrophe when the population reaches a limit of 8500 individuals. When this event takes place, it brings two disastrous consequences upon the population. First "God" (The Great Programmer), kills off 50% of all creatures that have lived more than a third of their maximum life span. If there still are more than 8500 creatures alive after this, the second disaster takes place: The creatures are killed off in reversed order of age until there remains 7500 creatures.

The consequence of this is a strong evolutionary pressure against long life spans, or in other words against large genomes. This since large creatures live longer and therefore is more likely to fall within the group of elderly creatures that are vaporised by flashes of lightning during the second part of "God's Wrath". Further discussion about this can be read in section 9.1.2 and 9.1.3.

5.2.3 Genesis

Since my hope was that creatures would develop higher complexity spontaneously given large enough series, I have started with a very low-level creature. "Adam" has a very limited repertoire of behaviour. He executes an "EAT" command in the hope that there is food at his current location. Then he takes one step forward in the hope that there is nothing there (this since staying in one place soon would deplete any quantity of food). Finally he stays put for one database pass (The NOP command does not draw any energy, but it increases the genotype length, of which many other creature parameters are dependent).

Cmd	Intention
EAT	Eat, hoping that there is food here
MOV	Move, hoping that nothing blocks the way
NOP	Do nothing for one database pass

Figure 4: This is "Adam", the initial creature in the habitat. He basically just moves in one direction and eats what he walks into.

Initially there is rather much food lying around in the habitat, so at first there will not be vary hard for creatures to find enough energy to survive and proliferate. The code for initialising the food content of the habitat looks like this:

```
s:=random(HABINITTH + HABINITMAX);
if s>=(HABINITTH-1) then s:=s-HABINITTH+1 else s:=0;
welt[x,y] := s;
```

Where HABINITTH is a threshold value of when to distribute food at all, and HABINITMAX is the maximum initial amount of food to place in one cell. The values of these constants are 9 and 5. Welt is the array that keeps check of the food contents in the habitat.

5.3 On Code and Documentation

As the source code is quite voluminous, and not necessary for the further discussion, I do not present it here. Still, some selected parts of it might be interesting for the further evaluation of the habitat runs. I do therefore present selected fragments in Appendix A. Please refer to this if you have more detailed questions about the habitat. For obtaining the complete source code digitally, please refer to Appendix D ("Correspondence").

5.4 The run

Given these premises I have started the system to see what comes out in the other end.

5.4.1 Some Graphics

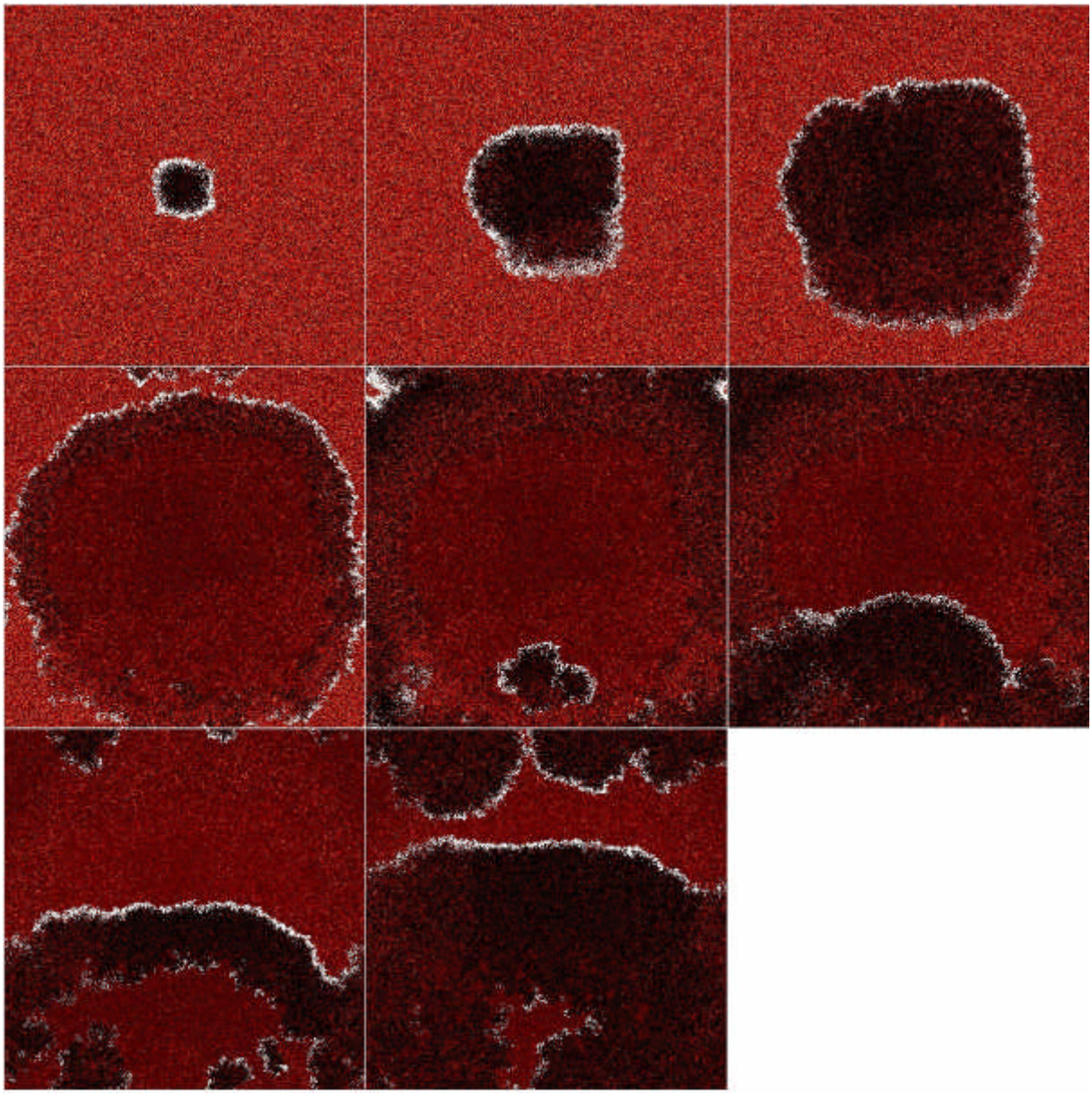

Figure 5: The habitat during a run. The pictures are snapshots of database passes 250, 500, 750, 1000, 1250, 1500, 1750 and 2000. The red on the pictures is food, the white is creatures and the black is where there is neither food nor creature.

As the population increases and eats food, the creatures are forced to look for food in the outer regions of the habitat. Of course, the individual creatures are too stupid to figure out in which direction they need to move. Instead all creatures that move into an area without food will starve, while creatures moving into an area with food will

survive and breed. The end result is that the population moves away from areas without food.

5.4.2 What Comes Out in the Other End

After having run the habitat for 2000 passes, we can examine the history table to see what creatures were produced. After all, this is what is the interesting result. To the right is one example of one of the more complex creatures that existed during the run.

```
EAT     -1      Try to eat here
JMP     0       Jump no lines
MOV     -1      Move one step
EAT     4       Try to eat again
MOV     0       Move again
```

Figure 6: Species number 7795, one of the more complex species. This species collectively managed to execute 1033 instructions and produce 29 children before becoming extinct.

There is a whole branch of this family (with two eats and two moves and a nonsense-instruction somewhere in between). I have not yet figured out why this is so efficient.

5.4.3 Some Descriptives

	250	500	750	1000	1250	1500	1750	2000
Creatures	2782	7783	7532	10216	5350	3564	8217	8272
Species	162	1049	2670	5258	7420	8075	9531	11445
Killed by god	0	0	37687	249248	382453	382453	430478	506707
Killed by peer	14	103	312	551	737	759	821	925
Starved	9088	77003	193644	248605	333847	405464	483767	586827
Dead of old age	982	5447	6712	6715	7454	9841	10718	10718

Figure X: Some descriptive data from the runs. The columns are from database passes number 250, 500, 750, 1000, 1250, 1500, 1750 and 2000.

As we can see, the number of creatures increases explosively until "God's Wrath" starts to come into action. At pass 750 we can see that that the most common cause of death is starvation, while it at pass 1000 is "Killed by God".

5.4.4 Some Statistics

		SPC	AGE	NUM	IND	AVG	VIAB
SPC	**Correlation**	1,000	-,031(**)	-,031(**)	,117(**)	-,013	-,164(**)
	Sig. (2-tailed)	,	,001	,001	,000	,155	,000
	N	11445	11445	11445	11445	11445	11363
AGE	**Correlation**	-,031(**)	1,000	,976(**)	-,003	,045(**)	,017
	Sig. (2-tailed)	,001	,	,000	,768	,000	,064
	N	11445	11445	11445	11445	11445	11363
NUM	**Correlation**	-,031(**)	,976(**)	1,000	-,005	,045(**)	,014
	Sig. (2-tailed)	,001	,000	,	,592	,000	,138
	N	11445	11445	11445	11445	11445	11363
IND	**Correlation**	,117(**)	-,003	-,005	1,000	,137(**)	-,154(**)
	Sig. (2-tailed)	,000	,768	,592	,	,000	,000
	N	11445	11445	11445	11445	11445	11363
AVG	**Correlation**	-,013	,045(**)	,045(**)	,137(**)	1,000	,235(**)
	Sig. (2-tailed)	,155	,000	,000	,000	,	,000
	N	11445	11445	11445	11445	11445	11363
VIAB	**Correlation**	-,164(**)	,017	,014	-,154(**)	,235(**)	1,000
	Sig. (2-tailed)	,000	,064	,138	,000	,000	,
	N	11363	11363	11363	11363	11363	11363
** Correlation is significant at the 0.01 level (2-tailed).							

Figure 8: A correlation matrix describing the most important variables from the run. SPC ("Species") is a counter that increases with each mutation. AGE ("Age") is the total age of that species. NUM ("Number") is the total number of individuals of that species. IND ("Complexity INDex") is an index describing the complexity of that species. AVG ("Average ") is the average number of offspring of individuals of that species. VIAB ("Average VIABility") is an index describing how large part of the maximum lifespan createtures of that species survive in average. Values discussed in the text below are marked yellow.

Here we can see that there is a significant correlation between the complexity of creatures and when they are born. In other words, we can see that the average complexity has increased in the population over time. We can further see that viability decreases over time. This is explained by the fact that there is more competition later in the run and that "God's Wrath" come into action later. Therefore creatures will have less life expectancy later in the run than earlier.

Another way of describing the complexity development over time is by looking at the serial number averages in each group of complexity indexes. The complexity index ("IND") is a discreet variable, and it has not taken on more than six valid distinct values in this run (I have removed one species with zero-length genotype):

IND	MEAN	CASES	STD. DEVIATION
1,00	5660,71	2321	3148,15
2,00	5285,91	3218	3245,09
3,00	5644,39	3593	3343,22
4,00	6006,47	1750	3345,65
5,00	8040,51	505	2838,21
6,00	8464,09	57	2488,58

Figure 9: The average SPC (species, a counter, described by the column "MEAN") is higher in groups of creatures which are more complex.

But let us continue with the correlation matrix. When we study the complexity index ("IND"), we can see that it correlates positively with average number of offspring. We can see that more complex creatures in average produces more offspring than less complex creatures. We can also see that a high complexity index correlates negatively with viability. This is because complex (which in most cases means larger) creatures have longer life spans and will therefore more often be subject to "God's Wrath".

Finally we can see that viability correlates positively with average number of offspring. This is hardly surprising since creatures that live more of their maximum life spans will have more opportunities to breed.

There are other significant correlations in the matrix, but I will ignore these for now since they have no bearing upon this discussion.

5.5 Running Unfair Runs

Just to see what happened, I did also run an "unfair" run. In this I had removed the possibility of subtract mutations, so in theory, mutations could only keep or increase the genotype length. My idea with this was that there would be a noticeably higher complexity increase in this system (and in the end, more interesting creatures). However, the results were alarmingly similar to the "fair" run (above) where the chance of subtract and addition mutations were the same. There are three interpretations of this that I can think of. The first is that "God's Wrath" constitutes such a strong pressure that not even being very unfair can help the situation. The second is that there are unknown implicit anti-complexity evolutionary pressures in the habitat that I don't know of. The third is that I have made a programming error. I intuitively find the first and third alternatives more likely, but I have not been able to locate the source of this phenomenon. See further discussion in section 9.1.3.

5.6 Discussion about the Demonstration

Now that we have seen the demonstration we can start to discuss some implications and interpretations of it.

5.6.1 Vectors in State Space

Consider the pattern made by the creatures in the graphical representation of the habitat (see figure 5). In the early states (passes 250, 500, 750 and 1000), we can clearly see a "movement" towards the outer areas of the habitat, or at least away from the place of initial birth. Clearly, the need of food constitutes a strong vector in the state space of the whole demonstration. I had not predicted this behaviour, and was rather amused to see it.

The phenomenon could be related to (or possibly the opposite of) a concept Chris Lucas calls "Attractor" (C. Lucas, 2000): a force toward a specific subset of state space. Rather than thinking about attractors, I would use the concept of vectors though, since the driving force not necessarily has to be towards something specific, but only a general direction in state space.

There are other vectors to be found in the demonstration. One of these is the drive towards higher complexity (see discussion in section 5.4.4), another the increased number of individuals.

Clearly, we have to consider directionality in the habitat. As an example, the direction away from the place of initial birth of the first creature could prove unfortunate if one, for some reason, had wanted an even distribution over the habitat.

If there are unwanted vectors, these have to be balanced by injecting opposing forces in the system. Although I have no proof that it is so, it is my belief that these vectors in state space could be treated in a manner similar to that of vectors made by forces in a physical space.

5.6.2 Staticity[5] Traps

In the beginning of the programming part of the project, I had not yet implemented the seasonal explosion of food. When running tests during this period, I noticed that creatures did move towards the limits of the habitat (as they do on the pictures above), and perished there. Food was added continuously at random co-ordinates, but obviously it was not sufficient as to aid a move back towards the middle of the habitat. This is explained by the fact that the creatures more or less cleared the area they moved over from food, and it would take more than a few units of food to aid survival there. The continuos adding would not provide food fast enough. This led to

[5] I use "staticity" in lack of a better word. I doubt that it is good English, nor has any equivalent in any other language (it doesn't in Swedish at least). I use "staticity" in the meaning "statichood", "staticness" or "staticality" (none of which exist in my dictionary).

a situation where there was plenty of food in the middle of the habitat while creatures were starving to death at the borders of it.

On a more general plane of discussion, we could say that what happened was this: As the state of the system moved in one direction of state space, it burned the bridges behind itself. The only way to go was forward, and eventually this path led to a dead end.

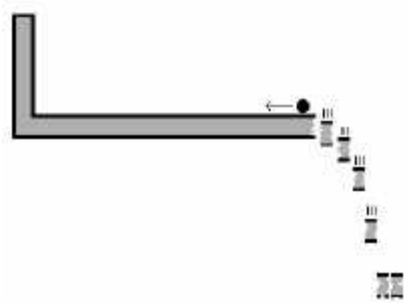

Illustration 3: State hurrying forward while state space is falling to pieces behind it.

With the state space looking as it did, one could say that the state vector was static: The state could only move into one direction.

The static vector forced the state into a very limited subset of state space and there was no way it could escape from there. As we (in the demonstration) wanted the state to continue through state space (to continue to develop interesting algorithms), this staticity trap was unwanted.

Falling into static behaviour and eventually getting stuck is not something that necessarily has to happen at once. One could imagine that there might be a deadlock later in the demonstration. As an example, if there were only two creatures alive, if they were standing on adjacent co-ordinates, and if they were facing each other, they would continue to try to move through each other until one or both (probably both) perished.

Clearly, if we want a continued dynamic development, staticity traps has to be avoided. This either by removing them altogether, or by making the probability of a state within them as low as possible.

5.6.3 "Black Box"-ness Contra Intervention

One point I found very distressing during programming, was the consequences of direct intervention. At one point I had managed to implement a bug where both XPOS and YPOS sometimes received the value of an old XPOS after a move. This resulted in a creature distribution narrowly conforming to a northwest/southeast axle. It took me three days to sort the situation out before I, after having called half of history's dark deities to my help, managed to find the erroneous code line.

> ***"The universe is in equilibrium; therefore he that is without it, though his force be but a feather, can overturn the universe"***

Illustration 4: The error of inserting unbalanced parameters into state space (Crowley, 1996)

During this threading and cursing, I found that with every new parameter I inserted in the habitat to steer the creatures away from the narrow axle, I locked something else into static behaviour. I found that the situation with thousands of creatures moving in all possible directions while breeding and eating was too complex to handle with direct intervention.

In retrospect it has struck me that the static locking is a logical consequence of inserting new forces. In theory, all new interventions could be considered as new constraints limiting the dynamic chaos of the system.

The purpose of all what I am doing here is to get away from tinkering with details. This because of the firm belief that chaos and a few general rules can take care of high complexity in a much better way than if I try to grasp it all and steer the details of that complexity. Given this, detailed intervention is the wrong way to go. There should be a balance between dynamic chaos controlled by general rules and detailed intervention, preferably so that it allows as much dynamic chaos as possible.

6. Specific Principles

This section is supposed to summarise some of the principles that (however subconsciously or randomly) have made the demonstration possible (see section 5.6). Please note that these principles are not all that are available; they are those that I have thought might be possible to generalise in the next section.

6.1 Where Do We Want to Go?

The first principle was that of directionality, to designate a direction in which we wanted to go. For the demonstration this was moving from "boring (unfit) algorithms" to "interesting (fit) algorithms". The direction, concluded from the earlier discussion on increasing complexity through evolution, was from a subset of state space where the system has low average complexity, to a subset of state space where the system has high average complexity.

The path there would be obvious if we did the whole thing by direct intervention: We would continue to step into the next state that had the highest complexity of the available choices. This was not possible though, since it would not take into account the viability and fitness of the creatures. These parameters were not as immediately recognisable as a complexity index. Obviously another path had to be taken.

6.2 Steering the Vessel

The second principle is that of intervention contra general rules. The path taken was that of evolution through non-intervention selection. The state moved through state space by hanging on to the natural tendency of increased complexity in evolutionary systems. Parameters (such as food increases and the genetic language of the creatures) were used to ascertain that the system survived while it developed on its own. There was no need to constrain the development more, since only fit and viable creatures would survive. This automatically without explicit intervention.

6.3 Avoiding the Traps

The third principle is that of avoiding being locked up in a situation without escape. Some traps on the path through state space has had to be avoided. These are mainly deadlock/staticity traps where the dynamism of the system is locked in equilibrium or halted. One such situation is the total depletion of food within the vicinity of all the currently alive creatures, another is situations where all creatures are deadlocked in trying to move through each other.

7. General Principles

From the specific principles I will here formulate some points that (I think) are generally applicable to systems of the class we are dealing with (evolutionary, self-organising systems).

7.1 The Black Hole Principle

One thing that is clear from the discussion on staticity traps (see section 5.6.2), is it this is something that may be general for all self-organising systems. Let us picture the current statespace as a plate and the system in question as a ball. Now, the ball can move anywhere on this plate, but it will not move over the borders, the limits of state space (see illustration 5).

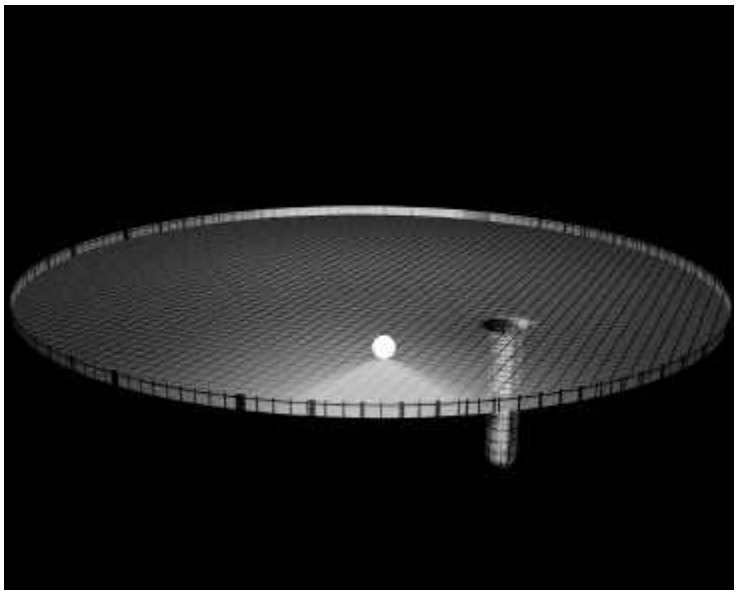

Illustration 5: State space percieved as a disc, and the system as a ball. The "Black Hole" curvature illustrates the trap of being caught in a subset of statespace from where there is no escape.

Let us further picture erratic movement in the ball to illustrate random changes in state of the system. This will enable the ball to move over the whole disc. Given completely random movement, the law of large numbers gives that the ball will eventually reach the hole and fall down into it.

Let us make a more formal description of this illustration. The "Black Hole" principle is in its essence mainly the axiom that if there is the slightest possibility that the dynamic system in question will lock in a single or limited number of states, it will according to the law of large numbers eventually do so. Just like a black hole in "real" space, the "Black Hole" subset of state space will grip everything that come within its influence and keep it there without any possibility of escape. In practise, this "black hole" could be a wide range of events, everything from a deadlock situation in a program to the depletion of virtual energy reserves.

When discussing this with one of my academic colleagues, he exclaimed "Aha! So this is mainly a formalisation of Murphy's Law?"[6]. This is exactly what it is: If there is the slightest possibility that anything can go wrong, it will, according to the law of large numbers, eventually do so.

[6] Mikael Ohlsson, during a discussion at Mid Sweden University

7.1.1 Consequences for Design

The obvious consequence of the above, is that these "Black Hole" traps in state space has to be avoided. When designing self-organising systems, some effort has to be put into analysing the state space to see if there is an (undesired) subset that might remove the dynamics of the system. Of course, it might also be so that the desired subset of state space has "black hole" properties, in which case these should instead be strengthened to keep the system from entering an undesired state. In this case the "black hole" has many properties with "attractors".

7.2 The Increasing Complexity Principle

As indicated by the data from the habitat run, and with reference to the discussion about the definition of evolution, we can see that complexity tends to increase over time in a dynamic system where subsystems interact and compete. This is something I perceive as a general principle applicable to all systems of this class.

It should be pointed out, though, that this is a principle formulated from a theoretical discussion about Ashby's law of requisite variety and indicated by a weak correlation in a badly made experiment. I shall therefore view this principle as plausible but not proven.

7.2.1 Consequences for Design

This principle could have important implications for the implementation of self-organisation. It could be possible to start out with simple components and use this principle to move through state space towards the area of interest. Effort should be spent on removing obstacles on the road to complexity. As an example, the strong evolutionary pressure created by "God's Wrath" in the Artificial Life demonstration earlier in the text, is such an obstacle that hamper the evolution of complexity.

7.3 The Directionality Principle

When handling self-organisation in the sense of moving from "bad" to "good" organisation, one has to consider the route through state space from the initial ("bad") state to the desired ("good") state or subset of state space (C. Lucas, 2000).

This route will lead through a number of states that were not known or considered when designing the self-organisation. The states actually concerned are designated by the direction through state space, its sum of state space vectors. These vectors, or preferred directions, can be everything from the increasing complexity in competition systems to attractors in Lucas' sense.

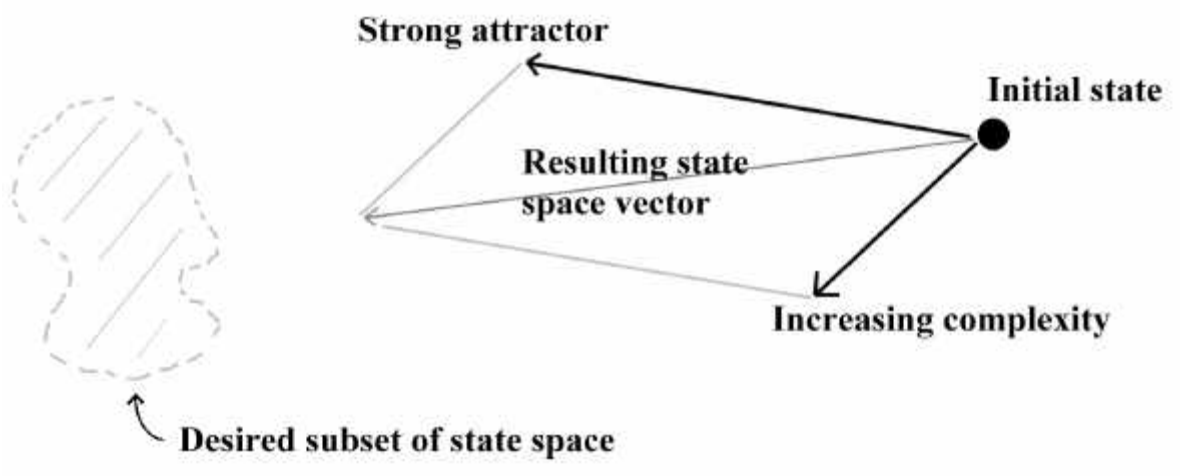

Figure 10: The direction in state space is the sum of all state space vectors. This direction hopefully points from the initial state towards the desired subset of state space.

In short, the direction of the whole system has to be considered. I do not know as of yet if this calculation is possible. I doubt that we have the tools or language for a detailed description of vectors in state space.

7.3.1 Consequences for Design

As efficient tools for this rather abstract calculation is not available, the analysis will have to be made the hard way by using brute force: The designer will have to consider all points he thinks relevant in state space. He will further have to consider all forces that could lead to or from these points and how they balance each other. In the end, I think that this will be mainly guesswork and trial-and-error.

7.4 The Requisite Intervention Principle

As every detailed intervention imposes a new constraint in state space - in practise diminishes it - some thought has to be spent on whether the intervention really is necessary. The balance between detailed control and dynamic self-organising chaos might not be easy to draw.

My thesis, with reference to previous discussions (in sections 5.6.3 and 6.2-3), is that to keep the systems' dynamic, the level of detailed intervention must be kept to an absolute minimum. As to not impose unnecessary static behaviour on the system, intervention should only be made when it is absolutely necessary. It is my belief that there is an (individual) distinct optimum point between detailed intervention and general rules for every new implementation of self-organisation.

7.4.1 Consequences for Design

This is a hard one, and in principle what the problem is all about. How do we go from tinkering with details to specifying general rules? One way is of course the use of evolution, but this only pushes the problem forward to how selection should take place. Selection is easy in a simulation of biological life, either the creature survives or it does not. Selection gets harder when we have to find (automatic) means of selecting which algorithm (object, component, principle...) that best calculates patterns or recognise pictures.

The consequence for the designer of self-organisation is that he will have to carefully consider which means, evolution or other, that best creates the momentum for movement through state space. What the result will be is probably individual for each specific case.

8. ...And Back

In this section I will use the general principles to re-formulate the specific principles and from there stress some points that should be considered. Of course, it will be impossible to be unbiased by the previous demonstration, but it might prove usable for the sake of discussion.

The below uses of the general principles are mainly results of brainstorming. The main purpose is to show the possibility of use of the principles, not to build a new demonstration

8.1 Black Holes and Traps

The first thing to consider is the various staticity traps that exist in state space. These include the total depletion of reachable food, deadlock situations in movement, and possibly the immediate extinction of all mutations. These are predictable "black holes", but there might be other as of yet unknown traps. My guess is that the only way to find most of this is to run the simulation until it falls into one of them. When this happens, careful analysis can be made of the path that lead into the black hole, and which other state space vectors that could be applied, preferably without breaking the requisite intervention level.

8.2 Increasing Complexity

In this case, the increasing complexity principle is a cornerstone in the demonstration, and one of the most important state space vectors. That is the way things are, and it should be let to work as unhampered as possible. Pains should be taken to remove vectors that work against it. As evolution (which this principle depends on) require very large series, there should be sufficient infrastructure for running millions of cases (and not just thousands as I did in my implementation). Evolution further depends on interaction and competition (C. Darwin, 1859), so these should be encouraged (as an example by providing sufficient instructions for creature interaction).

8.3 Directionality

Apart from the increasing complexity vector, we also want to move towards viability and fitness. The main means for this too is evolution. We can choose if we want to encourage growth by making it easier for large creatures (as an example by letting creatures eat more food with one instruction the larger they are). Or we can choose if we want to promote breeding (as an example by letting small creatures reach a required energy level with only one or two "EAT"s). These choices changes what should be meant by fitness.

8.4 Requisite Intervention

As the situation with all creatures, offspring and mutations will be too complex to grasp, direct intervention should be kept to a minimum. The best way is to hope that things will work out all right in the end if the force of evolution is used in a good way. To try to predict a detailed path through state space for the whole system will be impossible and pointless, since it is done better with chaos and "natural" selection.

"Consciousness is a symptom of disease. All that moves well moves without will"

Illustration 6: The error of trying to think our way through state space (Crowley, 1996)

9. Discussion

To wrap things up, I shall here discuss some points of interest, such as sources of error, and try to summaries the theories I have put forth.

9.1 Some Sources of Error

There are a number of reasons why the text in this document should be doubted and carefully evaluated. Below are those I find most important. There are probably other points that I am not aware of.

9.1.1 Bias Brought by the Bottom-Up Approach

As mentioned in the method section, I was forced to reconsider and use a bottom-up approach when studying self-organisation. Since this is the only way I have done it, it is hard to say what would have been different with a top-down model. I suspect that I might have seen more points that are general between different examples of self-organisation if I had studied these examples instead of immediately burying myself under the code of one example.

9.1.2 The Choice of DBMS and Programming Style

The choice I regret the most is the choice of database management system to use as infrastructure for the demonstration. It felt stupid to have to limit the investigation just because the database could not handle the load. Unfortunately, my own programming style made it impossible to change DBMS when this was discovered: I had hard-coded bad hacks that supposed local BDE-managed tables. I had broken about every rule of modularization, database modelling and documentation available. I guess my programming teachers would sit down and cry if they ever saw the code of the demonstration. They cannot see the documentation though, since there is none.

To find a new DBMS, learn how to use it, rewrite the code, and do all the runs all over again, would break the project schedule by weeks or months. This was not an option, since I was already behind schedule.

Well, you live and learn. I guess there might be a reason for all those programming style rules they teach you. Anyway, the programming style might have caused a lot of minor unintended "features" that might have effected the result of demonstration in a way that I am not aware of.

The main effect of the DBMS choice was that it forced me to implement the God's Wrath event. If I had to point out one possible reason for throwing the whole implementation part of this work on the city dump, I would choose the God's Wrath "feature". A program that is supposed to demonstrate self-organisation by producing complex algorithms must be said to have a somewhat limited value when the most salient feature of the demonstration is a principle that works against complexity.

The further effect of this is that I have *very* weak empirical data supporting my thesis about increasing complexity. I guess that the label most people would put on the complexity/time correlation would not exactly be "impressing".

9.1.3 Why Do Unfair Runs Produce Similar Results?

Another point about the demonstration that continues to bother me is that the "unfair" runs did not produce significantly different results than the "fair" runs. This leads to the feeling that there is something in the demonstration that I am unaware of, another of those unintended "features" that might have had an effect on the results. I simply do not know what causes the phenomenon, and that is a good reason to doubt the validity of the demonstration.

9.1.4 Are Life Simulations Too Easy?

The overall purpose of this work was to formulate some general principles for self-organisation. The means for doing this was mainly to do an implementation of self-organisation in the form of an AL demonstration. One might argue that this was a bad choice, since AL is not very controversial or even hard to implement. It was simply obvious that it would work through self-organisation since the whole concept has self-organisation as its base.

If another demonstration had been chosen, maybe other points about self-organisation had been found? I do not doubt it. In fact I find it likely. My only defence is that I could not think of another demonstration to do (I still cannot). With my limited experience of this area, it is not likely that I would have succeeded if I had chosen something more difficult. Better then to do a limited investigation where there is a good chance of finding something interesting.

The results and the reasoning based on them should be taken for what they are. The discussion is based on hints from observing one very limited area, and might very well have a strong bias towards that area. I do however think that the general principles are applicable outside the area of AL. All I am saying is that this is as of yet not proven.

9.1.6 Validity of the Reasoning about Specific and General Principles

Most of the discussions in the sections about the specific and general principles have a very limited empirical base. They are mainly theoretical and should be viewed as such. I hope that my line of thought is reasonably clear and possible to agree with, but if someone asks me "do you *know* this?", I would have to answer "no".

9.2 Points of Summary

These point are mainly summaries of the general principles, and could as such use a better empirical base. I state them here as propositions, not as fact.

The first point is about staticity: "*Most direct intervention will limit state space, make it more static*".

The second point is that about complexity: "*All dynamic systems whose subsystems compete and mutate, will have a vector towards increased complexity over time*".

The third is about state space vectors: "*All state space vectors influencing a system will combine into a direction through state space for that system. This direction is calculable if the input vectors are known*".

9.3 Proposed Further Research

I shall hereby use the opportunity to propose some ways of moving forward from this point. If someone finds these proposals interesting enough to write a thesis or paper about, they are welcome to discuss them further with me.

9.3.1 Requisite Variety and Evolution

As the simple evolution demonstration in this work does little more than hint that there is a positive correlation between complexity and time in an evolutionary system, this area could benefit from a more thorough investigation. I would, as a first step, propose doing the whole demonstration again, but this time using a capable database, running series several magnitudes larger that the ones here, and removing silly obstacles like "killed by god". It would be interesting to see if it can be *proven* statistically that complexity indeed increases in dynamic evolutionary systems. For now this principle is just an assumption based mainly on deduction and not on observation.

9.3.2 Vectors in State Space

It would be interesting to see if it was possible to build a collected model for the numerical handling of state space vectors. I have a feeling that this should be possible by exchanging physical forces in the classic vector models with state space probabilities. I suspect that it might be necessary to find a way of describing state spaces graphically first though, as classic vector models include calculating angles and parallelograms.

9.4 Wrapping it all Up

In the above I have pointed out a couple of points, or principles, about self-organisation, that I think might be general or of use across all implementations of self-organisation. These are currently pre-alpha to alpha to use open source terms, meaning that there is still a lot to do to make them usable in practise.

I think that the step that would bring things the farthest towards practical usability, would be if a formal language and model for calculating state space vectors came into being. With this kind of tool, most of the general principles I have pointed at here would be possible to apply formally and quantitatively to a wide range of implementations. I am certain that software development and possibly other systems development would benefit significantly from this.

APPENDIX A - SOURCE CODE

Below are selected parts of the source code. As mentioned earlier in the text, this source-code is ugly and mainly uncommented. The source is here as a means of reference (it should be possible to understand for someone who has some acquaintance with Delphi and SQL). If what you are reading now is the formal printed version of the thesis, there should be an accompanying CD-rom with (among other things) the source code.

Execute

The execute procedure is the main controlling thread in the application. It basically iterates through the creature database and delegates most of the action to other parts of the program.

```
procedure THabRun.Execute;
var tbs : TBlobStream;
    qry : TQuery;
    ins : TGInstruction;
    alldead : boolean;
    s   : string;
    bf,af : TDateTime;
begin
  runs := 0;

  tblstat := TTable.Create(application);
  tblstat.DatabaseName := dbhandler.dbname;
  tblstat.TableName := 'Status';
  tblstat.open;

  tbl := TTable.Create(application);
  tbl.DatabaseName := dbhandler.dbname;
  tbl.TableName := 'Creatures';
  tbl.open;

  alldead := FALSE;

  qry := TQuery.Create(application);
  qry.DatabaseName := dbhandler.dbname;

  qry.sql.add('select count(*) as num from creatures');
  qry.open;
  if qry.isempty then begin
    numcreatures := 0;
    alldead := true;
  end else numcreatures := qry['num'];
  qry.close;

  qry.sql.clear;
  qry.sql.add('select count(*) as num from history');
  qry.open;
  if qry.isempty then begin
    numspecies := 0
  end else numspecies := qry['num'];
  qry.close;
  qry.free;
```

```
  synchronize(synk);

  while (not self.terminated) and (not tbl.IsEmpty) and (not alldead) do
begin
    tblstat.edit;
    tblstat['numpasses'] := tblstat['numpasses'] + 1;
    tbl.First;
    numdeaths1 := 0;
    numbirths1 := 0;
    numinstrs1 := 0;
    numeaten   := 0;
    bf := now;
    while (not tbl.Eof) and (not terminated) and (not alldead) do begin
      if tbl['nextpos'] >= tbl['gsize'] then begin
        tbl.edit;
        tbl['nextpos'] := 0;
        tbl.post;
      end;
      tbs :=
TBlobStream(tbl.CreateBlobStream(tbl.fieldbyname('genome'),bmRead));
      tbs.Seek(tbl['nextpos']*8,soFromBeginning);
      tbs.Read(ins,8);
      tbs.Free;
      if (ins.cmd < 24) and (ins.cmd >= 0) and (1=2) then begin
        s:='CRE: ' + chr(9) + inttostr(tbl['serial']) + '/' +
inttostr(tbl['species']) + chr(13);
        s:=s + 'POS: ' + chr(9) + inttostr(tbl['nextpos']) + '/' +
inttostr(tbl['gsize']) + chr(13);
        s:=s + 'INS: ' + chr(9) +  GLanguage[ins.cmd].name + ' (' +
inttostr(ins.arg) + ')' + chr(13);
        s:=s + 'AGE: ' + chr(9) + inttostr(tbl['age']) + '/' +
inttostr(tbl['maxage']) + chr(13);
        s:=s + 'NRG: ' + chr(9) +  inttostr(tbl['energy']) + '/' +
inttostr(tbl['reqnrg']) + chr(13);
        s:=s + 'COR: ' + chr(9) +  inttostr(tbl['xpos']) + ',' +
inttostr(tbl['ypos']) + chr(13);
        s:=s + 'DIR: ' + chr(9) +  inttostr(tbl['direction']) + chr(13);
        s:=s + 'REG: ' + chr(9) +  inttostr(tbl['register']);
        showmessage(s);
      end;
      insts[ins.cmd](tbl,ins.arg);
      tblstat['numinstrs'] := tblstat['numinstrs'] + 1;
      inc(numinstrs1);
      tbl.edit;
      if tbl['addjump'] = 0 then tbl['nextpos'] := tbl['nextpos'] + 1;
      if tbl['addjump'] >= 0 then tbl['addjump'] := tbl['addjump'] - 1;
      tbl['nextpos'] := tbl['nextpos'] + 1;
      tbl['energy'] := tbl['energy'] - 1;
      tbl['age'] := tbl['age'] + 1;
      tbl.post;
      if tbl['energy'] >= tbl['reqnrg'] then begin
        birth(tbl);
      end;
      if (tbl['age'] >= tbl['maxage']) or (tbl['energy'] <= 0) then begin
        die(false);
        if tbl.IsEmpty then alldead := true;
      end else if (not alldead) and (not tbl.isempty) then tbl.Next;
    end;
    af := now;

    numsecs := round((24*60*60) * (af-bf));
```

```
    tblstat['tottime'] := tblstat['tottime'] + (af-bf);
    tblstat.post;
    if (tblstat['numpasses'] mod 250) = 0 then begin
      passes:=tblstat['numpasses'];
      habitat.Writeout;
      tblstat.close;
      tbl.close;
      synchronize(snapshot);
      tbl.open;
      tblstat.open;
    end;
    if not alldead then begin
      if numcreatures > WrathTH then GodsWrath;
      self.synchronize(synk);
      habitat.incfood;
    end;
  end;
  if tbl.IsEmpty then begin
    showmessage('No creatures to work with');
  end;
  tbl.close;
  tblstat.close;

  tbl.free;
  tblstat.free;

  showmessage('Leaving execute');
end;
```

Gods Wrath

The "Gods Wrath" event (of which I have spoken at some length earlier in the document), is in practise a procedure which scans the creature database and kill off elderly creatures.

```
procedure THabRun.godswrath;
var qry:TQuery;
begin
  tbl.first;
  tblstat.edit;
  while not tbl.eof do begin
    if (tbl['age'] > (tbl['maxage'] div 3)) and (random(2) = 0) then
die(true);
    tbl.next;
  end;
  tblstat.post;
  if tbl.RecordCount > WrathTH then begin
    qry := TQuery.Create(application);
    qry.DatabaseName := dbhandler.dbname;
    qry.sql.Add('select * from creatures order by serial');
    qry.RequestLive := true;
    qry.open;
    qry.first;

    tblstat.edit;
    while tbl.recordcount > 7500 do begin
      tbl.editkey;
      tbl.fieldbyname('serial').asfloat := qry['serial'];
      tbl.gotokey;
```

```
      die(true);
      qry.next;
    end;
    tblstat.post;
    qry.close;
    qry.free;
  end;
end;
```

Die

The "die" procedure handles the actual killing of the creatures, and the statistics about it.

```
procedure THabRun.die(wrath:boolean);
var qry : TQuery;
begin
  qry := TQuery.Create(application);
  qry.DatabaseName := dbhandler.dbname;
  qry.sql.Add('select * from history where species = ' +
inttostr(tbl['species']));
  qry.RequestLive := true;
  qry.open;

  qry.edit;
  qry['totage'] := qry['totage'] + tbl['age'];
  qry['totnum'] := qry['totnum'] + 1;
  qry['offspr'] := qry['offspr'] + tbl['offspring'];

  tblstat['numdeaths'] := tblstat['numdeaths'] + 1;
  dec(numcreatures);
  inc(numdeaths1);
  if not wrath then begin
    if (tbl['age'] >= tbl['maxage']) then begin
      tblstat['numoldage'] := tblstat['numoldage'] + 1;
      qry['oldage'] := qry['oldage'] + 1;
    end else begin
      tblstat['numstarve'] := tblstat['numstarve'] + 1;
      qry['starve'] := qry['starve'] + 1;
    end;
  end else begin
    tblstat['numgodwr'] := tblstat['numgodwr'] + 1;
    qry['godkill'] := qry['godkill'] + 1;
  end;
  qry.post;
  qry.free;

  habitat.Die(tbl['xpos'],tbl['ypos']);
  tbl.Delete;
end;
```

Birth

The birth procedure handles "mitosis" and checks that the split-off was valid. It is also here that mutations takes place. This is the birth of the "fair" run. The only difference in the "unfair" run is that the yellow-marked 999 below should be set to 1000 or higher.

```
procedure THabRun.birth(q:TTable);
var t,s:TTable;
    m:TQuery;
    xpos,ypos,d,i,l,r : integer;
    maxsp,maxse : double;
    tbs,tbs1:TBlobStream;
    p : pointer;
    mutation : boolean;
begin
try
  t:=TTable.create(application);
  t.DatabaseName := dbhandler.dbname;
  t.TableLevel := 7;
  t.TableName := 'Creatures';
  t.open;

  m := TQuery.create(application);
  m.DatabaseName := dbhandler.dbname;
  m.sql.add('SELECT MAX(serial) AS tops FROM creatures');
  m.open;
  m.first;

  maxse := m['tops'];

  m.close;
  m.sql.Clear;
  m.sql.add('SELECT MAX(species) AS tops FROM history');
  m.open;
  m.first;

  maxsp := m['tops'];

  m.close;
  m.free;

  xpos:=q['xpos'];
  ypos:=q['ypos'];
  randomize;
  d:=random(8);

  DirectionFix(d,xpos,ypos);

//  if (q['xpos'] = xpos) and (q['ypos'] = ypos) then begin
//    showmessage('Lika värden!');
//  end;

  if habitat.place(xpos,ypos) then begin
    mutation := false;
    t.Append;
    getmem(p,round(q['gsize']) * 8 * 2);
    tbs := TBlobStream(q.CreateBlobStream(q.fieldbyname('genome'),bmRead));
```

```
    tbs1:=
TBlobStream(t.CreateBlobStream(t.fieldbyname('genome'),bmWrite));
    tbs.seek(0,soFromBeginning);
    tbs1.seek(0,soFromBeginning);

    l:=0;
    for i := 1 to q['gsize'] do begin
      r:=random(700)+300;
      if r < 999 then begin       // if not mutsub
        if r < 997 then begin
          tbs.read(pbyte(longint(p)+8 * l)^,8);
          inc(l);
        end else begin
          mutation := true;
          pinteger(longint(p)+8 * l)^ := random(24);  // mutchange or add
          d:=random(10) - 5;
          if d < 0 then d := 0;
          pinteger(longint(p)+8 * l + 4)^ := d;
          inc(l);
          if (r = 998) then begin  // mutadd
            tblstat['mutadd'] := tblstat['mutadd'] + 1;
            tbs.read(pbyte(longint(p)+8 * l)^,8);
            inc(l);
          end else begin // mutchange
            tblstat['mutchange'] := tblstat['mutchange'] + 1;
            tbs.seek(8,soFromCurrent);
          end;
        end;
      end else begin
        mutation := true;
        tbs.seek(8,soFromCurrent);
        tblstat['mutsub'] := tblstat['mutsub'] + 1;
      end;
    end;
    tbs1.write(p^,l * 8);
    tbs.Free;
    tbs1.Free;

    tbs1:=
TBlobStream(t.CreateBlobStream(t.fieldbyname('memory'),bmReadWrite));
    tbs1.seek(0,soFromBeginning);
    if tbs1.write(p^,l * 8) = 0 then beep;//showmessage('Skrev 0, l är
'+inttostr(l));
    if tbs1.size = 0 then beep;//showmessage('Size är 0, l är
'+inttostr(l));
    tbs1.free;


    t['serial']      := maxse + 1;
    t['gsize']       := l;
    t['msize']       := l * 2;
    t['parent']      := q['serial'];

    if mutation then t['species'] := maxsp + 1
    else t['species']     := q['species'];

    t['age']         := 0;
    t['energy']      := l * 4;
    t['reqnrg']      := l * 8;
    t['offspring']   := 0;
    t['nextpos']     := 0;
```

```
    t['register']    := 0;
    t['maxage']      := l * 16;
    t['xpos']        := xpos;
    t['ypos']        := ypos;
    t['direction']   := random(8);
    t['attacked']    := FALSE;
    t['addjump']     := -1;

    t.post;

    if mutation then begin
      s:=TTable.create(application);
      s.DatabaseName := dbhandler.dbname;
      s.TableLevel := 7;
      s.TableName := 'History';
      s.open;
      s.append;
      s['species']     := maxsp + 1;
      s['totage']      := 0;
      s['totnum']      := 0;
      s['offspr']      := 0;
      s['oldage']      := 0;
      s['godkill']     := 0;
      s['peerkill']    := 0;
      s['starve']      := 0;
      s['gsize']       := t['gsize'];
      tbs :=
TBlobStream(s.CreateBlobStream(s.fieldbyname('genome'),bmReadWrite));
      tbs.Write(p^,t['gsize'] * 8);
      tbs.Free;
      s.post;
      s.close;
      s.free;
      inc(numspecies);
    end;

    tblstat['numbirths'] := tblstat['numbirths'] + 1;
    inc(numbirths1);
    inc(numcreatures);
    freemem(p);
//    tbs1:=
TBlobStream(t.CreateBlobStream(t.fieldbyname('memory'),bmRead));
//    if tbs1.size < 32 then showmessage(inttostr(tbs1.size));
//    tbs1.free;
  end;
  t.close;
  t.free;

  q.Edit;
  q['energy'] := q['reqnrg'] DIV 2;
  q['offspring'] := q['offspring'] + 1;
  q.post;
except
  on exception do ;
end;
end;
```

Instructions

Below are the code base for the instructions (listed in section 5.2)

```
Procedure G_NOP(q:TTable;arg:integer);
begin
  q.edit;
  q['energy'] := q['energy'] + 1;
  q.post;
end;

Procedure G_MOV(q:TTable;arg:integer);
var xpos,ypos : integer;
begin
  xpos := q['xpos'];
  ypos := q['ypos'];
  DirectionFix(q['direction'],xpos,ypos);


//  if (q['xpos'] = xpos) and (q['ypos'] = ypos) then begin
//    showmessage('Lika värden!');
//  end;

  q.edit;
  if habitat.move(q['xpos'],q['ypos'],xpos,ypos) then begin
    q['xpos'] := xpos;
    q['ypos'] := ypos;
    q['register'] := 1;
  end else begin
    q['register'] := 0;
  end;
  q.post;
end;

Procedure G_EAT(q:TTable;arg:integer);
begin
  q.edit;
  if habitat.eat(q['xpos'],q['ypos']) then begin
    tblstat['eaten'] := tblstat['eaten'] + 1;
    inc(numeaten);
    q['register'] := 1;
    q['energy'] := q['energy'] + foodworth;
  end else q['register'] := 0;
  q.post;
end;

Procedure G_TUL(q:TTable;arg:integer);
var d:integer;
begin
  d := q['direction'];
  d := d - 1;
  if d < D_EAST then d := D_NORTHEAST;
  q.Edit;
  q['direction'] := d;
  q.post;
end;
```

```
Procedure G_TUR(q:TTable;arg:integer);
var d:integer;
begin
  d := q['direction'];
  d := d + 1;
  if d > D_NORTHEAST then d := D_EAST;
  q.Edit;
  q['direction'] := d;
  q.post;
end;

Procedure G_CDR(q:TTable;arg:integer);
begin
  q.edit;
  q['register'] := q['direction'];
  q.post;
end;

Procedure G_SRE(q:TTable;arg:integer);
begin
  q.edit;
  q['register'] := arg;
  q.post;
end;

Procedure G_MEF(q:TTable;arg:integer);
var xpos,ypos : integer;
begin
  xpos := q['xpos'];
  ypos := q['ypos'];

  DirectionFix(q['direction'],xpos,ypos);

  q.edit;
  q['register'] := habitat.numfood(xpos,ypos);
  q.post;
end;

Procedure G_MEH(q:TTable;arg:integer);
begin
  q.edit;
  q['register'] := habitat.numfood(q['xpos'],q['ypos']);
  q.post;
end;

Procedure G_ATK(q:TTable;arg:integer);
var xpos,ypos : integer;
    qry,qry1:tquery;
begin
  xpos := q['xpos'];
  ypos := q['ypos'];

  DirectionFix(q['direction'],xpos,ypos);

  q.edit;
  if habitat.iscreature(xpos,ypos) then begin
    qry := TQuery.Create(application);
    qry.databasename := dbhandler.dbname;
    qry.RequestLive := TRUE;
```

```
    qry.sql.add('SELECT * FROM creatures WHERE xpos = ' + inttostr(xpos) +
' AND ypos = ' + inttostr(ypos));
    qry.open;

    tblstat['numattacks'] := tblstat['numattacks'] + 1;
    if qry['energy'] <= 10 then begin
      tblstat['numkills'] := tblstat['numkills'] + 1;
      qry1 := TQuery.Create(application);
      qry1.databasename := dbhandler.dbname;
      qry1.RequestLive := TRUE;

      qry1.sql.add('SELECT * FROM history WHERE species = ' +
floattostr(qry['species']));
      qry1.open;
      qry1.edit;
      qry1['peerkill'] := qry1['peerkill'] + 1;
      qry1['starve'] := qry1['starve'] - 1;
      qry1.post;
      qry1.free;
    end;

    qry.edit;
    qry['energy'] := qry['energy'] - 10;
    qry['attacked'] := TRUE;
    qry.post;
    qry.close;
    qry.free;

    q['energy'] := q['energy'] + 10;
    q['register'] := 1;

  end else q['register'] := 0;
  q.post;

end;

Procedure G_ATD(q:TTable;arg:integer);
begin
  q.edit;
  if q['attacked'] then q['register'] := 1
  else q['register'] := 0;
  q['attacked'] := FALSE;
  q.post;
end;

Procedure G_IFE(q:TTable;arg:integer);
begin
  q.edit;
  if not (arg = q['register']) then q['nextpos'] := q['nextpos'] + 1
  else q['addjump'] := 1;
  q.post;
end;

Procedure G_IFL(q:TTable;arg:integer);
begin
  q.edit;
  if not (arg < q['register']) then q['nextpos'] := q['nextpos'] + 1
  else q['addjump'] := 1;
  q.post;
end;
```

```
Procedure G_IFM(q:TTable;arg:integer);
begin
  q.edit;
  if not (arg > q['register']) then q['nextpos'] := q['nextpos'] + 1
  else q['addjump'] := 1;
  q.post;
end;

Procedure G_JMP(q:TTable;arg:integer);
begin
  if arg > 1 then begin
    dec(arg);
    q.edit;
    q['nextpos'] := q['nextpos'] + arg;
    q.post;
  end;
end;

Procedure G_SAY(q:TTable;arg:integer);
var xpos,ypos : integer;
    qry:tquery;
begin
  xpos := q['xpos'];
  ypos := q['ypos'];

  DirectionFix(q['direction'],xpos,ypos);

  q.edit;
  if habitat.iscreature(xpos,ypos) then begin
    qry := TQuery.Create(application);
    qry.databasename := dbhandler.dbname;
    qry.RequestLive := TRUE;

    qry.sql.add('SELECT * FROM creatures WHERE xpos = ' + inttostr(xpos) +
' AND ypos = ' + inttostr(ypos));
    qry.open;
    qry.edit;

    qry['heard'] := arg;
    qry.post;

    q['register'] := 1;

    qry.close;
    qry.free;
  end else q['register'] := false;
  q.post;
end;

Procedure G_LSN(q:TTable;arg:integer);
begin
  q.edit;
  q['register'] := q['heard'];
  q['heard'] := -1;
  q.post;
end;

Procedure G_GEN(q:TTable;arg:integer);
begin
  q.edit;
  q['register'] := q['energy'];
```

```
  q.post;
end;

Procedure G_GMS(q:TTable;arg:integer);
begin
  q.edit;
  q['register'] := q['msize'];
  q.post;
end;

Procedure G_GGS(q:TTable;arg:integer);
begin
  q.edit;
  q['register'] := q['gsize'];
  q.post;
end;

Procedure G_SEE(q:TTable;arg:integer);
var xpos,ypos : integer;
begin
  xpos := q['xpos'];
  ypos := q['ypos'];

  DirectionFix(q['direction'],xpos,ypos);

  q.edit;
  q['register'] := habitat.see(xpos,ypos);
  q.post;
end;

Procedure G_PUT(q:TTable;arg:integer);
var tbs : TBlobStream;
      i : integer;
begin
try
  i := q['register'];
  q.edit;
  if (arg < q['msize']) and (arg >= 0) then begin
    tbs :=
TBlobStream(q.CreateBlobStream(q.fieldbyname('memory'),bmReadWrite));
    tbs.Seek(4*arg,soFromBeginning);
    tbs.Write(i,4);
    tbs.Free;
    q['register'] := 1;
  end else q['register'] := 0;
  q.post;
except
  on exception do beep;// begin
//    showmessage('Put kraschade. ARG: ' + inttostr(arg) + '  MSIZE: ' +
inttostr(q['msize']) + '  SER: ' + inttostr(q['serial']));
//  end;
end;
end;

Procedure G_GET(q:TTable;arg:integer);
var tbs : TBlobStream;
      i : integer;
begin
try
  q.edit;
  if (arg < q['msize']) and (arg >= 0) then begin
```

```
    tbs := TBlobStream(q.CreateBlobStream(q.fieldbyname('memory'),bmRead));
    tbs.Seek(4*arg,soFromBeginning);
    tbs.Read(i,4);
    tbs.Free;
    q['register'] := i;
//    showmessage('Got ' + inttostr(arg) + ' = ' +
inttostr(q['register']));
  end else q['register'] := 0;
  q.post;
except
  on exception do beep;//begin
//    showmessage('Get kraschade. ARG: ' + inttostr(arg) + '  MSIZE: ' +
inttostr(q['msize']));
//  end;
end;
end;

Procedure G_CMP(q:TTable;arg:integer);
var xpos,ypos : integer;
    qry:tquery;
begin
  xpos := q['xpos'];
  ypos := q['ypos'];

  DirectionFix(q['direction'],xpos,ypos);

  q.edit;
  if habitat.iscreature(xpos,ypos) then begin
    qry := TQuery.Create(application);
    qry.databasename := dbhandler.dbname;

    qry.sql.add('SELECT * FROM creatures WHERE xpos = ' + inttostr(xpos) +
' AND ypos = ' + inttostr(ypos));
    qry.open;

    if q['species'] = qry['species'] then q['register'] := 1
    else q['register'] := 0;

    qry.close;
    qry.free;
  end else q['register'] := -1;
end;
```

Create Database

The create database procedure is where the database is initialised the first time the program is run (or when the database has been deleted).

```
procedure TDBHandler.CreateDatabase;
var tbs : TBlobStream;
begin
  mtbl.TableName := tables[0];
  with mtbl.fielddefs do begin
    add('serial',ftfloat,0,true);
    add('parent',ftfloat,0,false);
    add('species',ftfloat,0,false);
    add('age',ftinteger,0,false);
    add('energy',ftinteger,0,false);
    add('reqnrg',ftinteger,0,false);
```

```
    add('offspring',ftinteger,0,false);
    add('gsize',ftinteger,0,false);
    add('msize',ftinteger,0,false);
    add('nextpos',ftinteger,0,false);
    add('register',ftinteger,0,false);
    add('maxage',ftinteger,0,false);
    add('xpos',ftinteger,0,false);
    add('ypos',ftinteger,0,false);
    add('direction',ftinteger,0,false);
    add('attacked',ftboolean,0,false);
    add('heard',ftinteger,0,false);
    add('addjump',ftinteger,0,false);
    add('genome',ftblob,2,false);
    add('memory',ftblob,2,false);
  end;
  with mtbl.IndexDefs do begin
    add('','serial',[ixprimary, ixunique]);
    add('coords','xpos;ypos',[]);
  end;
  mtbl.CreateTable;
  mtbl.open;

  mtbl.Append;
  mtbl['serial']      := 0;
  mtbl['parent']      := -1;
  mtbl['species']     := 0;
  mtbl['age']         := 0;
  mtbl['energy']      := sizeof(adam) div 2;
  mtbl['reqnrg']      := sizeof(adam);
  mtbl['offspring']   := 0;
  mtbl['gsize']       := sizeof(adam) div 8;
  mtbl['msize']       := sizeof(adam) div 4;
  mtbl['nextpos']     := 0;
  mtbl['register']    := 0;
  mtbl['maxage']      := sizeof(adam) * 2;
  mtbl['xpos']        := 500;
  mtbl['ypos']        := 500;
  mtbl['direction']   := 1;
  mtbl['attacked']    := FALSE;
  mtbl['heard']       := -1;
  mtbl['addjump']     := -1;

  tbs :=
TBlobStream(mtbl.CreateBlobStream(mtbl.fieldbyname('genome'),bmReadWrite));
  tbs.Write(adam,sizeof(adam));
  tbs.Free;

  tbs :=
TBlobStream(mtbl.CreateBlobStream(mtbl.fieldbyname('memory'),bmReadWrite));
  tbs.Write(adam,sizeof(adam));
  tbs.Free;

  mtbl.post;

  mtbl.close;
  mtbl.FieldDefs.Clear;
  mtbl.IndexDefs.Clear;

  mtbl.TableName := tables[1];
  with mtbl.fielddefs do begin
    add('species',ftfloat,0,false);
```

```
    add('totage',ftfloat,0,false);
    add('totnum',ftfloat,0,false);
    add('offspr',ftfloat,0,false);
    add('gsize',ftinteger,0,false);
    add('oldage',ftinteger,0,false);
    add('godkill',ftinteger,0,false);
    add('peerkill',ftinteger,0,false);
    add('starve',ftinteger,0,false);
    add('genome',ftblob,2,false);
  end;
  with mtbl.IndexDefs do begin
    add('main','species',[ixprimary, ixunique]);
  end;
  mtbl.CreateTable;
  mtbl.open;

  mtbl.Append;
  mtbl['species']     := 0;
  mtbl['totage']      := 0;
  mtbl['totnum']      := 0;
  mtbl['godkill']     := 0;
  mtbl['peerkill']    := 0;
  mtbl['oldage']      := 0;
  mtbl['starve']      := 0;
  mtbl['offspr']      := 0;
  mtbl['gsize']       := sizeof(adam) div 8;
  tbs :=
TBlobStream(mtbl.CreateBlobStream(mtbl.fieldbyname('genome'),bmReadWrite));
  tbs.Write(adam,sizeof(adam));
  tbs.Free;
  mtbl.post;

  mtbl.close;
  mtbl.FieldDefs.Clear;
  mtbl.IndexDefs.Clear;

  mtbl.TableName := tables[2];
  with mtbl.fielddefs do begin
    add('key',ftinteger,0,true);
    add('numdeaths',ftinteger,0,false);
    add('numbirths',ftinteger,0,false);
    add('numstarve',ftinteger,0,false);
    add('numoldage',ftinteger,0,false);
    add('numkills',ftinteger,0,false);
    add('numattacks',ftinteger,0,false);
    add('numgodwr',ftinteger,0,false);
    add('numpasses',ftinteger,0,false);
    add('numinstrs',ftfloat,0,false);
    add('eaten',ftinteger,0,false);
    add('mutadd',ftinteger,0,false);
    add('mutchange',ftinteger,0,false);
    add('mutsub',ftfloat,0,false);
    add('tottime',ftfloat,0,false);
  end;
  with mtbl.IndexDefs do begin
    add('','key',[ixprimary, ixunique]);
  end;
  mtbl.CreateTable;
  mtbl.open;

  mtbl.Append;
```

```
  mtbl['key']        := 0;
  mtbl['numdeaths']  := 0;
  mtbl['numbirths']  := 0;
  mtbl['numstarve']  := 0;
  mtbl['numoldage']  := 0;
  mtbl['numkills']   := 0;
  mtbl['numgodwr']   := 0;
  mtbl['numattacks'] := 0;
  mtbl['numpasses']  := 0;
  mtbl['numinstrs']  := 0;
  mtbl['eaten']      := 0;
  mtbl['tottime']    := 0;
  mtbl['mutadd']     := 0;
  mtbl['mutchange']  := 0;
  mtbl['mutsub']     := 0;
  mtbl.post;
  mtbl.close;


  habitat.InitHab('y:\database\habitat.dat',true);
  habitat.place(500,500);
  habitat.Writeout;
end;
```

Genetic Language

The GLanguage data structure contains the definition of the instructions described in section 5.2.

```
Const GLanguage : array[0..24] of TGInstDef = (
       (nr:  0; name: 'NOP'; param: false), // No operation
       (nr:  1; name: 'MOV'; param: false), // Move
       (nr:  2; name: 'EAT'; param: false), // Eat
       (nr:  3; name: 'TUL'; param: false), // Turn Left
       (nr:  4; name: 'TUR'; param: false), // Turn Right
       (nr:  5; name: 'CDR'; param: false), // Check Direction
       (nr:  6; name: 'SRE'; param: true),  // Set register <VALUE>
       (nr:  7; name: 'PUT'; param: true),  // Put mem <ADDRESS>
       (nr:  8; name: 'GET'; param: true),  // Get mem <ADDRESS>
       (nr:  9; name: 'SEE'; param: false), // See forward
       (nr: 10; name: 'CMP'; param: false), // Compare species
       (nr: 11; name: 'MEF'; param: false), // Measure forward
       (nr: 12; name: 'MEH'; param: false), // Measure here
       (nr: 13; name: 'ATK'; param: false), // Attack
       (nr: 14; name: 'ATD'; param: false), // Attacked
       (nr: 15; name: 'IFE'; param: true),  // If equal <to value>
       (nr: 16; name: 'IFL'; param: true),  // If less <than value>
       (nr: 17; name: 'IFM'; param: true),  // If more <than value>
       (nr: 18; name: 'JMP'; param: true),  // Jump <lines down>
       (nr: 19; name: 'SAY'; param: true),  // Say <number>
       (nr: 20; name: 'LSN'; param: false), // Listen
       (nr: 21; name: 'GEN'; param: false), // Get energy level
       (nr: 22; name: 'GMS'; param: false), // Get Memory size
       (nr: 23; name: 'GGS'; param: false), // Get genome size
       (nr: 24; name: 'ILL'; param: false));// Illegal operation
```

Increase Food

The incfood procedure handles the continuos and seasonal food increases.

```
Procedure Thabitat.IncFood;
var i,x,y:integer;
begin
  if foodadd > 0 then begin
    randomize;
    for i :=1 to foodadd do begin
      inc(welt[random(999),random(999)])
    end;
    inc(pdmstep);
    if pdmstep >= foodpdm then begin
      pdmstep := 0;
      for y:=0 to 999 do for x:=0 to 999 do begin
        if random(foodpdmth) = 0 then inc(welt[x,y]);
      end;
    end;
  end;
end;
```

Initialise Habitat

The inithab procedure sets up the habitat with its initial food content.

```
procedure Thabitat.InitHab(f:string; forcenew:boolean);
var x,y,s : integer;
    hf    : file;
begin
  fname := f;
  randomize;
  if fileexists(f) and (not forcenew) then begin
    assignfile(hf,f);
    reset(hf,1);
    blockread(hf,welt,1000*1000,s);
    closefile(hf);
  end else begin
    for y:=0 to 999 do for x:=0 to 999 do begin
      s:=random(habinitth + habinitmax);
      if s>=(habinitth-1) then s:=s-habinitth+1 else s:=0;
      welt[x,y] := s;
      poss[x,y] := false;
    end;
  end;
  pdmstep:=-100;
end;
```

APPENDIX B - FIGURES AND ILLUSTRATIONS

Figures:

Illustrations:

APPENDIX C - REFERENCES

(All URLs were checked and valid 2000-01-21)

Ashby, W. R. : (1962) Principles of the Self-organizing System (from Klir (edt, 1991): Facets of Systems Science New York, USA: Plenum Press).

Ashby, W. R. : (1956) An Introduction to Cybernetics (From (2000) Principia Cybernetica *http://pcp.vub.ac.be/books/IntroCyb.pdf*)

Checkland, P. et al: (1997) Soft Systems Methodology in Action (Chicester, England: John Wiley & Sons)

Crowley, A. : (1996) The Book of Lies (York Beach, USA: Samuel Weiser)

Darwin, C. : (1859) On the Origin of the Species by Means of Natural Selection (from Huxley: (1946) The Living Thoughts of Darwin. London, Great Britain: Cassell)

Foerster, H. : (1960) On Self-organizing Systems and their Environments. (from Foerster (edt, 1984): Observing Systems. Seaside, CA, USA: Intersystems Publications).

Heylighen, F. : (2000a) Principia Cybernetica Web Dictionary (*http://pespmc1.vub.ac.be/ASC/INDEXASC.html*)

Heylighen, F. : (2000b) What is Complexity? (*http://pespmc1.vub.ac.be/COMPLEXI.html*)

Miagkikh, V. V. : (2000) Soft Computing and Artificial Life (*http://web.cps.msu.edu/~miagkikh/SC_AL/index.html*)

Lucas, C. : (2000) Self-organizing Systems FAQ version 2.2 (http://www.calresco.force9.co.uk/sos/sosfaq.htm)

Miller, J. G. : (1978) Living Systems. (USA: McGraw-Hill)

Miller, G. A. : (2000) The Magical Number Seven (plus minus two) (*http://www.well.com/user/smalin/miller.html*)

Minsky, M. : (1997, incomplete reference from S. Turkle, see below)

Penrose, R. : (1990) The Emperor's New Mind (London, Great Britain: Vintage Books)

Ray, T. : (2000) Tierra Home Page (*http://www.hip.atr.co.jp/~ray/tierra/tierra.html*)

Turkle, S. : (1997) Life on the Screen (New York, USA: Touchstone)

Varela, F. et al: (1974) Autopoiesis: The Organization of Living Systems, Its Characterization and a Model (from Klir (edt, 1991): Facets of Systems Science, New York, USA Plenum Press)

Varela, F. et al: (1987) The Tree of Knowledge (Boston, MA, USA: New Science Library)

APPENDIX D - CORRESPONDENCE

If you, after having read all this, still have not tired entirely of me, I can be reached by the following means:

Snail mail: Joel Palmius
C/O ITK
Mitthögskolan
S-831 25 Östersund
Sweden

Email: joel.palmius@itk.mh.se

This and related documents (and source code) should be possible to reach through a sub-link of **http://www.itk.mh.se/~joepal/** (If they are not, please mail me).

APPENDIX E - DICTIONARY

Some of the concepts below are more elaborately discussed in section 4, "Definitions".

4GL, "Fourth Generation Language". This is the kind of tools that builds much of the code for the programmer, so the programmer can ignore much of the routine work. One example of this is IDE/RAD tools.

AE, Artificial Evolution, simulating evolution or using evolution in design.

AI, Artificial Intelligence, the discipline dedicated to making computers do things that would be considered intelligent if done by people.

AL, Artificial Life, the discipline dedicated to building things that would be considered alive if found in nature.

ALIFE, see AL

ARTIFICIAL LIFE, see AL

AUTOPOIESIS, autonomous self-reproduction and self-maintenance. A systemic definition of life (see LIFE).

BDE, Borland Database Engine, is the DBMS handling the Paradox tables in the demonstration.

COMPLEX, Having many subsystems and internal relations, having great variety.

COMPLEXITY, The number of subsystems and internal relations, the level of variety.

EVOLUTION, Descent with modification combined with selection.

GENE, see INSTRUCTION.

GENOME, the total set of instructions the creature possesses (for an example of a genome, see figure 6, "Species number 7795".

INSTRUCTION, One of 25 things a creature can do. For a description of the 25 available instructions, please see figure 2, "Genetic Code".

LIFE, when a system's only purpose is to uphold and produce itself, and when its subsystems participate in this task.

ORGANISATION, how everything relates to everything within a system.

PARADOX, is a file format for storing relational data.

SELF-ORGANISING SYSTEM, a system that without explicit pressure from the environment moves through its state space so that we deem its later states as more efficient or better than its earlier states.

SELF-ORGANISATION, to move from bad to good organisation without explicit pressure from the environment.

STATE, how a system and everything within that system relates to everything relevant to that system.

STATE SPACE, the total set of states, no matter how unlikely, available to a system.

SYSTEM, an abstract concept describing a selected set of related things and a boundary grouping them together.

APPENDIX F - INDEX