Looping as a Means to Survival: Playing Russian Roulette in a Harsh Environment

Robert Davidge
School of Cognitive and Computer Sciences
University of Sussex
Brighton BN1 9QH
England
robertd@uk.ac.susx.cogs

Abstract

The *von Neumann* architecture has long been the heart of all computers. Its apparently rigid structure has led to it being accused as the antithesis of the new approaches to programming led by Artificial Life. However, slight modifications of this structure allow us to produce a processor which behaves like a simple organism. A population of virtual processors have been designed to live in an environment consisting of a 2-D memory of machine instructions. The processors execute these instructions and thereby affect their own state and the state of the world they inhabit. This first experiment shows that in random conditions and playing a game of russian roulette, populations survive and display a specific strategy of looping forming clusters of organisms. We also see evidence for a phenotypic shift under strong selection.

Introduction

Every processor in every computer that you use today has basically the same architecture. This initial design [*Burks, Goldstine and Neumann*, 1946] has changed only in minor details and the material of its implementation in nearly fifty years. So despite all the alternative possible processor architectures, none has actually come close to rivaling the practicality of the *von Neumann* design.

We need also to remember the sheer beauty of this design - perhaps something we forget since we first were introduced to its innards in college. The prevalence of this design and the programming methods which have arisen around it are so ingrained in the way we now think of computers that it has led one commentator to place the blame for the patterns of our thought about computers and programming upon the *von Neumann* design [*Brooks*, 1991]. However, we are responsible for our own patterns of thought. If we choose to think in the frame of mind of a biologist then we can begin to see biological types of processes in the artificial creations that surround us. To this respect, the automobile has long been used as an analogy to the organism. We can extend this model to actually ask the question what would an automobile require to actually make it an organism. This line of speculation is very much in tune with Rodney Brooks's actual approach to designing robots to function as organisms.

By applying the biologist's way of viewing the world to computer systems, we can immediately adapt these "rigid" systems to more flexible, or robust ones (see for example [*Fontana*, 1992; *Rasmussen et al.*, 1990; *Ray*, 1992]). Tom Ray's *Tierra* has now become famous as an illustration of this approach of viewing the mechanistic world through the eyes of a biologist rather than an engineer. This approach though does not end with Tierra. All of the creations of computer science are now available for examination in the light of the biological revolution in the informational sciences. John Koza's use of the Lisp sub-tree mechanism is a beautiful example of the adaptation of existing computing methods to a biological analogy [*Koza*, 1990]

In Tom Ray's *Tierra* the organism is considered as an area of contiguous 'assembler' instructions which when executed by a processor act as simple self replicating loops. Using the special operating system of *Tierra*, the ancestor organism allocates memory space and copies its own description into this daughter area. Through mutation and sloppy replication the populations have developed many interesting forms of commensal living.

Using the same basic computer system as the starting point, we can shift our view from the programs formed in the memory to the processor itself [*Davidge*, 1992]. Now we consider the actual processor to be the organism and the memory of instructions to be its environment for exploration. A simple reorientation of aspect and the static computer processor requesting instructions down the memory bus can become an organism moving through the memory. It is the same procedure, but it entirely changes the way we regard the results and what we can do with a computer. We have released the constraints on our thought processes and consequently on our computers.

To be of any interest biologically, the memory must be 2-dimensional not 1-dimensional as in all standard stored-program computers. Whenever we execute a

normal program the instructions are strung together in a sequence and the processor works through them one by one. This order is only broken by Jump instructions which enable us to write loops entered or exited by conditional instructions. This ability makes the *von Neumann* design Turing equivalent, able to execute any program described by a particular Turing Machine [*Minsky*, 1967].

If we think of the execution point moving through the instructions as representing the position of the organism, then it is easy to see that we shall not get very interesting behaviours. The point will move in a straight line occasionally jumping through space and starting again. But if we get rid of the idea that the processor exists to execute *our* program then we can let it move in a 2-D or 3-D space of instructions and the motional behaviour will become a continuous track through space. Such motion is shown by all animals to some extent, but is especially reminiscent of those animals inhabiting a basically homogeneous world, e.g. earthworms or protozoa.

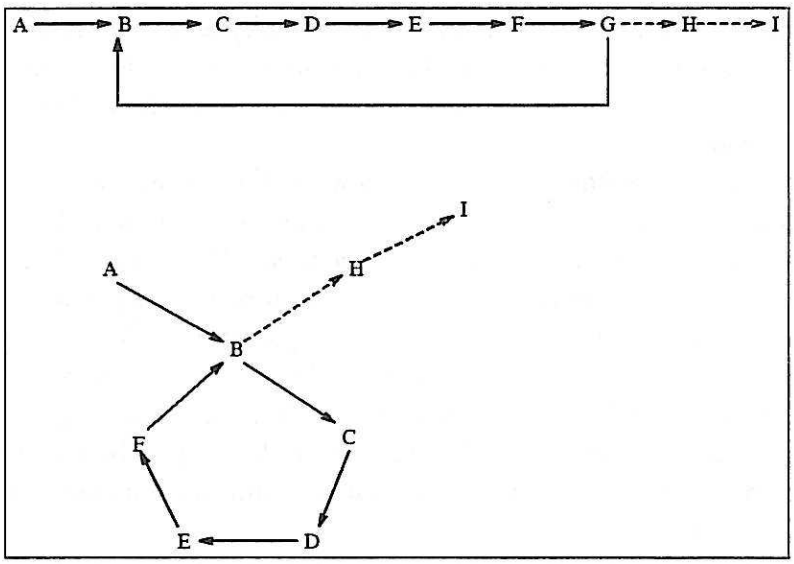

Figure 1: Turing Equivalence in 2-D

Of course, by removing the Jump instructions we have apparently lost our claim that this processor is still Turing Equivalent. Some might argue that this power was in some way essential to the quest for Artificial Life. Whether this is so or not, we can still demonstrate that the potential of this processor to execute the same program as one in 1-D is still present. Figure 1 shows the sequential program where B is the top of a while loop and G is the bottom. In 1-D the program is tested at B, passes through C-F and at G jumps back to B for retesting. In the 2-D track, the organism-processor passes in a *2-dimensional* loop from B through C-F and back to B where it is retested. One result of the test would be to carry on round the loop again and the other would be to alter its direction and pass to H. Note G is no longer needed, but B must have influence on the processor-organisms direction of travel. All this may be interesting to computer scientists, but it is irrelevant to the processor-organism. Only survival and possibly breeding is of consequence from its viewpoint.

The actions of the processor-organism are completely local. It fetches the next instruction and processes it. The effect of that instruction is entirely restricted to that processor-organism. Normally we can expect it to alter the state of its registers. If the organism can also read the environment directly into its registers, then the world can directly influence the state of those registers rather than just causing internal processing of the initial state. The same memory location can be regarded as both program and data. Here this is equivalent to both affecting the physiological process and actually forming the substrate for that physiological process.

The ability to write to the world is even more radical. Through this action the processor-organism is able to change its environment. That change is indicative of its own internal state. It has left its mark upon the world and through that mark the possibility of communication and co-evolution appears. Because of this a population of processor-organisms can interact indirectly with each other by changing the state of the world around them. Their actions have consequences for other processor-organisms passing through the same area at a later date. So just through the basic functions of a processor: fetching, execution, reading and writing we have the basis of behaviour for an organism displaying motion, physiology, ingestion of substrate, and action through egestion or secretion.

The environment itself has no power of its own. It is a receptacle for the processings of the organisms. But the processor-organisms themselves are controlled entirely by the instructions and data that they take in from the environment. Thus we can say that they are at the mercy of the environment, but rather it is better to say they are at the mercy of each other. Here then lies the potential for co-evolution, the possibility of competition or cooperation.

Finally I need to emphasize three things. Firstly the semantics of the processors and their organisms are entirely self-referential. Our view does not come into it other than how we create the initial structure of the system. These processors are not executing *useful* programs for us, but survival patterns for themselves.

Secondly, there is no natural analogy for resource limitation that can be carried out through local sensing in this model yet. Since this is assumed to be fundamental to the evolution of ecosystems, physiology and behaviour it will be interesting to see how far we can go without it.

Thirdly and lastly the processor-organisms need to be selected. In the experiment described selection is both through death and through sexual proclivity. This selection, though intrinsic [*Packard*, 1989], is still via '*the hand of god*'. It is not fully intrinsic in that it is not a dynamic part of the co-evolving system itself.

Anatomy of a Virtual Processor-organism

It is possible to design many variations on the basic *von Neumann* architecture. This design described here is the specific one used in the current implementation and is illustrated in Figure 2. It comprises two 4-bit registers, an ALU, a 2-D current address register, an instruction register with a special instruction-ID tag, special incrementers for altering the current address when fetching, reading or writing and a variant on a microprogram complete with microinstruction register and interpreter.

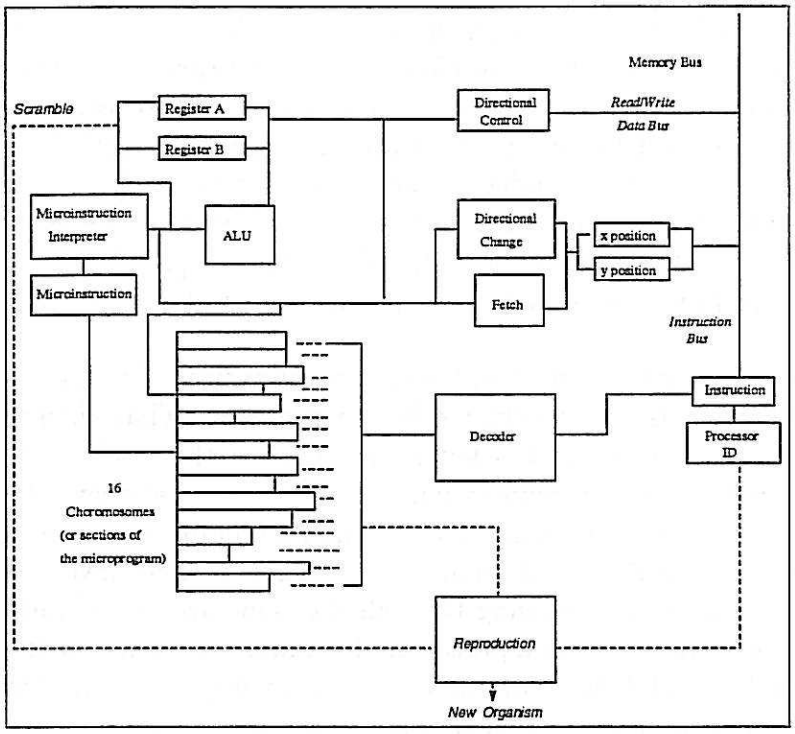

Figure 2: Processor Layout

The two 4-bit registers are implemented using modulo 16 arithmetic, e.g. 15 plus 1 becomes zero, with negative as well as positive numbers. The ALU is standard and has been taken from [*Cline*, 1981].

Each instruction fetched from the world is one byte long and the first 4 bits are used as the machine language for the processor. Each of the 16 codes causes execution of a microprogram interpreter to begin at the section of microprogram relevant to that code. When the end of the section is reached control returns to the fetch part of the processor cycle. Certain microinstructions act as conditional breaks upon the execution of the microprogram. They permit testing of the registers for zero or equality states and if satisfied prevent execution of the remainder of that section by returning control prematurely to the fetch cycle. These microinstructions allow the processor of 2 dimensions to retain its Turing equivalence as discussed in the introduction and illustrated in Figure 1.

Each section of the microprogram may be of any length, but in this experiment, using cloning, it is fixed at 10 microinstructions. We might like to think of these sections as 'chromosomes', so that the processors are organisms with 16 chromosomes!

Microinstructions are represented as bytes. The 8 bits are allocated into microorders which encode specific operations. For example, the first two bits determine whether the microinstruction refers to an ALU operation, a read/write operation, a change in the motion incrementer, or a conditional break. Within an ALU operation, say, the next 3 bits determine which particular operation is to be used and the 6th bit selects the register which will be updated. Thus, for example, the microinstruction 011110 adds registers A and B together and places the result in register A.

At each clock cycle one microinstruction is executed. Each processor is permitted one clock cycle in the parallel activity simulator and so every processor in the population progresses at the same rate executing microinstruction by microinstruction. But each processor will be at a different point in its microprogram dependent on its genetic makeup and past activities.

When the microinstruction interpreter has come to the end of a chromosome, the processor fetches another instruction to execute from the world. The instruction is fetched from its current position to which the processor has just moved by incrementing or decrementing either or both of its position registers which hold its x and y coordinates. The positional incrementers have the values -1, 0 or +1 and their current status is controlled by microinstructions which increment or decrement them. Thus when the new position of the processor is calculated just prior to the fetch, it may have moved in one of 8 compass directions.

A similar arrangement is used for reading from the environment or writing to it. The relevant microinstruction contains a 3 bit field which determines in which of 8 directions the read or write occurs from around the current position of the processor. The processor is both influenced by and influences only very local conditions.

When organism-processors reproduce they do so by the operating system allocating a new body with the same anatomy and position as its parent, and with random values in its registers.

Playing Russian Roulette

This model is highly maleable and many ideas can be tested within its framework in the future. The first experiment chosen was to investigate whether or not evolution could occur.

It is essential for evolutionary experiments to include variation and selection, but there is no inherent or truly intrinsic selection in the system. The processor-organisms do not compete for natural resources, nor do they prey one on the other. Also the processors do not age or die. Since we can find no natural population control within the system we must artificially introduce one.

In this experiment both death and reproduction are achieved when a processor has finished executing an instruction. If at that moment it has either the lucky or the unlucky number in its register A, then it either gives birth or dies. These two fateful numbers are chosen randomly at the beginning of the program. They must of course be different to allow organisms to either avoid death or become a parent of many offspring. The two numbers must also occur in the same register, otherwise processor-organisms would rapidly evolve that failed to operate on the death register and only operated on the birth register.

This set up seems to be a deadly game of Russian roulette for the processor-organisms. They can achieve immortality by never operating on the birth-death register, but though they influence their environment individually, they are sterile members of the evolving population. In order to chance reproducing, everyone must risk death.

In terms of computer science we can say that the selection is against programs which halt. So long as a program has not halted it may have a chance of reproduc-

ing either through its internal processing or its external actions. Once it achieves the global halting condition for all processors, it is removed. Thus we would hope for the evolution of processors which did not halt or for the evolution of 'lucky' processors which continue to hit the jackpot and spawn offspring processors.

The dynamics we might expect of such a system would be that they maintain a constant population size and average lifespan *unless* evolution of some sort had occurred. Since every processor in the initial population is given a random fixed-length genome; random values to its registers; and processes a randomly generated environment, we might expect that the death number would occur with the same frequency as the birth number, therefore maintaining the population to a relatively constant size. Indeed we could attempt to calculate the life expectancy of any processor. Given the proportion of microinstructions that affect the birth-death register, and the 1 in 31 chance of hitting the dread number. We would expect the average lifespan to be 31 divided by the proportion of 'active' microinstructions (those affecting register A). Significant deviation from this average lifespan or the initial population should then be an indicator of something other than random activity.

These conditions are extremely harsh — only the structure and operation of the processor being given and all else being random. We might expect the population to remain constant for many generations and if some deviation from normal were to occur, after a very long time, this could indicate that the seeds of evolution had appeared. If evolution was not going to be demonstrable, then no significant deviation would be seen within the running time of the experiment. The actual results came as a surprise which seemed at first inexplicable and perhaps due to a mistake, but on further investigation there seems to be a consistent explanation.

Twin basins

Every population examined so far has shown one of two general behaviours which are aspects of the same behaviour. From the initial starting population, which within the confines of our computing facilities is a maximum of 16,000, but was usually 4,000 or 8,000 then the population either dropped toward extinction, set at less than 20 remaining, or rose to its maximum permitted level say 16,000 or 32,000. More often than not it rose to the maximum and demonstrated an initial dip before rising (see Figure 3). Indeed this dip was common to both outcomes. Either the population recovered and went on to reach its maximum or failed to recover and stagnated or died out.

Phenotype Shift

This behaviour remained inexplicable until the register values were examined. If the phenomena were due to random causes then the distribution of values in the registers should be equal. No value should appear more often than any other value.

From Table 1 it is apparent that this is definitiely not the case. Indeed there is a large preponderance of zero with higher distributions of numbers close to it resembling a gaussian. By systematically changing the operators in the ALU it

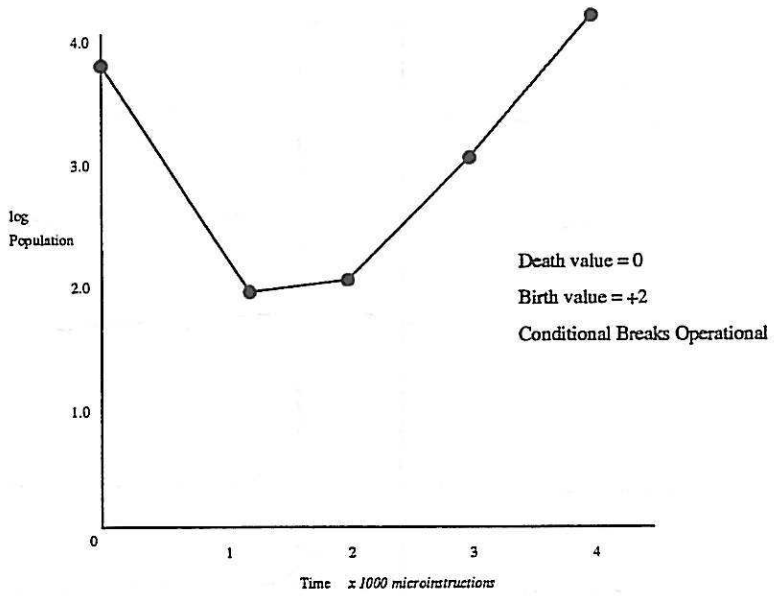

Figure 3: Population Change with Time

Register value	-15	...	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	...	15
% occurrence	0	...	0	2	4	56	12	5	2	2	1	1	0	1	0	...	0

Table 1: Distribution of Register Values without Selection Against Zero

could be demonstrated that they were responsible for the unusual distribution — most especially 'set zero', 'AND', 'shift right' and 'shift left'. Remember these operators are standard for a simple ALU (they were extracted from [*Cline*, 1981]).

Experimentation soon determined that the only populations not to succeed in reaching their maximum were those with the 'death value' set at zero or +1. We now have a means of introducing selection against the major phenotype of the population. What would happen when the 'death value' was set at zero and the 'birth value' varied?

Generally, if the 'birth value' was set beyond +7 or -7 then the phenotype of the population could not migrate downhill sufficiently far to reach the 'birth value' and cause a take off in reproduction. But if the 'birth value' was within this 'migrating range' then the phenotype would migrate toward the 'birth value' (see Figure 4). It was not necessary for the phenotype to reach the actual 'birth value', but rather to generate harmonic-like peaks leading off from the main migrating peak. These were sufficient to cause the population to eventually take off. This 'migrating range' within which the population succeeds and outside of which it stagnates is reminiscent of Langton's edge of chaos [*Langton*, 1990]. The migration down-hill away from the peak is a nice illustration of a converged population experiencing a change in the fitness landscape (see [*Harvey*, 1992 figure 7]).

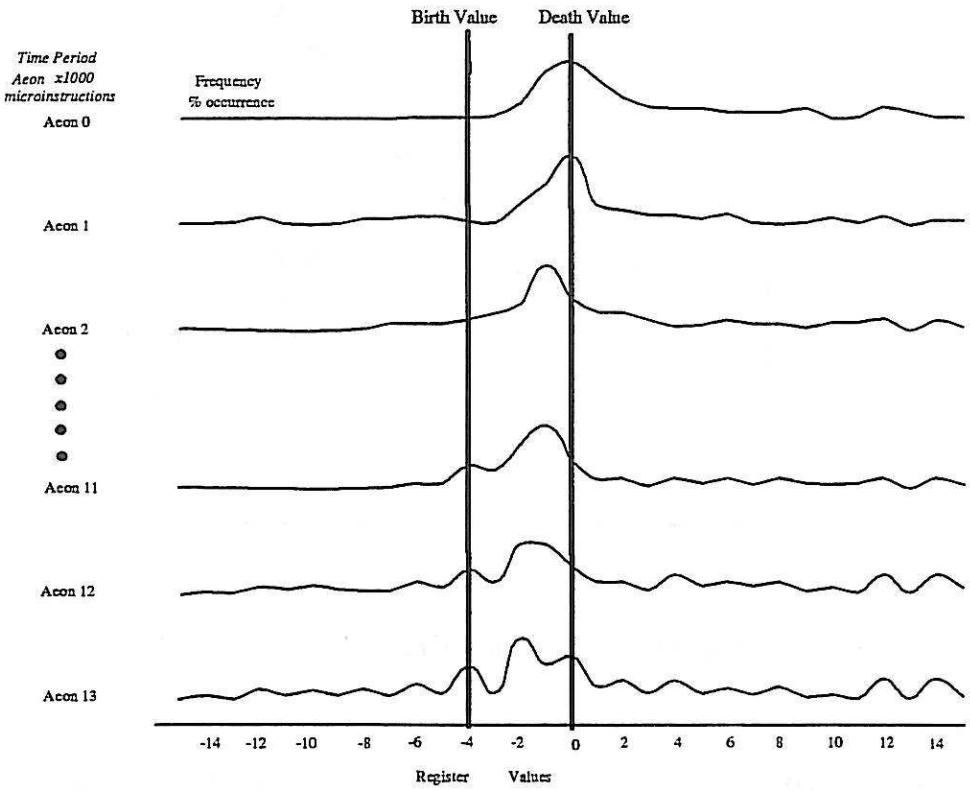

Figure 4: Migrating Phenotype: the mode of register values shifts from the death value towards the birth value

Behaviour at the Population Level

By measuring the average range traversed and the number of instructions executed before death (typically 10-200) it became apparent that the processors were turning rapidly within a short space — indeed they were looping to survive! All the populations, which displayed exponential growth, exhibited clustering as the only successful strategy for survival in this environment. Figures 5– 7 show a typical population changing from a random distribution to a clustered one of islands. In Figure 5 the organisms are distributed randomly across the environment and show up like waves upon a sea. Within one generation the basic pattern of islands in a barren sea has appeared (Figure 6) Each island represents a small colony of genetically similar individuals probably cloned from one ancestor. Figure 7 is taken from the final maximum population showing that the pattern of islands has not really changed only the size of the colonies.

This clustering behaviour is a function of localised birth. The offspring are given the same location as the parent but a random direction. If we set the offspring to be ejected into the environment randomly or even only a short distance away (radius of 5 steps) then generally the population fails to take off. Only in the most favourable conditions, e.g. 'birth value' set at +1, will the population survive despite this and then we see a random distribution of organisms rather than islands.

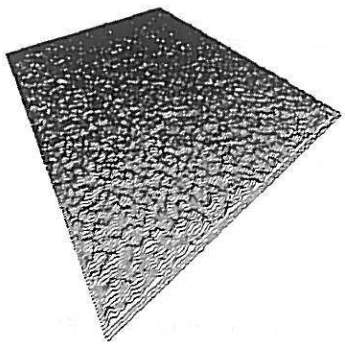

Figure 5: Initial Random Conditions

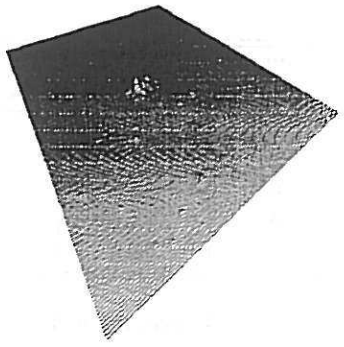

Figure 6: Rapid Formation of Islands

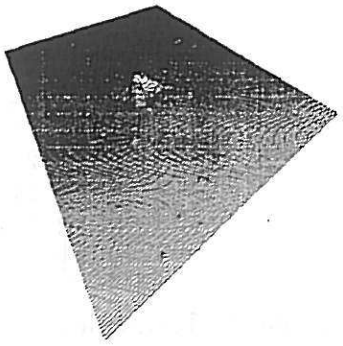

Figure 7: Growth of the Main Island

Aeon	Main Codes Used by Population														
	Population 1					Population 2									
2	0	2	3	5		0	1			5			11		15
3	0	2	3	5		0	1			5			11		15
4		2	3			0	1			5			11		15
5		2	3			0	1			5			11		15
6		2	3			0	1	4	5	6			14		15
11		2	3			0	1	4	5	6			14		15
12			3	5	8	0									15
13			3	5		0									15

Table 2: Change in Code Use by Two Populations over Time (Aeon = 1000 microinstructions)

Each population examined uses this basic strategy of islands to success. In any one population there are usually two or three islands with flourishing populations and perhaps 20-40 smaller islands of one or a few long lived individuals. However each population uses a different group of codes for its survival (see Table 2). This is as expected and constitutes an emergent relationship, a 'language of survival' between a population and its environment. All other behaviour is dependent upon the nature of the organisms and the cluster of codes they process.

The relationship between the clusters of organisms and their environments is remarkably robust. If the organisms had identified a particulary suitable niche within the environment, then when the environment is radically changed: such as being completely randomised after the population has shown some success, we would expect the population to crash dramatically. This is not the case, and though there is some set back it is relatively small and does not prevent the march to success. We can say then that the populations having been born into a random environement are completely adapted to such an environment and are not depending upon localised specific environments of adaptation.

If we look at genetic indicators at a gross level (Table 3) then we see little change in the distribution of genes governing ALU, movement, reading/writing or conditional breaking. However if we look at more detail within the genes respon-

Aeon	Frequency Percentage Occurrence			
	ALU	Read/Write	Movement	Conditional Breaks
0	25.2	25.1	25.2	24.4
1	25.0	25.2	25.3	24.5
11	24.8	29.8	23.9	21.9
12	25.0	31.0	24.2	19.8
13	27.1	26.0	24.3	22.6

Table 3: Frequency of Gene Groups Governing Basic Functions (Aeon = 1000 microinstructions)

sible for operations on the active register A either within the ALU or by reading from the environment then we see distinct changes within some groups of genes within the population. Frequently a gene will be pushed close to extinction reducing its frequency by up to 50 times its starting value. Which genes experience change cannot be predicted and it is not always the most expected — see Table 4 where in the first case 'set zero' is almost extinguished as might be expected, yet in the next case it actually increases in value and 'shift left' is reduced to near extinction.

Aeon	Frequency of Genes (%) Affecting Active Register																	
	Population 1									Population 2								
	0	-	<<	++	>>	&	+	Read		0	-	<<	++	>>	&	+	Read	
0	1.56	1.58	1.58	1.57	1.59	1.56	1.60	1.60	6.32	1.58	1.58	1.57	1.58	1.58	1.57	1.57	1.59	6.31
1	1.30	1.66	1.62	1.59	1.58	1.66	1.43	1.62	6.24	1.39	1.59	1.52	1.64	1.60	1.62	1.55	1.66	6.38
12	0.01	1.90	0.97	1.89	2.48	1.81	2.46	2.38	3.11	1.87	0.81	0.18	2.42	0.65	2.63	1.54	1.56	5.36
13	0.02	2.56	1.02	2.54	1.84	2.55	3.31	1.58	4.13	2.52	0.66	0.02	3.08	0.62	2.45	1.87	1.85	5.02

Table 4: Change in Frequency of Genes Affecting the Active Register (Aeon = 1000 microinstructions)

All these results have been shown using populations without the conditional breaks being operative. Such organisms can not be Turing-equivalent, but are in effect highly complex lookup tables. If we make the conditional breaks effective, then the same behaviour is shown, but only in the most favourable of conditions, such as 'birth value' set at -1 or +1. This increase in difficulty for the organisms to survive by rendering them potentially Turing-equivalent is as we would expect.

Behaviour at the level of the Individual

It remains only to look at the specific behaviour of a single processor functioning from the viewpoint of it as a computer. Looking at either the population level or the individual level cannot give us the full account of how ecosystems like this function any more than it can in Biology, but it gives us a limited explanation of what is a truly complex system.

Generally the organisms are to be found in tight loops such as the one illustrated in Figure 10, but we do also find organisms with a much wider range as in Figure 8. There are also examples of wanderers entering loops and others escaping from them.

Code	Microinstruction									
3	DEX	WRA	WRA	WRB	A&B	SLB	INA	ZTB	WRB	ZTB
14	ZTA	REB	REB	SLB	INX	EQT	INY	B|A	INY	INX
15	DEY	DEX	NEB	A|B	DEY	WRB	INX	ZTA	WRB	ZTA

Table 5: Relevant Chromosomes for Looping Organism

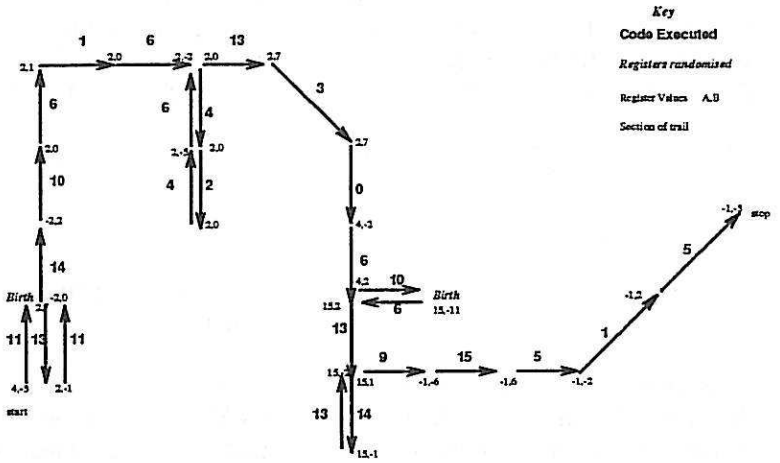

Figure 8: Wandering Organism

Figures 10 and 8 show the codes executed by each move and the values in registers A and B. From this and the microprogram it is possible to work out exactly what an organism is doing. The relevant sections of microprogram or genetic code for Figure 10 is found in Table 5 and that for Figure 8 is in Table 6. A key to the microcode is given in Table 7.In these examples the conditional break instructions were disabled, so they have no effect.

The loop in Figure 10 and Table 5 executes the following program: Mutation was by fixed probability of a copying error during reproduction and the resultant mutation was chosen randomly. The value used throughout the experiments was a 50% chance of a mutation somewhere in the genome. This is of the order suggested by [*Eigen et al.*, 1982]. There is also a copying error, by which some chromosomes are curtailed variants of their parents. This is a bug in the code, but a rather useful one!

Conclusion

I believe what has been shown here is evidence that despite these extremely harsh conditions, the processor-organisms have displayed survival under selective pressure against the dominant phenotype. The shift in phenotypic peak away from the dominant indicates evolution proceeding in a classical darwinian fashion. What was not expected was the rapidity with which these populations adapted and be-

Code	Microinstruction									
0	WRB	DEX	EQT	REB	WRA	SLA	NEA	DEX	REB	NEA
1	WRB	DEY	INY	DEY	WRA	ZTA	EQT	REB	INY	EQT
3	DEX	WRA	WRA	WRB	A&B	SLB	SRB	ZTB	WRB	ZTB
4	EQT	ZOB	WRB	EQT	WRA	EQT	EQT	WRB	WRB	REB
5	B&A	REB	DEY	SRB	WRB	DEY	EQT	REB	INY	B+A
6	DEY	DEY	DEX	WRB	SLB	DEX	INX	ZTB	REB	DEY
10	DEX	NEA	ZTB	ZTB	SLB	DEX	B&A	NEB	ZOB	SRB
11	NEA	ZTA	REB	WRB	REB	REB	B|A	DEX	INB	WRA
13	ZTA	INX	INX	ZTA	WRA	REB	REB	ZTB	WRA	DEY
14	ZTA	REB	REB	SLB	INX	EQT	INY	B|A	INY	INX
15	DEY	DEX	NEB	A|B	DEY	WRB	INX	ZTA	WRB	ZTA

Table 6: Relevant Chromosomes for Wandering Organism

Microin-struction	Explanation	Microin-struction	Explanation
A&B	A register becomes A AND B	DEX	Decrement x direction
B&A	B register becomes A AND B	DEY	Decrement y direction
B+A	B register becomes A Plus B	INA	Increment A register
B|A	B register becomes A OR B	INB	Increment B register
A|B	A register becomes A OR B	INX	Increment x direction
EQT	Test if both registers are equal	INY	Increment y direction
NEA	A register becomes minus A	SLA	Shift A reg 1 place left
NEB	B register becomes minus B	SLB	Shift B reg 1 place left
REB	Read from the environment into reg B	ZOB	Set register B to zero
SRB	Shift B register 1 place right	ZTA	Test A register is zero
WRA	Write the A reg to the environment	ZTB	Test B register is zero
WRB	Write the B reg to the environment		

Table 7: Explanation of Relevant Microinstructions

came successful.

It has been shown that under random conditions and with no concept of competition for resources a population of processors can survive and demonstrate indications of evolutionary processes underway. The strategy for survival is to form colonies from cloned offspring. These colonies are genetically similar, but not identical, i.e. a converged population. They exhibit a behaviour of looping over a short range executing a few codes. These codes are peculiar to the development of the population and their semantics are completely self-referential, i.e. important only to the colonies survival and having no meaning outside of this. The appearance of the islands is due to localised birth permitting similar organisms to expererience similar conditions. These populations are adapted to a random environment not to specific econiches and they are robust in the face of sweeping changes to their environment. It is easiest to see these effects on non Turing-equivalent organisms, but similar patterns can be found in populations of Turing-equivalent organisms given more favourable conditions.

```
DO
 A = A AND B
 B = Shift Left B
 Increment A
 Read into B
 B = Shift Left B
 B = A OR B
 A = A AND B
 B = Shift Left B
 Increment A
 B = -B
 A = A OR B
LOOP BACK TO DO
```

Figure 9: Program of an Organism Executing a Tight Loop

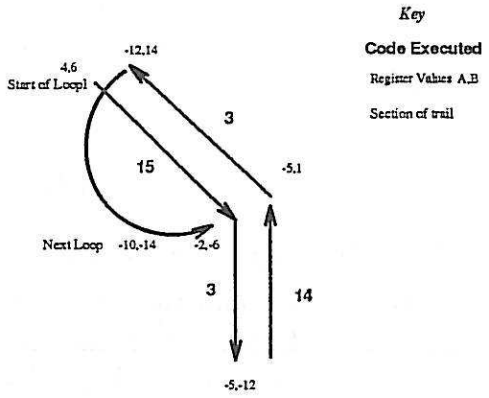

Figure 10: Organism executing a Tight Loop

It has not been shown what happens if the selective pressure is changed on an adapted population. Indeed removing the selective pressure against the dominant is likely to return the population to the original distribution. It is not possible within this system, due to the limits on the population size, to follow the phenotype shift from one peak to another stable dominant without the population springing back. If we had computer resources with two orders of magnitude more main memory than our current sun4 with 20 Mb, I believe we could follow this complete shift in the fitness landscape thereby demonstrating evolution in action. However, as it is, we must be content with an indication of evolution in progress.

Also I have not shown either the effect of sexual selection upon this population or the effects of different mutation rates. Indications are that neither of these affects the basic pattern of behaviour.

I am struck by, the at least superficial, resemblance of these patterns of behaviour to populations of bacteria on agar plates. Both have in common unlimited resources, localised reproduction, cloning with mutation and a stategy of restricted motility. This is at the population level. Previously I have discussed with Pedro Marijuan, by personal correspondence following the publication of [*Davidge*, 1992], the similarity of these processors to bacteria at the physiological level. In that paper I originally suggested the correspondence between the eukaryotic cell, or protozoan, and the processor. Pedro pointed out that the better correspondence would be to the prokaryotic cell, or bacterium. This experiment adds a little more weight to the view that we might have a very good model in the processor for the prokaryote.

In forthcoming work I intend to examine the effect on the populations when some form of limited energy consideration is introduced to the processor.

Acknowledgements

I would like to thank Phil Husbands, Tom Ray, Pedro Marijuan and Inman Harvey for suggestions and discussion. The 3-D display of population density and distribution is due to Homero Rios. The author is supported by an SERC Research Studentship.

References

Brooks, R. A., Intelligence without Reason, MIT, AI Memo No. 1293, April 1991.

Burks, A. W., H. H. Goldstine and J. von Neumann, Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, Institute for Advanced study, pt 1, vol 1, Princeton, NJ, 1946.

Cline, B. E., *Microprogramming Concepts and Techniques*, Petrocelli Books, 1981.

Davidge, R., Looking at Life, in *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, edited by F. J. Varela and P. Bourgine, MIT Press/Bradford Books, Cambridge, London, 1992.

Eigen, M., W. Gardiner, P. Schuster and R. Winkler-Oswatitsch, The Origin of Genetic Information, in *Evolution Now: A Century after Darwin*, edited by J. Maynard-Smith, pp. 10–33, W. H. Freeman, San Francisco, 1982.

Fontana, W., Digital Chemistry, in *Artificial Life: Proceedings of the second workshop on Artificial Life*, edited by C. G. Langton, J. D. Farmer, S. Rasmussen and C. Taylor, Addison-Wesley, 1992.

Harvey, I., Evolutionary Robotics and SAGA: the case for Hill Crawling and Tournament Selection, 1992.

Koza, J. R., Genetic Programming: A Paradigm for Genetically breeding Populations of Computer Programs to Solve Problems, STAN-CS-90-1314, Stanford University, 1990.

Langton, C. G., Computation at the Edge of Chaos: Phase Transitions and Emergent Computation, *Physica-D*, *42*, 12–37, 1990.

Minsky, M., *Computation: Finite and Infinite Machines*, Prentice-Hall, 1967.

Packard, N., Evolving Bugs in a Simulated Ecosystem, in *Artificial Life: Proceedings of the first workshop on Artificial Life*, edited by C. G. Langton, pp. 141–156, Addison-Wesley, 1989.

Rasmussen, S., C. Knudsen, R. Feldberg and M. Hindsholm, The Coreworld: Emergence and Evolution of Cooperative Structures in a Computational Chemistry, *Physica-D*, *42*, 111–134, 1990.

Ray, T. S., An Approach to the Synthesis of Artificial Life, in *Artificial Life: Proceedings of the second workshop on Artificial Life*, edited by C. G. Langton, J. D. Farmer, S. Rasmussen and C. Taylor, Addison-Wesley, 1992.