Emergent Behaviors in Artificial Life with Genetic Algorithms

John Hills

Department of Computer Sciences
Washington and Lee University
Supervisor: Professor Thomas Whaley
Submitted: May 26, 1998

Abstract

Artificial life is concerned with the understanding of various aspects of biological systems by artificially recreating them. An emergent property of any system is a simple or complex behavior that is beyond the original programming of that system; the whole is more than the sum of its parts. In this project, an artificial system was created to study emergence and other interesting questions associated with artificial life research. Results from various experiments were positive, demonstrating behaviors more suited to given environments would emerge in those environments. Artificial life research has a promising future in the understanding of evolution and intelligence in biological life. Where traditional artificial intelligence has failed, perhaps artificial life will eventually be the key to unlock the understanding of our own intelligence.

Table of Contents

I INTRODUCTION AND OVERVIEW OF KEY CONCEPTS 3

1 INTRODUCTION 3
2 ARTIFICIAL LIFE 4
2.1 History of Artificial Life *5*
2.2 Genotype / Phenotype Distinction *5*
2.3 Cellular Automata *6*
2.4 Emergence *6*
2.5 PolyWorld *7*
3 EVOLUTIONARY ALGORITHMS 7
3.1 Simple Genetic Algorithms *8*
3.2 Genetic Programming *9*
3.3 Classifier Systems *10*
4 ARTIFICIAL INTELLIGENCE 11
4.1 Artificial Neural Networks *12*

II EXPERIMENTS AND APPLICATIONS - A TIMED PROGRESSION 12

1 EXPERIMENT ONE - BEEPER FOLLOWING KARELS 12
1.1 First Implementation *12*
1.2 Results *13*
2 INTERACTIVE KARELS 14
2.1 Implementation *14*
2.2 Results *14*
3 ANNIMALS 14
3.1 Encoding Artificial Neural Networks *15*
3.2 The ANNimAL Brain *16*
3.3 Experiments and Results *16*

III OVERALL RESULTS AND FUTURE GOALS 18

1 FUTURE GOALS 18
2 CONCLUDING REMARKS 19

IV REFERENCES 20

V APPENDIX 22

VI ACKNOWLEDGEMENTS 23

VII TABLES, FIGURES AND PICTURES 24

I Introduction and Overview of Key Concepts

1 Introduction

The goals of this project were to research and understand various topics in artificial life and then to develop various applications to better understand and explore this relatively new subject. This paper will be dedicated to the understanding of some of the more important topics of this field and discussion of the implementation and results of each phase of the research. Before diving into the explanations of the individual topics studied in the course of this project, it would be wise to provide a general context in which to place each of the topics. Briefly, each phase of the research project will be described and the essential subjects identified. All descriptions will be brief and nonspecific, but should provide a general context for the research as a whole. Detailed explanations of each phase will be supplied later in this paper.

The first few weeks of research were spent in general study of genetic algorithms. After a good understanding was acquired, the principles of genetic algorithms were applied in a homemade application based partly on a popular computer science tutoring program, Karel the robot [23]. The program used a 50X50 grid containing a random path of beepers originating from the center, and a population of 200 robots. The robots were programmed initially with randomly encoded finite state machines allowing the robots to turn left or move forward based on their input. If they encountered a beeper, they would automatically pick it up. The goal of each robot was to collect as many beepers as possible in 200 turns. Those robots in each generation performing the best on this task were selected to mate with one another and populate the subsequent generation. The average performance of all the robots increased with each new generation. After a number of generations, the robots would usually find the best solution to the problem. This application was similar to one developed by Norman Packard which evolved bugs to follow food gradient trails [22]. While this application was less complicated, it was a good starting point and clearly displayed the power associated with genetic algorithms. After a series of similar experiments, a more complex application was developed to see the extent of what genetic algorithms could accomplish.

The next application maintained many of the same characteristics of the first, but was more typical of artificial life projects developed today. The environment was still a 50X50 grid, but instead of creating a path of beepers for the robots to follow, the beepers were randomly distributed throughout the grid. In the first experiment, the robots were tested independently of one another; but in the second, all the robots inhabited the environment at the same time. The robots were faced not only with the problem of finding the beepers, but they were also competing with their fellow robots for the beepers. Before, each robot's fitness was a function of how many beepers it could collect in 200 turns. Now, the fitness of every robot was more closely associated with its ability to survive. The robot had to expend a certain amount of energy in order to move, and beepers replenished this energy. The more beepers the robot found, the more energy it had, and the greater its fitness was. Robots finding no beepers eventually ran out of energy and died. Finally, the way the robots mated was altered. In the prior experiment, the most genetically fit robots were selected to mate. In the second experiment, the robots mated only when two occupied the same square on the grid.

After running the system, the optimal solution for the environment was quickly found. For this particular environment, the best solution was for the robots to move forward while occasionally turning left. This strategy exposed them to the most beepers (food), and the most possible mates. Unfortunately, the way the robot finite state machine brains were encoded did not allow for further expansion of their input (perception) or outputs (behaviors) beyond this simple system.

Most of the concepts uncovered with the transition from the first experiment to the second had to deal with the basic principles of artificial life. For example, in the second experiment, the system became more like a real biological system. However, many of the variables associated with both the environment and the robots themselves were still being manually manipulated. The eventual goal of an artificial life system is to eliminate all of the external variable control and allow the system to determine the optimal settings for each of its parameters.

The final phase of the project would encompass all the features of the previous systems as well as some additional elements from the study of artificial intelligence. With a finite state machine, it was not possible to create brains for the creatures that were very complex. By replacing these finite state machine brains with artificial neural network structures (ANNs), the complexity of the whole system was increased dramatically. One problem inherent to the use of artificial neural networks (or any other complex system) with genetic algorithms is deriving a method with which to encode a network so that it can be combined with other networks (bred with other robots) and still retain some of its functionality. For example, how do we take two independent robots and combine their brains in such a way that the offspring has the opportunity to benefit from behaviors exhibited by either parent. Solutions to this problem will be discussed in II.3.1.

One important concept in artificial life is the distinction between the *genotype* and the *phenotype*. In biological life, the genotype is analogous to the DNA, and the phenotype is analogous to the animal grown from that DNA. The animal is not just a function of the DNA, but of environmental factors, mutational factors, and a host of other influences through its development. Genetic algorithms, as might be implied by the name, work in much the same way that biological evolution works on the DNA. After genetic algorithms do their job on the genotype of an artificial creature, some function must then convert that genotype into the phenotype (the creature itself), and it must do so in such a way that a given genotype may not always produce exactly the same phenotype.

Because the creatures now possessed artificial neural network brains, they were dubbed "ANNimALs" (Artificial Neural Network, Artificial Life). These ANNimALs were given additional perceptive ability. They were able to sense any food or other ANNimALs present one cell in every direction. They were also given the ability to control more of their own behavior. They were no longer told when to eat or when to mate, as these actions would be emergent behaviors. As discussed below, these changes imposed some interesting problems and questions, and eventually sparked even more changes within the system.

2 Artificial Life

Artificial life (alife) is concerned with modeling biological evolution in silicon based machines; it is the study of artificial machines that behave like biological creatures. With roots in computer science, artificial intelligence, mathematics and biology, artificial life aims for a better understanding of the biological and theoretical basis of evolution and life. By modeling life within a machine, we can twist and pry into an artificial creature much more easily than we could its biological counterpart. And by simulation of thousands of generations in a few minutes, we can actively observe and study evolution first hand [16].

Biology has always been concerned with the study of carbon based life. The reason biologists have only looked at carbon based life is because this was the only type of life known to exist. However, the ability to model biological evolution in computers gives us not only the ability to study evolution first hand, but also allows us to take a much closer look into the internal systems of these creatures than we could get from a monkey or a fruit fly [16]. Alife is not concerned with the dissection and study of the parts of life as biology has tended towards. Rather, alife studies the complexity and behavior of systems made up of known parts.

Artificial life is very similar to artificial intelligence in that its goal is the same. Artificial intelligence originally set out to create a computer that could think like a human. Traditional artificial intelligence has gone wrong in that the field has continuously tried to model intelligence from the top down. Scientists would look at the brain, poke at it, make models of it, and then try to duplicate it within a machine. This has provided slight advancements over the years, but has fallen seriously short of its original goal. Artificial life takes the opposite approach to the same problem. Where artificial intelligence attacked the problem from the top down, artificial life attacks it from the bottom up. By starting with the simplest examples of intelligence and evolving them over generations, scientists hope to evolve intelligence within a machine. Just as intelligence is an emergent behavior within humans and other life, so should it be within a machine.

By simulating (very roughly) biological evolution, artificial creatures can be developed. The creatures are initially provided only the tools through which to evolve. They are given no initial instructions or behaviors at all; evolution is used to develop their systems in such a way that the creatures are suited to their environment. Instead of trying to model a frog by explicitly telling it what to do in any given situation, we put the frog in an artificial environment and give it the ability to evolve a survival strategy. Through many generations of frogs, only the fittest of each generation survive to pass their genetic code to the next generation. New, unique strategies are created when these genetic codes are combined and mutated. Every generation of frogs is better than the last, until an optimal behavioral strategy is found. This concept can be expanded to encompass a range of possible problems.

2.1 History of Artificial Life

Although a new era of artificial life has emerged from the computer age, the concept of artificial life is not a new one. With the development of clock making came the first advances in the development of artificial life. The earliest examples can be seen in the animated clocks or "Jacks", which would display lifelike movement of a man striking a bell every hour. Later, in 1735, Jacques de Vaucanson created his famous Vaucanson duck. According to spectators, this mechanical duck which was comprised of thousands of pieces amazed spectators with its lifelike behavior [17]. The replication of life has always intrigued and fascinated man, but it is only recently that we have been able to replicate life within electronic media, bringing artificial life that much closer to reality.

Progressing into the 1950's, John von Neumann was the first to think of the idea of the self-replicating machine. He theorized that a machine could exist which would contain an internal representation of itself, and the ability to build duplicates based on this internal representation. It could then copy the internal representation and bestow upon a duplicate this same ability [17]. This theory sparked new interest in the field, and led to his later discovery of *cellular automata.* Eventually, *evolutionary algorithms* such as *genetic algorithms* were developed, and the study of artificial life entered a new phase of existence.

2.2 Genotype / Phenotype Distinction

It is important to understand the distinction between the genotype and the phenotype. In biological systems, the genotype is the genetic code for that system. The phenotype is the system itself, having derived its characteristics from the genotype. In animals, the genotype is synonymous with the DNA, and the phenotype is the animal itself. A phenotype is grown from the genetic instructions within the genotype. There are often other factors such as mutation and the environment that also play a role in deriving the phenotype. Because of this, a given genotype may not always produce the same phenotype. However, the genotype should provide the instructions on which the basic characteristics of the phenotype are based [16].

This distinction poses an interesting problem to researchers in artificial life. Genetic operators associated with evolutionary algorithms operate on the genotype. If the genotype itself represents the solution to the problem such as in the example in 3.1, then the genotype and phenotype are the same. By eliminating this distinction, more precise control can be given to the genetic operators. But one of the primary goals of artificial life is to approximate true biological systems as closely as possible, so the genotype / phenotype distinction must be made. In most of the current artificial life systems, the evolutionary algorithms are put to work on the genotype of the creature. After these processes are complete, the phenotype (the creature itself) must be developed from the genotype. The phenotype will interact with the environment and determine the fitness of the genotype. After the development of the phenotype, the only time the genotype will be used again is to donate parts of its genetic makeup during reproduction.

Recursively generated objects describe a category of systems taking advantage of this genotype / phenotype distinction. Some of the systems in this category include *cellular automata* (see 2.3) and *Lindenmayer systems* [17]. Lindenmayer systems are recursively generated rule based systems similar to Chomsky's grammars. Starting with an initial seed and a set of replacement rules, a unique string can be generated by recursively replacing symbols with other symbols. The rules of these systems represent the genotype, and the final string derived by the rules represent the phenotype (see Figure 5).

There are many different research areas in artificial life. Some take advantage of this phenotype / genotype distinction and some do not. In the first two experiments described in this paper (II.1, II.2), there was no distinction between the genotype and phenotype. The genotype represented the exact encoding of a finite state machine; the genotype itself represented a solution to a problem so no phenotype could be derived. More recent approaches to the development of artificial ecosystems such as *PolyWorld* (see 2.5) use a genotype string to represent the DNA of the creatures and then use a specific function to develop the phenotype. This is the type of system used in the last experiment described in II.3.

2.3 Cellular Automata

Generalized cellular automata (CA) can be described as a lattice or grid of individual cells, each of which can possess a finite number of states. Each cell in the grid interacts by local functions or laws in some way with its neighboring cells causing state changes from one iteration of the system to the next. The current state of a cell and its neighboring cells effect that cells state in the next iteration, and all cells are updated concurrently. Cellular automata can be *n* dimensional, so they can have great complexity. These systems are used in a range of applications from image processing to modeling ant behavior [24]. Self-replicating cellular automata have also been discovered by Christopher Langton. His (shown in Figure 4) is the simplest example of a self-replicating cellular automata system [16].

Perhaps the most well known example of cellular automata is John Conway's game of life. This game is played out on a two-dimensional toroidal grid where each cell in the grid is designated alive or dead. If a space is alive and has too many neighbors, it suffocates and dies; if a space is alive and has too few neighbors, it gets lonely and dies; if a space is dead, and has exactly three neighbors, it is born. Developing these basic rules was a huge step forward for cellular automata, and greatly simplified the complex replicating machine von Neumann had earlier supposed. Conway's game of life is simple enough to be played on a checkerboard by hand, and yet is powerful enough to mimic all the necessary logic operators necessary to represent a universal computer [17].

2.4 Emergence

Emergence is one of the most interesting properties of artificial life systems. An emergent behavior is one that is more than the sum of its programming. Artificial creatures have the ability to perceive the world around them and to sense their internal states; based on this information they can act in a certain way. Initially, the way these creatures act in any given situation is random; there is no effort on the part of the programmer to instruct the creature what to do in a given situation. But as they live out their lives, only the ones with the best initial behaviors will prosper, living on to mate with other survivors. Over generations as the creatures evolve, certain behaviors will emerge for given situations. The creatures are never told to eat when standing on food, but that behavior will emerge in the population because only those creatures that do eat will survive to mate. When we create artificial life, the goal is to develop a system in which the behaviors of the system emerge. Programming behaviors into the system initially provides little insight into the process of evolution.

While the previous example is very simple, emergence can become extremely complex. The more complexity there is in the artificial system and creatures, the more complexity there will be in the behaviors that emerge. In the systems described in this paper, very simple emergent behaviors were observed. These behaviors were very consistent with expectations based on the initial environment. If the environment were more complex, more sophisticated behaviors would become positive survival characteristics and would emerge from the system.

2.5 PolyWorld

Developed by Larry Yaeger, PolyWorld is an excellent example of the complexity of current artificial life systems [27]. Within this system emerged characteristics such as basic communication, predator prey relationships, as well as other behaviors associated with biological life. PolyWorld was an excellent model for the development of the artificial systems described in this paper. While the incredible complexity of PolyWorld was not achieved, many of the more interesting aspects of the system such as the neural network architecture were utilized and expanded upon.

PolyWorld's environment is a relatively simple flat space containing the PolyWorld organisms and food for them to replenish their energies. The organisms can metabolize energy, process visual input from their point of view in the world, communicate through control of their own appearance, and reproduce with other PolyWorld creatures. Because of this wide range of built in abilities, the system could optimize itself in a variety of ways allowing for a great deal of behavioral strategies to surface. There was no specific fitness function to determine the most genetically fit individual of the system; rather the system was based on survival of the fittest. Those individuals expressing the best strategies would survive longer and be more likely to pass their genes onto the next generation.

3 Evolutionary Algorithms

Evolutionary algorithms (EAs) are computational models that use elements of biological evolution in their implementation [2,10]. Evolutionary algorithms model biological systems to quickly search large solution spaces for solutions to various problems. Starting with an initial population of possible solutions for a problem (the first generation), new generations are created through a process of natural selection. Each successive generation is better than the last; only the fittest of each generation survive to propagate the next. The first generation of possible solutions is created randomly. A *fitness function* is used to derive a relative fitness for each solution. A *selection* method is then used, considering each solution's fitness, to select those solutions that will propagate the next generation.

The three primary operators used in the propagation of a new generation are *recombination*, *mutation*, and *selection*. Recombination takes two *parent* solutions and creates a third *child* solution. Recombination is probably the most important of these operators. It can be thought of as the mating operator; by combining parts of two solutions, there is a chance the child will be better than either of its parents. Mutation simulates gene mutation that exists within real biological systems. This operator is very important within evolutionary algorithms because it can sometimes make a slight change to a solution that makes it superior to any of its competitors.

There are many variations of evolutionary algorithms. The five most commonly studied EAs are *evolution strategies*, *evolutionary programming*, *genetic algorithms*, *genetic programming*, and *classifier systems*. All of these systems are very intriguing to consider; however, only the last three will be addressed in this paper. Furthermore, genetic algorithms are the primary interest of this study and receive most of the attention.

3.1 Simple Genetic Algorithms

Conceived by John Holland in the early 1960's, genetic algorithms are probably the most widely known of the evolutionary algorithms [11]. Simple genetic algorithms are primarily concerned with fixed length binary strings (the genotype). Each of these strings can be thought of as a possible solution to a given problem. For example, assume the problem is to find the maximum binary value for a six character binary string (shown in Table 1). A very bad solution to this problem would be 001101, as this string only has a value of 13. However, the solution string 110100 is a much better solution with the value 52. The very best solution to this problem is 111111, where the value is 63. The goal of applying the genetic algorithm to this problem would be to find this maximum valued string, or at least a high valued string. One problem with genetic algorithms is that the best solution to a given problem may never be found; however, a reasonably good solution is usually found. In the example in Table 1, after only three generations the average fitness has increased from 30 to 51, the best being 55. Continuing to compute more generations would most likely increase the average fitness and produce an even better maximum valued solution string [9].

The fitness function of a genetic algorithm evaluates the solution strings in each generation. This function is designed to give some value indicating how well a given solution solves the problem. For instance, in Table 1, the fitness function takes each binary number and returns its decimal value. The higher the decimal value, the more fit the solution is at solving the given problem. After each generation, a percentage of solutions with the highest fitness values are kept. The genetic operators are then applied to produce the next generation.

Crossover is the primary genetic operator. When two solution strings are crossed over, different substrings of each parent combine to create a child string. Only the most genetically fit chromosomes of each generation are selected to be parents for the next generation. There are several different methods of crossover, but the most common method is *single point crossover*. Other types of crossover are *multi-point crossover* and *uniform crossover*.

In Single point crossover, shown in Figure 1, a random crossover point is selected on the chromosome. Parent A gives all of its characters up to the crossover point to the first child A, and all the characters after that point to child B. Parent B then gives the characters opposite to those that the first parent gave, filling in the gaps in each child. The two children chromosomes now have some of the characters (genetic information) from each of the parent chromosomes. This type of crossover was what primarily interested John Holland and is the basis for his *schema theory*.

Single point crossover does not provide optimal performance in all cases. One problem lies in *positional bias;* a bit is more likely to be exchanged as it nears either end of the chromosome [2]. One solution to this problem is to implement a multi-point crossover. Multi-point uses a random number *z* of crossover points {C_1, C_2, …, C_z}. The child chromosomes receive their bits in sections alternating at each crossover point. Additional crossover types such as *punctuated crossover* attempt to actually save C and *z* into the chromosome, allowing the algorithm to actually evolve the type of crossover most fit for the problem.

Uniform crossover is probably considered the best of any mentioned here. In this technique, a crossover point is inserted between every bit. Thus, for any position it its chromosome, a child could receive a bit from either of its two parents. It is difficult to know what type of crossover is appropriate for a given situation. However, because the experiments described later in this paper deal primarily with emergence and not with performance, Holland's original single point crossover was used in all the experiments.

The mutation operator is generally applied to only a small percentage of each population every generation. The percentage is typically around 0.01 percent, however this can vary. There are various types of mutations that can be applied if a chromosome is selected for mutation. In a binary chromosome such as in Figure 1, a mutation operator would randomly select and invert one bit in the string. The mutation operator applied depends on the characteristics and size of the chromosome and the size of the population.

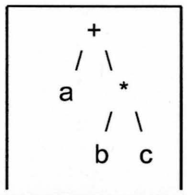
Before Holland's work, most scientists believed that the mutation operator was responsible for the majority of evolutionary progress. However, Holland realized that something more than mutation had to be the driving force of evolution. Mutation was rare at best in actual biological reproduction, occurring approximately once in every ten million genes [17]. With his schema theorem, he was able to show the power of crossover. If solution strings are thought of as chromosomes and its substrings as alleles within the chromosome, then some interesting things can be shown about the effect of crossover on these alleles. Holland theorized that because crossover occurs at a random point within the chromosome, blocks of information closer together in the chromosome are more likely to retain their integrity through crossover. Therefore, blocks of information responsible for the chromosome being more fit will tend to transcend from one generation into the next. Holland used a character set of {0,1,*} to represent strings in his schema theorem (* can match either 0 or 1). Some examples in a four character chromosome with one character alleles are, 1**1, 00**, 1110, and ***1. Holland's theorem would postulate that the first schema example, 1**1, would be less likely to hold up over several generations than would the second (00**). The two 1's in the first example would be more likely to be separated due to crossover than would the two 0's in the second.

Genetic algorithms primarily deal with strings of a constant length. However, it has been proposed that in order to evolve continuously emergent behaviors such as in biological life, the genotype (solution string) must be able to grow. For the purposes of the research described below, all the experiments have been created with constant length strings. Doing so imposes a limit on the eventual evolution of the system. However, making the string sufficiently long allows for growth well beyond what could be achieved in the environment. The constant length strings used in this experiment do not impose any real restriction on the growth of the system. Relying on this type of fixed chromosome also allows for a better evaluation and implementation of Holland's original simple genetic algorithms.

3.2 Genetic Programming

Genetic programming is the brainchild of John Koza [14,15]. Genetic programming differs from genetic algorithms in how they represent the solution to the problem. Where genetic algorithms use binary strings as genotypes to represent solutions, genetic programming uses Lisp or Scheme programs to give a solution to the problem. A solution's fitness is determined by how well it solves the problem. The best programs in each generation undergo crossover and mutation to populate subsequent generations.

The programs themselves can be thought of as trees of *terminals* and *functions*. Terminals are the variables of the program, and the functions are operations applied to the terminals. A program could then be as simple as "a + b * c". The terminals are a, b and c, and the functions are + and *. A tree could then be built to represent this program [7].



Lisp and Scheme are very popular among those who use genetic programming because they allow easy representation for these types of programs. Complex trees can form quickly with any complexity in the problem. Luckily, any tree can be represented as a string (with parentheses) and visa versa. Because of this, with slight modification, the same genetic operators we use in genetic algorithms can be used in genetic programming.

The fitness function in genetic programming can be relatively simple. The fitness function evaluates how well a given program (solution) solves a problem. If the outcome of the generated program can be evaluated simply, then the fitness function will be simple. For example, if the program is being evolved to be able to aim a cannon to hit a target, the fitness of each solution will be how close it came to hitting the target. On the other hand, some additional computation may be required to compute the fitness of programs with more complex outcomes. Genetic programming is best suited to problems with a large number of parameters and a simple fitness function.

Crossover can be thought of as the transposition of two branches, one from each parent, to create two new children. First, exact duplicates of each parent tree are created. As in Figure 3, one branch from each new tree is selected at random and switched from one new tree to the other. Because of the tree structure of the solutions, it is now possible to perform an asexual crossover producing new offspring. Two branches from one parent can be transposed to create a new child. In genetic algorithms this is not possible because a single parent will always produce identical offspring. It is important to note that the crossovers being performed always have at the root node similar character types. Either both will be function types, or both will be terminals. Violating this rule will produce erroneous programs.

There are two variations of mutations associated with genetic programming. The first is when a terminal or function is replaced by a random character of the same type. In the second, an entire branch is copied to replace another branch in the tree. Just as in genetic algorithms, mutation does not play as large a role as crossover, but it is essential to avoid local maxima in our solution space.

3.3 Classifier Systems

Also the discovery of John Holland, classifier systems represent one of the earliest implementations of genetic algorithms. They are best suited to adapting behaviors and strategies to a changing environment and so are very popular with those studying artificial life [10]. A classic example of a classifier system is an artificial frog. The goal of the frog would be to learn to catch and eat food while avoiding predators. The essential elements of any classifier system are: an artificial environment, ability to perceive the environment, ability to manipulate the environment, and an artificial creature which exhibits behaviors by manipulating the environment based on perception. In our frog example, the frog can perceive what bugs (prey) or crocodiles (predators) might be around it at any given time. The frog has the ability to kick its back legs (one at a time or both at once), turn its head to the left or the right, and stick out its tongue. Based on what the frog perceives, it will act a given way. The goal of the classifier system is to have the frog learn the optimal behaviors for any given situation. A frog that acts more appropriately to more situations will be better off than one that acts inappropriately (jumps toward its predator instead of away).

A classifier system can be thought of as a series of conditional statements. If the frog perceives a given condition, then the frog will act in a certain way.

IF	small, flying object to the left	THEN	send @
IF	small, flying object to the right	THEN	send %
IF	small, flying object centered	THEN	send $
IF	large, looming object	THEN	send !
IF	no large, looming object	THEN	send *
IF	* and @	THEN	move head 15 degrees left
IF	* and %	THEN	move head 15 degrees right
IF	* and $	THEN	move in direction head pointing
IF	!	THEN	move rapidly away from direction head pointing

This would be a possible optimal solution to our frog classifier system. It has learned to face and follow small flying objects while fleeing from large looming objects, generally a pretty good strategy for a frog. The encoding of a classifier system would look rather similar to a genetic algorithm encoding.

IF	THEN
0000, 00 00	00 00
0000, 00 01	00 01
0000, 00 10	00 10
1111, 01 ##	11 11
~ 1111, 01 ##	10 00
1000, 00 00	01 00
1000, 00 01	01 01
1000, 00 10	01 10
1111, ## ##	01 11

Example taken from the Evolutionary Computation FAQ [10]

Stringing all of these if...then strings one after another would give a chromosome of a fixed length. Decoding the chromosome would just be a matter of separating it back into the 12 character alleles such as show above. The chromosome would be filled initially with random bits and would, over many generations, converge toward a good (perhaps optimal) strategy for the frog.

One thing to notice about these systems is that they may not initially account for every possible situation; it could be the case that in a given situation, a frog would have absolutely no response. Desired behaviors help the frog survive to populate future generations. Negative behaviors most likely lead to the extinction of the frog. Additional features to these systems allow them to generalize. As seen above, the # symbol is a wildcard allowing for any variation of the characters. Another addition is the ~ (not) operator allowing negation of the characters.

Classifier systems were not directly implemented in any experiments described in this paper. However, they are still an exciting research area in artificial life and will definitely be a topic of future interest and study. The method of encoding implemented in the first experiment was perhaps less efficient than a classifier system, but as will be shown later, it accounted immediately for every possible situation.

4 Artificial Intelligence

The traditional approach to artificial intelligence is concerned with a top down representation of intelligence. Actual processes or systems were modeled in hopes these systems would somehow demonstrate intelligence. Artificial neural networks (discussed in 4.1) are a newer approach to model the biological process in animal brains, but they still lack the complexity to accurately model this process. Top down approaches have been successful in very limited aspects of artificial intelligence, but have not been able to completely model any aspect of biological life. Artificial life approaches this problem from the bottom up. Instead of trying to model an existing system, scientists develop a system that has the ability to develop its own processes. By allowing these processes to interact and evolve, intelligence could then be an emergent property of a system.

4.1 Artificial Neural Networks

Artificial neural networks (ANNs) are rough mathematical models of the animal brain. Using a vast array of connected nodes artificial neural networks are able to derive an output state based on given input. Because they are able to generalize input to come up with an output state, they are excellent for problems likely to have degraded input. There are many different types of networks and several learning strategies with which to train networks. They are currently implemented in countless applications from image processing and speech recognition to stock market forecasts. While the complexity of the artificial neural networks used today pale in comparison with that of their biological counterparts, it is the hope that the future will bring these systems continuously closer to accurately simulating the intricate interactions within the human brain.

A typical neural network has a number of nodes with specific connections to other nodes within the network. Each one of these connections has an associated connection weight. The input is fed into the nodes in the input layer of the network, which in turn create activation in every node to which they are connected; the stronger the connection strength is, the stronger the activation will be. If the activation in a node exceeds a threshold value called a bias, that node becomes active and will activate all nodes to which it is connected. These newly activated nodes go on to activate all the nodes with which they are connected and so on through the network. Eventually all the activation values in the network will stabilize. At this point, the activation that has filtered through to the output layer of the network will dictate the solution of the network [3,19,26]. Figure 7 is an example of an artificial neural network. While this network is very simple, it can easily be seen how the values from the input layer propagate through the network to decide on a given output.

II Experiments and Applications - A Timed Progression

1 Experiment One - Beeper Following Karels

The first experiment incorporated a few key topics of artificial life, yet it was simple enough to research and complete in a reasonable amount of time. It was difficult to decide exactly where to begin, but the project would include an artificial environment of some type and creatures that would learn through evolutionary selection and genetic algorithms. It was decided the first step should be to recreate an experiment done by an artificial life team at UCLA [12]. In this experiment, artificial ants were bred to traverse a special trail named the John Muir Trail. Random populations of ants were created which behaved in a given way based on their environments. Each ant was evaluated by judging their performance on the trail, and ants performing the best were bred to generate new populations. The UCLA artificial life team had great success with this experiment and after many generations an ant emerged that could follow the trail almost perfectly.

1.1 First Implementation

This experiment was generalized, and work began on construction of the artificial system. The world itself was modeled after a popular tutoring program *Karel the Robot* [23]. Karel's job in the application was to find as many beepers on a 50x50 grid as possible. Karel starts out in the middle of the grid, which is always also the location of the beginning of the path. The more beepers that Karel finds in 200 moves, the higher his fitness will be and the more likely he will reproduce in the next generation. Those robots that do not find many beepers are more likely to die and to be replaced by new robots. Initially every robot in our population of 200 is given a completely random set of behaviors for any particular set of inputs. The robot's brain is simply an encoded finite state machine (shown in figure 2). Inputs are determined by what the robot senses about the space that it is currently occupying and its current state. It knows if there is a beeper on its current location or not, and it knows what state it is in at that moment. This information is fed into the finite state machine representing the robot's brain, and the output is a behavior and a new state. The behavior can be either turn left or move forward. The state is used as the input state for the next behavior.

The finite state machine had to be encoded in such a way that the use of genetic algorithms would not only preserve some of the characteristics of the parents, but would also allow for the emergence of new characteristics. This problem did not have a good answer, but through trial and error, it was found that a direct binary encoding worked well. For these particular creatures, there was no distinction between the genotype and the phenotype. The genotype was the actual encoding of the finite state machine. There were a series of steps to follow to find the new action and new state for a given Karel. First, the sensory and internal information from the Karel had to be evaluated. This information consisted of four bits: one bit indicated whether there was a beeper on the Karel's current location, and three bits indicated the current state of the robot. To evaluate this information, the binary bit string was first converted to base 10. Taking this number and multiplying it by the number of bits in every solution in our chromosome (in this case 4), we get some number *x*. Finally, by moving *x* bits in our chromosome, the appropriate action and state are found.

The initial population consisted of a number of completely random finite state machines. For each trial, every individual in the population was run on the same path of beepers. Its fitness was a direct result of the number of beepers it found. After each trial, the top 30% of the robots with the greatest fitness were crossed-over randomly with one another, and a small percentage of the individuals were subjected to the mutation operator. These offspring replaced the 70% that died off in the previous generation. This process was repeated over many generations until the average fitness of the population converged and stabilized.

1.2 Results

This program gave some very positive results. With an initial population of 200, 50 generations, and 200 moves allowed in each trial, the robots seem to quickly evolve good strategies. Using a path of 20 beepers, trials tended to average a maximum score of 16-18 beepers found per trial. In some cases, a strategy would evolve that would find all the beepers. In the first trials, only one path was used. This produced robots very good at following that particular path, but their strategies did not always work well on other paths. The program was then expanded so that each trial had a new, randomly created path to follow. Running this trial between 50 and 100 generations seemed to always come up with the same good strategy. The robot would enter a loop in which it would try each direction. Having found a beeper in one direction, it would move in that direction and then enter its original loop. This was a very simple strategy, but it seemed to be the most efficient and effective for finding beepers.

The first experiment accomplished two vital goals. It provided the basis for an artificial life system implementation for both the robots and the environment itself. But more importantly, simple genetic algorithms were shown to perform exactly as they should have. The rapid success of this phase could not have been predicted, but greatly bolstered enthusiasm for ensuing stages of the project.

2 *Interactive Karels*

The next step of the project was to provide a more complex environment for the artificial creatures. The goal of artificial life is to gradually simulate natural biological life in the most accurate way possible. To make the system described in the first experiment more lifelike, the next logical progression was the elimination of the specific fitness function. This function (the number of beepers a robot could find) was eliminated for *survival of the fittest* [22]. The robots still had to be able to interact with the environment to find food, but now they were also able to interact with one another. The better a robot did at finding beepers, the longer it would live; the longer the robot lived, the more chances it would have to mate with other robots. Because these more successful robots reproduced more often, their behaviors were more likely to populate subsequent generations.

2.1 Implementation

There were many steps involved with this move away from a defined fitness function. First, the population of robots had to be put in the same environment at the same time. Before, they had been isolated from one another, but now they would be free to interact. Beepers were now considered energy necessary for the creatures to live. They were born with a certain amount of energy, and every move they made expended energy. If they expended too much energy before they could find another beeper, they would die. Beepers were no longer in paths; rather they were randomly distributed across the 50X50 toroidal world the creatures inhabited. Crossover of individuals within the population had been done by selection of the best fit 30% in the previous experiment, but now all mating was done by proximity. If two robots ran across each other on the grid, then they would mate and their offspring would be born in that cell. Because the creatures with the best survival strategy would live the longest, they would be more likely to mate.

2.2 Results

Upon running the new implementation, several problems became apparent immediately. One problem was the violation of the law of conservation of energy. Because robots were given a certain amount of energy when they were born, and reproduction cost very little, the most fit population of creatures evolved when the entire population lived around a single cluster of cells or moved in large swarms. Large numbers of creatures could stay in one position and reproduce with each other and their children every generation. The result was a very fit population of *lazy inbreeding cannibals*. They did not need to find beepers because they simply reproduced for energy.

To fix this, the entire system was reconfigured to conserve energy. Parents lost whatever energy they gave to their children, making beepers the only source of energy. After this modification, the system ran just as one might have expected. In the 50X50 world with randomly dispersed beepers, the best strategy was simply to move forward and to occasionally turn left. Not only did this behavior optimize the amount of energy collected, but it also increased the likelihood of running into a mate. This behavior was exhibited immediately in almost every trial.

At this point, the limit of the complexity of the finite state machine brain had been reached. It was not possible to expand any of the inputs (sensory data) or outputs (behaviors) because doing so would mean increasing the size of the genotype exponentially. The next step would have to be to replace this brain with a new one capable of greater complexity.

3 *ANNimALs*

The final experiment utilized all the features of the previous experiments in addition to several other concepts designed to increase complexity and biological similarity. The finite state machine brains were replaced with artificial neural networks (ANNs), more accurately simulating the brains of biological creatures. Using the ANN brains solved many problems that had been encountered. It reduced the constraint on the amount of input (internal and external sensory information) a creature could process, and provided a more accurate representation of true phenotypic representation. The ANN was not the genotype of the brain; rather it had to be created from the genotype. This raised one problem; the ANN had to be encoded in such a way that part of the structure of the brain would be passed on to its children after crossover. A *marker based* ANN encoding strategy was eventually implemented as a solution [8,20].

In addition to the ANN brains, many steps were taken to reduce constant parameters within the simulation. The system itself was redesigned to find optimal values for these parameters. Finally, the interface was redesigned and improved to allow for closer observation, and to provide more information about the creatures and environment within the system. Statistics such as neuron density and connectivity as well as behavior percentages and fitness values could be seen interactively along with the system's evolution.

3.1 Encoding Artificial Neural Networks

In order to use an artificial neural network representation as the brain for the ANNimALs, a method had to be found to encode an ANN into a genotype. The genetic operators would effect the genotype before the phenotype was expressed; the phenotype (the ANN brain) would then be decoded from the genotype (figure 8). Two different problems are inherent to this process. First, it is difficult to express ANNs of different node density and connectivity with a fixed length genotype. The decoding process would have to allow not only for different values between the connections of each node at each layer, but it would also have to allow for different numbers of nodes and connections within each node. The second problem was that some of the structure of the parent's brains had to be passed onto the children. This meant that parts of the decoding of the genotype would have to withstand crossover and mutation. Several methods were examined to deal with these problems, but one seemed to solve both in a rather straightforward manner based somewhat on biological systems.

Some encoding strategies for artificial neural networks deal strictly with the structure of the ANN, and not with the connections and bias values within the network. These strategies use a method similar to Lindenmayer Systems [18], recursively growing their networks from genotype rules. After the networks have been grown, they are inhabited with random connection values. The values are initially modified with back propagation and later through various learning strategies throughout the creature's life. Because these strategies produce only the node structures and provide no connection values, back propagation bears most of the brunt for developing the final structure [13]. This was not a desirable characteristic for the ANNimALs system; so these systems were not pursued.

Other encoding strategies focus on recreating the actual biological process of neuronal growth. Parisi, and Nolfi developed a method where the genotype encoded locations for nodes and projection directions for various connections within the network [4,21]. As the nodes migrated out from various neurons, they would connect only with other neurons the axons (connections) encountered. Their originating node determined connection weights from axons. This strategy was computationally intensive and did not suit this project well.

The strategy finally used was a marker based encoding strategy developed by Fullmer and Miikkulainen [8]. This strategy allowed not only for a varied number of nodes and connections, but preserved the structure of the network through crossover. By defining markers within the genotype string, it could be broken into many smaller strings. Each of the smaller strings could be used to represent a node within the system. By walking through the string and building the connections based on these substrings, important information is preserved even through crossover.

Figure 6 shows a substring of a genotype string broken down into the various components defining a node and its connections [20]. The genotype string is traversed from left to right and stops when a start marker has been found. The next value in the string indicates what node the current substring defines. Every pair of values after the node marker define another node with which to make a connection and the value of that connection. These connections are defined for a specific node until an end marker is encountered.

The strategy implemented in this experiment exemplifies pure evolutionary learning. Once the phenotype (neural network) has been decoded from the genotype, it remains unchanged throughout the course of the creature's life. There have been many studies to determine what is the best method of learning. Other systems have implemented ANN learning strategies such as back propagation and reinforcement learning to teach ANNs beyond what they learned through evolution. It has been suggested that the combination of a learning strategy and evolutionary learning rather than just one or the other is the best and quickest approach to evolution of intelligence in artificial life [1]. No learning strategies were implemented in the ANNimAL experiment. The primary interest was in the effect of genetic algorithms on evolutionary systems and additional learning strategies could have caused confabulating results.

For this experiment, the initial genotype strings consisted of 800 random integers from 0 to 51. The value 22 was reserved for the start marker, and 23 for the end marker. When a start marker was encountered, the next integer indicated the node that substring defined. All values after that served to define a connection originating from that node to another node defined as k_n with a value v_n. When an end marker was encountered, the definition of that node would stop, and the system would begin searching for the next start marker in the string. The ANN this produced had a fixed number (22) of input nodes, a variable number (up to 24) of hidden nodes, and a fixed number (5) of output nodes. Connections and connection strengths were random and could be very dense (many connections) or sparse (few connections).

3.2 The ANNimAL Brain

The input layer of the ANN was given activation values based on the internal and external perception of the creature. 18 of the input values correlated with the creature's vision. It could see all 8 surrounding cells as well as the cell it occupied. There were four possible values for each cell (nothing, another creature, food, or another creature and food); so 18 total input nodes were required for vision. The creature used two nodes to sense its hunger state (satiated, slightly hungry, hungry, or very hungry), and another two nodes to sense its current age (young, middle age, old, very old). Figure 9 shows the ANNimAL brain.

The 5 nodes in the output layer corresponded to a given behavior the creature was to exhibit. The behaviors were move left, move right, move forward, sleep or eat. The network would use the 18 input nodes as initial activation values and then calculate the activation in each node of the network. The behavior associated with the output node with the greatest activation would be chosen for that time cycle. As the creature moved, became hungry or satiated, and aged, its internal and external sensory values would change, causing different behaviors to become most strongly active.

3.3 Experiments and Results

Replacement of the finite state machine brain with the artificial neural network allowed for a great deal more complexity within the system. This in turn allowed much more room for experiment and study in different areas of alife. One previous restriction had been the number of inputs to the brain. Because the encoding of the finite state machine grew exponentially as more input was added, it was almost impossible for the artificial creatures to have any complicated sensory input because exponentially large genotypes would cause massive slowdown during crossover.

Switching to the ANN brain allowed for great expansion in this area because the inputs to the neural network had very little effect on the size of the genotype. Immediately after implementing the ANN the sensory field was expanded and enhanced.

It is crucial that the artificial creatures have adequate internal and external sensory input or complex emergent behaviors (following, avoiding, etc.) can not surface. It was hoped that expanding the sensory inputs would eventually lead to such behaviors, but none of these behaviors were ever exhibited. There was sufficient sensory input for these behaviors to exist; most probably there was simply no need for the behaviors within the system. The creatures did not evolve them because there was no evolutionary advantage to having them. This indicated that the environment was too simple, that it did not challenge the creatures into evolving behaviors that were more complicated. Further work to expand the environment would most likely result in demonstration of these behaviors.

Another advantage of moving to the ANN brain was the increased number of possible outputs. With the new brain, it was possible to have as many immediate behaviors (left, forward, sleep, etc.) as was necessary. This opened up many interesting questions, but two were the most intriguing. Would it be possible for the creatures to learn to eat, and would it be possible for them to learn to mate? These had been constant behaviors in the past experiments. Creatures ate when they encountered food, and mated when they encountered other creatures. Could these behaviors be learned through evolution?

In fact, eating was a very learnable behavior. Because it was so necessary for the creatures' survival, only those creatures having the behavior survived to reproduce. This was a good and simple example of an emergent behavior; the system behaved just as one might think it should. Unfortunately, the system did not make out so well in the case of the mating behavior. If mating was a behavior (the creatures were given the choice to mate instead of mating automatically), reproduction within the system almost completely stopped. Because the creatures initially had random survival strategies, the chance that one would want to mate when occupying a cell with another creature was miniscule. Since none of the creatures mated, natural selection never took place. There was no offspring to replace dying creatures and entire populations simply died without evolving. This posed an interesting question however: how exactly did biological life get started when it was so apparent that sexual reproduction was much too involved for such simple organisms? One possible solution was to incorporate asexual reproduction.

With asexual reproduction, the creatures no longer had to be occupying a cell with another creature to reproduce. It was hoped that the best creatures would have time to generate offspring before they died, giving the entire population more time to evolve before extinction. The implementation did prolong extinction, but it was still apparent that at this simple level the choice of any kind of reproduction was beyond the complexity of the system.

Another question had to do with the complexity of the ANN. With marker based encoding, it was possible for different neural networks to have different node densities (number of active nodes) and connection densities (number of active connections between nodes). Does neuronal complexity change with the complexity of the environment? For example, would creatures dealing with more difficult food distributions have to develop ANNs that were more complex (would they be smarter)? Initial experiments showed those slightly more complex environments (patches of food vs. random food) actually did result in creatures with greater connection density. Further work focusing on environments that are more complex would have to be done to verify such findings.

III Overall Results and Future Goals

1 Future Goals

The eventual goal of alife is to simulate biological life in computers as closely as possible. Unfortunately, the speed and complexity of computers today can not approach what would be necessary to accurately simulate the immensely complex process of biological life. The best that can be done is to create models roughly simulating some of the processes currently understood. But the boundaries of what the machines of today are capable of have yet to be really tested; the majority of the models still have a mathematical and logical basis, sometimes completely ignoring biology completely. The most promising future for research specifically directed at recreation of biological life is in the joining of evolutionary biologists and computer scientists. The future of alife is not in simulations of life designed to demonstrate certain behaviors; rather it is in the development of systems capable of taking simulations of the building blocks of the biology of life, and allowing them to interact and react and eventually develop into a new species completely independent of any constraints introduced by the programmer. In other words, no one should have to play god; the system should govern itself.

Possible extensions of this kind of research are limitless. Introduced in this paper were several current areas of research interest in evolutionary algorithms designed for a range of possible applications. Genetic algorithms can be used in a range of applications from the highly experimental and theoretical aspects of artificial life, to their practical utilization quickly finding optimal solutions to computationally difficult problems. Continuation of work in genetic algorithms in artificial life should concentrate on looking at increasingly complex alife systems. Work on more practical applications would be to solve increasingly computationally intensive (or NP) problems.

A very interesting future research problem would be the study and implementation of classifier systems in a machine learning application. Classifier systems can be generalized in both practical learning solutions as well as in an alife context. They would provide the stability of the finite state machine used in the first two experiments, without the problems of exponential growth of the genotype. A classic example of classifier systems in alife is in the learning of behavior strategies for an artificial frog. Implementation of this type of problem could serve to provide a basis for further research in this area.

One immediate continuation of the work described in this paper would be to incorporate some form of ANN learning to the artificial neural networks used in the third experiment. That experiment was primarily concerned with Darwinian evolution, in which learning is purely evolutionary. Lamarckian evolution states that information learned in the lifetime of a single individual would effect later generations. This concept has been discredited for the most part, but there has been research to indicate the addition of learning strategies actually aid in the evolution of the population as a whole [1]. The interaction between the evolutionary learning of the population and the individual learning of creatures to aid in evolution is called the Baldwin effect [1,25]. The genotype of the creature is used to express a phenotype at birth. During the life of the individual creature, this phenotype remains unchanged. The only processes having any effect on changing the networks from one generation to the next are crossover and mutation operators working on the genotype. If some form of learning were incorporated into the system, after expression of the phenotype, the network would not remain constant; rather it would learn based on interaction with the environment. Survival would then not only be dependent on the network at birth, but also by how well that network could learn to deal with its environment. The knowledge a creature learns during its lifetime does not transcend generations, but its ability to collect that knowledge make it more likely to survive, passing on its genetic code to future generations.

2 Concluding Remarks

The results from the progression of this project were very positive for each phase. In the first experiment, the path following robots immediately displayed the power of genetic algorithms and evolution as a method of learning. The implementation of the interactive robots and the elimination of the fitness function brought the whole project into the field of artificial life, instituting the concept of survival of the fittest. Finally, in the last experiment, the addition of an artificial neural network brain increased the complexity of the system dramatically allowing more complicated, more interesting, and more original questions to be examined.

The original goal of the research was not clearly defined, but this project served not only to collect a great deal of information about the topics, but also to dive into the most current research areas of the field. Doing this not only made the research more interesting, but also allowed for the possibility of novel research and ideas to be explored. At the end of this project, I am by no means at the end of the research associated with this project. Hundreds of unanswered questions have been raised providing a great range of possible future research topics in this field. I am grateful for the opportunity I had to study in this area, and sincerely hope others will follow my interest in studying this new and vastly interesting topic.

IV References

[1] Ackley, D., and M. Littman. "Interactions Between Learning and Evolution." In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer, and S. Rasmussen. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. X, 487-510. Redwood City, CA: Addison-Wesley, 1992.

[2] Bäck, T. *Evolutionary Algorithms in Theory and Practice.* NY: Oxford University Press, 1996.

[3] Belew, R. K., J. McInerney, and N.N. Schraudolph. "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning." In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer, and S. Rasmussen. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. X, 511-548. Redwood City, CA: Addison-Wesley, 1992.

[4] Cangelosi A., Parisi D., and Nolfi S. "Cell division and migration in a 'genotype' for neural networks". *Network* **5** (1994): 497-515.

[5] Clarkson, M. *Windows Hothouse: Creating Artificial Life with Visual C++*. Addison-Wesley, 1994.

[6] Collins, R. J., and Jefferson, D. R. "AntFarm: Towards Simulated Evolution." In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer, and S. Rasmussen. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. X, 579-602. Redwood City, CA: Addison-Wesley, 1992.

[7] Fernandez, J. *The Genetic Programming Tutorial Notebook.* http://www.geneticprogramming.com/Tutorial/index.html.

[8] Fullmer, B. and Miikkulainen R. "Evolving finite state behaviour using marker-based genetic encoding of neural networks." *Proceedings of the First European Conference on Artificial Life*. Cambridge, MA: MIT Press, 1992.

[9] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

[10] Heitkötter, J. and Beasley, D. *The Hitch-Hiker's Guide to Evolutionary Evolution*. Q1.4. March, 1998. http://www.cs.purdue.edu/coast/archive/clife/Welcome.html.

[11] Holland, J. H. *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press, 1975.

[12] Jefferson et al. "Evolution as a Theme in Artificial Life: The Genesys/Tracker System." In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer, and S. Rasmussen. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. X, 549-578. Redwood City, CA: Addison-Wesley, 1992.

[13] Kitano, H. "Designing neural networks using genetic algorithms with graph generation system." *Complex Systems* **4** (1990): 461-476.

[14] Koza, J. R. "Genetic Evolution and Co-Evolution of Computer Programs." In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer, and S. Rasmussen. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. X, 603-630. Redwood City, CA: Addison-Wesley, 1992.

[15] Koza, J. R. "Artificial Life: Spontaneous Emergence of Self-Replicating and Evolutionary Self-Improving Computer Programs." In *Artificial Life III*, edited by C. Langton. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XVII, 225-262. Reading, MA: Addison-Wesley, 1994.

[16] Langton, C. G. "Artificial Life." In *Artificial Life*, edited by C. Langton. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. VI, 1-48. Redwood City, CA: Addison-Wesley, 1989.

[17] Levy, S. *Artificial Life: The Quest for a New Creation*. NY: Pantheon Books, 1992.

[18] Lindenmayer, A. "Mathematical models for cellular interaction in development, parts I and II." *Journal of theoretical biology* **18** (1968): 280-315.

[19] McClelland, J. L., and Rumelhart, D. E. "Distributed memory and the representation of general and specific Information." *Journal of Experimantal Psychology: General* **114** (1985): 159-188.

[20] Moriarty, D., and Miikkulainen, R. "Evolving complex Othello strategies using marker-based genetic encoding of neural networks." Technical Report AI93-206, University of Texas at Austin, Department of Computer Science, 1993.

[21] Nolfi, S., Elman, J. L., and Parisi, D. "Learning and Evolution in Neural Networks." CRL Technical Report 9019, Center for Research in Language, University of California, San Diego, La Jolla, CA, 1990.

[22] Packard, N. "Intrinsic Adaptation in a Simple Model for Evolution." In *Artificial Life*, edited by C. Langton. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. VI, 141-156. Redwood City, CA: Addison-Wesley, 1989.

[23] Pattis, R. E. *Karel the Robot: A Gentle Introduction to the Art of Programming.* John Wiley & Sons, 1981.

[24] Santa Fe Institute. Santa Fe Artificial Life Homepage. http://alife.santafe.edu/alife/.

[25] Simpson, G. G. "The Baldwin Effect." *Evol.* **7** (1953): 110-117

[26] Waltz, D. and Feldman, J. A. *Connectionist Models and Their Implications: Readings from Cognitive Science.* NJ: Ablex Publishing Corporation, 1988.

[27] Yaeger, L. "Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision and Behavior or PolyWorld: Life in a New Context." In *Artificial Life III*, edited by C. Langton. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XVII, 263-298. Reading, MA: Addison-Wesley, 1994.

V Appendix

While much detail has been provided about the different aspects and stages of this project, little has been mentioned about the specific nature of their implementations. This appendix will be used to briefly describe the applications behind of each stage of the project.

Visual C++ was the chosen language of the project. In the first phase, Visual C++ code was compiled into a console application for an IBM compatible 200 MHz Pentium II. Because it was difficult to output visual data in ASCII code in such a way that useful information could be provided, data from the trials was saved to a data file after each successful run. A separate Visual Basic program was developed to interpret this data file, graphically displaying the patterns of the most successful robots in the population for any given generation.

In the second phase, the robots were able to interact with one another. This meant that no longer could the computations be independent of the visual interface. In every generation, a vast amount of information was being updated and a data file was not adequate to store all this information. To solve this problem, a visual interface was designed in Visual C++ allowing the robots to be updated immediately as the generations were being computed. Not only did this simplify running the program, but also made it apparent how the creatures were evolving to cope with their environment.

The final phase involved not only the addition of many additional features, but also a restructuring of the code to make it more easily modifiable and readable. The same interface was used, but buttons and pull down windows were added to allow certain variables to be changed while the system was running. A new code object structure was implemented based on a project in Mark Clarkson's Windows Hothouse [5]. This new structure allowed for more sensible relationships to exist between the various classes in the code. A screenshot of the final application is shown in picture 1.

VI Acknowledgements

I would first like to thank Dr. Thomas Whaley for all his assistance and support with this project. I would also like to thank Dr. Pamela Vermeer, Dr. Kenneth Lambert, and Dr. Joseph Thompson for all their excellent tutelage throughout my time at Washington and Lee. I must also mention my fellow students and friends Noah Egorin, Anurag Chandra and John Thrall, without whom all the late nights in the computer lab would not have been nearly as fun.

VII Tables, Figures and Pictures

Table 1 Three generations of a sample genetic algorithm using the Max() fitness function

Initial Population	Fitness	Generation 2	Fitness	Generation 3	Fitness
1) 0 0 1 1 0 1	13	**(1=2X5) 1 1 0 1 | 1 1**	**55**	(1=1X3) 1 | 1 0 0 1 0	50
2) 1 1 0 1 0 0	**52**	(2=5X2) 1 0 0 1 | 0 0	36	**(2=3X1) 1 | 1 0 1 1 1**	**55**
3) 0 1 0 0 0 0	16	**(3=2X6) 1 1 | 0 0 1 0**	**50**	**(3=1X6) 1 1 0 | 1 1 1**	**55**
4) 0 1 1 0 1 1	27	(4=6X2) 1 0 | 0 1 0 0	36	(4=6X1) 1 0 0 | 1 1 1	39
5) 1 0 0 1 1 1	**39**	(5=5X6) 1 0 0 | 0 1 0	34	(5=3X1) 1 1 0 0 | 1 1	51
6) 1 0 0 0 1 0	**34**	**(6=6X5) 1 0 0 | 1 1 1**	**39**	**(6=1X3) 1 1 0 1 | 1 0**	**54**
Average Fitness:	30		42		51

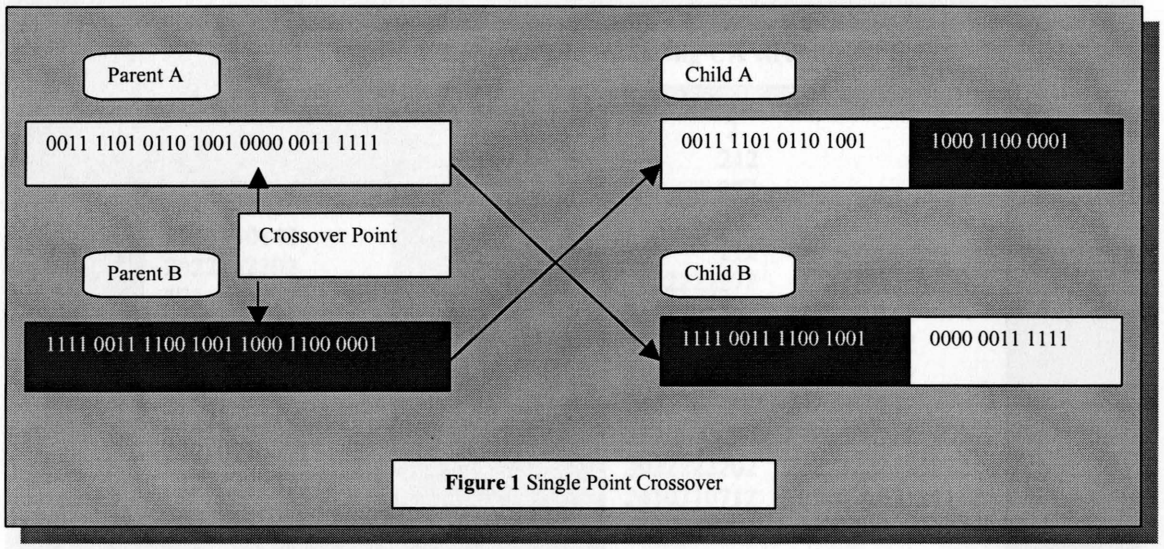


Figure 1 Single Point Crossover

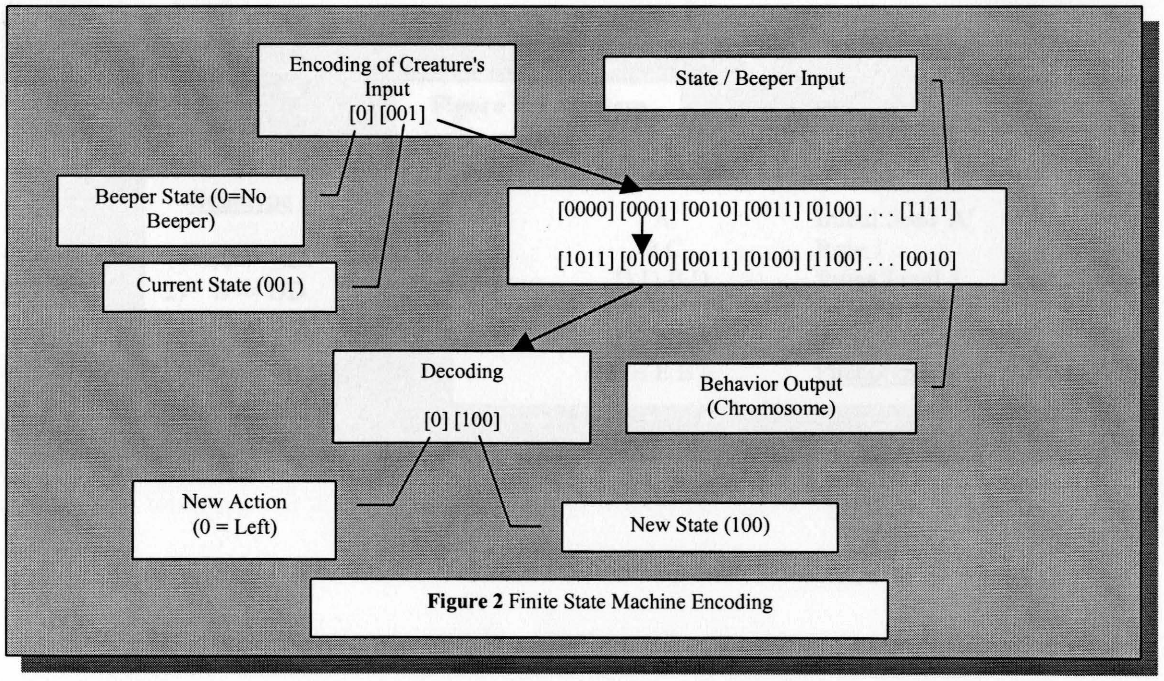


Figure 2 Finite State Machine Encoding

Figure 3 Genetic Programming Crossover

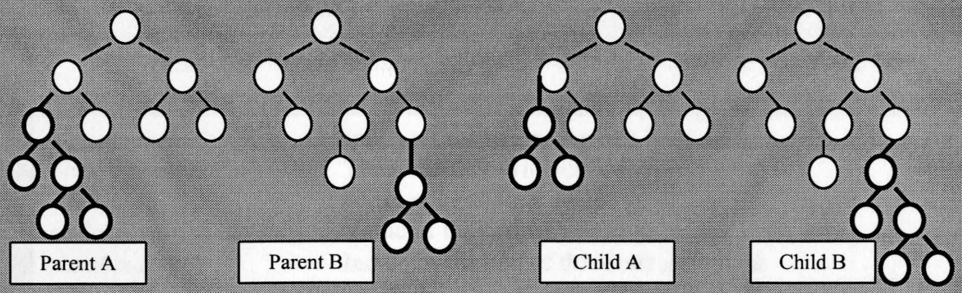


Figure 4 Langton's reproducing CA structure

```
 22222222
2170140142
2022222202
272     212
212     212
202     212
272     212
21222222122222
207107107111112
 2222222222222
```

After 151 Iterations $\longrightarrow$

```
            2
          212
          272
          202
          212
 222222272    22222222
2111701702   2170140142
2122222212   2022222202
212     272  272     212
242     212  202     212
212     272  272     212
2022222202   21222222122222
2410710712   207107107111112
 222222222    2222222222222
```

Figure 5 L-System

Genotype

1) A -> BC
2) B -> DD
3) C -> BD
4) D -> E

0	A	Initial Seed 'A'
1	B C	Rule 1
2	D D B D	Rules 2 and 3
3	E E D D E	Rules 4, 4, 2, 4
4	E E E E E	Rules 4, 4, 4, 4, 4
	E E E E E	Phenotype

Figure 6 Marker Based Encoding

<start marker><node number><k1><v1><k2><v2><kn><vn><end marker>

<start marker>	Indicates the beginning of a new string
<node number>	The node this string defines
<ki>	Number of another node
<vi>	Value of the connection to *ki*
<end marker>	Indicates the end of this node definition

Figure 7 Simple ANN

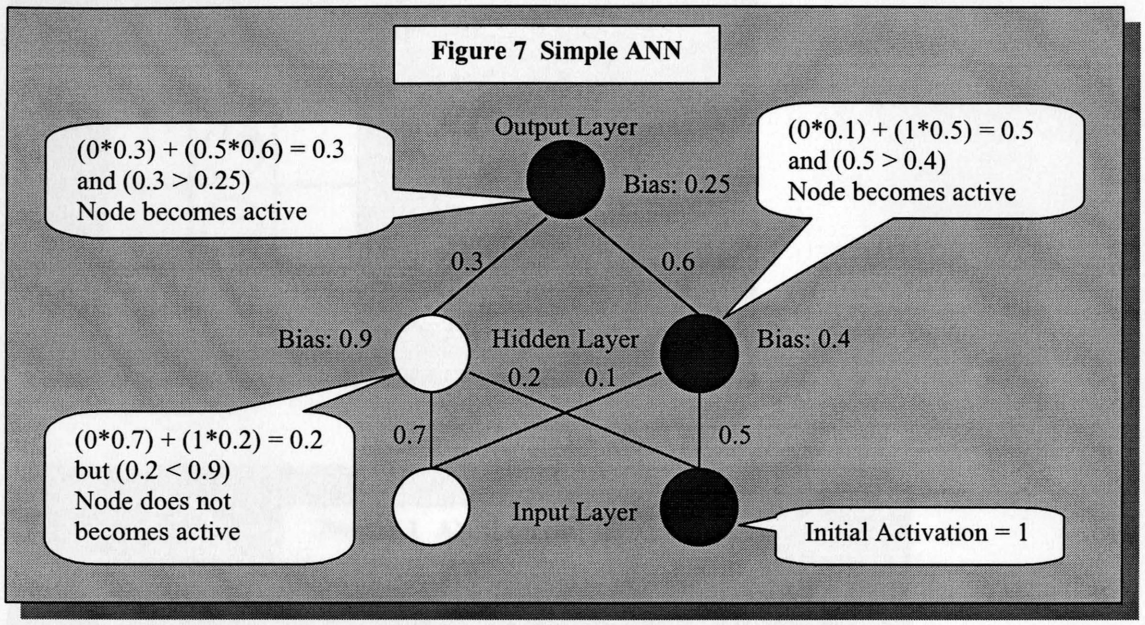

Output Layer
(0*0.3) + (0.5*0.6) = 0.3 and (0.3 > 0.25) Node becomes active
Bias: 0.25
(0*0.1) + (1*0.5) = 0.5 and (0.5 > 0.4) Node becomes active
0.3
0.6
Bias: 0.9
Hidden Layer
Bias: 0.4
0.2
0.1
0.7
0.5
(0*0.7) + (1*0.2) = 0.2 but (0.2 < 0.9) Node does not becomes active
Input Layer
Initial Activation = 1

Figure 8 Genotype / Phenotype

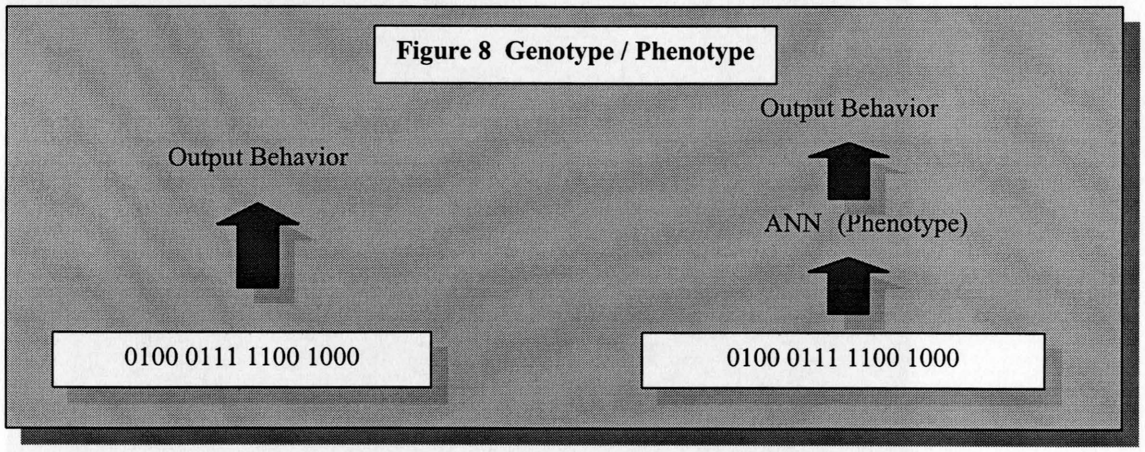
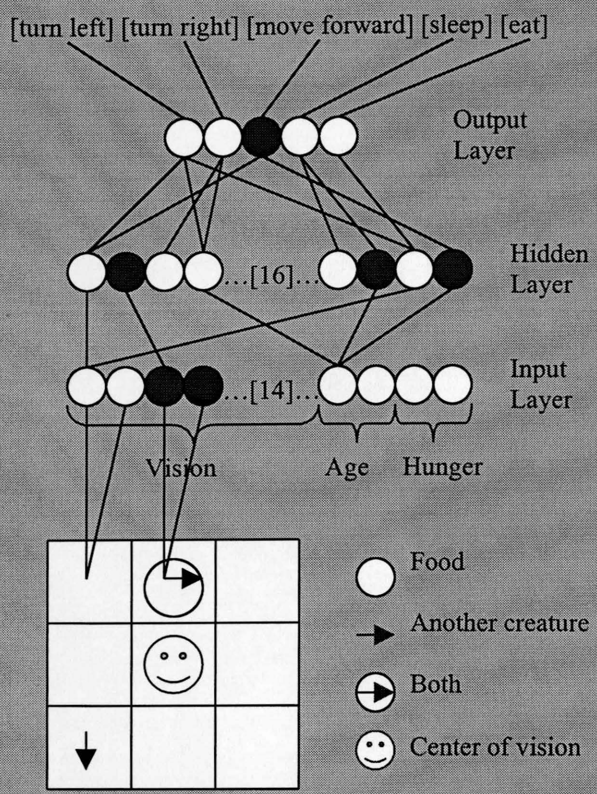


Figure 9 ANNimAL Brain



Picture 1 ANNimALs

Artificial Life - ANNimALs!
File Edit View Help Controls Speed Zoom World Generations