ABSTRACT

EVOLVING INTELLIGENT EMBODIED AGENTS WITHIN A

PHYSICALLY ACCURATE ENVIRONMENT

By

Gene D. Ruebsamen

December 2002

This thesis explores the application of evolutionary reinforcement learning techniques for evolving behaviorisms in embodied agents existing within a realistic virtual environment that are subject of the constraints as defined by the Newtonian model of physics. Evolutionary reinforcement learning uses evolutionary computation techniques, which are based to some degree, on the evolution of biological life in the natural world. These techniques are generally stochastic in nature and involve random decisions that guide the optimization process via processes of selection, mutation and reproduction. A common problem of using evolutionary computation techniques to evolve intelligent behaviors in embodied agents is the simplicity of the environment and overall system often precludes any life-like behaviors from emerging. Furthermore, the commonly used supervised learning techniques are extremely difficult to apply to embodied agents that employ a complex control system. This thesis proposes a methodology, based on neuroevolution, that effectively addresses this issue of environmental complexity and learning; thus, allowing for the emergency of like-like and efficient behaviors.

EVOLVING INTELLIGENT EMBODIED AGENTS WITHIN A PHYSICALLY ACCURATE ENVIRONMENT

A THESIS

Presented to the Department of

Computer Engineering and Computer Science

Calfiornia State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By Gene D. Ruebsamen

BS, 1997, California State University, Pomona

December 2002

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
A-Life	Artificial Life
ANN	Artificial Neural Network
CA	Cellular Automata
EA	Evolutionary Algorithm
EC	Evolutionary Computation
GA	Genetic Algorithm
NE	Neuroevolution
RNN	Recurrent Neural Network

CHAPTER 1

INTRODUCTION

Overview

This thesis proposes a methodology for evolving intelligent behaviors of embodied agents within a physically realistic environment as a significant step toward devising evolutionary techniques for the emergence of complex intelligent behaviors that utilize techniques inspired by biological evolutionary systems. Evolutionary computational techniques allow the embodied agents to learn to interact with their environment in such a way as to produce complex emergent behaviors. Evolutionary computation is an umbrella term often used to describe problem solving systems that incorporate computational models of biological evolution. Evolutionary computation consists of a variety of evolutionary algorithms such as genetic algorithms, classifier systems, evolutionary strategies and genetic programming. All of these algorithms share the common theme of simulating biological evolution of individual structures via the modeling of selection, mutation and reproduction. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms (Spears et al., 1993).

To better understand the evolutionary computation paradigm, some discussion of biological evolution is warranted. In nature, evolution consists of several different processes. The generation of biologically diverse organisms that compete with each

other for limited resources in the environment, otherwise known as natural selection, is the fundamental process that drives biological evolution. Individuals that are better able to obtain those resources are more likely to survive and propagate their genetic material to their progeny.

The genome represents the encoding of genetic information in nature. Sexual reproduction allows for 2 individuals to produce offspring that contain a combination of genetic information inherited from both parents. The crossover operator in the genetic algorithm models what occurs at the molecular level in biological systems and is also known as recombination. This crossing over of information from the 2 parents, along with random bit mutation, is part of the driving force being natural evolutionary processes. Through these processes nature selects for the fittest individuals (ie. survival of the fittest) in a population to procreate and produce offspring.

An embodied agent is an autonomous living creature, subjected to the constraints of its environment. The term "embodied" differentiates these agents from regular software agents, which are pieces of software that perform tasks in an intelligent way as defined by their authors. Examples of software agents include web spiders or IRC bots. Embodied agents in their simulated environment can be subjected to the same physical forces that govern bodies in our natural environment, thus allowing for a less difficult physical realization of the embodied agent upon completion of the evolution process. Every embodied agent has its own Artificial Neural Network (ANN) that acts as the brain that processes sensory input and generates motor output. The study of embodied software agents evolving within a virtual environment has been an active research area in Artificial Intelligence (AI) and Artificial Life (A-Life). Recently, the field of Artificial

Life has produced several such systems that demonstrate evolution; however, applying the results to physical systems has proven to be difficult, if not impossible due to the often-simplified nature of the environment the agents are evolved within.

Problem Statement

This thesis attempts to solve the problem of evolving intelligent behaviors that are readily transferable to real world machines existing in the natural world. Doing so requires the use of an accurately simulated environment utilizing evolutionary computational paradigms. Due to the evolutionary nature of such systems, the solutions evolved can often be more robust than the solutions we engineer ourselves; however, solutions from existing systems are generally not transferable to machines of the natural world due to the overly simplistic virtual environments employed during development and training. Currently, much effort is being devoted to designing control systems for a whole slew of electro mechanical devices; these control systems require the ability to effectively process the sensor inputs and generate motor outputs that allow the device to properly and intelligently interact with its environment. The more complex the device, the more effort is needed in designing the control system for even such basic tasks as locomotion. Much of this effort can be alleviated if a framework existed where such a device could be accurately modeled and its control systems simultaneously evolved. The significance of this thesis is to provide a framework for such a system capable of modeling a variety of physical devices and/or biologically plausible agents and simultaneously employing the techniques of evolutionary computation to evolve intelligent control solutions. The accurate modeling of our natural environment is an often-neglected area of A-Life research; however, we demonstrate that devoting the

computational resources to an accurate environmental model and simultaneously utilizing complex biologically inspired algorithms can have a positive influence on the effectiveness of the emergent behaviors demonstrated by the embodied agents.

Another important problem of A-Life research addressed in this thesis is the difficulty of training the embodied agents. Artificial Neural Networks (ANNs) are models of biological Neural Networks; however, most algorithms for training an ANN employ supervised learning algorithms and require known solutions to the problem be presented to the ANN during the training stage. The sheer complexity of the embodied agents, with their complex control systems, make it impossible to provide the ANN with the accurate training data to allow for supervising learning methods to be employed.

This thesis will present a general methodology for evolving optimal or near optimal solutions to complex control problems that would otherwise require enormous effort if approached using conventional methodologies. The difficulties and pitfalls of traditional methods can often be avoided in their entirety by the novel use of Neuroevolution (NE) techniques to train an embodied agent without requiring the use of a supervised learning algorithm or an external agent directing the process.

Due to the complexity of modeling real world physics, as well as the enormous overhead required for the complex evolutionary algorithms, earlier simulations required the vast computational resources of supercomputers; however, today with the widespread availability of powerful commodity computing hardware, the evolution of intelligent behaviors of embodied agents within a physically accurate environment has become computationally feasible.

Approaches of This Thesis

This thesis approaches the problem by addressing two separate issues: (1) environmental complexity and (2) learning algorithms. It is important to provide sufficient environmental complexity and accuracy in modeling the natural environment if we ever wish to have a physical manifestation of the embodied agent. The virtual environment employs the Newtonian model of physics as it applies to the natural world. Bodies constrained within the virtual environment behave as they would in the natural world, with forces such as inertia and friction acting upon them. The embodied agents that live within this virtual environment are not able to violate any of the constraints of the environment. The evolutionary algorithms employed in the evolution of the embodied agents brain allow them to learn via a reinforcement style of trial and error known as neuroevolution (NE). Furthermore, the embodied agents operate individually by collecting information from a variety of sense organs, further effectuate changes in the environment by exerting forces through muscles and pass on their knowledge from generation to generation. The embodied agent is endowed with a fully connected ANN with feedback (recurrent) connections; a genetic algorithm is employed to search for an optimal weight structure for the ANN that will tend to achieve a higher fitness score. The learning process for the individual embodied agent is a gradual process of adaptation and mutation that occur when a generation of embodied agents mate and produce offspring. Highly fit individuals have a greater probability of mating than that of less fit individuals.

An initial population of embodied agents are set lose in the environment without any instructions or external stimulus. Since we initially wish to elicit locomotive behavior, each individual is evaluated and assigned a fitness score based upon its distance

traveled during its brief life. Individual embodied agents whom are able to traverse a long distance quickly are given a higher fitness than slower individuals or individuals who are unable to maintain the correct course. In this fashion, natural selection tends to favor individuals that are able to learn how to move quickly and efficiently. The individuals with intelligent locomotive behaviors tend to mate with each other and as a result may generate even faster and more efficient locomotive behaviors.

Rationale for This Approach

Competition alone is not enough to allow for complex locomotive behaviors to emerge. By introducing a sufficiently complex and realistic environment, the individual embodied agents can evolve infinitely complex behaviors. Furthermore, by allowing a population of agents to compete with each other in nature's game of survival of the fittest, enormous pressures are placed upon each individual to evolve effective locomotive behaviorisms or die. The individual embodied agents start with no knowledge of their environment; however, with each subsequent generation, knowledge of how to survive within the environment is passed to the offspring. In this way, knowledge is preserved and passed on to each subsequent generation. Though an embodied agent learns individually; collectively, the gene pool contains the body of knowledge of the population as a whole, thus through reproduction an effective agent can share its knowledge with other individuals in subsequent generations.

The use of evolutionary algorithms to optimize the structure of the ANN allows for a rapid convergence on an optimal solution for a given problem. The evolutionary algorithms approach employs stochastic processes to generate results that significantly outperform results that would otherwise be obtained through a random search or

conventional optimization techniques. Though the evolutionary algorithms approach makes use of random processes such as mutation, it cannot be stressed enough that the results obtained are distinctly nonrandom. The benefits of the evolutionary approach are twofold: the embodied agents do not need to contain any prior knowledge of their environment and it is sufficient to define the problem and let the embodied agents come up with the optimal solutions.

CHAPTER 2

BACKGROUND AND RELATED WORKS

Motivation and Introduction to the Background

This chapter provides background material for the reader and presents similarly related works by other researchers.

Related Fields

Several branches of research concern themselves with studying intelligent behavior, complexity, evolutionary computation or emergence of complex behaviors from simple interactions. Stephen Wolfram provides an excellent summary (Wolfram, 2002) of no less than 17 distinct disciplines relating to those fields of study. Many of these areas have influenced, in some way, the directions taken in this thesis.

Classical Artificial Intelligence

Although most scientific disciplines, such as mathematics, physics, chemistry and biology, are well defined, the field of artificial intelligence (AI) remains enigmatic (Fogel, 2000, p. 1). Many of the proposed definitions of AI rely upon comparisons to human like behavior. We choose to define Artificial Intelligence as the branch of science that deals with designing machines that can find solutions to complex problems in a more human-like fashion. Whatever the definition, most researchers agree that classical methods of AI have taken a top down approach. The top down approach treats cognition as a high-level phenomenon that is completely independent of the low level details of the

implementing mechanism. The classical, or top-down approach to AI is deductive (i.e., given a set of basic rules, the system is to deduce what combination will produce the desired result) and deals with descriptions of relevant features of the task.

In 1950, Alan Turing pondered the question "Can machines think?" Rather than attempt to answer the question, he devised a test that still bears his name. The Turing test begins with 3 people, a man (A), a woman (B) and an interrogator. The interrogator, in a room separate from the man and woman allowing for no sensory input, may ask questions of both the man and woman. The interrogator's objective is to discern which person is the man and which is the woman. Participant (A) may be deceitful; however, the object for participant (B) is to help the interrogator. Turing then pondered what would happen if a machine were to take the place of participant (A) or (B). Should the machine perform as well as a human participant, it was declared to have passed the test. The original question of whether the machine should then be judged as being capable of thinking was left unanswered by Turing.

The popularity and acceptance of the Turing test focused early efforts on simulating aspects of human behavior. In 2000, Fogel wrote, "At the time (1950), it was beyond any reasonable consideration that a computer should pass the Turing Test. Rather than focus on imitating human behavior in conversation, attention was turned to more limited domains of interest." Thus, classical AI focused its energies on problems of limited domain such as game playing strategies and expert systems.

The initial focus of AI was to eventually create general problem solving programs; however, after several unsuccessful preliminary attempts, the focus of AI narrowed considerably through the 1960s to the early 1980s. Specific search algorithms

were applied to very narrowly defined problems that were typical of human experts in their field of expertise. It was discovered that knowledge in a particular field could be represented in a form that allowed a computer to perform reasoning activities upon it. Researchers again employed the top-down approach of the classical AI paradigm in the development of the computer "expert system." It was believed that such a system would offer many advantages over a human expert, such as permanence and convenience.

In the late 1980s, the branches of AI split into several directions, and are often difficult to classify; however, 2 recent branches of AI that rely heavily upon a bottom up approach are: Machine Learning and Search and Optimization. The Search and Optimization branch deals with planning, constraint satisfaction and function optimization. The Machine Learning branch concerns itself with Neural Networks, Inductive Programming, Data Mining, Bayesian Networks and Decision Tree Learning. The evolution of embodied agents incorporates methods from both the Machine Learning and Search and Optimization branches of AI. The bottom up approach takes its cues from biological evolutionary systems.

Artificial Life

Artificial life (A-Life) is the scientific field of study that attempts to model living biological systems through complex algorithms. A-Life and Evolutionary Computation both make use of evolution and the bottom up approach; however, A-Life emphasizes the development of intelligence through emergent behavior of complex adaptive systems, whereas Evolutionary Computation focuses on providing a framework for optimization processes in general.

The primary objective of A-Life is to produce intelligence or life through local interactions among a large population of virtual agents, as well as the study of the evolutionary process of life in general. The process of producing intelligence or global behavior from local interactions is called an emergent property. Life can be classified as something capable of reproducing itself and adapting to its environment. Life is also capable of independent actions that are not decided by an external agent. These are 2 properties that are shared in common with A-Life.

The environment used in A-Life experiments does not have to mirror the natural world. Rather, most of the A-Life simulations make use of a simplified environment or an environment with entirely different rules than that of our natural world. These simplified environments allow the researcher to concentrate on studying the emergent properties of life rather than having to deal with the complexities of the environment.

Chaos Theory

The field of chaos theory concerns itself with the study of unstable a-periodic behavior in deterministic nonlinear dynamical systems. The basic principle that describes chaos theory is the "Butterfly Effect." The Butterfly Effect states that small variations in initial conditions result in huge and dynamic transformations in the resulting events. The term butterfly effect arose from the claim that a butterfly flapping its wings can, given enough time, affect the direction of a hurricane on the other side of the planet. The main significance of Chaos Theory is the implication that any small uncertainties in the initial conditions of a system will eventually lead to behavior that is impossible to accurately predict.

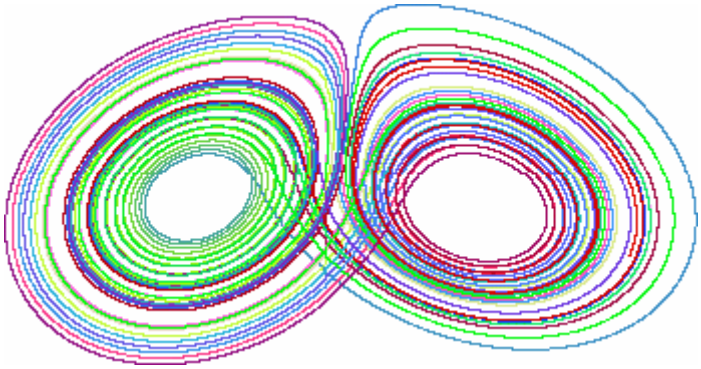


FIGURE 1. Lorentz strange attractor

The most identifiable symbol, forever linked to the Butterfly Effect, is the famed Lorentz Strange Attractor (see figure 1). With the aim to describe convection in meteorological systems, Lorentz (1963) came to identify 3 nonlinear equations that show chaotic behavior can arise from simple deterministic models. The strange attractor is a plot of these 3 equations. Starting from any initial condition, the calculations will approach the path displayed in figure 1; however, the actual path followed by the equations is highly dependent upon the initial conditions. Lorentz apparently discovered the chaotic nature of these equations after quickly recalculating his simulations with input data that had the fractional parts truncated to save data entry time on his computer system. He successfully demonstrated sensitive dependence upon initial conditions.

Chaos theory states that many apparently random events can be represented using simple computations, when iterated, that produce complex results. The values from each stage of the iterated computations are generally fedback into the next stage. Different algorithms produce different amounts of complexity when iterated. Lorentz was the first to show this type of chaotic behavior (Lorentz, 1963).

The results of a chaotic system, while deterministic and predictable in theory, are in reality random and unpredictable due to the sheer complexity of dependencies. Results from a chaotic system can appear to be far from rational.

Chaotic Systems can only be predicted if all the inputs to the system and all the rules of the system are known. Even chaotic systems with known rules will likely remain unpredictable due to the butterfly effect. Slight errors in measuring the inputs of these systems will cause large deviations in the predicted results vs. the observed results over time. Many natural processes, as well as many man made processes, are chaotic systems. Examples include a dripping water faucet, the financial markets and global weather patterns.

Cellular Automata

Cellular automata (CA) are discrete dynamical systems whose behaviors are completely specified in terms of a local relation. A uniform grid usually represents space, with each cell location containing some bits of data. Time in a CA universe advances in discrete steps and the laws of such a universe is generally expressible in a small lookup table of rules.

A cellular automata is basically an array of cells that interact with their neighbors. This array can take on any number of dimensions. One-dimensional CAs are popular with researchers due to their simple to analyze rules. Each cell has its own state and receives input from connected cells, by using its set of rules it can then determine what its reaction will be. The reaction a cell takes will be a change of state and can also trigger a cell to send messages to other cells in more complex CAs.

FIGURE 2. Conway's game of life

A key feature of CAs is their ability to allow complexity to emerge by interaction of simple individuals following simple rules. Similar to the butterfly effect of chaotic systems, the dynamical systems defined as CAs often show huge resultant effects due to small changes in the starting conditions. The best-known Cellular Automaton is John Conway's Game of Life and is played on a 2-dimensional grid as demonstrated in figure 2.

Chris Langton (1986) proposed the lambda parameter for the standard CA as the percentage of nonzero transitions in the CAs state transition table and established its significance as a dynamical measure. The lambda parameter, when increased from 0 to 1, shifts the behavior of the CA from simple monotonic (Class I as defined by Wolfram) to periodic (Class II), then to chaotic (Class III), while passing briefly through an intermediate stage of long transients and complex structures (Class IV). Langton theorized that living systems and other self-organizing systems display qualitative features associated with this brief interlude between periodicity and full chaos (Langton, 1986, pp. 120-149).

The Neo-Darwinian Paradigm

The Darwinian paradigm uses the theory of Natural Selection or "survival of the fittest" as the mechanism for evolution and creating diversity among species. Variations in an individuals genome that provide advantages in reproductive success will be favored, while other variations that decrease reproductive success will tend to be eliminated. The Neo-Darwinist theories of evolution form the underpinnings of the classical genetic algorithm.

Application of Research Areas to This Thesis

This thesis makes use of many of the research areas described above. Neuroevolution (NE), the primary means of training the embodied agents, has benefited from research into evolution, chaotic systems, and cellular automata. The genetic algorithm as employed by NE models biological evolution and the artificial neural network models biological neural networks, both borrow heavily from the Neo-Darwinist theories of evolution.

A pivotal requirement for the embodied agents is the ability to learn from their mistakes, and beget offspring with the tendency to have a higher fitness than their parents. NE employs reinforcement learning techniques that utilize Genetic Algorithms, Artificial Neural Networks and Evolutionary Computation. The classical AI approach cannot be utilized due to the rigid rules and lack of learning and evolutionary algorithms. A classical approach would likely not produce the results we are looking to achieve; therefore, we chose to implement techniques from the newer bottom-up approach to AI.

Adopting the use of Evolutionary Algorithms and utilizing the bottom-up approach rather than the top-down approach of traditional AI place great pressure upon

the population of embodied agents to learn to improve and adapt their behaviors otherwise face certain extinction. This use of evolution and survival of the fittest allows the fittest embodied agents to evolve efficient locomotive behaviors along with their corresponding control systems without the need for an external agent to direct the learning process.

The Genetic Algorithm

Genetic algorithms (GAs) are evolution inspired algorithms used for optimization and machine learning based loosely upon biological evolution. Invented by John Holland in the 1960's, his original goal was the study of adaptation as it occurs in nature and to simulate such mechanisms in a computer system. As in the case of AI, there exists no clear definition of a GA. However, it can be said that most methods called "GAs" have at least the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring and random mutation of new offspring (Mitchell, 1999, p. 8). Unlike many other optimization algorithms, the GA is probabilistic rather than deterministic. The genetic algorithm works by creating a large population of individuals, each of which is represented by a chromosome(s) that are analogous to the chromosomes present in human DNA. The individuals in the population then go through a simulated process of natural evolution involving fitness calculation, mating, reproduction and generation of offspring. A simple GA can be broken down into 6 steps: 1) Define and capture the problem in an objective function that will allow the calculation of fitness of any potential solution, 2) then create an initial population of random individuals whose genes represent a solution to the problem defined above, 3) then decode the chromosome and compute the fitness for every individual based upon the

predetermined fitness function, 4) assign each chromosome a probability of reproduction based upon its fitness score, 5) according to the probabilities of reproduction, create a new population by performing the crossover operation to each pair of mating individuals to produce a set of offspring which are then subject to a small probability of bit mutations, 6) if a suitable solution has been found, halt the process, otherwise proceed at step 3 with the new set of chromosomes generated in this step. One iteration of the above loop is called a generation. The first generation of this process operates on randomly generated individuals; however, subsequent generations of this process operate to improve the population by employing the genetic operations in concert with the measure of fitness. The GA effectively searches the fitness landscape for a peak that represents the highest fitness score obtainable. Exploiting the parallel nature of the search, as well as the directed nature of the search itself, the GA is generally able to converge upon an optimal solution. The repetition of the above steps for several generations is the process that drives for the selection of individuals of successively higher fitness with each passing generation (i.e., evolution). By allowing the simulation to run for a sufficient number of generations, highly optimized solutions will tend to evolve.

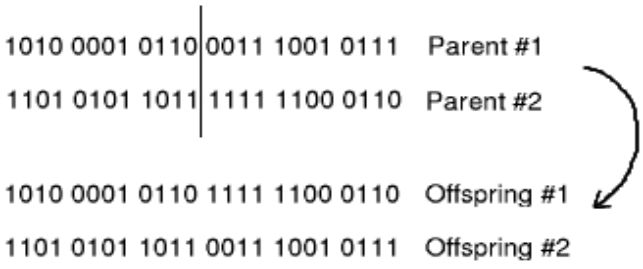


FIGURE 3. The crossover operation

A simple genetic algorithm that yields good results in many practical problems is composed of three operators: (1) reproduction, (2) crossover and (3) mutation (Goldberg,

p. 10). Reproduction is the process in which individuals mate according to their fitness values. Individuals with a higher fitness have a higher chance of reproducing and producing offspring; hence, promoting their good genes. Simple reproduction generally uses a method called roulette wheel selection where the probability of an individual mating is proportional to its fitness. Simple single-point crossover takes a pair of mating individuals and combines their genes. The point at which the 2 chromosomes cross is generally randomly determined and is known as the crossover point and is demonstrated in figure 3. The simple mutation operator randomly modifies a gene to produce a new value mainly for the purposes of exploration of the fitness landscape.

The fundamental theory behind the genetic algorithm is coined: the Schema Theorem. The theorem implies that GAs work by discovering and manipulating short, low-order "building blocks" of solutions of above average fitness in an implicitly parallel fashion. In other words, good solutions tend to be made up of good building blocks.

The simple GA is prone to premature convergence due to multiple peaks in the fitness landscape. At the start of a GA run it is common to have a few extraordinary individuals in a population of mediocre colleagues. If left to the normal selection rule, the extraordinary individuals would take over a significant proportion of the finite population in a single generation, and this is undesirable, a leading cause of premature convergence (Goldberg, 1989, p. 77). By scaling the fitness of the individuals, we can help prevent premature convergence by giving less weight to large fitness variations early on in a run, and giving greater weight to small fitness variations later in a run.

Because the GA is an optimization algorithm, it can be used to optimize the structure of other Evolutionary Algorithms, such as Artificial Neural Networks. A

slightly modified genetic algorithm that implements fitness scaling is used in this experiment, with the purpose of optimizing the neural connection weights of the embodied agent's ANN to produce intelligent locomotive behaviors. The use of a genetic algorithm to train an ANN is a novel and relatively new approach to ANN learning known as neuroevolution.

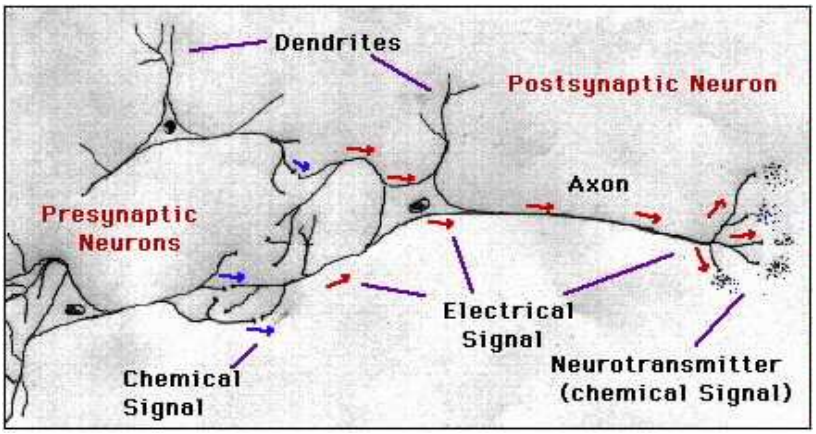


FIGURE 4. Biological neural network

The Artificial Neural Network

Artificial Neural Networks (ANNs) are information-processing systems that have certain performance characteristics in common with biological neural networks. Biological neural networks have neurons that emit electrical signals along an axon to the dendrites of other neurons. Figure 4 demonstrates how a biological neural network transmits data. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology (Fausett, 1994, p. 3). In other words, Artificial Neural Networks are a different paradigm for computing that exploit the parallel architecture of biological brains. According to Fausett, a neural network is characterized by 3 things: (1) the architecture or structure of the ANN, (2) the learning method used to determine the weights of the network, and (3) the activation function of

the neurons. A typical artificial neural network consists of a collection of "neurons" or simple processing elements that attempt to mimic on a much smaller scale, the massively parallel structures and connections of biological neural networks (i.e., biological brains). Every neuron in an ANN contains an internal state that is defined by an activation function. Activation functions generally employ a nonlinear squashing function such as the hyperbolic tangent, or the sigmoid function to the weighted sum of the input signals on a neuron. The architecture of a simple feed-forward ANN consists of a collection of neurons that are arranged into distinct layers that forward propagate their output signals into the inputs of the next layer of neurons. Generally, 3 layers are used: an input layer, a hidden layer, and an output layer. A set of inputs are fed into the first layer of the network with each neuron taking the weighted sum of its inputs and applying the activation function before propagating its output to the next layer of neurons and eventually reaching the output of the neural network. Multilayer nets as described above can be trained to perform nonlinear mappings from an n-dimensional space of input vectors (n-tuples) to an m-dimensional output space (Faucett, 1994, p. 16). Another important characteristic of ANNs that are shared with biological neural networks is their fault tolerance characteristics. In biological systems, damage to the neural system itself can often be tolerated with little ill effect and other neurons can often be trained to take over the functions of the damaged cells. Similarly, artificial neural networks can be designed to be insensitive to small amounts of damage to the network, and retraining can occur with larger amounts of damage. The typical feed forward ANN configuration is demonstrated in figure 5.

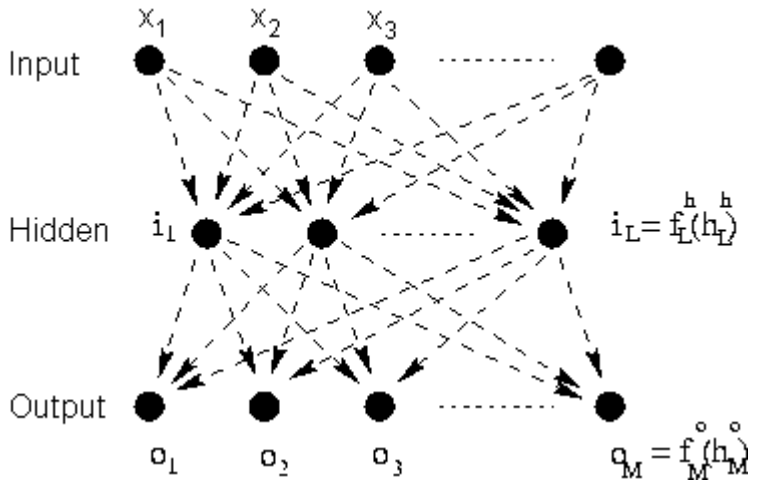


FIGURE 5. Simple fully connected feed forward neural network

The architecture of the ANN determines its method of learning or training. An ANN can be trained using either a supervised approach or an unsupervised approach. In the most typical neural network setting, a simple feed forward ANN architecture, learning is accomplished via supervised training using one of several different learning algorithms, the most common of which is the back propagation (of errors) algorithm. Back propagation is the basis for training a supervised ANN and is used to produce a mapping of static (time independent) input to a static output. The back propagation algorithm works by its application of the chain rule for ordered partial derivatives to calculate the sensitivity that a cost function has with respect to the internal states and weights of a network. In other words, back propagation is simply a gradient descent method to minimize the total squared error of the output computed by the net. The prerequisites for training an ANN using the back propagation algorithm include sample-training data consisting of inputs to the ANN and the corresponding expected outputs. A back propagation net (a multilayer, feed forward ANN trained using the back propagation algorithm) is generally used for solving static classification problems such as optical

character recognition. A significant disadvantage of the back propagation training method is the requirement of training data. For several applications, especially those involving control systems, it is often impossible to generate a sufficient quantity of training data.

Time is clearly important in cognition and is inextricably bound up with many behaviors (such as language), which express themselves as temporal sequences. Indeed, it is difficult to know how one might deal with such basic problems as goal-directed behavior, planning, or causation without some way of representing time (Elman, 1990). Another type of ANN that makes use of time and is both biologically more plausible and computationally more powerful is the Recurrent Artificial Neural Network (RNN). RNNs are artificial neural networks with adaptive feedback connections. Each time a pattern is presented on the inputs of an RNN, the respective neurons compute their activation functions just as in a feed forward network; however, the net inputs to each neuron now contain a term that reflects the state of the network before the input pattern was seen. The obvious advantage an RNN offers over the traditional feed forward network is "memory." The use of feedback connections allow the RNN to have a "memory" of past events; thus, pattern presentation to the RNN will now take into consideration what moment in time the pattern occurs. Biological neural networks process information in a similar fashion to the RNN. Figure 6 demonstrates the feedback connections and context units of the RNN.

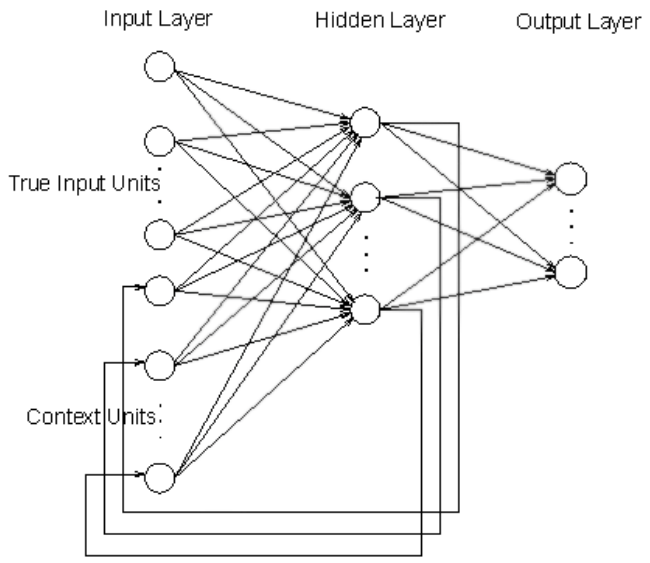


FIGURE 6. A simple recurrent neural network (RNN)

Drawbacks of the RNN include: they are computationally more intensive than feed forward ANNs and the standard method of learning via back propagation of error does not work with RNNs. New methods of training RNNs have been devised; one approach has become known as the Back Propagation Through Time (BPTT) method. BPTT can be seen as an approximation to the ideal of computing a gradient that takes into consideration all the inputs seen so far by the network. The disadvantages of BPTT are similar to the back propagation training method and include the requirement of large amounts of storage, computation, and training examples consisting of known solutions.

The embodied agent of this thesis is endowed with an Elman (1990) Recurrent Neural Network for its biological plausibility and powerful memory capabilities. The BPTT algorithm is unusable in this application because of the requirement of training data. Furthermore, biological neural networks do not make use of back propagation for learning. Because of our desire to use evolutionary algorithms to evolve intelligent

locomotive behaviors that give each embodied agent the highest likelihood of survival, we chose to utilize NE as a relatively new reinforcement learning approach for RNNs that make use of GAs.

Neuroevolution

In difficult real-world learning tasks such as controlling robots, pursuit & evasion tasks, or game playing, it is impossible to specify correct actions for each situation. In a complex control system, such as those used by the embodied agents in this thesis, specifying the correct outputs for each possible input combination and state is practically impossible. In these situations, optimal behavior must be learned by the exploration of different actions, the reinforcement of good decisions based upon some feedback from the system itself and the exploitation of learned knowledge of the environment. NE techniques evolve neural networks using the optimization process of genetic algorithms, and allow for the evolution of robust solutions to difficult real-world learning tasks without the need to supply additional information or for an external agent to direct the process. NE is often more robust and less susceptible to noisy input than traditional back propagation training methods due to NEs evolutionary nature.

Traditionally, NE begins with a fixed topology for the evolving neural networks. A fully connected network topology with a hidden layer of neurons is typically used. Once the topology is chosen, evolution searches the connection weights of this network by allowing the reproduction of several networks and evaluating their performance. A Genetic Algorithm controls the evolution and reproduction of the neural networks.

The exploration of the weight space is done via the crossover of network weight vectors and through mutation of single networks' weights (Stanley and Mikkulainen, 2002). Only allowing the best performing networks to reproduce reinforces robust solutions.

In this thesis, NE makes use of the powerful optimization capabilities of the GA by encoding the weights of the RNN into a chromosomal format compatible with the GA. The GA is run for several generations until a successful solution (RNN weight structure) is evolved. Though this method is very computationally intensive, it holds a distinct advantage over the traditional learning methods of ANNs: unsupervised reinforcement learning. In other words, NE does not require the use of training data or an external agent to direct the learning process and will eventually produce an optimal (or near optimal) solution after several generations with the only requirement being a well-defined fitness function. Furthermore, NE is more biologically plausible than the BPTT method of learning. The user may sacrifice some control, especially with a procedurally defined fitness function; however, the potential gain in automating the learning process and the creation of a complex control system compensates for the loss (Sims, 1994).

Standard NE is highly effective in reinforcement learning tasks such as robotic control; however, a significant advantage can be gained by evolving neural network topologies along with the weights. Algorithms that expand upon the basic principles of NE include SANE (Moriarty, 1997) which evolves the topology of the network, as well as the weights, and NEAT (Stanley and Mikkulainen, 2002) which starts with a minimal network and "complexifies" the network as necessary to evolve an optimal and efficient solution to the problem. Both the above methods increase the efficiency of the NE system, and make it possible to evolve ever increasingly complex solutions over time.

The NEAT algorithm can further make use of competitive co-evolution. In competitive co-evolution, the goal is to establish an "arms race" that will lead to increasingly sophisticated strategies (Stanley and Mikkulainen).

Environmental Complexity

Even disregarding issues of biological plausibility, the coupling of embodied agents with the environment brings with it a major methodological problem: results reporting behaviors of different organisms in different environments are incommensurate. It is, therefore, difficult to assess whether an apparently superior behavior is the consequence of more sophisticated adaptive techniques, or is due to the relative complexity of the environments. There exists a great desire to be able to define artificial environments of controlled complexity, within which a wide range of A-Life techniques may be directly compared (Menczer, 1998).

Godfrey-Smith attempts to characterize the generic conditions on complexity of environments by the number of states they present to their organisms, the frequency of their change, and their overall heterogeneity (Godfrey-Smith, 1996). Several other researchers propose food density as another generic condition of environmental complexity; however, we define the only generic condition of environmental complexity to be the accurate modeling of natural physics. In the natural world, this is one of the few conditions we cannot change and this condition alone is sufficient to allow the emergence of complex behaviors. Furthermore, as stated earlier, our long-term goal is the transfer of evolved intelligent behaviors into physical manifestations of the embodied agents within the natural world.

The accurate modeling of the physics of the natural world allow for commensurate comparisons to be made between the physical and virtual manifestations of the embodied agents.

Evolutionary Computation and Emergence

Evolutionary computation (E.C.) is an umbrella term that encompasses several computational techniques that are to some degree based upon the evolution of biological life in the natural world. The term is relatively new and represents an effort to bring together researchers who have been working in closely related fields, yet following different paradigms. EC involves research into genetic algorithms, evolutionary strategies, evolutionary programming, and artificial life. EC struggles with the same ideas as Artificial Life: determining how to represent "solutions" to an environment, determining which "solutions" are able to reproduce, determining how the reproduction mechanisms work and determining which life forms should die. EC also concerns itself with the global behaviors that emerge from simple and local interactions.

The relevance of EC to this thesis is demonstrated in the emergent properties of many of the computational techniques that we utilize. This thesis does not directly make use of the emergent properties of EC; however, by employing a Genetic Algorithm, we can take advantage of past behaviors and make use of a "global knowledge" when the population reproduce and indirectly take advantage of the emergent properties of EC. One possibility is to allow the population of individuals themselves to interact with one another and develop a group mentality or behavior. With the entire population cooperatively (or competitively) working on a solution to a problem, novel global solutions may emerge.

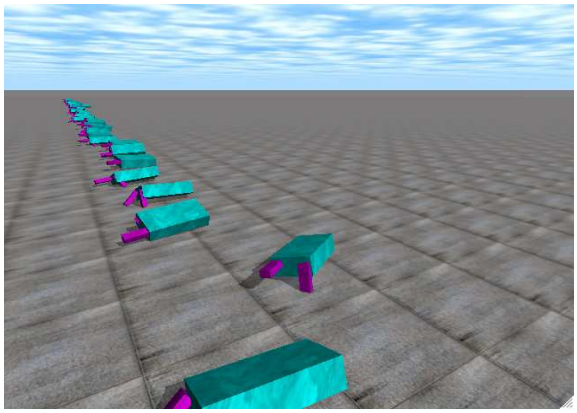


FIGURE 7. Evolving a population of embodied agents

Parallel Processing and Fault Tolerance

Parallel processing is the use of multiple processing elements to execute different parts of the same program simultaneously. The main goal of parallel processing is to reduce the overall time required to complete a computational task. In theory, a thousand simple processing elements working in unison toward the same goal would be equivalent to one processing element that is a thousand times more powerful than the single simple processing element. Biological neural networks make use of parallelism by their implementation of millions or billions of simple neurons (processing elements) and their vast network of connections working together to comprise an extreme parallel processing system. Both GAs and ANNs are inherently parallel algorithms; however, their implementation is often on serial computational devices such as a uniprocessor computer, thus the speed advantages of the implicitly parallel structure of these algorithms are often lost on such devices. The genetic algorithm is inherently parallel with each individual population member having few if any dependencies upon the other individuals. There is no question that a biological neural network, with their densely interconnected parallel structure is a parallel information-processing device. The artificial neural network, an

information-processing paradigm inspired by the biological neural network is also an inherently parallel structure. Though both the GA and ANN are based on parallel structures or algorithms, they may not be ideally suited for implementation on today's parallel machines. Communication delays and synchronization issues within their structures can hinder the implementation of these algorithms onto a parallel machine.

Much research has been done on the implementation of both GAs and ANNs on parallel computers. Issues such as synchronization, global communication, and the amount of implicit parallelism of the fitness function all need to be considered when implementing a genetic algorithm or a neural network on a parallel machine.

The Embodied Agent

An agent, in the context of computer science, often refers to a piece of software that can perform its tasks in an intelligent manner. An agent that exists in a virtual world, like their biological counterparts, requires an interface (or body) to interface with the environment. Agents that exist within a virtual world are often known as animats or virtual creatures. An embodied agent is an autonomous living creature, subject to the constraints of its environment. The consequence of giving a software agent a body to control subjects the agent to the forces of its programmed environment. The impetus for creating embodied agents is realism and modularity. A typical embodied agent with 2 appendages composed of several rigid bodied interconnected via multiple hinge joints is shown in figure 8.

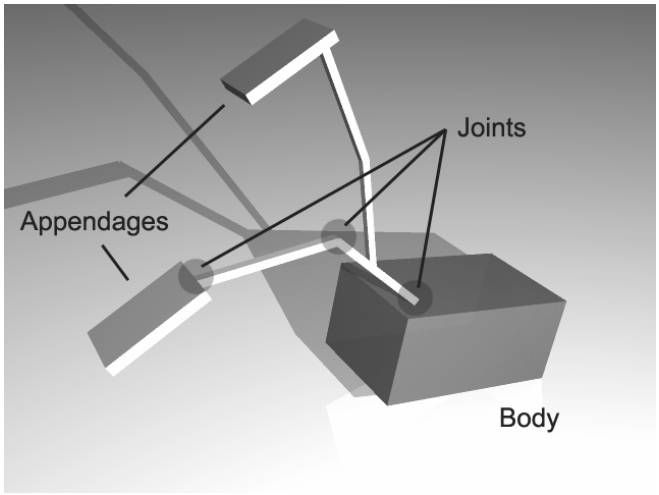


FIGURE 8. An embodied agent

The embodied agent can model a biological creature with its sensors and outputs (i.e., muscle output). Vision and movement are the 2 most common attributes an embodied agent can posses. The embodied agents in this experiment were endowed with sensors to provide the ANN with input from the environment, and effectors that allow the ANN to control the movement of the agent.

Artificial Life and Virtual Creatures

There has been little work in the field of A-Life involving simulations that utilize a physically accurate environment that models natural world rules of physics. Many recent studies in artificial life make use of a simplified environment to evaluate the learning process within the embodied agent; however, there have been a few studies aimed at accurately recreating a physically plausible environment.

The most complex and realistic simulation to date was performed by Karl Sims and presented at Siggraph in 1994. Sims's Virtual Creatures described a novel system for creating virtual creatures that move and behave in a simulated 3-dimensional physical world (Sims, 1994). Sims's GA evolved both the creature morphology and control system

simultaneously to perform a variety of tasks ranging from walking to competing with other virtual creatures for control of a block. The brains of Sims virtual creatures resembled that of a data flow computer program more than a neural network, and were able to process sensory information from the environment and produce motor output. Sims ran his simulations on a Connection Machine supercomputer because of the high computational complexity of his simulations.

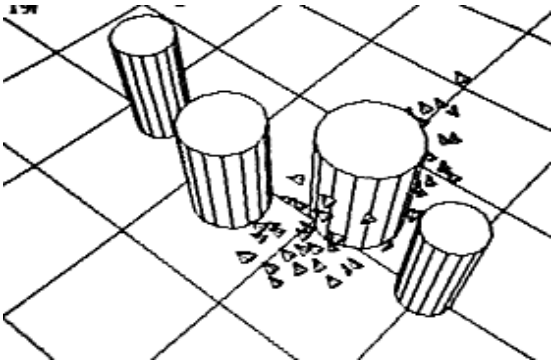


FIGURE 9. Craig Reynolds boids simulation (Reynolds, 1987)

One of the first 3-dimensional computer simulations of embodied agents was demonstrated by Craig Reynolds who simulated the aggregate motion of a flock of birds within a 3-dimensional environment (Reynolds, 1987). The simulation was an elaborate particle system, with each bird represented by a particle with the aggregate motion of the simulated flock controlled by a distributed behavioral model. The flocking behaviorisms demonstrated by the Boids were similar to those of birds in the natural environment.

The Boids made use of 3 basic rules: (1) Steer to avoid getting close to neighbors, (2) Steer to keep on the average heading of the flock, (3) Steer to stay near the average position of the neighbor. The emergent global behavior that arose from the simple interactions was astonishing at the time. The Boids were able to navigate, as a flock, across an area of columns.

They were able to flow around the columns, and remained in a group formation by speeding up or slowing down.

CHAPTER 3

DESIGN APPROACH AND IMPLEMENTATION SPECIFIC DETAILS

Evolutionary computation methods are borrowed from natural living systems. Because of our desire to evolve intelligent behaviors for embodied agents in a life-like environment, we employ several different Evolutionary Algorithms. As stated earlier, the goal of this thesis is to develop a framework for evolving life like and optimal behaviors in virtual creatures that can be easily transferred to real world machines. This framework consists of the evolutionary algorithms employed in evolving the brain of each embodied agent, the virtual environment the agents "live" within, and the morphology of the agents themselves. In this chapter we discuss the details of the environment, the agent morphologies, and the union between the GA and RNN.

Virtual Environment and Physics Engine

Environmental complexity plays a large role in the successful evolution of complex biological organisms. In an effort to allow for commensurate comparisons between physical agents of the natural world and the agents of the virtual environment, an accurate physics model is employed. The virtual environment utilizes a sophisticated physics engine to accurately simulate rigid body dynamics, joints, contacts/collisions, friction, inertia and gravity by simulating natural world physics. Due to the relatively low velocities expected, and to reduce the computational complexity of the simulation, the physics engine uses Newton's model rather than Einstein's equations. By employing

an environment that accurately models real world physics, we can ensure that the embodied agents that exist within the virtual environment are unable to generate any motions or forces that would otherwise be irreproducible in our natural environment.

The embodied agents that live within this environment are composed of a series of rigid bodies interconnected via joints. The physics engine acts directly upon the rigid bodies of the agents; therefore, the agents themselves are subject to the rules of physics that govern the environment. Each rigid body can be further constrained by the use of a joint. A joint can connect 2 or more rigid bodies and can be permanent (such as a hinge or slider joint), or can be temporary and a result of the collision of 2 rigid bodies (such as a contact joint). When an embodied agent's appendage collides with the ground plane, a temporary contact joint is created.

Gravity in the virtual world is set at approximately 9.8 m/s^2, to provide a close approximation of real world gravitational forces at sea level. Ground friction is necessary for locomotion, and is modeled in the temporary contact joint created during a collision. Friction within the agent's joints is also modeled by the use of a small negative torque within the joint itself. Other forces such as wind resistance are so minor that we choose not to simulate them in order to reduce the computational complexity of the system. The landscape of the virtual environment is completely barren; an infinite horizontal place with a coefficient of friction similar to that of asphalt acts as the ground upon which the embodied agents can use their appendages to generate forces.

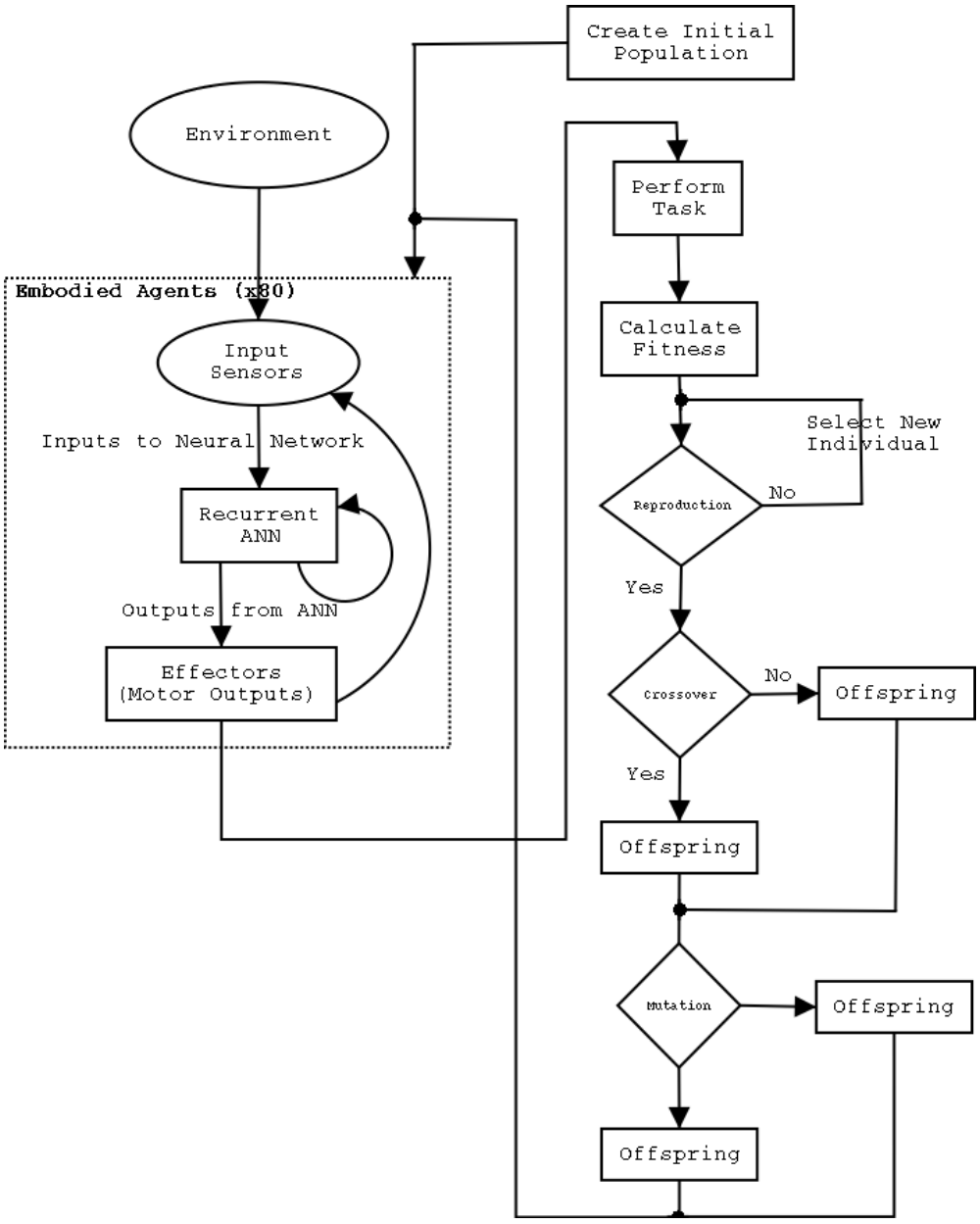


FIGURE 10. System flow diagram

As stated earlier, the physics engine interacts directly with all the rigid bodies within the virtual environment; therefore, the embodied agents themselves are subject to the constraints imposed by the physics engine. The system flow diagram is shown in figure 10, and represents an overview of how the environment interacts with the embodied agents to allow for locomotive strategies, such as walking, to develop. The

system flow diagram can be broken down into six distinct steps: (1) Creation of a new population of embodied agents whose actions are immediately constrained by the physics engine. (2) Agents perform tasks dictated by the predefined fitness function. In this case, the agents employ a variety of input sensors to gather environmental data, which are processed by the RNN to generate effector outputs, which physically move the agent's joints and limbs, thus generating motion. (3) Calculate the Fitness scores based upon how well the agents were able to perform their tasks in step 2. Each individual is given a fitness score proportional to the distance traveled along a predetermined axis and then the entire population is linearly fitness scaled to reduce large variations in fitness. (4) Select individuals for reproduction with the chances of being selected being proportional to the fitness score. This method is also known as roulette wheel selection. (5) Crossover selected individuals from step 4 by performing the single point crossover operator as shown in figure 3. The probability of being select for crossover Pc is 80% as defined in table 2. (6) Mutate selected individuals from step 4 by employing a Gaussian mutation operator with the probability of mutating any given gene Pm at 0.10%, generate new population and repeat at step 2. The symbiotic relationship between the RNN and the GA can be clearly seen in the system flow diagram as the GA works to optimize the RNN.

A population of 80 embodied agents begins each generation at their initial position of $(0,y_0)$ on the horizontal ground plane as shown in figure 11. The population is given a fixed amount of time to move as far along the x-axis in the negative direction as possible. After a fixed amount of time, each individual embodied agent is evaluated and assigned a fitness value corresponding to how well it performed the task.

Agents that travel longer distances are assigned proportionately higher fitness scores than agents that are unsuccessful at traveling.

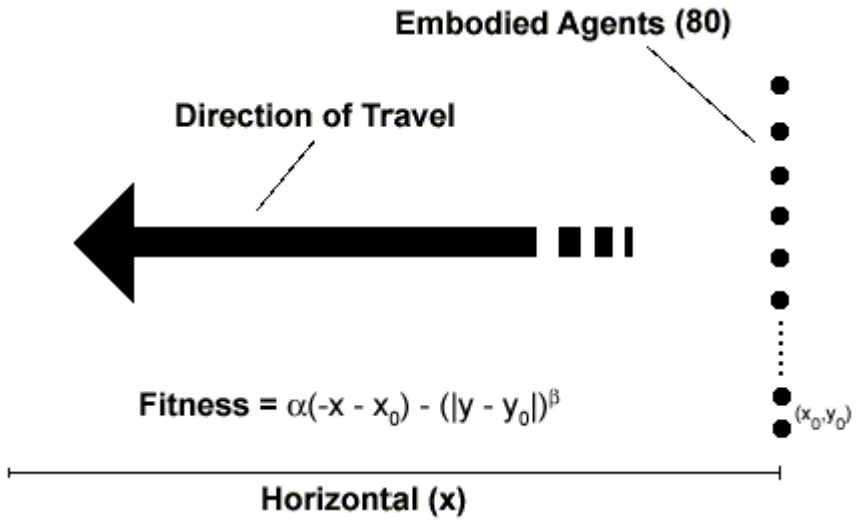


FIGURE 11. Initial starting conditions for each generation

Rigid Bodies

A rigid body has various properties from the point of view of the simulation. Four properties of rigid bodies that change with time are: (1) Position vector (x,y,z) of the body's point of reference corresponding to a bodies center of mass, (2) linear velocity vector of the point of reference (vx,vy,vz), (3) orientation of a body, represented by a quaternion (qs,qx,qy,qz) or a 3x3 rotation matrix and (4) angular velocity vector (wx,wy,wz) that describes how orientation changes with respect to time. Rigid body properties that remain constant over time include: (1) Mass of the body, (2) position of the center of mass and (3) inertia matrix describing how the body's mass is distributed around the center of mass (Smith, 2002). These properties are used internally within the physics engine to calculate the forces and torques that affect the rigid bodies. The calculations are described in detail later in this chapter.

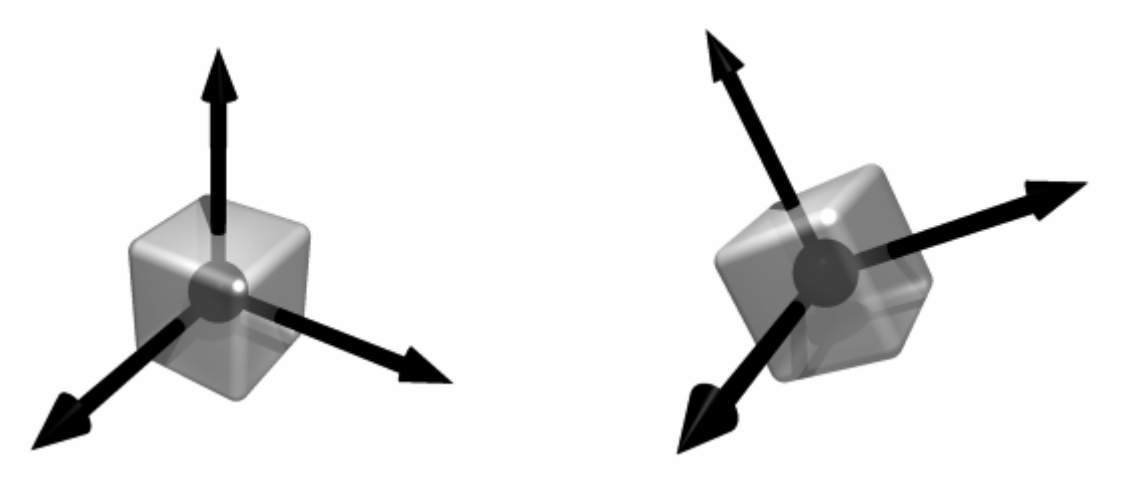


FIGURE 12. Representation of rigid bodies

Conceptually, each body has an x-y-z coordinate frame embedded in it that moves and rotates with the body, with the origin of this coordinate frame at the body's point of reference as shown in figure 12. A rigid body can be represented, for collision detection purposes, by a rectangular box-like structure or a cylinder.

Hinge Joint

The hinge joint, as shown in figure 13, constrains the motion of the two attached rigid bodies to rotate about the axis. The hinge joint is the simplest joint with only 2 sensors: (1) current hinge angle and (2) current hinge angle rate. Let *h(t)* be the current angle between 2 bodies at time t, then the hinge angle rate *r(t)* is defined as the time derivative of the hinge angle:

$$r(t) = \frac{d}{dt} h(t)$$

The value returned for the hinge angle will be between –pi and pi. The hinge joint has 2 sensors that can provide the current angle and angular velocity to the embodied agent. The joint also has 1 effector that accepts the desired velocity as its input.

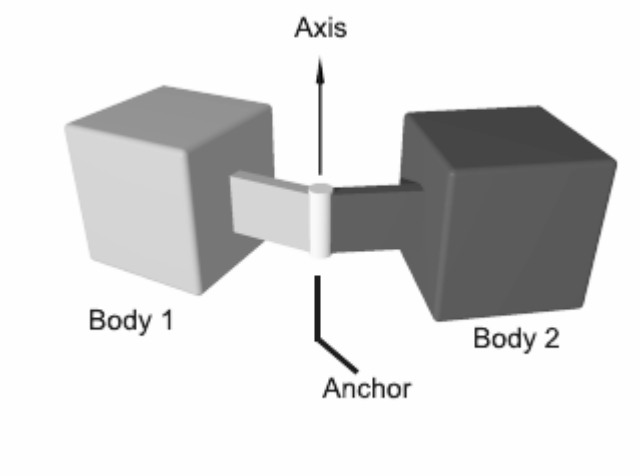


FIGURE 13. Hinge joint

Slider Joint

The slider joint, as shown in figure 14, constrains the motion of the attached bodies to move along the axis. The slider joint is another simple joint that allows for only 1 degree of freedom. The 2 sensors of the slider joint provide the following information: (1) the slider joint's current position and (2) the rate of position change. Let *s(t)* be the current slider position between 2 bodies at time t, then the slider position rate *p(t)* is defined as the time derivative of the slider position:

$$p(t) = \frac{d}{dt} s(t)$$

The current position is returned as a number between –1 and 1, with 0 representing the midpoint between the 2 joint extremes. The sliders 2 sensors provide linear position and linear velocity information to the RNN of the embodied agent. The slider has 1 effector that can accept the desired velocity as its input.

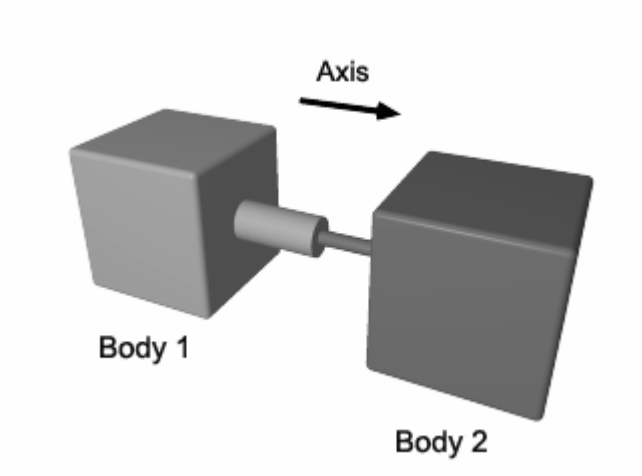


FIGURE 14. Slider joint

Ball and Socket Joint

The ball and socket joint, as shown in figure 15, is the most complex joint currently used by the embodied agents. The ball and socket joint allows for 3 degrees of freedom about the 3 axes. The ball and socket joint provides 2 feedback sensors for each axis. The 2 sensors provide the following data: (1) ball and socket angle for the given axis and (2) ball and socket angle rate for the given axis.

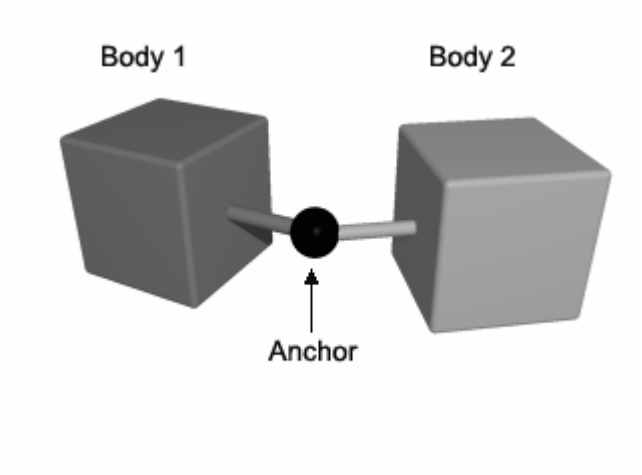


FIGURE 15. Ball and socket joint

The angle rate for an axis is calculated by taking the derivative with respect to time of the current angle on the particular axis. Due to the 3 degrees of freedom, the 2 sensors can provide up to 6 pieces of information to the RNN.

Contact Joints

The contact joint prevents body 1 and body 2 from inter-penetrating at the contact point. It does this by only allowing the bodies to have an "outgoing" velocity in the direction of the contact normal. Contact joints are typically created and deleted in response to collision detection. Contact joints simulate friction at the contact by applying forces in the 2 friction directions that are perpendicular to the normal (Smith, 2002).

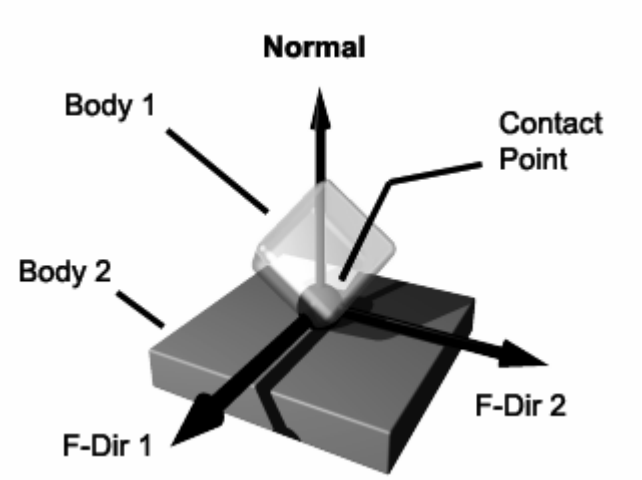


FIGURE 16. Contact joint

Sensory Input

The embodied agents employ a host of sensors to collect data from the environment and feed the data to the RNN. Table 1 demonstrates the variety of sensors employed in this simulation as well as the internal RNN representation of the sensory data. The table also demonstrates the range of values that the sensors are able to detect.

The sensors update their output values at every simulation time-step and are perfect in the sense that they do not generate false or erroneous data.

Four of the 8 sensors perform a linear scaling of their output before presenting it to the RNN. The reason for the linear scaling of the sensory data is to keep the data within the range of the RNNs squashing/activation function. The 8 sensors output a range of values based upon their inputs except for the touch sensor, which can only output 2 discrete values. The touch sensor is modeled after a momentary toggle switch, with 2 positions: on and off.

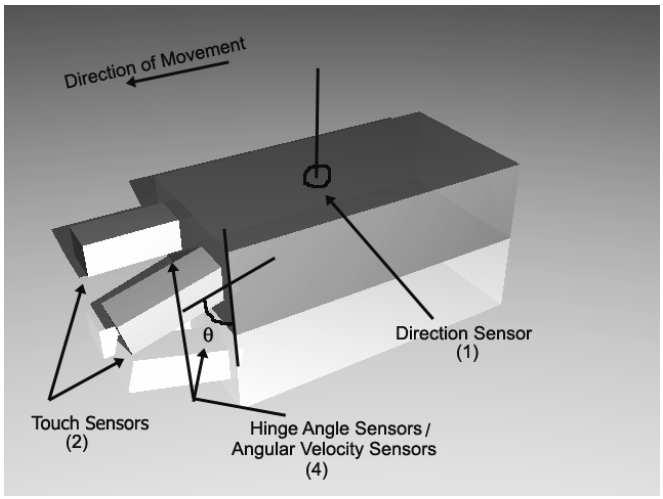


FIGURE 17. Sensor arrangement

The arrangement of the sensors is shown in figure 17. In this figure, the embodied agent has 7 sensors located throughout its body. The touch sensors indicate when the frontal appendages make contact with the ground plane, and the angle sensors measure the angle the front appendages make with respect to the agents body. The hinge joints also contain angular velocity sensors that feedback the rate of angular change to the RNN. Finally, a direction sensor provides the ANN with a sort of compass.

The other morphologies have a similar sensor arrangement, with the only difference being the type or number of sensors.

TABLE 1. RNN Representation of Sensory Data

Sensor Type	Output Range	RNN Representation
Hinge	[-pi,pi]	[-3.1416,3.1416]
Slider	[-1,1]	[-1,1]
Ball/Socket	[-pi,pi]	[-3.1416,3.1416]
Linear Velocity	[-300,300] cm/sec	[-1,1]
Angular Velocity	[-100,100] rad/sec	[-1,1]
Height	[0,1000] cm	[0,1]
Direction	[0,2pi] rad	[-1,1]
Touch	on / off	-1,1

Effector Outputs and Angular Dynamics

Every joint has a motor associated with it known as an effector. The effector applies torque to a joints degree(s) of freedom to get it to pivot or slide at the desired speed. Effectors have limits to the maximum amount of torque that can be generated, and will be unable to apply more than a given maximum force or torque to a joint. The embodied agents effectors allow them to control the relative angular or linear velocities of two bodies connected via a joint, thus enabling them to control their appendages and produce motion.

TABLE 2. RNN Representation of Effector Data

Effector/Joint	Max Force/Torque	Input Range	RNN Representation
Hinge	800 gm-cm	[-10,10] rad/sec	[-1,1]
Slider	1200 gm-cm	[-100,100] cm/sec	[-1,1]
Ball/Socket	600 gm-cm	[-10,10] rad/sec	[-1,1]

The RNN representation of effector data is shown in table 2. Due to the activation function chosen for the output layer of the RNN, the RNN can only effectively output values within the range of –1 to 1. These outputs are fed directly to the inputs of

the effectors. The effector takes the input value and converts it into a desired speed by linearly scaling the effectors inputs to the input range as shown in table 2. The maximum force/torque is predetermined for each type of effector, and is designed to be similar to several common types of DC electric motors. Using the equations below, the physics engine can quickly determine the acceleration experienced by the bodies attached to the joint based upon the desired speed set by the RNN.

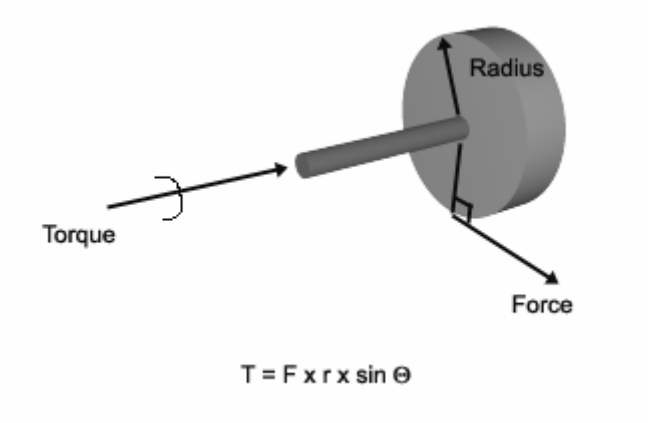


FIGURE 18. Effector model for hinge and socket joints

Effectors employ a simple model of real life motors, as shown in figure 18, with two parameters: (1) the desired speed and (2) the maximum force that is available to reach the desired speed. Effectors can also be used to model geared motors (motors attached to gearboxes). Such devices are often controlled by setting the desired speed, and can only generate a maximum amount of power to achieve that speed (Smith, 2002). By employing the geared model, we can effectively reduce the number of outputs required by the RNN to just one per effector: the desired velocity.

Slider joints are simply modeled by employing Newton's second law to calculate Force **F**:

$$\text{Force:} \quad F = m \cdot a$$

Where **m** represents the mass of the rigid body and **a** is the acceleration. The hinge and socket joints are modeled utilizing Newton's second law for rotation to come up with the basic torque equations as a function of force, angle, and radius is defined as:

$$\text{Torque:} \quad \tau = F \cdot r \cdot \sin\theta$$

where **τ** is torque, **r** represents the radius, **F** is the force, and **θ** is the angle between the line made by r and F. The basic torque equation is used to calculate how much torque the effector will apply at either a hinge or ball and socket joint.

The other rotational force the physics engine must consider is the moment of inertia. The moment of inertia of a rigid body is its measure of how difficult it is to start rotating and it depends upon where the axis of rotation is (either a joint or the center of gravity for a freely rotating body) and the mass of the object. The physics engine computes the moment of inertia **I**, by first breaking up the rigid body into several small pieces, then multiplying the mass of each piece by the square of the distance from its axis of rotation **r**, and adding all these products up:

$$\text{Moment of Inertia:} \quad I = \sum m \cdot r^2$$

The rigid bodies of this simulation have a homogenous distribution of mass about the body's center of mass.

ANN Implementation Details

The recurrent artificial neural network (RNN) is a fully connected multi-layer neural network also known as an Elman Network. The topology of the network is made up of 4 layers: (1) an input layer, (2) a hidden layer, (3) an output layer and (4) a context layer. The number of neurons contained in the input and output layer differ per morphology; however, the hidden layer consists of between 6 to 10 neurons that are fully connected to both the inputs and the outputs. Furthermore, the hidden layer has 6 or more layers of recurrence (context layers) depending upon the morphology of the embodied agent. Table 3 describes the specifics of each RNN for a particular morphology with the chromosome size representing the number of inter-neuron connections within the RNN. Each inter-neuron connection within the RNN is assigned a weight.

TABLE 3. RNN Implementation Details

Agent	Inputs	Hidden	Outputs	Context Layers	Neurons	Chrom. Size	Name
1	5	6	2	6	44	266	Crawler
2	9	8	6	8	78	646	Long Arms
3	13	8	4	8	76	660	Hopper
4	10	10	6	8	96	976	Runner

The RNN is a feed forward network employing hidden units and context (recurrent) units, which develop internal representations for the input patterns and recode those patterns in a way that enables the network to produce the correct output for a given input. The context units remember the previous internal state and provide the embodied agent with a short-term memory. The internal representations that develop are sensitive to temporal context; the effect of time is implicit in these internal states. They represent a memory that is highly task and stimulus dependent (Elman, 1990). This short-term

memory is necessary for the embodied agent to develop locomotive behaviors that can involve repetitive motor strategies (i.e., walking).

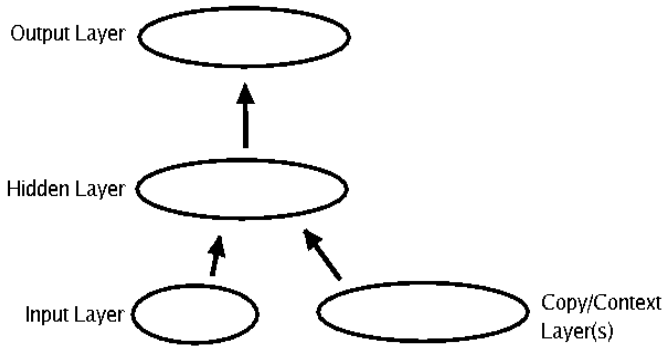


FIGURE 19. The structure of a generalized recurrent neural network

The activation at time *t*, for an arbitrary unit in a recurrent neural network is defined as:

$$y_i(t) = f_i(net_i(t-1))$$

At each time step, the activation propagates forward through 1 layer of connections only, where *net* represents some nonlinear activation function. Once a certain level of activation is present in the network, it will continue to flow around the units, even in the absence of new input. The activations in the hidden units are just the activations at time *t-1*. A generalization of this technique is used in this thesis: copy the input and hidden unit activations for a number of previous time steps. The more context (copy) layers that are maintained, the more history we are explicitly including in the activation computations. This approach takes into consideration not just the most recent inputs, but also all the inputs seen so far by the network. Figure 19 and figure 20, illustrates this approach.

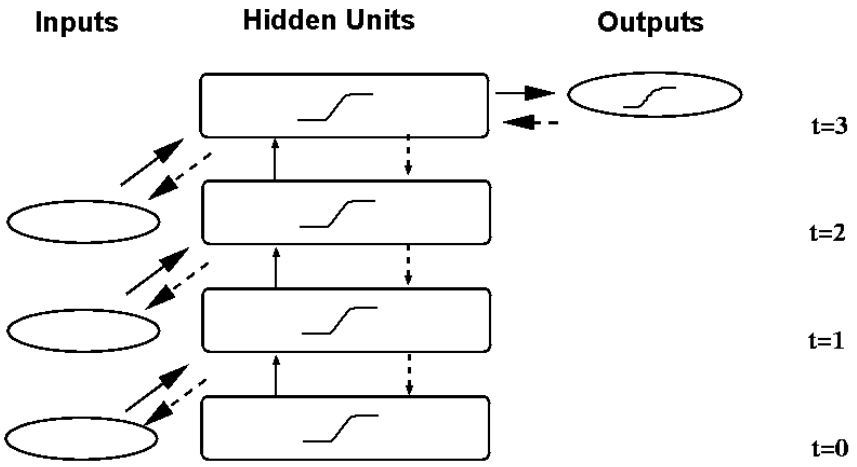


FIGURE 20. Hidden unit outputs are fed back into the input

The activation function of the RNN is the bipolar sigmoid function. The bipolar sigmoid was chosen over the standard sigmoid due to the bipolar nature of the sensory input data and effector outputs. By employing a bipolar sigmoid activation function, the RNN is able to effectively process the bipolar sensory data and generate bipolar output necessary for the correct operation of the embodied agents effectors. Let **w** be the weight matrix with n rows and n+m columns, where w_i is the weight to unit i, b is the bias, and z_i is the input into the unit i. The activation for each unit can now be calculated, by first computing the weighted sum of their inputs:

$$net_k(t) = b + \sum_i w_i z_i(t)$$

Units then compute the nonlinear bipolar sigmoid function of their inputs:

$$y_k(t+1) = \frac{2}{1+\exp(-\sigma \cdot net_k(t))} - 1$$

The bipolar sigmoid function is plotted in figure 21. The derivative of the bipolar sigmoid function is:

$$y_k'(t+1) = \frac{\sigma}{2}[1 + y_k(t+1)][1 - y_k(t+1)]$$

The hidden, context, and output layers of the RNN all use the same bipolar sigmoid activation function. The parameter σ determines the steepness of the bipolar sigmoid function and in this thesis was fixed at σ = 2. The external input at time *t* may still influence the output of any unit until time *t*+*n*, where *n* represents the number of context units. The output from the output layer is fed directly into the effectors of the embodied agent.

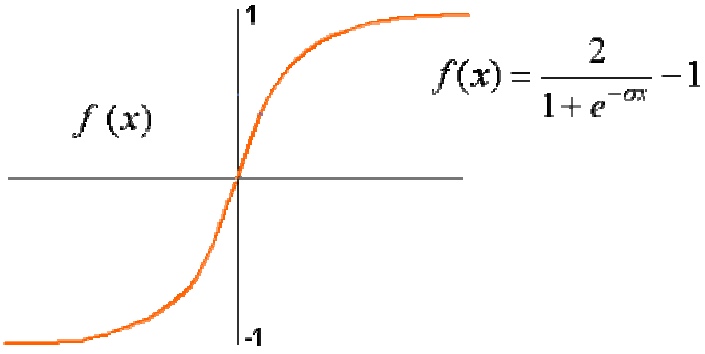


FIGURE 21. Plot of the bipolar sigmoid function

The bipolar sigmoid is closely related to the hyperbolic tangent function, which is often used as an activation function when the desired range of output values is between –1 and 1 (Faucett, 1994). The hyperbolic tangent is

$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

This thesis utilizes the bipolar sigmoid as the activation function utilized in both the hidden and output neuron layers of the RNN.

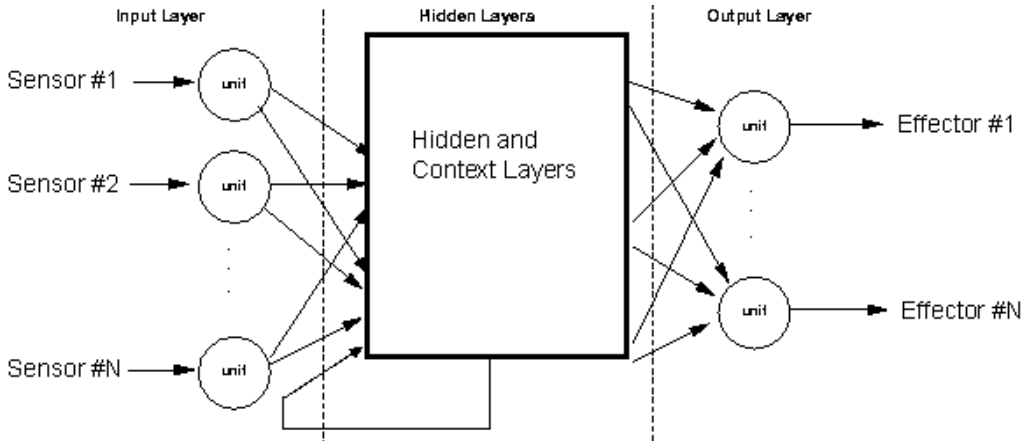


FIGURE 22. Embodied agent RNN control system

Sensory input is first fed into the RNN, as illustrated in figure 22. The RNN then calculates the activations based upon the sensory inputs and the context layers. The values produced at the output layer are fed directly into the embodied agents effectors. The agent's effectors in turn control its appendages, thus producing movement. For every time step in the simulation, this process is repeated.

GA Implementation Details

The purpose of the genetic algorithm is to optimize the weights of the neural network to evolve efficient locomotive behaviors for the embodied agent. A symbiotic relationship exists between the GA and the RNN. The GA optimizes the RNN, and the RNN produces agent behavior that is then scored and fed back into the GA. At startup, the population's chromosomes are initialized to random values. The chromosome length varies per embodied agent morphology, with 1 gene per RNN weight. The number of connections, in table 1, represents the number of genes in the chromosome; a floating-point number represents each gene. To help avoid the problem of premature convergence, linear fitness scaling is utilized in this thesis. The GA of this thesis makes

use of the standard single-point crossover operator as described by Goldberg (1986). After the crossover operation, the gene has a probability of being mutated. The mutation operator utilizes a Gaussian perturbation rather than a random mutation. The Box-Mueller transform is employed to generate a perturbed weight with a mean of the original weight and a standard deviation of 0.25. By perturbing the weights rather than randomly selecting values for the mutated weights provides for a gradual change. Table 4 presents the static parameters used for the GA.

TABLE 4. Genetic Algorithm Parameters

GA Parameters	
Population Size	80
Crossover Rate	80.00%
Mutation Rate	0.10%
Fitness Scaling	Linear
Mult. Factor (a)	9
Penalty Factor (b)	4

The GA uses a predefined function to evaluate the fitness of each individual member of the population. The purpose of the fitness function is to accurately evaluate the genetic health or fitness of a particular individual, and thereby either increase or decrease the probability of that particular individual reproducing and generating offspring. The fitness function determines how an individual is rated in terms of genetic fitness, and indirectly influences the behaviors of the embodied agent. The fitness function was carefully chosen such that it would tend to award efficient locomotive behaviors and penalize wasted effort and is based upon the distance traveled by the embodied agent within a certain period of time. A higher fitness score is awarded to embodied agents that are able to travel large distances in a given amount of time; traveling off course is penalized via a reduction in the fitness score.

Let x_0 and y_0 be the initial starting position of the embodied agent, α be the multiplication factor for the distance traveled and β be the penalty factor, then the agents score F is defined as:

$$F(x) = \alpha(-x - x_0) - (|y - y_0|)^{\beta}$$

The agent's fitness score *Fitness* is defined as:

$$Fitness = \begin{cases} F(x) & F(x)>0 \\ 0 & F(x)\leq 0 \end{cases}$$

The distance traveled along the x-axis is represented by x-x_0, while the distance traveled along the y-axis (representing an agent traveling off course) is determined by y-y_0. The fitness function positively reinforces agents whom are able to travel great distances along the x-axis while maintaining course. The multiplication factor $\alpha = 9$ and the penalty factor $\beta = 4$ are used in the fitness calculations in these experiments.

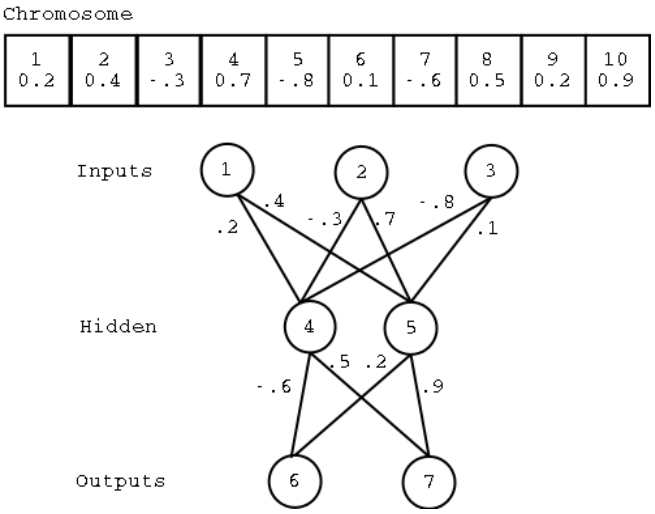


FIGURE 23. Chromosome format

The representation of the chromosome of an embodied agent is shown via a simple example in figure 23. In this example, a 7 node ANN contains ten inter-neuron connections. Every gene in the chromosome represents one inter-neuron connection

weight. Each inter-neuron connection tying two neurons together must have a connection weight associated with it. The weight structure is represented as a chromosome that is optimized by the GA to produce intelligent behaviors via selection. The selection mechanism used to determine which of the individuals will mate is based upon the roulette wheel algorithm modified with a linear fitness scaling mechanism as described by Goldberg (1986).

Embodied Agent Morphology and Details

The morphology of the embodied agent is completely predetermined and is designed to elicit a variety of different locomotive behaviors and test the generalization abilities of our framework. Four different morphologies are introduced into the simulation; each morphology represents an entirely different species of embodied agents with its own unique genotype and is composed of a number of joints, joint types, sensor inputs, effector outputs and rigid bodies as shown in table 3.

Sensors allow the intake of data from the environment and the effectors act as motors attached to the appendages and allow the embodied agent to physically interact with the environment. Each agent is composed of a hierarchy of 3-dimensional rigid body parts that are connected via joints. As described earlier, each joint has effectors (muscles) between them that exert a pulling or pushing force in any of the degrees of freedom (up to 3 DOF for a ball and socket joint) and can model the flexion and extension forces that pairs of biological muscles exert or the rotational torque of a geared motor.

The environmental data collected by the sensors is represented as a floating-point number, which is then fed into the RNN. The use of a bipolar sigmoid as the activation

function of the RNN allows the network to accept a range of input values that can be both negative and positive valued numbers. The bipolar sigmoid activation function also allows for the network to produce an output compatible with the embodied agent's effectors, which also require data to be presented in a bipolar format. The continual cycle of input and output to and from the RNN is the mechanism that can produce intelligent behaviors.

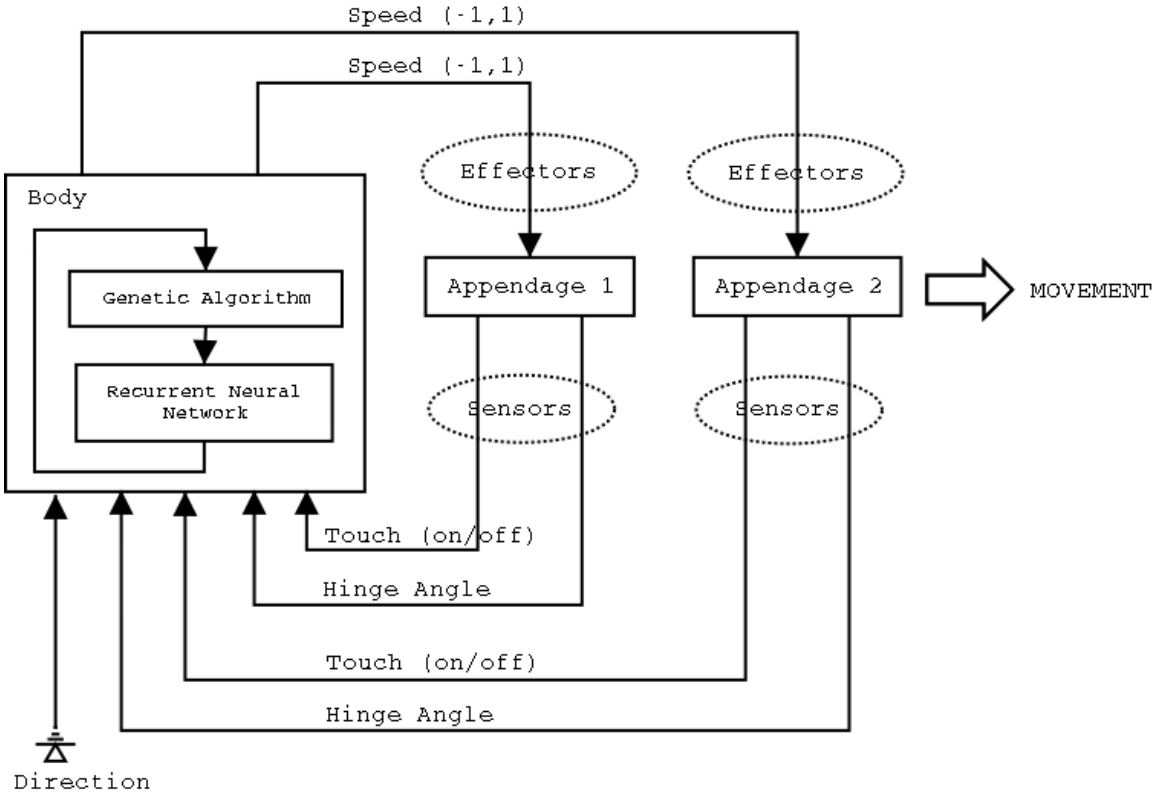


FIGURE 24. Embodied agent flowchart

The embodied agent flowchart as shown in figure 24, describes how an embodied agent can evolve intelligent locomotive behaviors. Though the flowchart describes the Crawler morphology, it remains applicable to all the morphologies, as the only difference from the standpoint of the system is the number of inputs and outputs. The flowchart can be broken down into 4 steps: (1) the agent receives environmental feedback from the

built in array of sensors attached to its appendages, (2) the RNN processes the incoming sensory data, (3) the RNN sends output signals to the effectors and (4) the effectors control the movement of the appendages and then repeat at step 1. The 4 steps are continuously repeated to generate movement. The structure of the RNN determines the types of movement behaviors demonstrated by the embodied agent. Upon completion of a generation, the RNN is updated and optimized by the GA and the process begins anew.

TABLE 5. Embodied Agent Morphologies

Agent	Joints	Bodies	Appendages	Joint Type	Sensors	Effectors
Crawler	2	3	2	Hinge	5	2
Long Arms	6	7	2	Hinge	9	6
Hopper	4	5	2	Hinge/Slider	13	4
Runner	2	3	2	Ball & Socket	10	6

The purpose of introducing multiple morphologies is twofold: (1) test the generalization capabilities of the NE process in its ability to evolve intelligent solutions of locomotion when presented with a variety of morphologies, and (2) prove the system is capable of modeling a variety of joints and structures that exist within the natural world. As stated earlier, the accurate modeling of natural world machines can conceivably allow for transplantation of the evolved neural structures from embodied agents of the simulated environment to the machines of the natural environment. The composition of the agent's morphologies is detailed in table 5.

Species #1 Simple Crawler Morphology

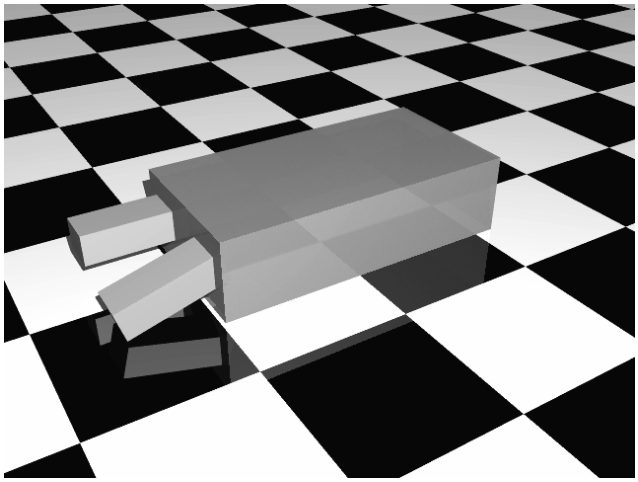


FIGURE 25. Embodied agent species #1 (simple crawler)

The first embodied agent design we experimented with is a simple box like creature consisting of 3 interconnected rigid bodies as shown in figure 25. Two appendages protrude from the front of the torso and are attached to the torso via hinge joints and allow for 1 degree of freedom about the agents y-axis. Feedback to the neural network is provided via a set of 5 inputs sensors that feedback the angular velocity of the 2 hinge joints, touch sensors on each appendage, and a direction sensor acting as a sort of compass. The RNN has 2 outputs that control the angular velocity of the hinge motors (muscles). The creature morphology was designed with the thought the creature would use the 2 front appendages to drag or pull itself forward.

Species #2 Long Arm Morphology

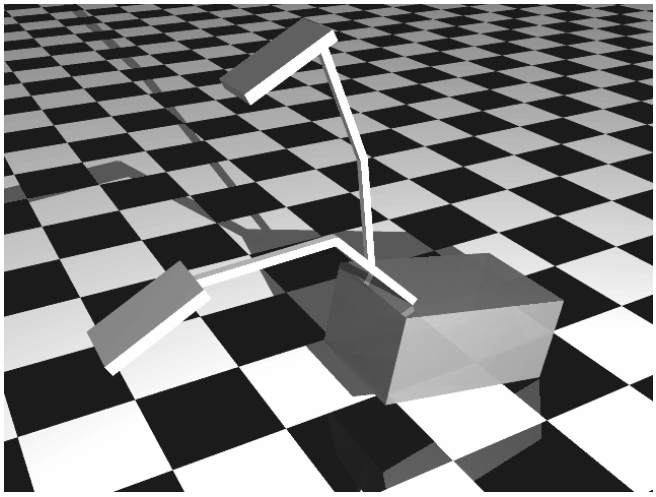


FIGURE 26. Embodied agent species #2 (long arms)

The morphology of this embodied agent resembles that of the Box creature; however, instead of 2 simple appendages protruding from the front of the torso, this creature has 2 complex "arm-like" appendages protruding from the top of its torso as shown in figure 26. A total of 7 rigid bodies make up the structure of this agent. Each complex "arm-like" appendage is composed of 3 rigid bodies interconnected with hinge joints and connected to the torso with a hinge joint. The "hands" are shaped like a paddle to provide greater contact area with the ground.

The RNN of this embodied agent processes nine input sensors feeding it a continuous stream of data collected from the environment. The information processed includes angular velocity of all 6 hinge joints, touch sensor feedback on each "hand," and a compass like direction sensor. The RNN has an associated 6 outputs that are used to control the muscles of each joint. The outputs determine the angular velocity that the muscles will attempt to maintain at any given moment in time. This morphology was introduced to determine if the RNN framework would be able to cope with a more

complex embodied agent and learn how to effectively manipulate complex appendages composed of multiple rigid bodies and joints.

Species #3 Hopper Morphology

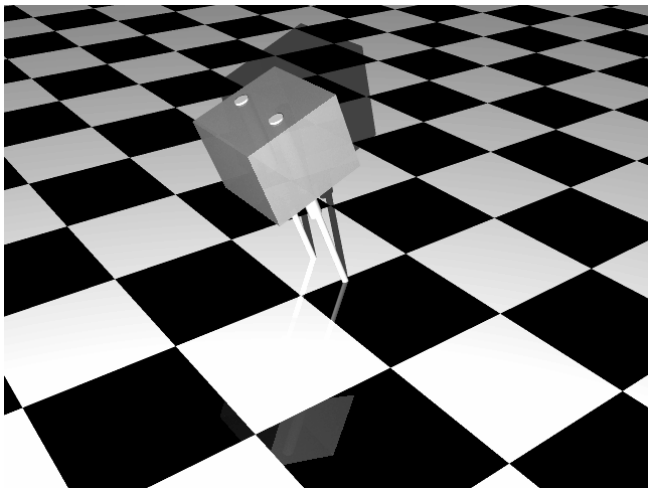


FIGURE 27. Embodied agent species #3 (hopper)

The third species of embodied agent introduces a morphology that is designed to elicit a method of locomotion that involves jumping and is composed of 5 rigid bodies interconnected via hinge and slider joints as shown in figure 27. This is a virtual creature that has a "box-like" torso with 2 legs that attach to the underside of the upper torso via hinge joints. Each "leg" is composed of 2 rigid bodies interconnected via a slider joint. The slider joint gives the legs the ability to quickly extend. The only conceivable means of locomotion this embodied agent can perform would have to involve some type of jumping behavior. This morphology was introduced to see how well the framework copes with a large fitness landscape with multiple peaks.

This embodied agent has 13 input sensors providing information to its RNN. These sensors provide the following environmental data: angular velocity of each hinge joint, extension velocity of each slider joint, linear velocity of the torso with respect to

the environment, angular position of each hinge joint, linear extension distance of each "leg", angular position of the torso with respect to the horizontal ground plane, contact sensors for each foot, and height of the torso above the ground plane. The 4 outputs from the RNN that are fed to the effectors include: velocity of the "leg" extensions, and angular velocity of the 2 hip joints.

Species #4 Runner Morphology

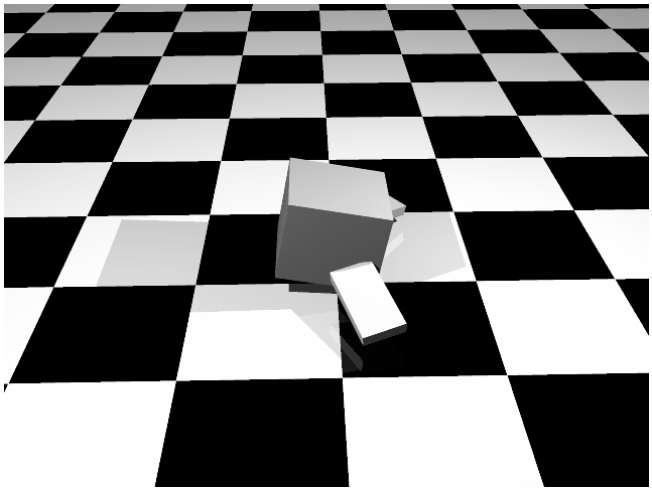


FIGURE 28. Embodied agent morphology #4 (runner)

The fourth embedded agent morphology introduces a new complex joint type with increased degrees of freedom and is composed of 3 rigid bodies interconnected with ball and socket joints. The virtual creature has 2 "arm-like" appendages protruding from either side of the torso as shown in figure 28. The 2 appendages are connected to the torso via ball and socket joints that provide 3 degrees of freedom for each "arm". Due to the 2 additional degrees of freedom vs. the hinge joint, this agent's RNN is required to process significantly more information than the previous creatures utilizing hinge joints with only 1 degree of freedom. The agent possesses 10 input sensors that feed the following data into the RNN: angular velocity for each degree of freedom for both

"arms" (6 inputs), contact sensors at the tip of each arm (2 inputs), contact sensor at the underside of the torso (1 input), and a direction sensor (1 input). The 6 outputs from the ANN provide velocity information, for each axis, to the effectors controlling the 2 "arm-like" appendages.

CHAPTER 4

EXPERIMENTS AND RESULTS

General Results From the Evolution of Locomotive Behaviors

The problem of evolving intelligent locomotive strategies for an embodied agent can be addressed using the techniques of evolutionary computation. In this particular problem with a Newtonian physics based environment constraining the movement and actions of the embodied agents, the NE paradigm performed admirably in its ability to evolve efficient locomotive strategies within the constraints of the system. The most important aspect of the NE techniques is its ability to learn without an external influence directing the process. NE exploits the ANN for its learning capacity and uses the GA to obviate the need for the traditional ANN learning strategies, such as BPTT, that require the use of training data and external direction.

The locomotive strategies evolved are unique for each species (morphology), and demonstrates behaviors that are generally smooth, fluid and quite like-like. Furthermore, the Hopper species developed a method of locomotion completely unforeseen in its initial design. The evolution of intelligent behaviors occur due to the fitness function of the GA optimizing the recurrent ANN of the embodied agent to evolve strategies that tend to receive high fitness scores. The fitness scores represent the total distance traveled in feet, minus a penalty for traveling off course, in the time period of 7.5 seconds. The fitness graphs are representative of the average of 5 simulation runs per morphology.

The evolved move sequences of the four agent morphologies are documented in appendix A.

Results are summarized in table 3, with the difficulty parameter representing the percentage of the simulation time required to evolve an intelligent and efficient locomotive behavior for a particular morphology. We choose to define an intelligent locomotive behavior as a method of locomotion making optimal (or near optimal) use of the morphology, sensors, and effectors to produce forward motion. Intelligent locomotive behavior involves the embodied agent being able to learn enough about its environment and how to manipulate its physical body to generate efficient movements. Certain methods of locomotion, such as jumping, require more intelligence and knowledge of the environment than other simpler methods of locomotion.

Results for the Embodied Agent Simple Crawler

In this experiment, a population of the species "Simple Crawler" begins learning how to move toward a target. Figure 29 shows the progress made during the initial 300 generations. During the first 20 generations, the agent quickly begins to learn how to interact with the environment to generate effector outputs that tend to produce ever-higher fitness scores. After 72 generations, the maximum fitness has reached 85% of the eventual peak fitness achieved within the 300-generation simulation. The simple crawler morphology achieves an eventual maximum fitness rating of 30.79.

Initially, the first generations of embodied agents perform entirely random movements; however, the population quickly learns the strategy of timing their arm movements in alternating sequences, thus resulting in a higher fitness score. Within 20 generations almost the entire population has adopted the strategy of alternating "arm"

movements to pull the rest of the body forward. The maximum fitness was recorded at generation 295 when a fitness score of 30.79 is achieved.

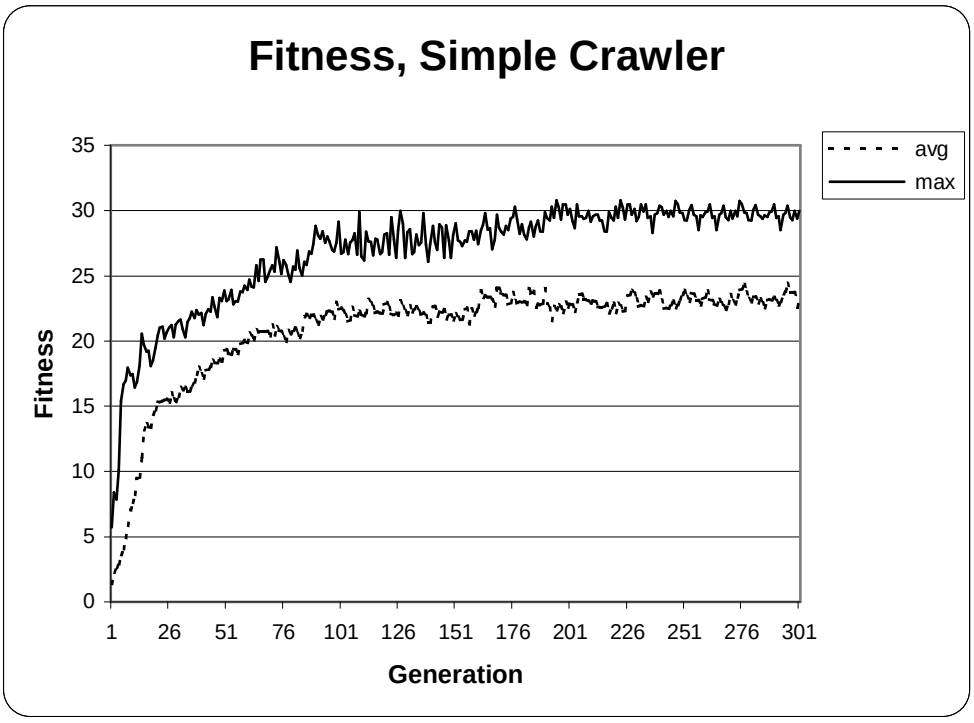


FIGURE 29. Fitness graph of species #1 (simple crawler)

This species, with its smaller brain and fewer context layers, quickly learned how to manipulate its effectors to produce movement that results in a high fitness score. Within 100 generations, the fitness peak has almost been found using NE techniques and little improvement is realized in subsequent generations. The results from the simulation show that NE is an effective method of training simple embodied agents to perform tasks that involve manipulating a simple control system. The simple controls, consisting of simplistic hinge joints (with their 1 degree of freedom) and effectors contribute to the quick convergence upon a solution for this morphology.

Results for the Embodied Agent Long Arms

The introduction of a more complex morphology with several additional inputs, outputs, and a larger ANN dramatically increases the fitness landscape of the NE process. The control system (ANN) of the embodied agent successfully learned how to manipulate the agent's "arms," each constructed of 3 rigid bodies connected via a total of 6 hinge joints, to produce intelligent locomotive behaviors. The NE algorithms are once again able to process and make use of the sensory data coming back from the joints and input sensors to produce effective locomotive solutions as evidenced in figure 30.

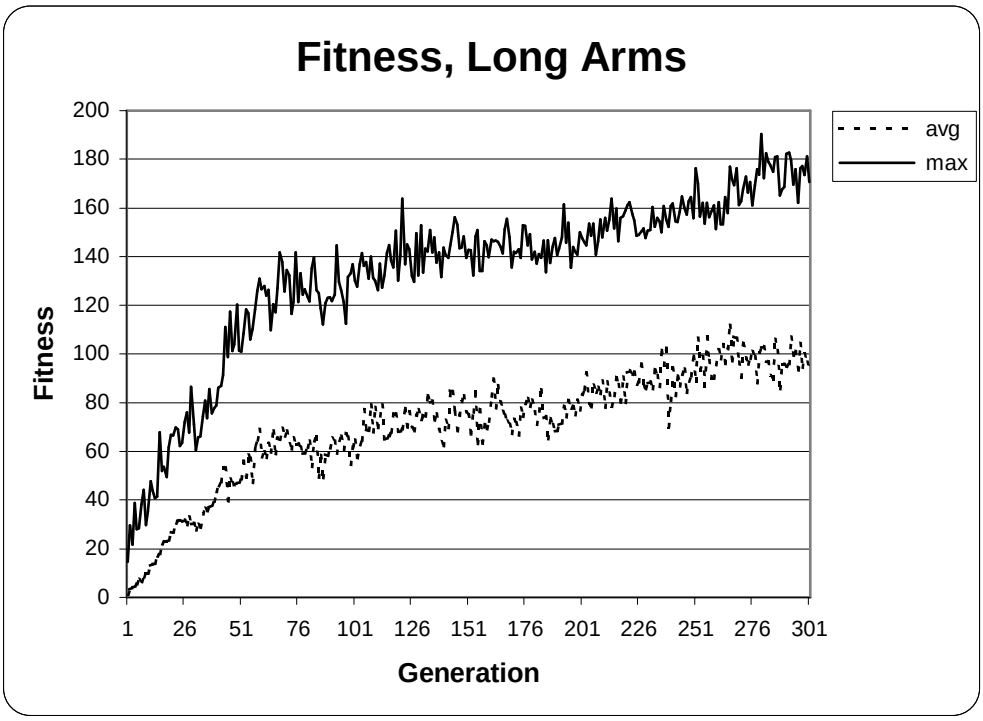


FIGURE 30. Fitness graph of species #2 (long arms)

The initial population again consisted of random effector outputs; however, the population quickly learned to time its "arm" movements to generate forward motion through a variety of techniques. For approximately the first 50 generations, several somewhat successful locomotive solutions arise. The method of locomotion that initially

evolved made use of the agent swaying its long and heavy "arms" to generate inertia to produce forward motion without the "arms" touching the ground. This locomotive behavior is quickly supplanted by individuals that began to use one "arm" to pull themselves forward by digging their "hand" into the ground and pulling their torso forward, and swaying their other "arm" to generate inertia to try and propel the body forward at the same time; however, a disadvantage of this method of locomotion was the little directional control it offered, which resulted in many of the agents traveling off course and being penalized. This semisuccessful locomotive strategy lasted for several generations until approximately the 50th generation when several individuals simultaneously learn how to alternate their "arm" movements to allow each "arm" to pull the torso forward and time their movements such that it affords directional control. This effective behavior lasts until generation 200, when some of the agents learn a more effective behavior of using both arms simultaneously to quickly pull the torso forward. By quickly repeating this movement, the agent could produce forward motion without the need for directional control. At generation 221, the population reaches 85% of its maximum fitness during the simulation. The individual with the maximum fitness score of 190.3 is born during the 279th generation.

Once again, the results from this simulation demonstrate the effectiveness of NE in producing intelligent and efficient locomotive behaviors for embodied agents with complex articulated structures employing several distinct joints. NE is able to develop a successful solution that effectively utilizes the more complex articulated "arm-like" structures to produce a higher fitness score than the simpler crawler species.

Furthermore, the results again demonstrate that NE is an effective means of evolving solutions to complex control problems in an environment that models the natural world.

Results for the Embodied Agent Hopper

The third species of embodied agent we examine has a morphology that was designed to only allow for locomotive strategies that involved jumping. Though this species may not be the most complex morphologically, this type of locomotion represents the most complex movement behavior of all the species. Successful jumping behavior is very complex; the agent not only needs to estimate the correct force vector to generate the desired results, it also needs to estimate the landing point and be able to manipulate its effectors in such a way as to have the feet in position for a landing. In other words, the successful agent needs to lean how to perform equations that can calculate the projectile motion of the body. Once again, using NE, we have successfully demonstrated it is possible to evolve complex locomotive behaviors. The embodied agent was able to successfully learn how to apply forces to its effectors in order to control its trajectory.

Initially, the population of agents all did very poorly due to their random effector outputs, which would result in the agent either quickly falling to the ground, or jumping in a random direction and failing to land on their feet. After about 15 generations, some of the individuals learned how to control the direction of their initial jumping motion, thus improving their fitness scores over the agents who jumped in a random direction; however, after their initial jump, they would still fail to land on their feet. This behavior continued until at generation 270, when several individuals learned how to time their initial jump so their bodies would do a full rotation in midair (summersault) and then land upon their feet to continue the jumping and summersault behavior. Several of the agents

were able to perform up to 3 of the summersault sequences before finally failing to land on their feet. The fitness score of 119.89 was the highest obtained within the first 300 generations of evolution. The population produced an individual that reached the 85th percentile at generation 276.

Due to the complexity of this form of locomotion, we decided to allow the simulation to run for a total of 1,500 generations. Very little progress was made from generation 300 to 1,350; however, as the population entered its 1,375th generation, certain individuals became good at learning how to control their summersaults and performing numerous summersaults in a row, timing them so they would perfect the landing with each summersault. Several individuals from this and subsequent generations can effectively and intelligently perform 10 or more summersaults before running out of time.

The results, as graphed in figure 31, demonstrate a type of evolution known as punctuated equilibrium. In punctuated equilibrium, the population undergoes long periods of stasis, with short and dramatic increases in fitness between these longer periods of stasis. The difficulty of evolving intelligent behavior is evidenced by the volatility in the maximum fitness score, as well as the punctuated equilibrium with fitness bursts occurring at approximately generation 275 and generation 1,400. The volatility demonstrated in the maximum fitness graph is likely due to the embodied agents testing minute variations in effector outputs that result in wide variations in fitness score due to the precision timing required to performing multiple summersault sequences.

The results from this simulation again demonstrate the robustness of solutions that are evolved using NE. The fitness graph appears to demonstrate a punctuated equilibrium that occurred within this population of embodied agents, with long periods of

stasis with abrupt and dramatic increases in fitness. NE was only semi successful in generating a good solution within 300 generations; however, good solutions start to appear by the 1,400th generation.

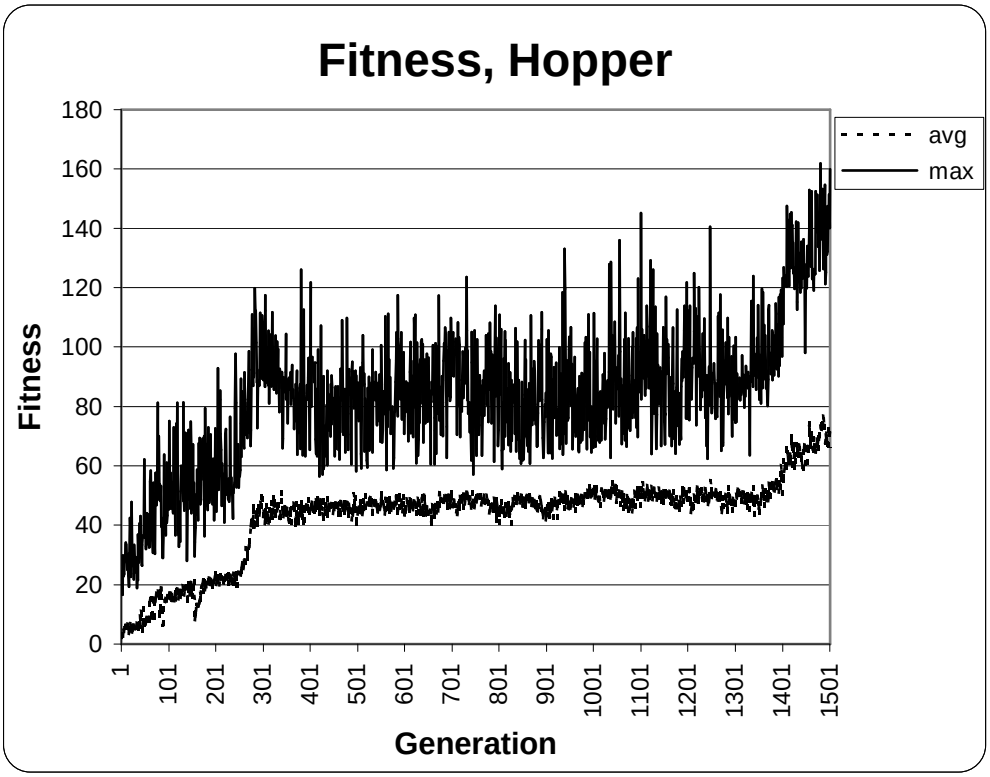


FIGURE 31. Fitness graph for species #3 (hopper)

The likely reason for the extended time period required to evolve a good solution is the sheer complexity of the task due in part to the large fitness landscape. As stated earlier, a successful embodied agent would require at least some knowledge of projectile motion and be able to determine where it will land after each successive jumping motion. Furthermore, the agent also requires the ability to precisely time the rotation of its body perfectly to allow the feet to impact upon landing, so that it can immediately perform another summersault sequence.

Results for the Embodied Agent Runner

The Runner species illustrates the most sophisticated genome in this experiment, with a chromosome consisting of approximately 976 genes. The embodied agent successfully learns to manipulate 2 "arms," extending from either side of its torso connected via ball and socket joints with 3 degrees of freedom, to produce intelligent and effective locomotive behaviors. This simulation demonstrates the effectiveness of neuroevolution at evolving intelligent behaviors in embodied agents employing complex joints with up to 3 degrees of freedom. A near linear improvement in performance is observed, as evidenced by figure 32.

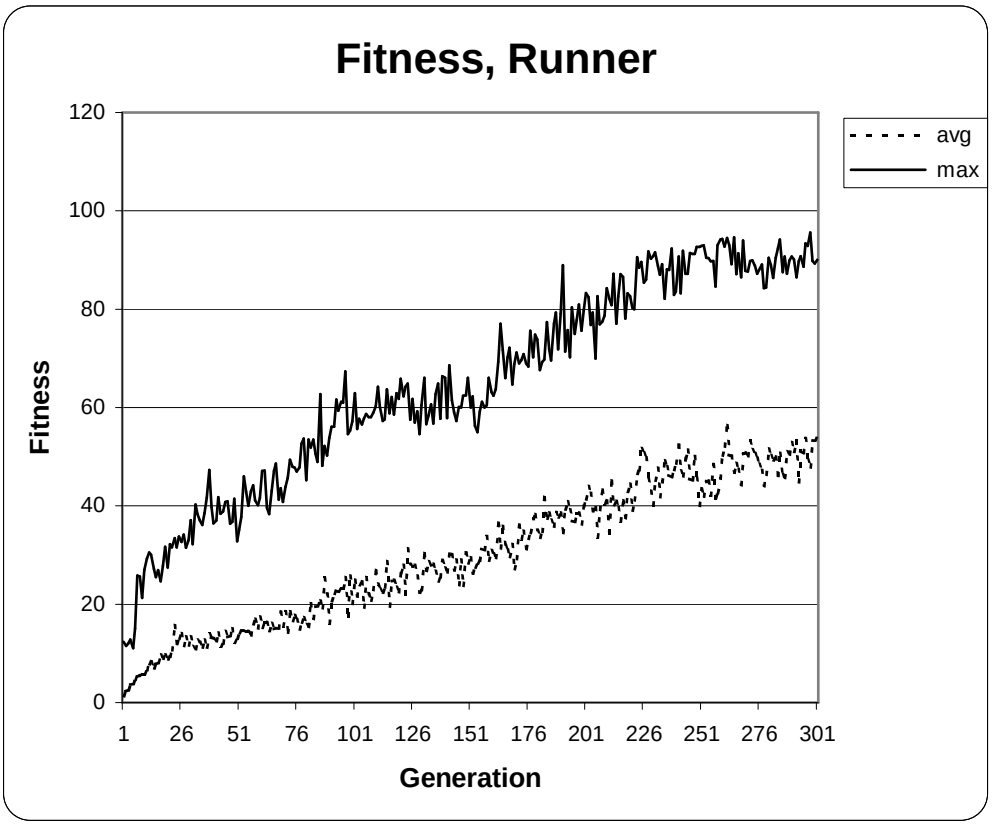


FIGURE 32. Fitness graph of species #4 (runner)

Within the first 10 generations, the population quickly improved its maximum fitness scores by learning to manipulate its joints and appendages by applying varying forces to the effectors. After the initial fitness gain, the population's fitness increased almost linearly as the number of generations passed. The agents quickly learned to use both arms by simultaneously swinging them forward, placing them on the ground and thrusting them back to produce forward motion. With each passing generation, the gradual improvement in locomotive abilities was almost unnoticeable; however, the distance traveled with each passing generation increased progressively. Sometime around generation 150, the arm movements of the embodied agent became more aggressive and the torso began to lift off the ground with each successive step the agent made. The peak fitness of 95.64 was obtained in generation 284, with the 85^{th} percentile occurring during generation 200.

The results for this simulation once again demonstrate the ability of neuroevolution to evolve robust and intelligent solutions to a given problem. We have demonstrated the ability of neuroevolution to generate solutions in a reasonable amount of time for complex morphologies involving appendages connected via ball and socket joints, very much how human arms or legs are attached. The results further demonstrate the ability of neuroevolution to deal with large and complex neural networks involving close to 1,000 inter-neuron connections.

Concluding Remarks

The goal of this thesis is to develop and demonstrate a framework for evolving intelligent behaviors in embodied agents. The virtual environment is modeled after the natural environment to provide for evolved solutions that can be transferred to physical

manifestations of the embodied agents. Results of the experiments show that the evolution of intelligent behaviors in a virtual environment constrained by the rules of physics is feasible. Agents display the ability to learn via a reinforcement style of learning. Agents also learn to coordinate and time their movements to generate effective locomotive behaviors, and in doing so, they are able to learn to adapt to their environment. Furthermore, they are able to evolve these effective methods of locomotion without any external agent directing the process or the need for training data. Although the simulations make use of four predetermined agent morphologies, the agents demonstrate robustness and the ability to make the best use of their morphology to evolve locomotive behaviors.

TABLE 4. Summary of Fitness Results

	Max	85th Percentile	Generation	Difficulty
Crawler	30.79	26.17	72	0.24
Long Arms	190.30	161.76	221	0.74
Hopper	119.89	101.90	276	0.92
Runner	95.64	81.29	200	0.67

The NE algorithm is not optimized for any particular morphology; however, certain morphologies lend themselves to simpler locomotive solutions as is demonstrated in table 3. The most surprising results were obtained with the Hopper morphology, which turned out to be the most difficult morphology to evolve an intelligent locomotive solution for. In comparison, the simplest morphology, the Crawler, only required only about 24% of the simulation time to evolve an intelligent behavior. The Runner, employing a more complex neural network than the Hopper, with over 976 genes, still requires less computational time to evolve an intelligent behavior than the less complex Hopper agent.

The Hopper, with its 660 genes required 92% of the simulation time to evolve a locomotive behavior that can only be called "adequate." In reality, the Hopper had not yet reached its full fitness potential within the 300 generations provided for in our initial simulations. Even after extending the simulation run out to 1,500 generations, the Hopper morphology appears to still be evolving. It is likely the difficulty the NE process is having with the Hopper morphology is due to the restrictive means of locomotion that the morphology dictates. The Hopper can only move by jumping (or hopping) from one location to the next; jumping requires planning, timing, and knowledge of the environment. Furthermore, the type of jumping performed by the hopper (performing 360 degree summersaults in midair), requires precision timing to perfect the landings to allow multiple jumps to be strung together. This type of intelligent behavior involves precision control over the effector outputs and an advanced knowledge of the inner workings of the environment that takes time to evolve. This is likely the reason the population of Hoppers is still evolving after 1,500 generations.

The results demonstrate that the complexity of the genome may not be as an important factor as the constraints imposed by the morphology of an embodied agent. Agents that evolve locomotive behaviors requiring intimate knowledge of the environment, such as being able to predict how much force is required to jump a distance based upon gravitational pull, require increased amounts of computational time to evolve an intelligent locomotive behavior. These agents use the additional time to evolve models of their environment that may include projectile motion, friction, and rotational and rigid body dynamics.

Furthermore, the agents evolve the ability to make use of these models by learning how to control their physical structures by the application of forces via their effectors.

Additionally, the results show that neuroevolution may possibly be a viable solution to evolving complex control solutions for physical machines. Future developments should take into consideration new advancements in neural network technologies and neuroevolution techniques while at the same time exploring the possibility of evolving ever more complex behaviors such as path following and competition amongst the agents.

CHAPTER 5

FUTURE DEVELOPMENTS

This chapter describes the future vision of this thesis. As explained earlier, the long-term goal of this study is to evolve intelligent behaviors that can be implanted into machines and robots that inhabit the natural world. The following is a list of recommendations for future enhancements to the proposed system.

Embodied Agents

Morphology dictates how successful the embodied agent will be at completing its task. If the goal is to embed intelligence in a machine that exists in the natural world, the embodied agent morphology should model that of the physical machine. With an accurately modeled machine, the embodied agent's ANN should be transferable to the machine existing in the natural world. In this section, we first examine how different morphologies allow for interaction with the environment and what type of processing is required to handle the vast quantity of incoming sensory data.

Morphologies

An embodied agent with a complex morphology generally has more opportunity to interact with the environment in ways previously unexplored. An example would be an agent with a complex articulated "arm-like" structure that modeled a human "arm" and "hand." The agent with such a structure could theoretically perform many of the same coordinated movements and tasks that a human could; however, the feedback from all the

sensors within the articulated "arm-like" structure would likely require an ANN with an extremely large number of neurons to process the input data. Such an ANN would require copious amount of time to compute the movement in the next time step. Furthermore, training such a monstrosity could be next to impossible due to the extreme size of the fitness landscape.

Currently, a compromise must be maintained. A morphology that is too complex will result in an unusable system due to the vast number of feedbacks into the ANN. Biological entities make use of their complex morphologies to develop and demonstrate intelligent behaviors. Learning about how neurons are organized and their interaction in biological entities can lead to a better understanding of behavior. The brain of a biological entity may only concern itself with the general control of its body rather than having to process each and every sensory input. An example of this is the movement in the body of a chicken after its head has been severed. The body will often continue to run about for several minutes after the brain and head have been completely removed from the body. Some form of control system must still be interacting with the body that does not rely upon the brain for control of certain motor movements. By modeling such a control system in the embodied agents, it may be possible to relieve the ANN from handling the multitudes of inputs present with a complex morphology and thus reduce the overall fitness landscape of the system.

Another area the future developer may consider is simple morphologies. Most robotic vehicles of the natural world utilize a wheeled platform. Simplifying the embodied agents by removing the complex articulated joint structures currently used for locomotion and replacing them with a wheeled platform should allow the ANN to devote

more of its resources to developing intelligent behaviors. A simple 4-wheeled platform would require a minimum of inputs and outputs to control locomotion and the rest of the inputs can be devoted to sensors that can provide the ANN with additional environmental data. The 4-wheeled platform would likely require at most 2 inputs: (1) current linear velocity and (2) current front wheel angles. One output should suffice for powering the rear wheels in either a forward or reverse direction. This type of morphology can be modified to model a variety of different wheeled vehicles.

Artificial Neural Network

Currently, the recurrent Artificial Neural Network utilizes evolvable weights; thus, the actual structure of the ANN is fixed. An evolvable ANN structure may provide some benefits such as quicker convergence upon a solution, or more efficient ANNs that complexify as necessary to provide the optimal solutions. Neuroevolutionary algorithms that begin with a minimal network and complexify the structure as needed may also obviate the current trial and error method of determining the correct number of hidden and context layers to start the simulation with. Kenneth Stanley's NEAT (2002) algorithm may be of use to the future developer in this regard.

Further consideration of different Neural Network models may provide a more biologically plausible model of the brain. Neural Networks are simulations of biological brains, with the main idea being that by simulating the various aspects of a biological brain, replication of some aspects of the brains capabilities (decision making, pattern recognition, etc.) should occur.

The current recurrent neural model employed uses the notion of clock timing (similar to that of a digital computer) that specify moments at which inter-neuron transmissions can occur and it further assumes the amplitude of the signal sent by a neuron is constant over the full period between clock timings.

The approach taken by traditional neural models is very different from how an actual human brain works. The human brain's neurons send out signals in brief "spikes, " lasting approximately one millisecond and reaching its peak amplitude for only a very brief moment. Spiking Neural Networks (SNNs) are neural network simulations that attempt to use the more accurate "spiking" model of neural output signals. The implementation of a SNN is more biologically plausible and may allow for further novel behaviors to emerge.

Environmental Issues

The environment in which the evolution occurs is extremely important to the final results. The virtual environment places the same constraints on the embodied agents as the natural environment would if the embodied agents were realized in physical form. It is likely any physical realization of the embodied agents would encounter "obstacles" in the natural environment; thus, in order to allow the embodied agents to learn to cope with obstacles, they should be modeled within the virtual environment.

A complex environment with obstacles that the embodied agents will require knowledge of how to avoid (or make use of) may provide for robust evolved behaviors. The future developer may consider embedding obstacles within the virtual environment to elicit new behaviorisms. A further consideration may be to allow the embodied agents to interact or compete with each other.

One example would be a maze like series of walls the embodied agents would have to navigate through.

Fitness Function

The fitness function directly determines how an agent will be rewarded based upon its performance within the virtual environment. By modifying the fitness functions, a variety of behaviors may evolve. One such modification a future developer may consider is to allow for a dynamic fitness function that evolves the embodied agents in a series of steps, the 1st of which may be to evolve locomotive behaviors. After the population of agents has evolved sufficient locomotive behaviors, the dynamic fitness function can be changed to evolve for a new behavior such as path following in the hopes the agents will not forget how to move. In a step-by-step fashion, it may be conceivable for the future developer to evolve complex behaviorisms in small developmental steps.

Parallel Implementations

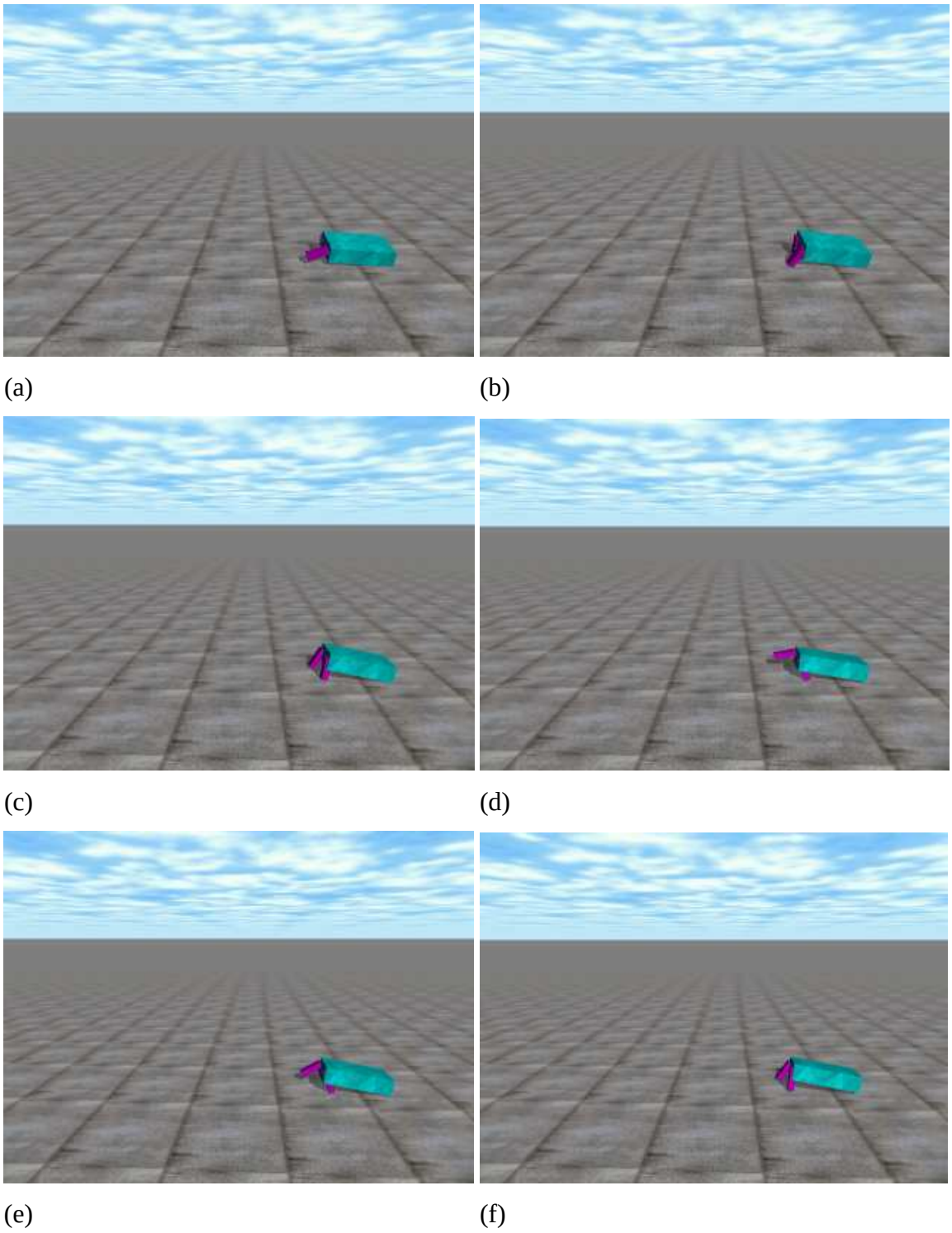
As described earlier, the evolutionary algorithms are inherently parallel processes; thus, the notion of a parallel implementation is the next logical step. Learning in a parallel-distributed environment reduces the computational load on one computer and enhances the overall performance. However, the communications necessary for the proper functioning of the evolutionary algorithms is more effective on one computer.

The majority of the processing time is spent performing physics computations on the eighty individual agents. Any parallel implementation of this system that hopes to afford a speed-up would have to consider the physics engine. The newer CPUs from AMD and Intel include SIMD instructions that the future developer may wish to optimize the physics engine for. In certain situations utilizing the SIMD unit rather than the

floating-point unit of the CPU will allow for 4 floating-point calculations per clock cycle and may result in an enormous speed advantage.

APPENDIX A

APPENDIX A

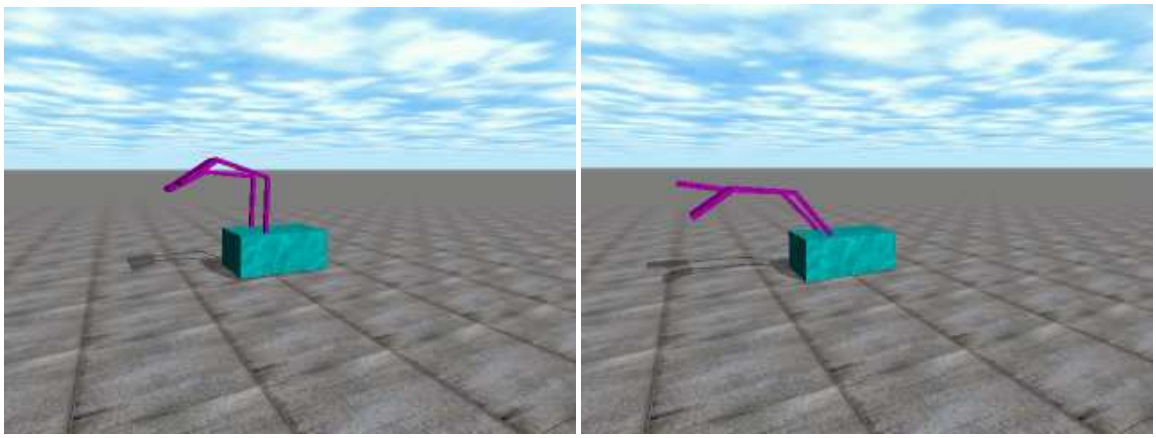


(a) (b)

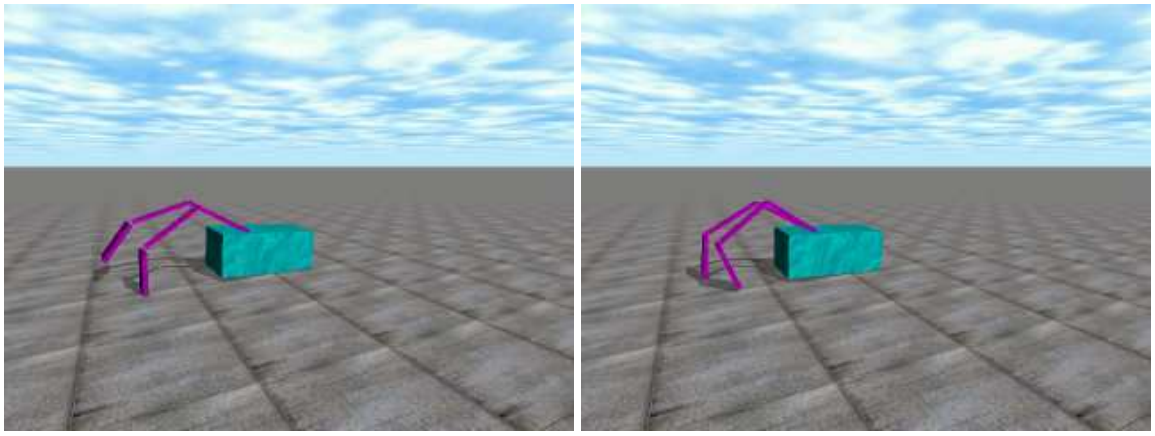
(c) (d)

(e) (f)

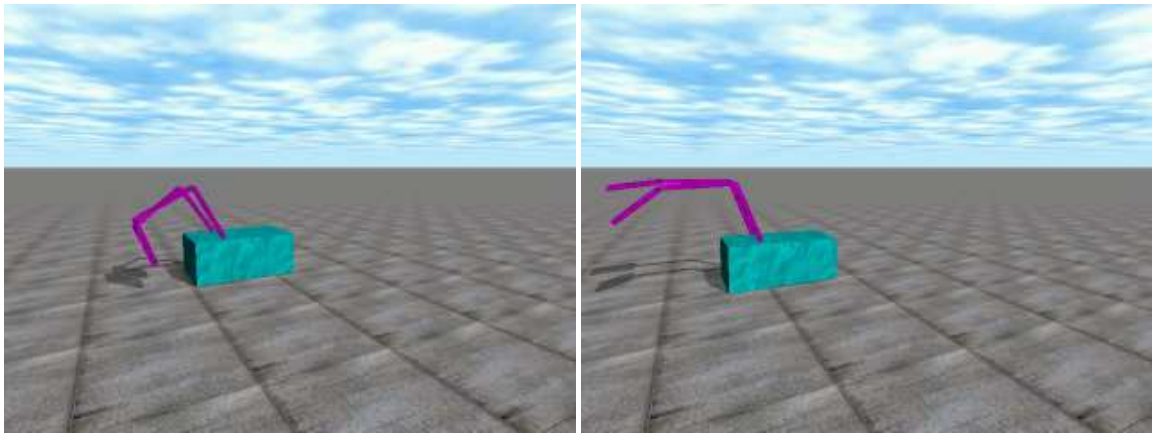
FIGURE 33. Set of evolved move sequences for the crawler morphology



(a) (b)

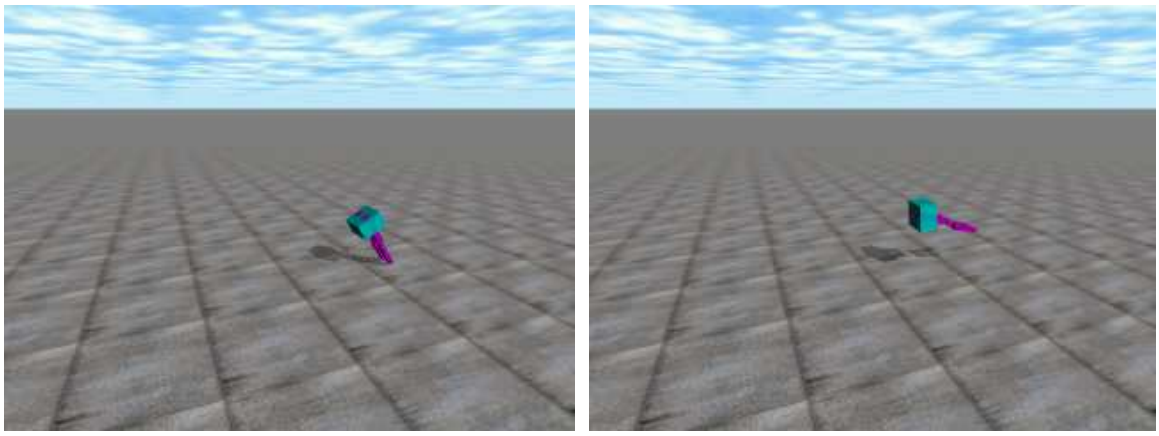


(c) (d)

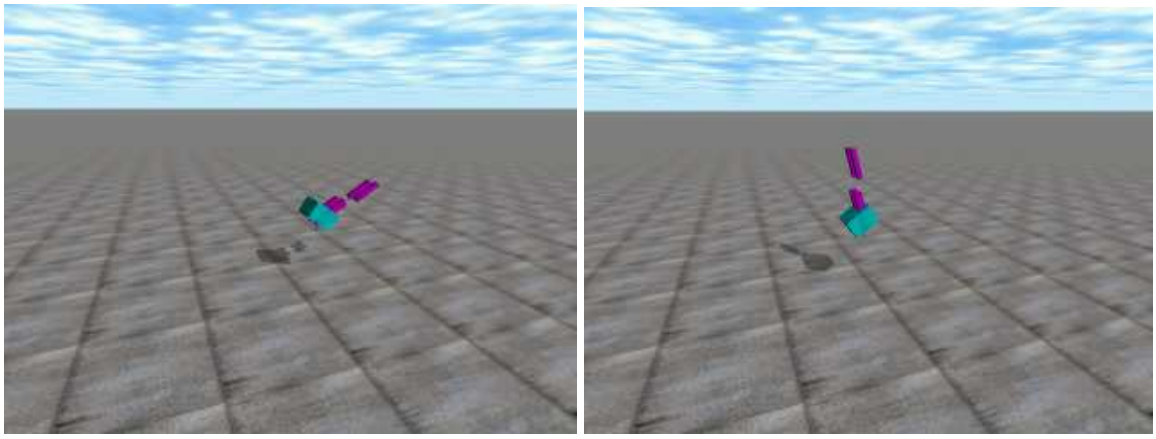


(e) (f)

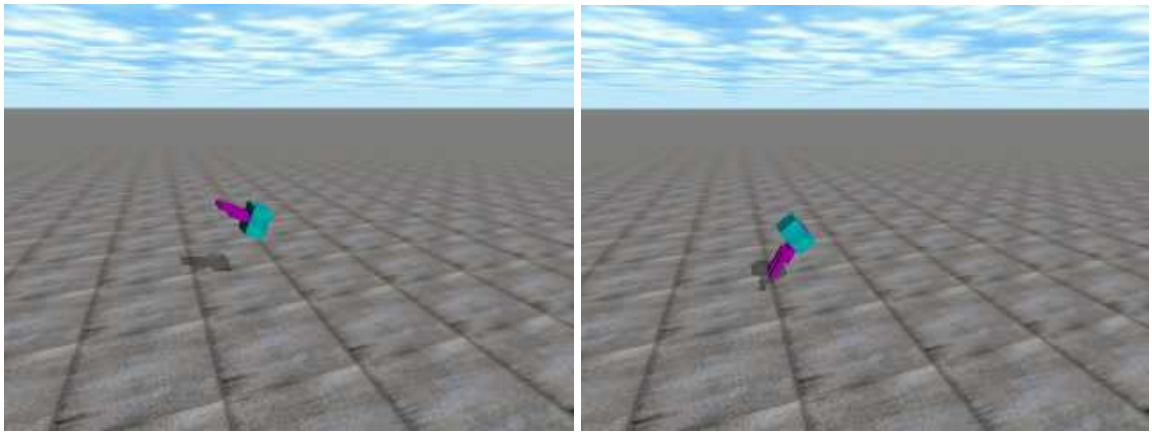
FIGURE 34. Set of evolved move sequences for the long arm morphology



(a) (b)

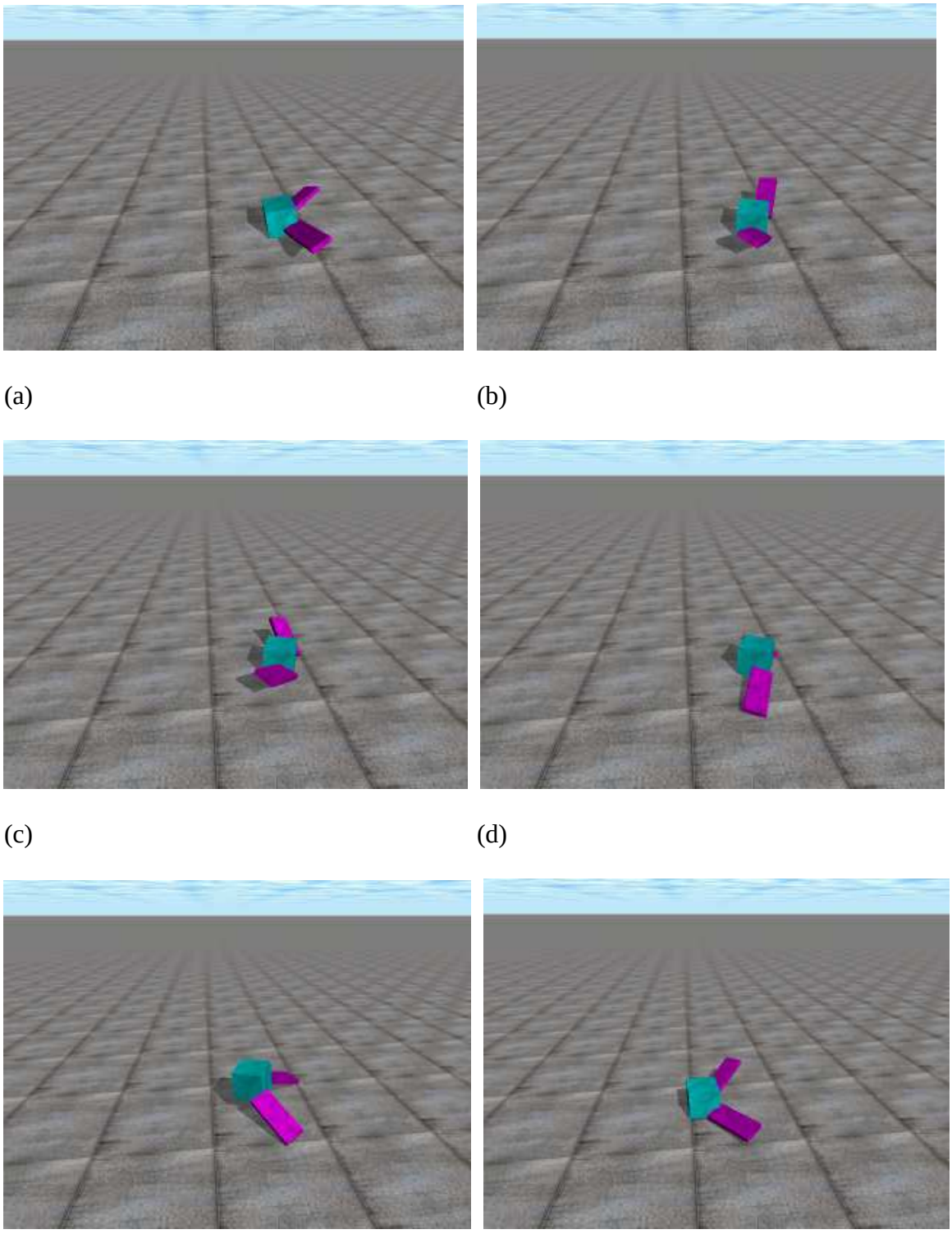


(c) (d)



(e) (f)

FIGURE 35. Set of evolved move sequences for hopper morphology



(a) (b)

(c) (d)

(e) (f)

FIGURE 36. Set of evolved move sequences for the runner morphology

REFERENCES

REFERENCES

[1] Spears, W., De Jong, K., Baeck, T., Fogel, D., and de Garis, H. "An Overview of Evolutionary Computation." European Conference on Machine Learning. (1993) 1.

[2] Wolfram, Steven. 2002. *A New Kind of Science.* Champaign, IL: Wolfram Media, Inc.

[3] Fogel, David. 2000. *Evolutionary Computation: Toward a new Philosophy of Machine Intelligence.* Piscataway, NJ: IEEE Press.

[4] Lorentz, E. N. "Deterministic nonperiodic flow." Journal of Atmospheric Science. 20. (1963) 130.

[5] Langton, C. "Studying Artificial Life With Cellular Automata." Physica 22D (1986) 120-149.

[6] Mitchell, M. 1999. *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press.

[7] Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning.* Reading, MA: Addison-Wesley.

[8] Fausett, Laurene 1994. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications.* Upper Saddle River, NJ: Prentice-Hall.

[9] Elman, Jeffrey L. "Finding Structure in Time" Cognitive Science 14 (1990) 179-211.

[10] Sims, Karl, "Evolving Virtual Creatures" Computer Graphics (Siggraph '94) Annual Conference Proceedings (1994) 43-50.

[11] Moriarty, David E. "Symbiotic Evolution of Neural Networks in Sequential Decision Tasks." PhD Dissertation; Technical Report AI97-257, Department of Computer Sciences, The University of Austin Texas. (1997)

[12] Stanley, Kenneth and Miikkulainen, Risto, "Efficient Evolution of Neural Network Topologies," Proceedings of the 2002 Congress on Evolutionary Computation (2002).

[13] Menczer, Filippo, "Life-like agents: Internalizing local cues for reinforcement learning and evolution," PhD Disseration; Department of Computer Science and Cognitive Science, The University of California, San Diego. (1998).

[14] Godfrey-Smith, Peter, 1996. *Complexity and the Function of Mind in Nature.* London: Cambridge University Press.

[15] Reynolds, Craig W. "Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics," SIGGRAPH '87 Conference Proceedings. 21(4) (1987) 25-34.

[16] Smith, Russell, 2001. *Open Dynamics Engine v0.03 User Guide.* Available from Russell Smith, Ph.D. site, http://www.q12.org/ode/ode-0.03-userguide.html