Representing Virtual Creatures as Neural Network

Benjamin Behar
behar006@umn.edu

May 11, 2016

Abstract

The design and animation of virtual creatures can prove to be a significant investment of resources, as it is both difficult and time consuming. However, recent work in artificial intelligence and neuroevolution have succeeded in generating virtual creatures that are sufficiently novel and effective at fulfilling the metric with which they are judged. This represents a significant advancement in the field of animation and video game design that has yet to take root in the current industry. Because of this, this paper seeks to further advance this field by providing a much more generalized method of representing virtual creatures in such systems by treating the virtual creature as just a neural network and defining the physical features of the creature as extensions upon this.

Introduction

The creation of characters and models represents a significant investment of effort in almost all computer graphics applications, primarily within the video game and film industry. Aside from this, the animation of such models naturally takes an even larger expenditure of resources, as the quality of the model must then be judged at each moment of time for the duration of the motions being tested. Of course, this may be considered reasonable when witnessing the level of scrutiny with which these applications are being evaluated by consumers. Those applications which do not rise above their competition quickly become overlooked as more attractive options present themselves. However, the level of scrutiny applied to one model may greatly differ from another when considering the context in which the model and its animations are viewed by the consumer. For example, an insect fluttering away in response to a humanoid walking through the underbrush will likely draw less viewer focus than the humanoid disturbing it. By recognizing situations like these, animators and artists commonly avoid investing unnecessary effort into the design of such individual models, and are able to spend the remainder of their resources into the improvement of other more critical models. Unfortunately, this approach usually results in these commonly overlooked models to remain unappealing and uninteresting to the viewer.

While this issue is constantly being addressed with advancements in the field of computer assisted design, a more interesting approach is to use artificial intelligence to enable the computer to perform the design on its own. Research in papers such as [5], [4], and [2] have explored the use of genetic algorithms to develop virtual creatures with morphologies that are capable of performing according to a fitness metric that a user may design specifically to describe the needs that this creature must fulfill. This process enables a designer to delegate the actual construction of a virtual creature to the computer, while retaining some control over the outcome. In addition, [5] and [4] also addressed the problem of animating such a creature by including a neural network which would serve as a controller for the creature, reacting to stimuli and driving the creatures motion. This controller evolves alongside the creature during the execution of the genetic algorithm, and this enables the computer to alleviate even more of the effort required to design a model.

Since genetic algorithms go about searching for a solution in a very different way than humans generally do, the results of such an algorithm producing virtual creatures are often surprising. In [5], it was noted that very strange creature morphologies were generated which upon first glance would appear ineffective, but during the simulations these creatures demonstrated unnatural efficiency, resulting in a very high fitness score. Examples include creatures that rocked back and forth to scoot forward at alarming speeds, those which developed a strategy of curving back and forth to roll in the required direction, and some that evolved a sidewinding behavior. Since these virtual creatures are so unusual, they provide a sense of novelty in whatever simulation they inhabit. This novelty is extremely valuable to the consumers of products such as video games, who often crave new experiences and quality entertainment. This method of generating virtual creatures may provide a substantial interest in the video game industry, as it provides a method of allocating a large amount of effort onto the computer and its ability to perform far faster than any human alone, but also introduces a certain amount of novelty into their productions that might not have otherwise been imagined.

Related Work

While the history of genetic algorithms can be traced back very far, even to some of the works of Alan Turing in [8], the concept of applying genetic algorithms to the generation of virtual creatures is relatively new [5]. Published in 1994, the work of Karl Sims in [5] illustrated both the possibility and effectiveness of generating virtual creatures and their controllers via genetic algorithms. Over the course of the next two decades, a great deal of advancement was made in both this method and other areas in which this method could be applied, such as [1]. Research has begun to follow a particular trend of dividing the evolution of virtual creatures into four distinct parts: morphology, controller, environment, and algorithm. These four sections represent areas of the system that exist relatively independently of one another, and as such, most of the work that has been done since [5] has focused on only one or two of these in any single paper, such as [3], [2], and [4]. By contrast this paper will provide focus on all of these four sections simultaneously, by designing each section on each of the focal points of theses papers, respectively. However, as all of these sections were first defined in [5], the significance of Karl Sims work must be made clear to properly understand the field of study that has followed it.

The work in [5] came about as an attempt to alleviate some of the effort required to make virtual creatures for applications in computer graphics. This is done by reducing a virtual creature down to two parts, the morphology and the neural network that controls it. These two parts are represented as a directed graph with nodes representing different possible features. For example, a trivial creature might have one node with a self reference, thereby representing a snake like morphology, while at the same time having a complicated neural network, such as a sine function driving an undulating motion until an sensor is triggered. With this encoding for a virtual creature, it is possible to use a genetic algorithm to generate multiple creatures with little to no input from a user, aside from the provision of a fitness metric. In the case of [5], this fitness metric was the effectiveness of the decoded virtual creature placed into a physical simulation with its respective neural network guiding its actions. The fitness function was further customized to judge functional behavior such as locomotion in a viscous fluid or the ability to move other objects. The method presented in [5] was successful and capable of producing new creatures that exhibited novel strategies for succeeding at their respective tasks. Some of the results were incredibly novel, such as creatures which walked on their torso by flailing limbs back and forth in order to scoot the base of the creature forward. The resulting strange-looking creatures ignited much interest in this area of study, and led the way in this particular focus of artificial intelligence and computer graphics.

Quite a few works that are largely based on [5], come to the consensus that the system used does not allow for sufficient complexity in regard to the morphology or neural network of the creatures produced [4] [2]. [2] attempts to improve upon the morphological representation in this type of system by using parameterized L-systems. L-systems were originally developed as a tool to describe biological organisms by taking advantage of the prevalence of fractal-like morphologies that tend to develop in nature. Since L-systems are capable of describing a great deal of structure with very little data, larger and more complex creatures can be generated with this system. In addition, an L-system that is parameterized allows for an even greater reduction in size of the encoding for structure, by allowing traits like size and speed to be specified numerically, such making a limb ten units in size with forward(10) or adding a rotation joint with a speed of three units with revolute-1(3). However, to make this change, [2] had to omit the complexity of the neural network directed graph by both removing the ability to react to stimuli and reducing the motor functions of the creature to a position as a function of time. Since L-systems describe structure as a set of rods, the generated creatures remained very lightweight in terms of computation intensity, allowing for a much larger amount of segments to be simulated at once. With this newly defined system, [2] was successful at generating creatures with morphologies in much larger complexity than what was possible in [5]. However, the cost of doing so was in the reduction of behavioral complexity in creatures and in the unnatural appearance of the creatures produced [2]. However, [4] attempted to improve morphology representation in a way that would emulate nature more closely.

[4] describes a much simpler improvement to the work of Sims: instead of changing the method of encoding the morphology or neural network of a creature, both could be incorporated into the same directed graph. This approach more closely resembles that of nature, in which DNA is the only form of encoding needed to define the entirety of an organism. [4] was successful in recreating the experimentations of [5] with this system, which is called Evolving Robotic Organisms (ERO) [4]. This ERO system was used in [4] to reduce the complexity involved during the encoding and decoding of virtual creatures during the execution of [4]s dramatically altered genetic algorithm.

Despite the improvements to the various encodings of virtual creatures, morphology and neural network, [4] sought to improve upon the genetic algorithm that was being used by most of the papers that had come before it. [4] expresses that there is a fundamental difference between natural evolution and genetic algorithms which restrains the effectiveness of its use in the generation of virtual creatures. This difference stems from the fact that not all organisms directly compete with each other, a bear and a bacteria both have very different environmental niches [4]. However, it is the nature of a genetic algorithm search to converge to a single solution instead or retaining different solutions [7]. Normally to acquire multiple novel creatures one would have to run the genetic algorithm multiple times so as allow the population to start over and possibly advance toward another optimum. [4] provides a suitable answer to this by providing a fitness metric that evaluates both the novelty and efficiency of each creature when compared to the other creatures that have developed in the simulation. The system provided was extremely successful in the generation of generation of creatures that maintained a sufficient level of diversity, and this additional complexity that was incorporated into the fitness metric reduced the speed of the system by only a marginal amount.

All of the discussed papers approach the problem of creating virtual creatures automatically through use of evolutionary algorithms (genetic algorithms, etc.) with the intent of simulating evolution found in nature. However, works such as [1] have begun to apply these virtual creature generation techniques to the field of robotics. While genetic algorithms have commonly been employed to provide artificial intelligence in robotics such as in [6], the mechanical design of robots has largely been reserved for human expertise until recently [1]. Since machines need to be both designed and controlled, and are created to satisfy some sort of performance metric, the genetic algorithm approach used to generate creatures seems to be a viable place of future research. However, since the problems in robotics are much more complex and require such delicate precision, such applications of autonomous design of robotic morphology remains largely academic, and will likely remain so for a significant amount of time.

Overall, the capabilities of computer generation of virtual creatures are impressive, but they remain fairly primitive in regards to complexity as they require substantial resources for even the simplest designs to be produced. Since the genetic algorithms involved must perform a physical simulation of every creature for each generation, the computational demand remains too large for any sufficiently practical application to be established. As such, the future of this field must largely be dominated by the search for novelty over functionality. Since video games correlate very strongly with this constraint, most of the development of this technology will likely remain focused on the production of virtual creatures and objects, and will find its strongest support from the video game community.

Approach

To fully implement a system capable of generating virtual creatures, each of the following tasks must be accomplished:

1. Defining the way in which virtual creatures are represented and encoded
2. Implementing a genetic algorithm that can be applied to creature encodings
3. Simulating creatures inside of a physical simulation to evaluate their fitness

Each of these individual pieces represent a significant expenditure of effort, and all of them together greatly exceed the scope of a final project. Because of this, this work will be limited to implementing a novel approach to the first task and discussing its implications on the other two tasks.

The task of defining the way virtual creatures are represented and encoded is the most fundamental of the three major tasks and is perhaps the most important. In the related work section above, various types of data structures used to represent virtual creatures are described, however, most of them involve the representation of a virtual creature as some extension of a simple directed graph [5] [2] [3] [4]. This is due to the fact that most species of animals function according to the neural connections that exist throughout their body, whether that be a centralized network (creatures with a brain) or a decentralized network (creatures with only various nerve bundles). In nature most creatures fall under one of these two categories, however, by defining both the morphology and control of a creature in the same graph, as shown in [4], a hybrid model can exist. This results in a directed graph that acts essentially the same way as a neural network, albeit with the unique modalities to interact with an environment. Occams razor entails that representing both the controller and morphology as one directed graph in the general form of a neural network is the best option, and this approach is what guided our creature representation.

To properly illustrate the manner in which virtual creatures were represented, the data structures that are used must be properly understood. Since Java is a heavily object oriented language, the representations of creatures were simplified into interacting object models that partitioned the code into succinct parts. The classes implemented are as follows: Creature, Node, and Connection. The Creature class is quite simply a collection of all of the Nodes that belong to and only to the virtual creature represented. The concept of the Node is similar to that of a vertex in a directed graph, in that it possesses a unique identifier which allows associations between Nodes called Connections. However, for the sake of efficiency, Nodes also contain a collection of all of the Connections which direct away from them. In a close parallel with a standard directed graph, the Connection is very similar to that of a directed edge. Connections describe only the node which they direct toward and the weight of this connection. The origin of any Connection is assumed to be the Node which contains it within its collection. Altogether, when properly constructed a Creature object is capable of defining every aspect of a virtual creature, including the current state of the neural network.

With the data structure of a virtual creature defined, execution of the neural network can be performed with a standard set of rules and assumptions. By using the collection of Nodes provided by a Creature object, a single run of the neural network can be accomplished by iterating over this collection. In this run each Node is evaluated, determining if its current value is above a particular threshold. If so, all Connections from this Node are followed and each resulting Node is signaled by incrementing its value by the value found for the first Node multiplied by the weight found in each respective connection. In this way, the neural network can be executed in individuals steps. This is important because in this neural network both cycles and self-references are supported, meaning that it is not guaranteed that any neural network will ever be completely terminated.

Since the method in which a virtual creatures morphology and control structure is implemented remains extremely generic, various input and output modalities can be built on top of the existing data structure with very little additional complexity. This can be done by a standard class extension in Java by overriding the signal method of the Node class. In order to create an input modality Node, one simply needs to alter the value field of the Node class to correspond to the input data. Similarly, to create an output modality Node, one may override the signal method to include additional code that performs some action which can be modified by the value of the current Node or based on the signal value being sent to the Node. However, extensions such as these add additional complexity during the encoding and decoding process, and for the sake of clarity and focus these will be considered optional extensions of this work.

With both the method of representation and execution well defined, the final piece of this puzzle is to define the method of encoding the data structure of a creature into a manner more acceptable to a genetic algorithm. Since almost all genetic algorithms focus specifically on altering data that can be represented as a binary string, it was decided that the method of encoding would be one of a direct representation. Starting with a byte buffer each Node contained in a creature is encoded into a subsequence of bytes in the following format: unique id, number of connections, 1st connection id, 1st connection weight, , Nth connection id, Nth connection weight. Each of these encodings are fed into the buffer, and a byte array is produced which contains all the necessary information to reconstruct the virtual creature.

Despite using a direct encoding this system expresses traits that are commonly found only in indirect encodings. This is due to the fact that Connections only identify nodes based on a unique identifier in contrast to direct reference. When signaling another Node, the first Node with an identifier matching the connection is selected. While this approach is slower as the number of nodes increases, it provides a method by which certain traits can linger in the genotype while not being expressed in the neural network. Normally this sort of non-expression is limited to encodings that are indirect.

Experimentation

Since this project focused almost exclusively on the representations and execution of a virtual creatures neural network, the design of experimentation remained relatively simple when compared to that of an entire neuroevolution system, such as in [5] and [4]. The only hypothesis that is introduced by this paper is that the creature representations that were constructed are able to function correctly given the extraordinary search space that genetic algorithms provide. To properly validate that the system of representation and execution can be interfaced with a genetic algorithm, two very important requirements must be met. First, the search space of the genetic algorithm may not cause any issue with the decoding of a byte sequence into a Creature object. Second, the execution of the neural network must be guaranteed to not produce any unhandled exceptions despite the genetic algorithms possible introduction of meaningless data.

In order to validate that these two requirements have been met, it was necessary to emulate the effects of a genetic algorithm, particularly a bad one. To accomplish this, it was decided to use a pseudorandom number generator to populate a byte array that can be decoded into a Creature. However, the creation of such a method proved to be much more difficult than originally anticipated, and required multiple attempts to produce an adequate emulation.

The first attempt at creating this genetic algorithm emulator was in fact successful at the task of emulating the search space of a genetic algorithm. By simply choosing a random size for the ByteBuffer and then populating it one byte at a time with a pseudorandom byte, a very simple emulator was created. The problem with this emulator was with the distribution of randomness. Since the bytes that determined the amount of buffer which belonged to the each node were also random, the decoding process would fail more often than it would succeed as it would almost always run out of buffer to work with. This repeated failing entailed that the amount of time necessary to produce one successful Creature would greatly outway the utility of testing the system at all. Because of this a second attempt was made. In this next attempt, the number of Nodes to be added was generated before the length of the buffer was defined. Then for each of these nodes, the number of connections that each node was to have was generated. In this way, it was possible to avoid adding any randomness to the bytes that could cause fatal errors. While this approach worked for continuing the testing process, it merely covered up a severe limitation in this project.

Using this genetic algorithm emulator, a virtual creature could be encoded fairly quickly enabling effective testing of the neural network execution. The network was executed in a similar manner to that described in the previous section, but with some small modifications. First, the threshold for all Nodes is set to 1, meaning that if any Nodes value exceeds 1 then its connected Nodes will be signaled. Second, before any execution occurs the first Node is signal manually with a value of 10, as to ensure that at least one Node will fire a signal during execution. Third, each time that a Node is signalled a message will be printed to standard out so that the activations of each Node can be seen. This message includes both the sender of the signal and the recipient. Lastly, the neural network simulation is only executed for 100 steps at which point the creature is decoded and checked for consistency with the previous byte array that was populated before execution.

This method of testing was performed by hand, as the output from each neural network proved to be sufficiently daunting in length. Fortunately, since the mathematical accuracy of the neural network execution is a low priority, the floating point values that were indicated did not have to be determined in extraordinary detail, instead checked for reasonability. However, merely checking each value proved to be substantially tedious as well. Despite this, some very interesting observations were made. First, it appeared that most neural networks had a significant amount of complexity as the first poke performed started a fairly significant reaction in the neural network. Second, that some networks despite having a great deal of connections did not have any significant relation with the first Node. Third, networks that demonstrated high activity had a dichotomy between those whose activity resulted in fairly low values with respective to absolute value, and those whose values quickly reached to positive and negative infinity. Lastly, very few neural networks featured self referential connections which were capable of continual firing, i.e. a Node which could send itself a strong enough signal to ensure its value stayed above 1 indefinitely.

Analysis

During the implementation of a genetic algorithm emulator a large number of issues were made apparent with respect to the encoding process that was presented. The most significant of these is the amount of search space that resulted in a encoding failure. This sort of issue would make such a system of representing virtual creatures largely undesirable in and of itself. While, of course, the system is capable of interfacing with a genetic algorithm if each failed creature signals a request for another creature to take its place, the amount of performance lost in these failure handling cases would prove to be extremely detrimental. Fortunately, there are two ways in which this type of failure could be minimized. The first and most effective would be to remove the byte sequence that describes how many more bytes belong to the currently described Node. This could be replaced with a specific byte that serves to indicate the end of a particular Nodes byte sequence in a manner similar to the concept of a null terminated string. In this way, more byte sequences could be considered valid without being completely functional. The second method of reducing this failure handling is to silently ignore any Nodes that were not properly encoded. This approach would make fatal defects in a creature non fatal by removing the offending part. However, this approach is quite excessive as mutation would render most creatures devoid of large portions of their morphology.

While the issues involved with the encoding process are significantly debilitating to this project, the observations made on the neural network execution are rather promising. The first two observations made relate specifically to the choice to signalling the first Node of the Creature before execution. Since such initial signaling is not essential to the execution of the neural network, these observations do not suggest anything that would indicate that the system is performing inadequately. In fact, these observations merely show that the genetic algorithm emulation provides an adequate distribution of creature possibilities, since the probability that a neural network may not interact very much with a small set of Nodes is quite common.

The third observation raises some concern as to the effectiveness of these neural networks, but it also demonstrates the resilience of this the algorithm which executes the neural network. Despite Nodes having non-number values, they still function as intended due to the defensive nature of the conditionals placed in the signal method. In Java, positive and negative infinity both act as expected in a mathematical sense both when used in comparisons and when used during multiplication and addition, meaning that the neural network will continue to function without significant issue. Additionally, in the case of NaN, Java will always return false during a comparison, and return NaN during mathematical computation. This entails that having a NaN for a value effectively turns off the node that it enters, which may provided additional functionality to the neural network and provides evidence that the neural network cannot fail in an unsafe way.

The last observation serves mostly as a point of intrigue. It was originally expected that quite a lot of neural networks would have Nodes which connected to themselves, and there was a concern that such Nodes might introduce infinite values into the simulation extremely quickly. However, as infinite values are handled effectively during the execution of the neural network, there is no concern about such self referential Nodes. However, it is quite likely that when creatures are being developed in a physical simulation with a genetic algorithm tuning them for efficiency that such nodes will be used to perform repetitive tasks rather simply.

Conclusion

The system described by this paper has been been shown to be effective at providing a method with which to describe virtual creatures, both in data structure and in encoding methods. By maintaining focus on this creature representation system, instead of the genetic algorithm or the physical simulation, a much more robust system has been developed and explored than could have been otherwise. However, this system does have significant room for improvement, specifically in the encoding and decoding processes, and as such, it is not ready to be implemented with a complete virtual creature evolution system such as [5] or [4]. While the system is not fully developed, it does provide a novel approach to the field of neuroevolution since it explores the concept of a genotype and phenotype that have a direct encoding but also allow for a sort of recessive expression when the same unique identifier is used. However, the effectiveness of these approaches has still yet to be determined.

Some future work ideas specific to this project have already been described in the experimental section as possible reasons for error, such as implementing an encoding format that uses a special byte or byte sequence that entails the termination of a specific Node subsequence. However, there are slight improvements that could be made to the system that are not critical to its success, such as altering the neural network execution to avoid order dependency issues. Since the algorithm handles each Node in a Creature in a set order those at the front of the collection are able to affect the graph independently of those after them, but those that come afterwards may have been altered by this previous Nodes execution. This has a small chance to introduce order dependency and could be improved by better simulating the simultaneous evaluation of all nodes.

The most obvious extension of this work is to fully implement a neuroevolutionary system complete with a physical simulation and fitness metrics with which to search for more ideal virtual creatures. The testing that was performed during this project is no substitute for a full implementation and genetic algorithm search for virtual creatures. Issues such as runtime performance and memory efficiency are likely to become apparent only after the entire system is complete. Of course, the benefits of all of this work only show up once the virtual creatures have become substantially evolved, and begin succeeding at the tasks designated for them.

References

[1] G. S. Hornby, H. Lipson, and J. B. Pollack. Evolution of generative design systems for modular physical robots. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 4146–4151. IEEE, 2001.

[2] G. S. Hornby and J. B. Pollack. Evolving l-systems to generate virtual creatures. *Computers & Graphics*, 25(6):1041–1048, 2001.

[3] P. Krcah. Evolutionary development of robotic organisms. *Master's thesis, Charles University in Prague*, 2007.

[4] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218. ACM, 2011.

[5] K. Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.

[6] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[7] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[8] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.