Modeling Self-Adapting and Multicultural Non-Player Characters in Artificial Worlds

Samuel Marin
Computer Science Institute
University of Namur, FUNDP
Rue Grandgagnage 21, B-5000 Namur, Belgium
sma@info.fundp.ac.be

Abstract

Modeling the behavior of Non-Player Characters (NPCs) so that they can act properly in an artificial world is a difficult, yet essential task in the development of modern computer games, often requiring never-ending adjustments of handwritten rules in a way to enable NPCs to perform well in the game world. This paper introduces an evolutionary method relying on high-level rule-based behaviors, genetic algorithms and speciation. By interconnecting these three areas, we propose a technique that has the potential to provide multicultural NPCs with the ability to adapt themselves to an artificial world.

Introduction

At a time when players are always expecting more realistic game worlds, we propose a technique aiming at creating interesting and appealing societies of multicultural and self-adapting NPCs. We define the "multicultural" concept as the ability for our NPCs to belong to different kinds of social groups, each group adopting a distinct strategy, permitting their members to adapt themselves to the world. Indeed, we are convinced this diversity can raise realistic worlds.

In order to create NPCs that can live in such worlds, a simple solution would be to add manually a large number of realistic interactions between NPCs and objects of the world. However, this process can lead to the creation of enormous finite state machines that are difficult to manage and maintain.

Our idea is to develop an evolutionary method aiming at providing realistic artificial worlds populated by multicultural NPCs with the ability to adapt themselves to their world.

This paper is organised as follows: first, we present our method, describing the various techniques that we will use, and explaining why they are relevant to reach our goal. The following section presents the experiments and their results. We then take a look at the related works. The last section gives some concluding remarks and plans for future work.

Method Description

This section describes the different concepts we combine to implement our approach, i.e. high-level rule-based behaviors, genetic algorithms (GAs) and speciation. It also motivates this choice.

High-Level Rule-Based Behaviors

We assume that each NPC is based on a specific behavior which is simply a set of 'If condition - Then action' rules. Each rule is evaluated in a specific order and the first fulfilled condition triggers the corresponding action, thus ending up the evaluation process until the next time step. Even though this technique is simple, it has the potential to provide complex and realistic behaviors.

We also want to incorporate in NPC behaviors as much prior and well-defined knowledge as possible. As mentioned in (Manslow 2002), this prevents NPCs from having to learn behaviors for which non-evolutionary implementations already exist in the literature.

Genetic Algorithms

We now explain why using GAs is suitable for our study.

We want a powerful tool to explore a large amount of possible NPC behaviors and only grab those that are really relevant for our world. GAs are known to perform well in poorly understood and complex search spaces.

We hope our NPCs will adapt themselves to our world and show interesting behaviors. However, finding the best possible ones is not an absolute requirement. With this respect, using GAs seems to be appropriate since it is a promising technique for finding solutions that do not have to be necessarily the best ones.

We only focus on the behavior of NPCs as a whole. We do not want to evaluate each rule separately, like classifier systems do for instance. Even if our approach may lead to strange combinations of rules, we think that creating possibly unexpected but interesting emergent behaviors is a significant requirement. Our GAs approach meets this requirement since each genome represents a complete behavior, i.e. the whole set of 'If-Then' rules.

All these reflections convinced us to use GAs in the implementation of our technique.

There are two important points to note. First, both the condition and the action of each rule may evolve. We think

that such a level of granularity is the best way to be surprised by the system, providing us interesting emergent behaviors. Of course, to prevent a too large search space, we limit the process by providing a kind of grammar that specifies what a "legal" genome is. We then also combine this search for the appropriate value for each behavior rule with the modification of the behavior structure itself, thus favoring the emergence of varied behaviors through generations. This last technique was inspired by the EANNs (Evolutionary Artificial Neural Networks) which aim at providing a way to evolve the weights of a neural network along with the topology of the neural network itself, i.e nodes or links. Since the structure of a neural network plays an important role in how effective the network is, those methods perform very well. Of course, we do not use here a neural network but we are convinced that the idea behind this evolving topology can be mapped to our structure of 'If-Then' rules.

Each genome (Figure 1) must be encoded to represent the entire behavior of a NPC.

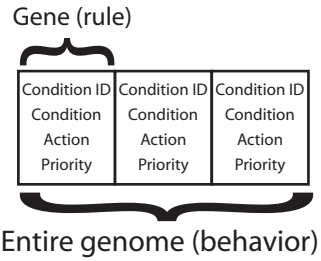

Figure 1: Example of a NPC genome made of three genes.

The behavior contains several rules. Each gene (rule) is composed of a condition and an action. In addition, each rule has also a priority number which defines in which order the rule will be evaluated within the set of rules of the behavior. Finally, the condition ID is a special information integer helping us to deal while evolving our behavior through our evolutionary method. More on this later.

We then need mechanisms to make our set of rules evolve through crossover and mutation, thus creating offsprings for the next generation. However, due to our specific encoding style, classical crossover systems might not provide valid offsprings. Indeed, our genomes are not necessarily the same length. Moreover, care must be taken as each genome can not simply recombine with another one using traditional approaches. For example, consider two genomes. The first has two rules with first condition A and second condition B while the second has also two rules but with first condition B and second condition A. Then, using typical crossover operations and using the crossover operator at the middle of the genome, we might face an undesirable situation in newly created offsprings because the second rule is duplicated and therefore useless.

A solution, largely inspired by NEAT (Stanley & Miikkulainen 2002) historical markings, but applied to our context, is to maintain a global database for all the conditions. Each condition now owns a unique identification number, the condition ID. Each time a rule is added to a behavior, the database is checked to verify if the rule condition has previously been added to it. If so, the rule containing the condition is assigned the existing condition ID. Otherwise, a new entry is created into the database, corresponding to this new condition and the rule containing the condition is assigned the new condition ID. This database remains alive during the entire execution of the evolutionary process.

This condition ID thus comes in handy and provides an easy way to deal with crossover. We simply sort genomes according to their condition IDs so as to reach the array-shaped alignment of Figure 2. Then, we iterate through each genome, applying our crossover policy. The policy we choose for our method is for an offspring to inherit genes from their fittest parent. However, actions of matching genes (meaning with the same condition) are randomly inherited from one of the two parents. Figure 2 depicts two genomes making the crossover operation to generate an offspring.

Figure 2: Performing the crossover.

As regards mutation, we have selected four operators in our implementation: add a rule, remove a rule, change the action associated with a rule and finally change the priority number associated with a rule.

Speciation

In addition to self-adapting NPCs, we also want multicultural NPCs. Speciation can help us. Indeed, as mentioned in (Buckland 2003), speciation is a niching technique that groups similar genomes into species (niches), only allowing genomes to mate with other genomes of the same species. Speciation thus provides an easy way to retain diversity among our population of genomes, permitting to converge towards multiple solutions at the same time. Since our genomes represent behaviors, we can expect the creation of distinct groups of behaviors among our population, each representing a cultural group with its own strategy permitting their members to adapt themselves to the world.

About the implementation, we group similar genomes into species with the help of a compatibility function. Again, the use of condition IDs eases the comparison between genes while iterating through genomes, and rapidly reports which genes are matching together. Then we adjust the fitness of each genome by sharing it amongst all the members of that species, preventing one species from taking easily over the rest of the population. This technique is known as explicit fitness sharing (Goldberg & Richardson 1987).

Experiments

Testbed Description

Our testbed is based on a set of rule-controlled NPCs which are living in a 2D world. NPCs can perceive information from their environment and can perform actions accordingly. All our experiments are inspired by a fairly simple ecology. We considered two kinds of NPC types: predators and preys and two kinds of objects: apple trees and apples.

Note that only predators have learning abilities and so use our evolutionary process. Each prey will receive a predetermined set of rules which can not evolve, giving them the ability to flee before predators and eat the apples that they can see.

Simulation and Results

Our main scenario was simply to initialize the world with preys, apple trees giving apples at random interval and some learning predators. We then ran the evolutionary process and checked what happened. During the first 2000 generations, we isolated various strategies among the population of predators. For instance, some of them learned to walk at random through the world seeking for preys. Another group of predators learned to stay near apple trees (obviously preys eat apples fallen from apple trees). Finally, another group learned that apples can feed them and thus constitutes a community of vegetarian predators. All of these cultural groups dynamically co-evolve, new cultures appear while others die out. Of course, various cultures can co-exist at the same time.

Related Works

NEAT: A large part of our ideas and implementation have been inspired by NEAT (Stanley & Miikkulainen 2002) and rtNEAT (Stanley, Bryant, & Miikkulainen 2005). The authors also use a GA and speciation, thus providing co-evolving groups of solutions, qualifying a multicultural ecology. The big difference between NEAT and our work is that NEAT deals with agents controlled by a neural network while we investigate rule-based controlled behaviors. Another difference is they focus their experiments on action-based games for which neural networks seem to be particularly suited while we investigate the creation of artificial societies for which rule-based behaviors seem to be more appropriate.

Growing Artificial Societies: Starting from simple rule-based agents, Epstein and Axtell (Epstein & Axtell 1996) build complex artificial societies reproducing such important sociocultural phenomena like segregation, trade, culture etc. We also have the ambition to raise realistic artificial societies of NPCs. Epstein and Axtell use agents with a predetermined set of rules which can not evolve. Instead, our system is based on evolving rules.

Evolutionary Tactics: In (Ponsen & Spronck 2004), the authors use a GA to find tactics for RTS games. Their genomes also represent a set of 'If-Then' rules. The main difference is that they are looking for one promising tactic at a time while we try to make evolve different solutions at the same time.

Conclusion

We presented our evolutionary method. We exposed how it works by means of its underlying components: high-level rule-based behaviors, GAs and speciation. We also developed a testbed aiming at supporting relevant properties and helping us to produce first promising results.

Future Work

In the near future, we would like to complexify the world and our NPCs in a way to develop more realistic multicultural societies of NPCs that can adapt themselves to their world. After that, we will think about its actual applicability in a computer game.

Acknowledgments

Special thanks to Stephane Bura who provided us with a rich stream of original ideas which has inspired us in this paper.

Our testbed implementation would not have been so efficient without the help of Mat Buckland's books (Buckland 2002; 2004) which provided us with a good background in genetic, game AI programming and NEAT implementation.

References

Buckland, M. 2002. *AI Techniques for Game Programming*. Premier Press.

Buckland, M. 2003. Building better genetic algorithms. *In Steve Rabin (Ed.), AI Game Programming Wisdom 2. Charles River Media.* 649–660.

Buckland, M. 2004. *Programming Game AI by Example*. Wordware Publishing, Inc.

Epstein, J. M., and Axtell, R. 1996. *Growing Artificial Societies*. The MIT Press.

Goldberg, D. E., and Richardson, J. 1987. Genetic algorithms with sharing for multimodal function optimization. *In Grefenstette, J. J., (Ed.), Proc. of the Second Intl. Conf. on Genetic Algorithms, San Francisco, CA: Morgan Kaufmann.* 148–154.

Manslow, J. 2002. Learning and adaptation. *In Steve Rabin (Ed.), AI Game Programming Wisdom. Charles River Media.* 557–566.

Ponsen, M., and Spronck, P. 2004. Improving adaptive game ai with evolutionary learning. *In Q. Mehdi, N.E. Gough, S. Natkin, and D. Al-Dabass (Ed.), Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004), Wolverhampton, UK. University of Wolverhampton.* 389–396.

Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation, 10(2).*

Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation, Vol.9, No. 6.*