Retaining Learned Behavior During Real-Time Neuroevolution

Thomas D'Silva, Roy Janik, Michael Chrien, Kenneth O. Stanley and Risto Miikkulainen

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78704 USA
tdsilva@mail.utexas.edu, roy@janik.org, mschrien@mail.utexas.edu, kstanley@cs.utexas.edu, risto@cs.utexas.edu

Abstract

Creating software-controlled agents in videogames who can learn and adapt to player behavior is a difficult task. Using the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) method for evolving increasingly complex artificial neural networks in real-time has been shown to be an effective way of achieving behaviors beyond simple scripted character behavior. In NERO, a videogame built to showcase the features of rtNEAT, agents are trained in various tasks, including shooting enemies, avoiding enemies, and navigating around obstacles. Training the neural networks to perform a series of distinct tasks can be problematic: the longer they train in a new task, the more likely it is that they will forget their skills. This paper investigates a technique for increasing the probability that a population will remember old skills as they learn new ones. By setting aside the most fit individuals at a time when a skill has been learned and then occasionally introducing their offspring into the population, the skill is retained. How large to make this milestone pool of individuals and how often to insert the offspring of the milestone pool into the general population is the primary focus of this paper.

1 Introduction

Non-player-characters (NPCs) in today's videogames are often limited in their actions and decision making abilities. For the most part such agents are controlled by hard-coded scripts (Buckland 2002). Because of this limitation, they cannot adapt or respond to the actions of a player or changes to the environment. Behavior of script-based agents is therefore predictable and hinders a game's replay value.

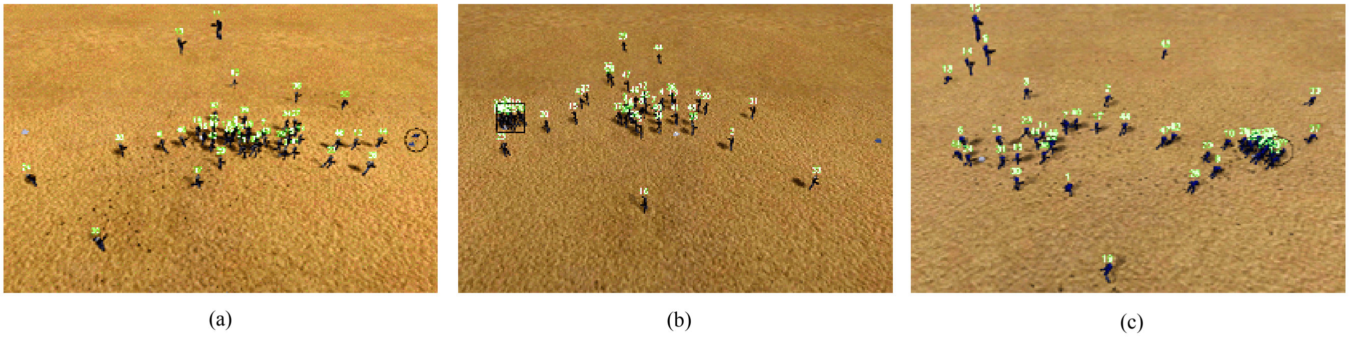
Employing machine learning techniques in a videogame environment can potentially allow agents to learn new skills (Geisler 2002). Neuroevolution in particular, which has had success in the domain of board games (Chellapilla and Fogel 1999, Moriarty and Miikkulainen 1993, Pollack and Blair 1996, Stanley and Miikkulainen 2004) is also well suited for videogames where a NPC's behavior needs to be flexible. Neuroevolution has made possible a new genre of videogame in which the player first teaches agents directly in the game and then releases them into a battle or contest.

NERO (Stanley, Bryant, and Miikkulainen 2005) is the first such videogame. NERO uses the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) method to allow the player to train agents in a variety of tasks. Typical tasks include running towards a flag, approaching an enemy, shooting an enemy and avoiding fire.

The main innovation of the rtNEAT method is that it makes it possible to run neuroevolution in real-time time, i.e. while the game is being played. It can therefore serve as a general adaptation engine for online games. One significant challenge for rtNEAT is training agents in multiple unrelated behaviors. In such scenarios, the player trains each behavior separately. For example, training soldiers to both go towards a flag and simultaneously go towards an enemy may result in a population that moves to the midpoint between the two goals. If instead the behaviors are trained separately, the first behavior may be *forgotten* while subsequent behaviors are trained.

The focus of this paper is to solve this problem using a technique called *milestoning*. Milestoning saves individuals into a separate cache called a milestone pool. Offspring from this pool of individuals are then inserted into the general population from time to time. Milestoning has the effect of retaining the behavior the population had when the milestone pool was created while still learning the new behavior. In this manner, multiple behaviors can be trained separately and combined through milestoning. The result is individual agents that can perform *both learned behaviors.*

The main purpose of this paper is to demonstrate that milestoning is effective at training agents in multiple tasks. Second, the appropriate sizes for milestone pools, and how often the evolution should draw from them are determined experimentally.



(a) (b) (c)

Figure 1: The three phases of the experiment. a) Phase I: The soldiers spawn in the center of the arena and are trained to approach the flag (circled). b) Phase II: The soldiers spawn and are trained to approach the enemy (identified by the square). c) Phase III: The soldiers from Phase II are evaluated based on their ability to approach the flag (circled). The question is whether networks that learn Phase II will forget what they learned in Phase I. Milestoning is intended to prevent this. The numbers above soldiers' heads are used to identify individuals.

The experiments consist of three phases:

- Phase I: Train agents to go to flag (figure 1a).
- Phase II: Train agents to approach an enemy without a flag present (figure 1b).
- Phase III: Test agents from Phase II in approaching the flag again *without* evolution (figure 1c).

This procedure was attempted with several combinations of probabilities and pool sizes. The results establish that milestoning helps in remembering learned tasks while learning new ones.

The next section describes NEAT, rtNEAT (the real-time enhancement of NEAT), and NERO. Section 3 explains how milestoning is implemented within NERO. Section 4 outlines the experiment in more detail and analyzes the results. In Section 5, the appropriate applications for milestoning are discussed and future work is outlined.

2 Background

The rtNEAT method is based on NEAT, a technique for evolving neural networks for reinforcement learning tasks using a genetic algorithm. NEAT combines the usual search for the appropriate network weights with *complexification* of the network structure, allowing the behavior of evolved neural networks to become increasingly sophisticated over generations.

2.1 NEAT

NEAT evolves increasingly complex neural networks to match the complexity of the problem. NEAT evolves both connection weights and topology simultaneously. It has been shown to be effective in many applications such as pole balancing, robot control, vehicle control, board games and videogames (Stanley 2004, Stanley and Miikkulainen 2004).

NEAT is based on three fundamental principles: (1) employing a principled method of crossover of different topologies, (2) protecting structural innovation through speciation, and (3) incrementally growing networks from a minimal structure.

Mating, or the crossing over of genomes of two neural networks of possibly differing structure, is accomplished through innovation numbering. Whenever a new connection between nodes is created through mutation, it is assigned a unique number.

Offspring produced with the new connection inherit the innovation number. Whenever networks are crossed over, those genes that have the same innovation number can be safely aligned. Genes of the fit organism with innovation numbers not found in the other parent are inherited by the offspring as well.

Speciation occurs by dividing the population into separate, distinct subpopulations. The structure of each individual is compared dynamically with others and those with similar structure are grouped together. Individuals within a species share the species' overall fitness (Goldberg and Richardson 1987), and compete primarily within that species. Speciation allows new innovations to be optimized without facing competition from individuals with different structures.

Networks in NEAT start with a minimal structure, consisting only of inputs connected to outputs with no hidden units. Mutation then grows the structures to the complexity needed to solve the problem. Starting this way avoids searching through needlessly complex structures.

2.2 rtNEAT

rtNEAT was developed to allow NEAT to work in real time while a game is being played. In a videogame environment it would be distracting and destructive for an entire generation to be replaced at once. Therefore, rather than creating an entirely new population all at once, rtNEAT periodically selects one of the worst individuals to be replaced by a new offspring of two fit parents.

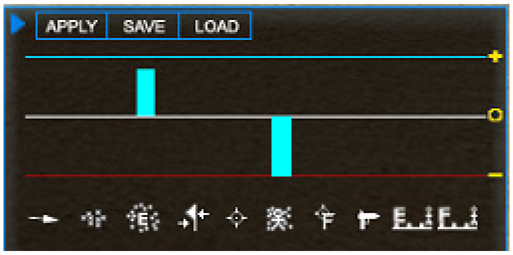


Figure 2: Sliders used to set up training scenarios. The sliders specify what behaviors to reward or punish. For example, the “E” icon means, “approach enemy” while the icon depicted with the soldier and the dots represents “get hit”. The current setting rewards approaching the enemy, but places even more emphasis on avoiding getting hit. The flag icon means “approach flag” and is used in Phase I and Phase III of the experiments.

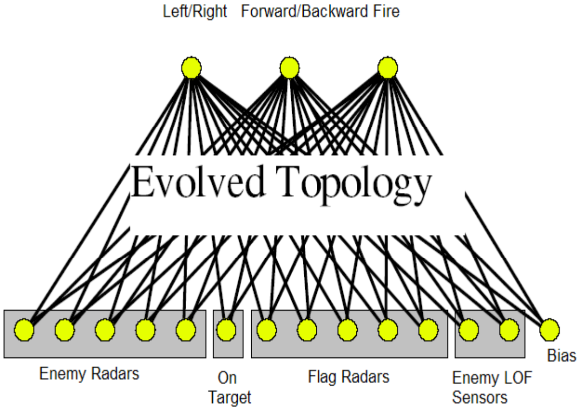


Figure 3: Nero sensors and action outputs. The soldier can see enemies, determine whether an enemy is in its line of fire, detect flags, and see the direction the enemy is firing. These inputs are used to evolve a network to specify when to move left/right, forward/backward, and fire. Because different sensors are used to detect the enemy and the flag, approaching the enemy and approaching the flag cannot rely on the same network connections.

The worst individual must be carefully chosen to preserve speciation dynamics. Determining the correct frequency of replacement is also important: if individuals are replaced too frequently they are not alive long enough to be evaluated properly, and if they are replaced too infrequently then evolution slows down to a pace the player does not enjoy. In this way, evolution is a continual, ongoing process, and well suited to an interactive environment. rtNEAT solves these and several other problems to allow increasingly complex neural networks to evolve in real time (Stanley and Miikkulainen 2005).

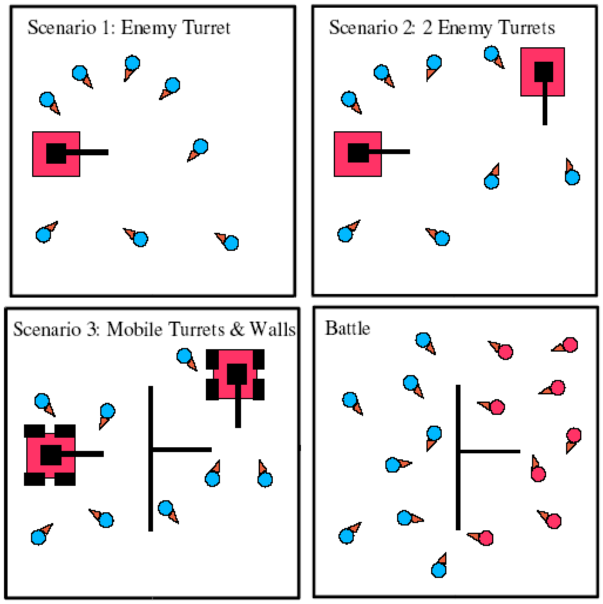

Scenario 1: Enemy Turret
Scenario 2: 2 Enemy Turrets
Scenario 3: Mobile Turrets & Walls
Battle

Figure 4: A turret training sequence. This figure depicts a series of increasingly complicated exercises. In Scenario 1, there is only one stationary enemy (turret), in Scenario 2 there are two stationary turrets, and in Scenario 3 there are two mobile turrets with walls. After the soldiers have successfully completed these exercises they are deployed in a battle against another team of agents.

2.3 NERO

NERO (http://dev.ic2.org/nero_public/) is a videogame designed to be a proof of concept for rtNEAT. The game consists of teaching an army of soldiers to engage in battle with an opposing army. The game has two distinct phases: training and combat.

During training, soldiers can be rewarded or punished for different types of behavior (figure 2). The player can adjust a set of reward sliders at any time to change the soldiers' goals, and consequently the equation used to calculate the fitness of each individual. The behaviors include (1) moving frequently, (2) dispersing, (3) approaching enemies, (4) approaching a flag, (5) hitting a target, (6) getting hit, (7) hitting friends, (8) and firing rapidly.

Sliders can be used to reward or punish these behaviors in any combination. Technically, the sliders are a way to set coefficients in the fitness function.

The soldier’s sensors are presented to the neural network as inputs. The network has outputs that determine which direction to move and whether to fire or not (figure 3).

The agents begin the training phase with no skills and only the ability to learn. In order to prepare for combat, the player must design a sequence of training exercises and goals. The exercises are increasingly difficult so that the agents can begin learning a foundation of skills and then build on them (figure 4).

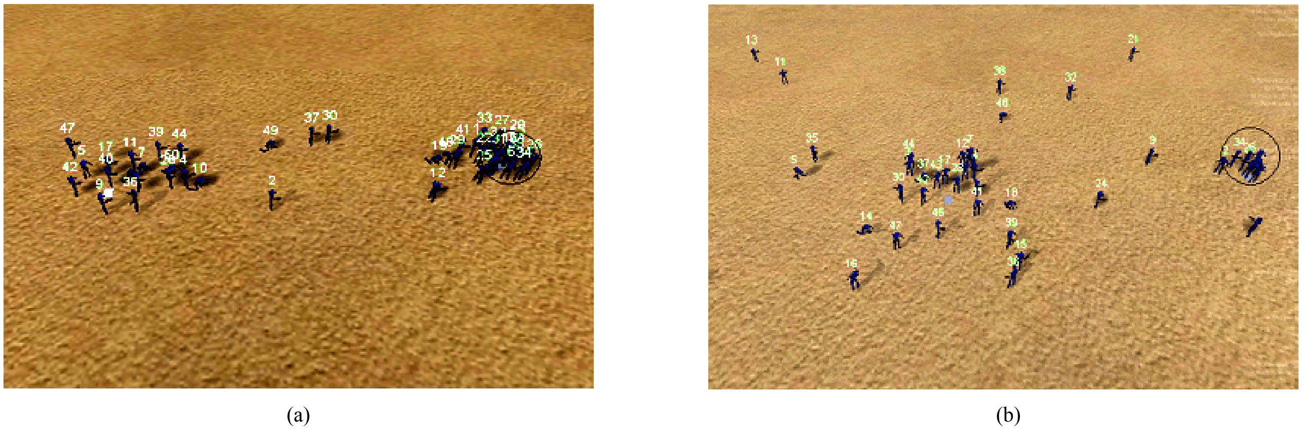


Figure 5: Approaching the flag during the third phase. a) Using milestoning, 80% of the population remembers how to approach the flag (encircled) in the third phase. The entire population can be seen traveling from the spawn point (at left) to the flag. b) Without milestoning, only 30% of the population remembers how to approach the flag (encircled) in the third phase. Milestoning clearly helps the individuals retain their prior skills.

During the training phase, the player places objects in the environment, sets the reward and punishment levels for the behaviors he or she is interested in, and starts evolving an army. While there are a limited number of explicit fitness parameters (i.e. sliders), choosing the right combination of rewards and punishment can result in complex behaviors. For instance, training soldiers to shoot the enemy and avoid getting hit results in soldiers that fire at an enemy and then run away quickly.

The army size during training stays constant at 50. After a predetermined time interval, an individual is removed from the field and reappears at a designated spawn point. In addition, rtNEAT will periodically remove the least fit individual from the field and replace it with a new network, the offspring of two highly fit networks (Stanley, Bryant, and Miikkulainen 2005).

As the neural networks evolve in real time, the agents learn to do the tasks the player sets up. When a task is learned, the player can save the team to a file. Saved teams can be reloaded to train in a different scenario or deployed in battle against another team trained by a different player (e.g. a friend, or someone on the internet).

In battle mode, a team of heterogeneous agents is assembled from as many different training teams as desired. Some teams could have been trained for close combat, while others were trained to stay far away and avoid fire. When the player presses a “go’ button, the two teams battle each other. The battle ends when one team is completely eliminated. If the two team’s agents are avoiding each other, the team with the most agents left standing is the winner.

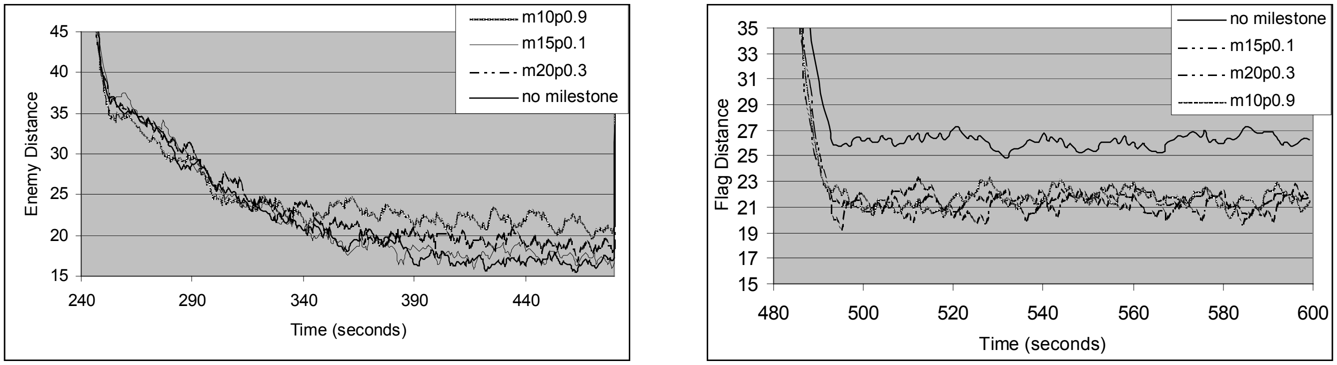
Although the game was designed partly to imitate real-time adaptation technology, it has turned into an interesting and engaging game in its own right. It constitutes a new genre where learning is the game.

3 Milestoning Method

A *milestone pool* is a cached collection of individual’s genomes pulled from a population of agents at a specific point in time. Milestoning depends on two parameters: (1) the number of individuals to copy into the pool and (2) the likelihood that a new individual will be bred from the pool. The most fit agents according to the fitness function in effect at the time of milestoning are put into the pool.

In subsequent evolution, there is a chance that an offspring will be generated from the milestone pool. In such cases, one parent is selected at random from the milestone pool, and the other is chosen as usual from the current population. Thus, the offspring is a hybrid of a modern individual and one of its ancestors that knew how to perform a prior task. The parent from the milestone pool is chosen using the roulette method with preference towards networks that had higher fitness *at the time of milestoning*.

Milestoning is best suited for situations where the agents are being trained for two unrelated tasks, such as learning to approach a flag and to approach an enemy (different network sensors are used to detect flags and enemies). Milestoning is based on the intuition that periodically mating with genetic material that solves a prior, unrelated task, will force the population to retain the older skill even if it is not necessary for the current task. Unlike many ideas in evolutionary computation, there is no strong biological analogy to this process: In natural evolution, the genomes of distant ancestors do not remain available for mating.



(a) (b)

Figure 6: Evaluating the population's performance in Phase II and Phase III: a) Distance from the enemy in the second phase. This figure compares the three best parameterizations with the non-milestoned case ('m' denotes the milestone pool size, 'p' denotes the probability). (a) The average distance continues to decrease because the population is evolving. Parameterizations m15p0.1 and m20p0.3 perform as well as the population without milestoning, while m10p0.9 performs only slightly worse. b) Average distance from the flag in the third phase is graphed. All three parameterizations perform significantly better than the non-milestoned population. The average distance reaches a steady state, because the population is not evolving. The reason the average distance to the flag is higher for the non-milestoned population is that only 30% of the individuals on the field remember how to approach the flag. The remaining robots are aimlessly wandering elsewhere (figure 5).

However, in an artificial context it is certainly feasible to retain old genetic material indefinitely, and by periodically inserting that material into the population, the information stored within it can be maintained by the majority of the population, even as they learn a new skill.

At first, the toll for mating with ancestors that are obsolete for the current task may somewhat slow or inhibit learning a new task, but the population should eventually evolve the ability to incorporate the old genetic information without losing its current abilities. Those individuals that are capable of combining their skills with those of the milestone pool will be most successful and the population will *evolve* to incorporate the milestone. The next section describes experiments that test the feasibility of this idea.

4 Experiments

NERO was set up to train for two tasks. The first task consisted of training agents to approach a flag, and the second was to train them to approach an enemy. The spawn point and the locations of the flag and the enemy were fixed so that the agents could be consistently evaluated across different milestone pool sizes and probabilities. The agents were trained to approach the flag for 240 seconds, a milestone was set, and the agents were trained to approach the enemy for 240 seconds. After these 480 seconds, the agents' ability to approach the flag was evaluated for the next 120 seconds with evolution turned *off* so the level of skill retained could be measured (figure 1). In order to determine the optimum probability and milestone pool size, 25 experiments were run with probability 0.1, 0.3, 0.5, 0.7 and 0.9, and milestone pool size 5, 10, 15, 20 and 25. One experiment was also run without using the milestone pool. Ten runs of each variation of probability and milestone pool size were conducted and the results were averaged. The instantaneous average distance from the flag in Phase III is used to measure how well the soldiers approach the flag

There is significant variation among the different parameterizations, but populations with higher probabilities remember to approach the flag in the third phase better than the non-milestoned population. This makes sense intuitively since higher milestone probabilities make it easier to remember the milestoned task but difficult to learn a new task because the population is continually mating with the milestone pool. Also, the higher milestone pool sizes do not perform as well in the third phase as smaller pool sizes because large pools include undesirable individuals who cannot perform the milestoned task.

Of the 25 combinations that were evaluated the population with pool size 10, probability 0.9 (m10p0.9) performs the best in the third phase, but it has a small but significant drop in performance on the second phase. In general those settings that best retain the ability to approach the flag result in slower evolution in the second phase because of the overhead of retaining prior skills. Therefore, the best parameterizations should be a compromise between the ability to retain previous skills, yet still learn new tasks.

Ten of the 25 combinations showed no statistically significant loss of performance in Phase II (figure 6a) as compared to the non-milestoned case. This was determined by performing a Student's t-test comparing the average distance of the milestoned populations with the non-milestoned population at the end of the second phase, at which point the soldiers have learned to approach the enemy. Of these ten combinations, the populations with pool m15p0.1 and m20p0.3 are the best at approaching the flag in phase III (figure 6b). Agents that used milestoning with these parameterizations were on average closer to the flag than those that did not use milestoning. The average distance to the flag without using milestoning was 27.86 units, while the average distance for m15p0.1, m20p0.3 and m10p0.9 were 23.22, 22.84, and 23.18 respectively. These three parameterizations resulted in agents that could approach the flag significantly better ($p<0.05$, according to Student's t-test) than the non-milestoned case. The reason for the difference is that 80% of the populations with these parameterizations know how to approach the flag in the third phase as compared to only 30% of the non-milestoned population (figure 5a, 5b).

While these results demonstrate that the same level of performance is achieved in phase III as in phase I, it is important to also note that at this level, the majority of agents actually solved the task. For example, during the m10p90 test run, 94% of the population successfully reached the flag after phase I. After phase II, even with the milestone pool continuously recombining with the evolving population, 70% of the population reached the enemy; in this same population, in phase III, 82% of the population retained the ability to reach the flag. Therefore, with the right parameterizations, milestoning performs significantly better on the third phase than evolution without milestoning, and causes no significant drop in performance in the second phase.

5 Discussion and Future Work

Milestoning works as a procedure to learn multiple tasks effectively without interference. Milestoning is best suited to learning unrelated tasks in which there is a chance of forgetting earlier tasks if they are trained sequentially. Many more agents properly respond to a milestoned situation than a population that was not trained using milestoning, and milestoned agents do not confuse the two tasks. Milestoning may therefore be a useful technique for video games that require online learning with quick response times, in which the non-player character has to learn the human player's behaviors and quickly respond without forgetting what it has learned in the past.

In the future, milestoning can be extended to more than two behaviors. Such a procedure consists of training for a task and then creating a milestone, then training for another task and creating a milestone, and so on. In this way, it should be possible to train for a number of tasks and not lose the ability to accomplish any of the tasks along the way. The goal of such research is to determine the number of tasks that can be learned using milestoning and the kinds of tasks for which it is suited.

Conclusion

The experiments reported in this paper show that when training for multiple tasks, milestoning helps in retaining learned knowledge. The population can continually absorb and maintain ancestral genetic information while at the same time learning a new skill. This technique can be used to build interesting games where agents adapt to changing environments and player's actions.

Acknowledgements

This research was supported in part by the Digital Media Collaboratory at the University of Texas at Austin and by DARPA under NRL grant N00173041G025.

References

Buckland, M. 2002. *A1 techniques for game programming.* Cincinnati, OH: Premier-Trade.

Chellapilla, K., and Fogel, D. B. 1999. Evolution, neural networks, games, and intelligence. In *Proceedings of the IEEE* 87:1471–1496.

Geisler B. 2002. An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a "First Person Shooter" Video Game. MS Thesis, Department of Computer Sciences, The University of Wisconsin at Madison.

Goldberg, D. E., and Richardson, J. 1987. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, 148–154. San Francisco: Kaufmann.

Moriarty, D., and Miikkulainen, R. 1993. Evolving complex Othello strategies with marker-based encoding of neural networks. Technical Report AI93-206, Department of Computer Sciences, The University of Texas at Austin.

Pollack, J. B., Blair, A. D., and Land, M. 1996. Coevolution of a backgammon player. In *Proceedings of the 5th International Workshop on Artificial Life: Synthesis and Simulation of Living Systems,* 92-98. Cambridge, MA: MIT Press.

Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10: 99-127.

Stanley K. O., Bryant B. D., and Miikkulainen R. 2005. The NERO Real-Time Video Game. To appear in: *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games.*

Stanley, K. O., and Miikkulainen, R. 2004. Evolving A Roving Eye for Go. In *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY: Springer-Verlag.

Stanley, K. O. 2004. Efficient Evolution of Neural Networks through Complexification. Doctoral Dissertation, Department of Computer Science, University of Texas at Austin.