Backpropagation without Human Supervision for Visual Control in Quake II

Matt Parker and Bobby D. Bryant

Abstract— **Backpropagation and neuroevolution are used in a Lamarckian evolution process to train a neural network visual controller for agents in the Quake II environment. In previous work, we hand-coded a non-visual controller for supervising in backpropagation, but hand-coding can only be done for problems with known solutions. In this research the problem for the agent is to attack a moving enemy in a visually complex room with a large central pillar. Because we did not know a solution to the problem, we could not hand-code a supervising controller; instead, we evolve a non-visual neural network as supervisor to the visual controller. This setup creates controllers that learn much faster and have a greater fitness than those learning by neuroevolution-only on the same problem in the same amount of time.**

I. Introduction

A large variety of complex tasks can be solved by humans using only simple two-dimensional image data for input. A vehicle can be adequately controlled remotely by a human using as input a transmitted image from a single mounted camera. Doctors are able to remotely perform intricate and detailed surgeries by controlling a robotic arm and using camera images for input. Virtual worlds in computer games are accessed mainly by using the 2-dimensional screen on a monitor, yet there are a huge variety of games that require that the player perform many different tasks to complete. Humans are able to use this image information because their complex brains are designed to quickly process raw vision. Computers can also process raw visual data but generally must use complex algorithms that are computationally intensive due to the large number of color values in an image that must be processed. Processing visual data in real-time is usually very difficult because the information must be processed within short time-constraints.

Neural networks have previously been used as controllers in research using raw visual input in a real-time environment. Pomerleau trained a neural network with backpropagation to drive a car on a road using a 30x32 grayscale input; the controller learned to imitate a human driver in real-time [1]. In research by Baluja, the same experiment was modified to use an evolutionary computation model instead of backpropagation. The new method evolved more robust controllers, but trained with recorded data sets rather than in real-time [2]. A car controller was trained by Floreano et al. to use active vision to race around a track in a realistic driving simulator [3]. Kohl et al. trained another virtual car racing controller by using NeuroEvolution of Augmenting Topologies and a 20x14 grayscale input to evolve a vehicle warning system [6]. They then proceeded to try out this technique on an actual robot with a mounted camera and found that the system was able to adequately warn the robot about obstacles. These experiments all show that neural networks are viable controllers for real-time raw vision applications.

Matt Parker (mparker@cse.unr.edu) is a graduate student at the Department of Computer Science and Engineering, University of Nevada, Reno.
Bobby D. Bryant (bdbryant@cse.unr.edu) is an assistant professor at the Department of Computer Science and Engineering, University of Nevada, Reno.

Synthetic vision, which is a method that adds extra-visual data into the raw visual field, has been used by many researchers for experiments that exist entirely in a virtual world [4][5]. This technique might change the color of objects to help with identification to specify some field such as distance. For example, in research by Enrique et al., the visual input was re-rendered into two separate color-coded views: one displayed the color according to entity identification and wall-angle, and the other displayed colors corresponding to the velocities of objects. This higher-level information simplified the raw visual field so that a hand-coded controller that navigated a map and picked up health boxes could more easily be created [4]. An agent's controller was hand-coded by Renault et al. to walk down a hallway using extra-visual pixel data: each pixel included distance, object identification, and color values [5]. Because these experiments used virtual worlds, the extra-visual information could be easily accessed; in the real world, however, such information is only accessed through great difficulty and by expensive sensors; for that reason synthetic vision techniques cannot be easily transferred to real-world robotics.

First Person Shooters (FPS) such as Quake II are popular video games that put a player in control of a character that generally must shoot many enemies to win. The player plays the game looking through the eyes of the in-game character and usually can see the aim of the character's gun. FPS's are often used for research because they usually have some native interface for programming AI for the game. Research was conducted by Bauckhage et al. that trained multiple neural networks to imitate behavior of human players whose actions had been previously recorded. One neural network was used for combat and another was used to traverse the map when not in combat [7]. An agent was trained in Quake III by Zanetti and El Rhalibi to collect items, engage in combat, and navigate a map. The controller was a neural network that learned by backpropagation on pre-recorded demos of human players, using the player's weapon information and location as inputs [8]. In research by Graham et al., controllers were trained using neuroevolution for obstacle avoidance and pathfinding strategies in the game Quake II [10]. A Neural Gas method was used by Thurau et al. to train agents to imitate the waypoint-navigation of human players in Quake II, also by observing the humans' movements from pre-recorded demos [9].

In our previous research we trained vision based controllers for agents in Quake II. First, we trained a simple recurrent network [12] by neuroevolution to control an agent to shoot a moving enemy in a visually-simple room [13]. The input to the neural network was a 14x2 grid of grayscale input blocks, each activated to the average grayscale value of the pixels covered by the block. We tested two different retinal layouts for the 14x2 grid of blocks: in the uniform retina we used blocks that were all the same size across the controller; in the graduated density retina we made the blocks thinner in the center of the retina and wider in the periphery. We found that the graduated density retina learned more quickly to shoot the enemy because the enemy could more easily be seen in the higher-resolution center of the retina.

In more recent research we took the graduated density controller and used it in a more visually complex room that contained varying lights and shadows (figure [15]). We found that the graduated density controller was unable to learn in the complicated environment using only neuroevolution. Instead, we hand-coded a controller that used exact location inputs rather than visual input to determine its behavior. We used this controller as the supervisor for backpropagation of the evolving neural network controllers: each controller was given 12 seconds to learn via backpropagation and then was given 24 seconds to test its fitness. The individuals were then bred according to their fitnesses. The idea that behavioral changes learned during an individual's lifetime can be genetically passed on to its offspring was first proposed by Jean-Baptiste Lamarck [16], and has been implemented previously by many researchers [17][18][19], including the use of backpropagation with neuroevolution [20][21]. Our experiments using a hand-coded controller to supervise backpropagation learned much more successfully than did the controllers that used pure neuroevolution on the same problem. However, a problem arises that a human must manually program the supervising controller, and a human might not know the optimal solution. In order to bypass the human-element completely, we have extended the research by evolving, rather than hand-coding, the non-visual supervisory controller; we then use that evolved controller to help train the visual controller.

II. The Quake II Environment

This research uses Quake II by Id Software, Inc. as a platform for testing visual controllers. Quake II is an FPS that has the player take control of a space marine who must fight his way out of a hostile alien world, using a wide variety of weapons and powerups. The game can be played either in single player or with many players over a network. The game's textures, maps, rules, models, and sounds can all be easily changed to create custom modifications.

Fig. 1. An in-game screenshot from the game Quake II.

We chose Quake II as our research platform for several reasons. The most important is that the game is open source, released by Id Software, Inc. under the General Public License (GPL), and can be compiled on many UNIX-like operating systems [11]. Because it is open source, we were able to create an easy interface to any values and functions that might be useful for AI; particularly, we are able to control the behavior of the player and read color values from the rendered screen buffer. Another important requirement for our platform is that it does not render using a 3D graphics card; if the engine used 3D acceleration then we could only run one copy of the game per graphics card. Instead, Quake II can render the 3D scene using only the regular CPU, so that we can run several copies of the game on one machine. We run the game in real-time and distribute the evolution over several copies of the game, so it is essential that we run as many copies as possible per computer.

Many FPS's, including Quake II, allow users to create AI opponents, called *bots*; generally these bots only run on the game server and cannot access the screen render because they do not use visual input for the controllers. For this research we use the client, which is normally controlled by a human player, and control it with our AI and read the visual data from the screen render.

III. The Experiment

For this experiment we have parted from the simple open room used in our previous research [13][15] and have inserted a large square pillar into the midst of it (figure 3). The room is the same size as in our previous research, with the same style of shading, and the task is also the same: shoot and kill the enemy as many times as possible. The pillar in the middle of the room occludes the player's view of most of the map, so often the enemy is not visible and the hallways around the pillar must be traversed in order to find him. Because of this new setup, the agent can no longer use its previous optimal behavior, which was to spin around in circles shooting bursts of shots at the enemy.

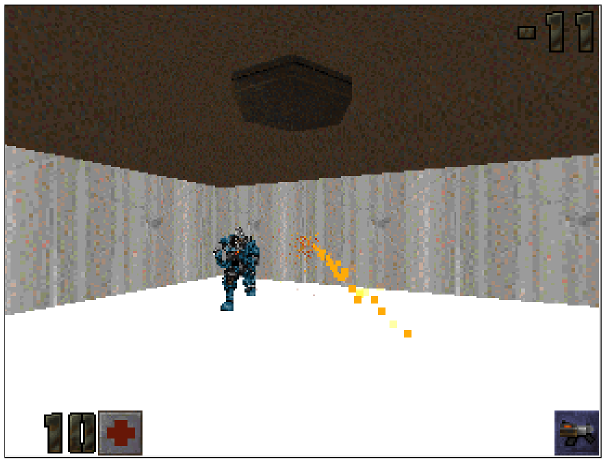

Fig. 2. An in-game screenshot from the simplified environment used in our previous experiments.

In this map we place the spawn portals for the learning agent over the empty hallways surrounding the pillar, so that he drops to the floor and can immediately begin hunting for the enemy. The enemy's spawn portals are over the top of the pillar, and he is programmed to automatically walk to the opposite side of wherever the agent happens to be. Once the enemy drops to the hallway floor, he begins moving about in a random pattern.

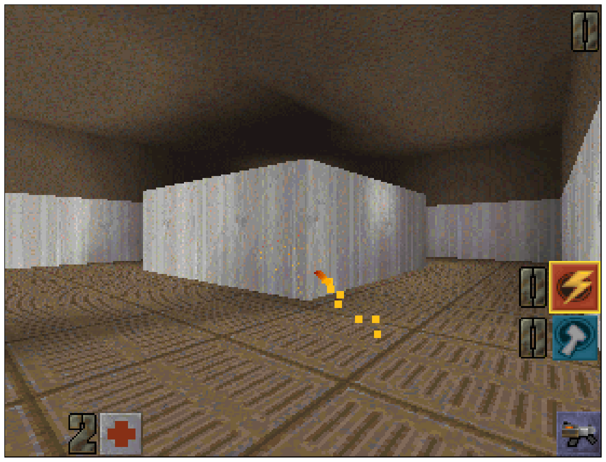

Fig. 3. An in-game screenshot of the environment used in this experiment. The floor and ceilings are brown, the walls are gray, and the enemy is dark blue. The room is dimly lit with varying shadows. A large square pillar is placed in the center of the room. The display of the shooter's own weapon has also been removed.

To kill the enemy opponent, the learning agent is equipped with a blaster that shoots out deadly beams of plasma. The blaster stores up to 25 energy units and uses 5 units for each shot; the energy units recharge after some time. This setup allows the agent to shoot out bursts of shots and is implemented to encourage the agent to wait to shoot until the enemy is in range.

IV. The Neuro-Visual Controller

The neural-visual controller consists of an array of visual inputs and a neural network. The visual input array consists of 28 grayscale blocks of pixels whose grayscale color values have been averaged together. The blocks are arranged in a 14x2 grid that spans across the width of the game screen and takes up about 8.4% of the height of the screen. We use the graduated density retina from our previous research [13]; the blocks in the center of the retina are thinner than those nearer to the periphery, with each block, from inmost to outmost, being approximately 1.618 times larger than the previous.

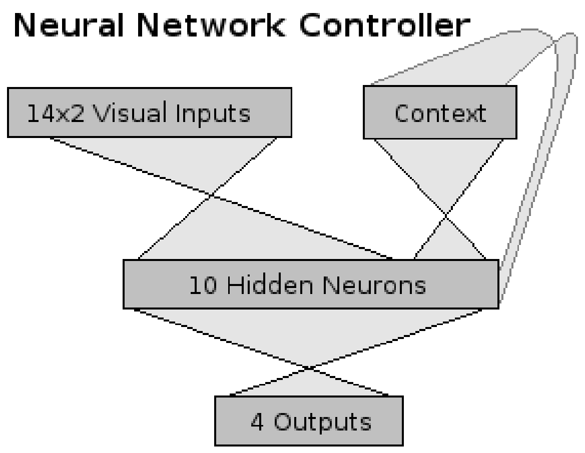

Fig. 4. The neural network control is a standard recurrent network with a hidden/context layer.

The neural network controller used in this experiment is a simple recurrent network [12] with 28 inputs, 10 hidden/context units, and 4 outputs neurons (figure 4). A bias unit with a value of 1.0 is input to the hidden and output layers. The weights of the neural network are floating point numbers that start in the range $[-2.0, 2.0]$, but through learning and mutation may exceed the bounds of that range. The inputs are the 28 visual blocks from the retina, as well as the recurrent hidden layer neurons. The summation of the products of the inputs and their corresponding weights is squashed by the $tanh$ function. The neural network outputs 4 values, corresponding to right/left movement, forward/back movement, right/left turning movement, and shoot or not-shoot action. The neural network calculates the movement of the agent for every frame of gameplay, 40 frames per second.

V. Supervising Bots for Backpropagation

In order to perform backpropagation learning, the agent must have some supervising behavior to imitate. As explained in the introduction of this thesis, many real-time backpropagation experiments in FPS's involve imitating pre-recorded human behaviors. A problem with using human gameplay is that humans are not very behaviorally consistant, and it is a hassle to build up a large library of recorded human players; moreover, unless they are particularly good, human players perform less than optimal.

Instead of using a human player our first solution was to use a hand-coded controller that would tell the neural network what it should have done for every frame. If we were able to program this supervising controller to do exactly what we wanted using the same inputs as the visual controller then we would not have much need to evolve a neural network that does the same thing with the same inputs. However, hand-coding visual controllers is particularly difficult, so instead we "cheat" and use non-visual inputs, like the enemy's exact X and Y location, to easily hand-code a controller that does what we want. It's permissible to let the supervising controller use non-visual inputs because in the end we still end up with a visual-only controller, but it will have learned its behavior from a non-visual controller.

In our previous Lamarckian research [15], we hand-coded the non-visual controller, which worked well because we already had a good idea of the optimal solution to the problem. We had learned the solution by observing the strategies evolved by neuroevolution in the retinal layout experiment, which used a room with no shadows. We did not know, however, the optimal solution for the pillar room. Rather than making a hand-coded controller, we evolve a neural network that uses non-visual data for inputs. The non-visual neural network controller is a simple recurrent network with a hidden layer. There are 11 non-visual inputs: 7 are wall distance sensors at 0 degrees and at 10, 25, and 60 degrees on either side; there are 4 enemy inputs which tell the x and y distance and x and y velocities of the enemy, relative to the agent's heading angle. Whenever the enemy is not on screen, is occluded behind the pillar, or is not near the center of the field of view, since some input must be given to the network, fictional inputs are used that indicate an imaginary enemy who is very far away. It is important that the non-visual controller cannot use information that is not somehow accessable to the visual-controller, so that the non-visual does not attempt to train the visual to do the impossible. The non-visual controller has 4 outputs that are exactly the same as the visual controller's, so the outputs can easily be used to teach the visual controller through backpropagation.

VI. Training

To evolve the weights of both the visual and non-visual neural networks, we represent the weights as a chromosome and evolve the population of chromosomes using a Queue Genetic Algorithm (QGA), which is a steady-state first-in-first-out genetic algorithm [22]. The QGA uses a queue (figure 5) to represent the population of chromosomes and arranges them from youngest to oldest. Whenever a new chromosome is needed, two individuals are stochastically selected (roulette wheel), according to fitness, and are used to form a new child through crossover and mutation. This new child is tested for fitness and, upon returning, is inserted onto the queue as the youngest; at the same time, the oldest individual is deleted from the queue. The QGA allows for easy distribution of evolution over a network because many individuals can be sent out to be tested and may return at their leisure without forcing the QGA to wait. It is essential to distribute the evolution because we run Quake II at normal game speed; by running several copies at once over multiple processors and over the network, we are able to evolve our controllers fairly quickly.

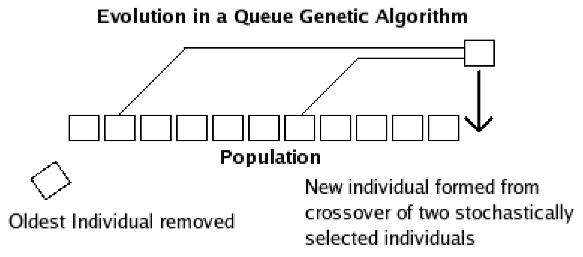
Evolution in a Queue Genetic Algorithm
Population
Oldest Individual removed
New individual formed from crossover of two stochastically selected individuals

Fig. 5. The Queue Genetic Algorithm (QGA). New individuals are bred from parents chosen from the current population by roulette wheel selection according to fitness. After each new individual is evaluated it is enqueued and the oldest individual is dequeued and discarded.

For this experiment we used a population size (queue length) of 128 chromosomes, consisting of 434 genes for the visual controllers, and only 136 genes for the non-visual controller; the genes are representing by floating point numbers. Crossover is uniform, with equal chance of drawing each gene from either parent. The resulting chromosome is mutated with an independent 10% chance of mutation per gene, and the mutations are calculated by drawing a delta from a sharply peaked random distribution, $\frac{log(n)}{10} * random(-1or1)$, where n is a random number in the range $[0, 1]$. This function has a low probability of generating large deltas and a high probability of generating small deltas.

We compare the Lamarckian controller to a neuroevolution-only controller. The setup of each test is the same, except that each individual in the Lamarckian tests are given 12 extra seconds to learn through backpropagation to imitate the non-visual supervisory controller before their fitnesses are tested. During these twelve seconds the non-visual controller completely controls the agent. The visual neural network controller, which consists of the weights given by the chromosome as dispensed by the QGA, is given the visual inputs of the agent as the agent performs the control specified by the non-visual controller. The visual neural network outputs what it would do to control the agent for each frame of gameplay, 40 per second, and the error is backpropagated with a learning rate of 0.0001. The weights are permanently modified throughout the backpropagation; after the backpropagation the updated chromosome remains static and is tested for 24 seconds and is returned to the QGA.

For the supervising controllers, we evolve one non-visual controller for each Lamarckian test, selecting the best individual from the 1000th generation. We then use it as supervisor for the 12 seconds of backpropagation in the Lamarckian tests, and run those tests for 1000 generations. To accurately compare the neuroevolution-only controller to the Lamarckian controller, we must allow it to evolve for the sum total of the time used to evolve the non-visual supervising

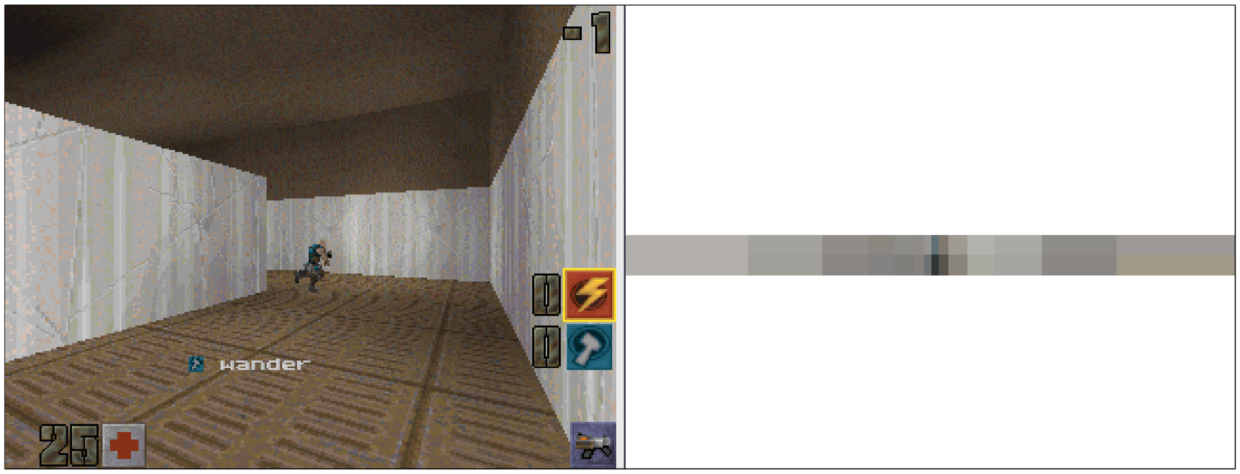

Fig. 6. An in-game screenshot of an agent looking at the enemy opponent in the map. The left side of the screen shows the Quake II game screen and the right side shows what the agent actually sees through its retina.

controller and the Lamarckian visual controller, as well as the 12 seconds of backpropagation for each individual. This totals to 2500 generations of evolution for the neuroevolution-only controller, at 24 seconds per individual.

Each individual chromosome is tested as follows:

1) The learning agent appears in the room and drops to the floor.
2) In the Lamarckian tests, the visual controller learns via backpropagation for 12 seconds.
3) The agent is given 24 seconds to kill as many enemy 'bots' as it can; kills are counted.
4) Whenever an enemy is killed, it promptly respawns at some random location in the room.
5) After the 24 seconds the current learning agent is removed and the fitness for its chromosome is reported.

We calculate the fitness for each individual solely based upon the number of kills achieved within the 24 second period. This number is modified by the equation $(5n)^2$, where n is the number of kills, to increase selection pressure. The maximum number of kills in the 24 seconds is about 12 due to the duration of the enemy's respawn time.

Rather than starting the enemy at full speed, we slowly increase the speed over the evolution. Whenever the average fitness of the population reaches a certain point, the speed increases by a small percent; this usually results in the fitness of the population decreasing again until it learns to hit the enemy at the new speed; then, the fitness once again reaches the speed increase level and the process continues until the enemy moves at full speed. This shaping of the difficulty allows the population to learn incrementally.

VII. Results

We tested 25 different populations for each of the two learning schemes. The Lamarckian controller's non-visual supervisor was also evolved 25 separate times, one for each Lamarckian population. The Lamarckian learning tests performed much better than did the neuroevolution-only tests. The fitness chart of the Lamarckian tests (figure 7) shows that the enemy's movement speed is up to 27% of fullspeed by the end of the test; comparatively, the enemy's movement speed is only at 10% in the neuroevolution-only test (figure 8). The average fitness of the neuroevolution tests gradually reaches just under 250. The average fitness of the Lamarckian test exceeds 250 very early in the tests and wavers around there for the remainder of the testing period.

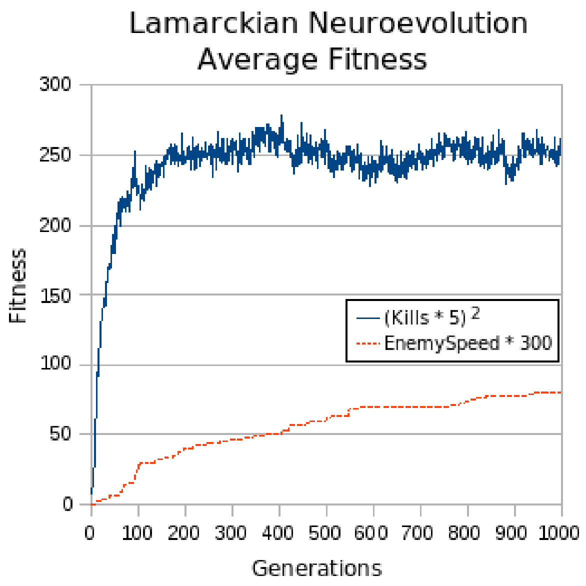

Fig. 7. Average of the average fitnesses of the 25 populations that used Lamarckian neuroevolution. The top dark line shows the fitness according to the number of kills, and the dashed line shows the enemy's speed, which increased whenever the fitness reached a certain point.

The Lamarckian controller not only achieved higher fitness, but its observable behavior is also superior and much different than the behavior of neuroevolution-only tests. Both controllers learned to walk around the pillar in one direction, which seemed to be optimal movement strategy because the enemy always dropped on the opposite side of whatever location the agent happened to occupy. The neuroevolution controller seems to have learned a "sprinkler" pattern for

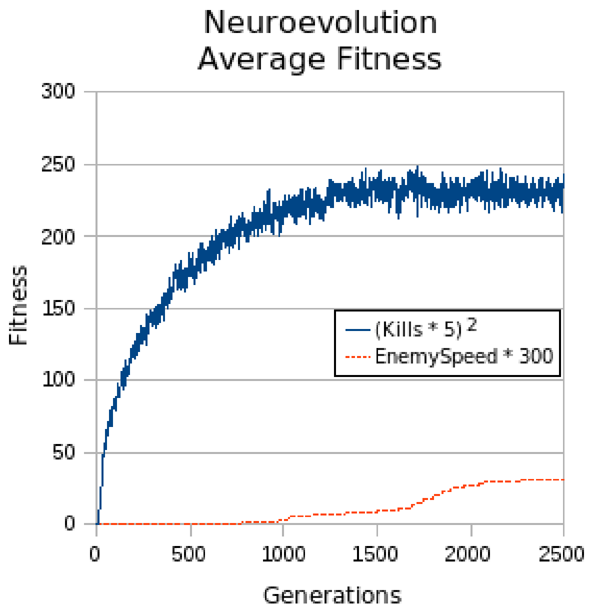

Fig. 8. Average of the average fitnesses of the 25 populations that used neuroevolution alone. The dark top line shows the fitness according to the number of kills, and the dashed line shows the enemy's speed, which increased whenever the average fitness reached a certain point.

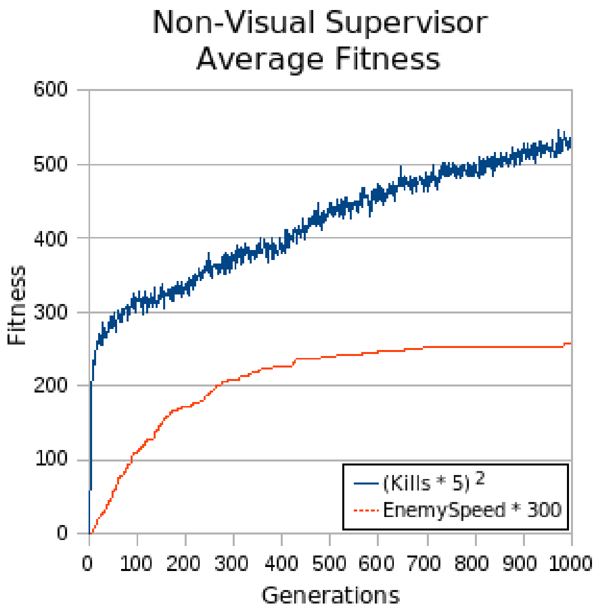

Fig. 9. Average of the average fitnesses of the 25 populations of evolved non-visual controllers. The dark top line shows the fitness according to the number of kills, and the dashed line shows the enemy's speed, which increased whenever the average fitness reached a certain point. The non-visual controller's fitness is nearly double that of either visual controllers (figures 8 and 7).

shooting at the enemy, by which the agent merely sprinkles the hallways in a pattern that is likely to hit any enemies that may be in it. The majority of the Lamarckian agents, however, learned to shoot bursts of blaster fire only when the enemy was in the aim. The non-visual supervising controllers easily learned that the best strategy is to save the shots until the enemy appears, then shoot a burst at the enemy, thereby increasing the chance that one of the shots will hit the enemy. These supervising agents pushed the visual Lamarckian controllers towards this strategy until they were able to do likewise. The neuroevolution-only tests, however, became stuck in a "sprinkler" strategy local optimum and had no supervisory controller to push them out.

Figure 9 shows the average fitness of the non-visual supervisory controllers. The non-visual controllers do much better than either of the visual controllers, and seem to continually be improving. Almost immediately the average fitness of the populations is higher than the highest average fitnesses of the visual tests. We arbitrarily chose to pick the best individual from the 1000th generation, though it appears that we could have picked one from a much earlier generation, since it still would be outperforming the visual controllers at anything above 250 fitness. However, since the best individual is sometimes just a lucky one in the earlier generations, choosing the best from a later generation provides a greater chance of it really being a good controller.

VIII. Conclusion

The research in this paper compared a visual controller trained by neuroevolution-only and a visual controller that used a combination of neuroevolution and backpropagation in a Lamarckian learning scheme. Each individual in the Lamarckian test was trained shortly with backpropagation before it was tested by the fitness function; the supervising input for the backpropagation came from an evolved neural network controller that used non-visual inputs, such as distance sensors and the exact enemy location, which was previously trained by neuroevolution. The controllers' task was to learn to shoot and kill an enemy as many times as possible in a 24 second period in a room with a large central pillar. Both the neuroevolution-only and the Lamarckian tests evolved controllers that learned to walk around the pillar, but they differed in that the Lamarckian tests learned to shoot bursts when the enemy was in view, while the neuroevolution-only tests seemed to ignore the enemy and shot in a random "sprinkler" pattern. The neuroevolution-only tests probably were stuck in a "sprinkler" local optimum, and were unable to learn more complex strategies because the problem was too difficult to learn from scratch with visual inputs. The Lamarckian tests, however, were heavily influenced through backpropagation by the strategy that the supervising controller learned, which it was able to learn from scratch because it used simpler non-visual input; these supervising controllers were able to guide the Lamarckian controllers out of any local optimums like the "sprinkler" strategy.

The most important aspect of this research is the idea of using controllers with high-level inputs to help train controllers that use lower-level inputs. This idea may be particularly useful in real-world robotics as well as in simulations. For example, suppose that someone needs a robot that can navigate through some hallways using only laser rangefinders for inputs. It may be too difficult to hand-code or to evolve a successful controller for such low-level inputs; instead, a controller that used some extra information such as GPS and an internal map could be hand-coded or trained to perform successfully, and that new controller could then be used to help train the controller that used the low-level laser rangefinders.

In this research we were fortunate that the evolved non-visual supervisory neural network learned a strategy that could transfer well to a visual controller. To make sure that the evolved supervisory controller could teach the visual controller we had to restrict the non-visual inputs to use only information that could be derived from the visual input. For this problem it was not too difficult to limit the supervisory inputs, but it may be very difficult to do so for other controllers that have more complex inputs. It would be helpful in the future to devise some system to automatically limit the supervisory controller so that it does not attempt to train the learning controller to do some impossible thing, given its limited set of inputs. Our work is now being directed to find such an automated solution to further remove requirements of human intuition from this Lamarckian style learning process.

IX. Acknowledgments

This work was supported in part by NSF EPSCoR grant EPS-0447416. Quake II is a registered trademark of Id Software, Inc., of Mesquite, Texas.

Our modifications to the Quake II source code, including a general-purpose API for client-based AI, is available from the Neuroevolution and Behavior Laboratory at the University of Nevada, Reno, http://nebl.cse.unr.edu/.

References

[1] D. Pomerleau, "Efficient Training of Artificial Neural Networks for Autonomous Navigation", *Neural Computation*, Vol. 3, No. 1, 1991, pp. 88-97.

[2] S. Baluja, "Evolution of an Artificial Neural Network Based Autonomous Land Vehicle Controller", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 26 No. 3, 450-463, June 1996.

[3] D. Floreano, T. Kato, D. Marocco, and E. Sauser, "Coevolution of active vision and feature selection", *Biological Cybernetics*, 90(3), 2004, pp. 218–228.

[4] S. Enrique, A. Watt, F. Policarpo, S. Maddock, "Using Synthetic Vision for Autonomous Non-Player Characters in Computer Games", *4th Argentine Symposium on Artificial Intelligence*, Santa Fe, Argentina, 2002.

[5] O. Renault, N. Magnenat-Thalmann, D. Thalmann, "A Vision-based Approach to Behavioural Animation", *Journal of Visualization and Computer Animation*, Vol.1, No1, 1990, pp.18-21.

[6] N. Kohl, K. Stanley, R. Miikkulainen, M. Samples, and R. Sherony, "Evolving a Real-World Vehicle Warning System", In *Proceedings of the Genetic and Evolutionary Computation Conference 2006*, pp. 1681-1688, July 2006.

[7] C. Bauckhage, C. Thurau, and G. Sagerer, "Learning Human-like Opponent Behavior for Interactive Computer Games", In B. Michaelis and G. Krell, editors, *Pattern Recognition*, volume 2781 of LNCS, pages 148-155. Springer-Verlag, 2003.

[8] S. Zanetti, A. El Rhalibi, "Machine Learning Techniques for First Person Shooter in Quake3", *International Conference on Advances in Computer Entertainment Technology* ACE2004, 3-5 June 2004, Singapore.

[9] C. Thurau, C. Bauckhage, and G. Sagerer, "Learning Human-Like Movement Behavior for Computer Games", In *Proc. Int. Conf. on the Simulation of Adaptive Behavior*, pages 315-323. MIT Press, 2004.

[10] R. Graham, H. McCabe, and S. Sheridan, "Neural Pathways for Real Time Dynamic Computer Games", *Proceedings of the Sixth Eurographics Ireland Chapter Workshop, ITB June 2005, Eurographics Ireland Workshop Series*, Volume 4 ISSN 1649-1807, ps.13-16

[11] "Q2 LNX stuff," Nov 14, 2005. http://icculus.org/quake2/

[12] J. L. Elman, "Finding structure in time", *Cognitive Science*, 14:179–211, 1990.

[13] M. Parker, and B. Bryant, "Neuro-visual Control in the Quake II Game Engine", *Proceedings of the 2008 International Joint Conference on Neural Networks* (IJCNN 2008), Hong Kong, June 2008.

[14] M. Parker, and B. Bryant, "Visual Control in Quake II with a Cyclic Controller", *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games* (CIG 2008), Perth, Australia, December 2008.

[15] M. Parker, and B. Bryant, "Lamarckian Neuroevolution for Visual Control in the Quake II Environment", *Proceedings of the 2009 International Conference on Evolutionary Computation* (CEC 2009), Trondheim, Norway, May, 2009.

[16] J.-B. Lamarck, *Pilosophi Zoologique*, 1809.

[17] J. Grefenstette, "Lamarckian Learning in Multi-Agent Environments", *Proceedings of the Fourth International Conference on Genetic Algorithms*, 303-310, San Mateo, CA, 1991.

[18] D. Whitley, S. Dominic, R. Das, and C.W. Anderson, "Genetic Reinforcement Learning for Neurocontrol Problems", *Machine Learning*, 13:259-284, 1993.

[19] K. Ku, M. Mak, and W. Sui, "A Study of the Lamarckian Evolution of Recurrent Neural Networks", *IEEE Transactions on Evolutionary Computation*, 4:31-42, 2000.

[20] D. Rumelhart, G. Hinton, and R. Williams, "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*', Cambridge, MA: MIT Press, 318-362, 1986.

[21] B. Bryant, and R. Miikkulainen, "Acquiring Visibly Intelligent Behavior with Example-Guided Neuroevolution", *Proceedings of the Twenty-Second National Conference on Artificial Intelligence* (AAAI-07), pp. 801-808. Menlo Park, CA: AAAI Press.

[22] M. Parker, and G. Parker, "Using a Queue Genetic Algorithm to Evolve Xpilot Control Strategies on a Distributed System", *Proceedings of the 2006 IEEE Congress on Evolutionary Computation* (CEC 2006), Vancouver, BC, Canada, July 2006.