Lamarckian Neuroevolution for Visual Control in the Quake II Environment

Matt Parker and Bobby D. Bryant

Abstract— **A combination of backpropagation and neuroevolution is used to train a neural network visual controller for agents in the Quake II environment. The agents must learn to shoot an enemy opponent in a semi-visually complex environment using only raw visual inputs. A comparison is made between using normal neuroevolution and using neuroevolution combined with backpropagation for Lamarckian adaptation. The supervised backpropagation imitates a hand-coded controller that uses non-visual inputs. Results show that using backpropagation in combination with neuroevolution trains the visual neural network controller much faster and more successfully.**

I. Introduction

SIMPLE two-dimensional image data is sufficient for humans to accomplish a large variety of complex tasks. Doctors perform intricate surgery remotely using a camera image. A camera mounted on a vehicle is sufficient input for a human to accurately control the vehicle. There is a wide variety of interactive simulated worlds in computer games that use the two-dimensional screen as the main informational output. Humans can effectively use this visual data because their complicated brains are designed to quickly interpret raw visual data into usable information. Computer algorithms that are designed to interpret raw visual data are generally also complex and computationally intensive due to the large amount of data that must be processed. Processing raw visual data in real-time is particularly difficult because each frame must be processed in a small amount of time.

Some previous research has used neural networks with raw visual input as controllers in a real-time environment. In research by Pomerleau, a neural network was trained with backpropagation to drive a car along a road, using a 30x32 grayscale input [1]. The training was performed in real-time to imitate a human driver. In other research by Baluja, the experiment was modified to replace backpropagation with an evolutionary computation model that allowed for more robust control, but could only be trained with recorded sets of driving data rather than in real-time [2]. Floreano et al. trained a car controller to use active vision to race around a track in a realistic driving simulator [3]. Another virtual car-racing controller was trained by Kohl et al. that used NeuroEvolution of Augmenting Topologies and a 20x14 grayscale input to evolve a vehicle warning system [6]. Furthermore, they used this technique on an actual robot with a mounted camera. Neural networks were shown through these experiments to be viable controllers for real-time control using raw visual input.

Synthetic vision, which is a method of representing extra-visual data within the visual field, is used by many research experiments that take place in a virtual world [4][5]. This method might change colors of objects to reflect their distance or angle or to show their specific object type. Enrique et al. re-rendered the visual input into two separate color-coded views, one of which displayed the color of an object according to its velocity and one which displayed color according to entity identification and wall-angle. This high-level visual data was used to simplify creation of a hand-coded robot controller that could navigate a map and pick up health boxes [4]. In other research by Renault et al, an agent used extra-visual pixel data and a hand-coded controller to walk down a hallway. Each pixel included a distance value, an object identification value, and a color value [5]. The extra-visual information used in synthetic vision can be accessed because they are using a virtual world, where such information is easily available; because it relies on this extra-visual information, synthetic vision 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 gun-wielding character who must survive in a hostile world. The player controls the game character from the viewpoint of the character's eyes and generally always views down the barrel of a gun. FPS's often natively support some form of programmable AI and are often used for research. Bauckhage et al. conducted research that first recorded demos of humans playing the game Quake II, then taught two neural networks to imitate them. One network learned to aim during combat situations by changing the yaw and pitch; another network learned to change its x and y velocities during combat, and another network learned to traverse the map [7]. Zanetti and El Rhalibi trained neural networks with backpropagation for Quake III to control an agent to collect items, engage in combat, and navigate a map. They also used pre-recorded demos of human players, using the player's weapon information and location as inputs [8]. Graham et al. trained controllers using neuroevolution for path-finding strategies and obstacle avoidance in the game Quake II [11]. Thurau et al. used a Neural Gas method to train agents in Quake II to imitate the waypoint-navigation of real players by observing their movements from pre-recorded demos [9]. Behavioral grids were evolved in Quake III in research done by Priesterjahn et al., and the resulting agents were able to

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.

amply defeat the hand-coded AI robots supplied with the game [10].

In our previous research we trained vision based controllers for agents in Quake II. First, we trained a simple recurrent network [13] by neuroevolution to control an agent to shoot a moving enemy in a visually-simple room [14]. 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 much quicker to shoot the enemy because the enemy could easier be seen in the higher-resolution center of the retina. In further research using Quake II, we designed a new controller in which the genetic chromosome represented a control program which included jump and action instructions [15]. The jump instructions determined whether or not to jump forward in the chromosome, and how far to jump, according to the grayscale value of some block of pixels on the screen, the size and location of which were specified in the instruction. The action instructions specified the behavior of the agent for that particular frame of gameplay. With these two instruction types the evolution produced control programs for agents that would effectively attack the enemy. We found that this controller learned much faster than our previous neural network controller, and used less computation.

In the research presented in this paper, we again train a neural network controller to shoot an enemy in a room but we have drastically increased the difficulty of the problem by adding uneven shadows to the room, as well as a darker floor. This added visual complexity hinders learning to the point that our previously-used graduated density controller [14] is unable to learn satisfactory behavior when trained only with neuroevolution. Instead, we combine neuroevolution with backpropagation to help train the network and are able to achieve the desired behavior. Our process is similar to Lamarckian evolution, an idea proposed by Jean-Baptiste Lamarck in the early nineteenth century which hypothesized that phenotype adaptations learned within an individual's lifetime were passed onto its offspring [16]. This type of evolution can easily be added to neuroevolution by changing the weights of the neural network over an individual's lifetime and returning the modified weight structure to the evolutionary gene-pool [17][18][19]; the Lamarckian adaptations can be made by using backpropagation [20]. Lamarckian neuroevolution that uses backpropagation has previously been used in research by Bryant and Mikkulainen, which trained a controller to play the strategy game Legion II by imitating pre-recorded human players [21]. Instead of backpropagating on pre-recorded human players we use a hand-coded bot that does not access any visual input, but rather directly accesses world entity information. The neural network is corrected against this hand-coded bot and learns to imitate it using only visual inputs.

II. The Quake II Environment

The platform used in this research is the Quake II game engine by Id Software, Inc. Quake II is an FPS that requires a player to take control of a space marine who is stranded on a hostile alien planet. The space marine must fight his way out of the planet, using a wide variety of weapons and powerups to kill many aliens along the way. The graphics in Quake II realistically present a dim and dreary environment. Id Software has provided many tools to customize the game by changing maps, models, sounds, textures, and even game rules.

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

Quake II was chosen for use in this research for several important reasons. Most importantly, the code of the game engine is open source, licensed under the General Public License (GPL), and can run on UNIX-like operating systems [12]. Because code of the engine is available, we are able to modify the engine to easily access game environment variables and visual screen data, as well as to directly control the agent. Many open source FPS's are currently available, but most of them use OpenGL graphics acceleration and thus require a devoted graphics card. Quake II, however, is able to render in software-only mode, which allows us to run multiple copies on one machine. This is essential for our distributed evolution scheme, which can run hundreds of Quake II simulations in parallel across a network of computers to speed up evolution.

Quake II natively includes a method for creating pre-programmed opponents, which are called *bots*. These bots usually run on the game server and cannot access the screen buffer because the server does not render the game world; it only calculates physics and implements game rules. A Quake II client receives game-world information from the server and renders the view of the world to the video buffer. We modified the Quake II client to allow us to control the player's behavior and to access game data and pixel color values from the rendered video buffer.

III. The Experiment

In this experiment we use the same game-world setup as our previous experiments [14][15], excepting that we use a more visually complex map. In our previous research the map had a white floor, gray walls, a dark ceiling, and a dark enemy; moreover, the map was fully lit with no shadows (figure 2). In our current research, the floor and ceiling of the map are brown, the walls are gray, the enemy is dark, and the room is lit dimly with varying shadows (figure 3). This map is much more realistic when compared to real-world rooms, and is also more difficult for good visual controller performance. The map is a single open room that is about the size of half a basketball court.

In multiplayer mode, players and bots enter or re-enter the map through *spawn portals*. If an agent spawns in the exact location of another player already in the map, then the living player is killed by the entering player. To prevent these accidental spawning deaths, we placed the spawn portals near the ceiling of the map so that the spawning players fall to the floor without killing any living players.

In this experiment the learning agent's task is to kill a hard-coded enemy opponent. The enemy opponent moves around the room in a random pattern and never shoots. The random pattern is created by performing randomly chosen forward/back, right/left, and turn speeds for random periods of time. This scheme produces random movement that is reminiscent of the movement of a human player.

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

The learning agent is equipped with a blaster handgun that fires energized plasma bolts. If one plasma bolt hits the enemy then the enemy will die instantly. The blaster shots travel at a non-infinite speed and therefore proper aiming often requires leading the target. In order to encourage shooting only when necessary, we changed the blaster to use a burst mode. The blaster has a reservoir of 25 energy units; whenever the blaster is fired, the energy units are depleted by 5. The blaster recharges after some time, so that by not shooting the agent can save up shots to fire a burst.

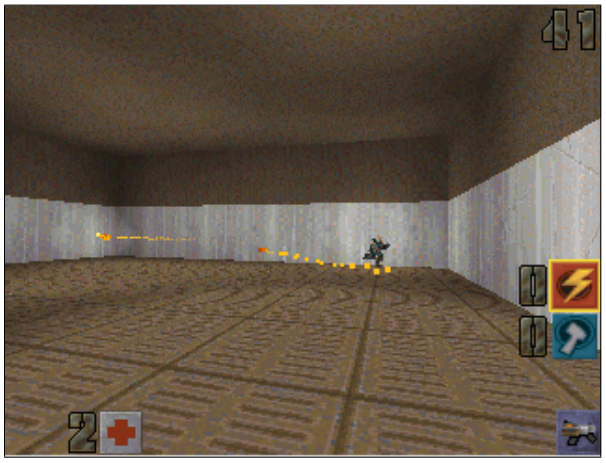

Fig. 3. An in-game screenshot of the environment used in this experiment. The floor and ceilings are brown, the walls are gray, and the enemy is dark blue. The room is dimly lit with varying shadows. The display of the shooter's own weapon has also been removed. (Cf. figure 1.)

IV. The Neuro-Visual Controller

The neuro-visual controller consists of a neural network and a set of visual inputs. The visual inputs are taken from the game screen as blocks of pixels whose color values are the average color values of the grayscale pixels within their described area. There is a 14x2 grid of blocks that make up the visual retina, which is similar to the graduated density retina used in previous research: the blocks near to the center of the screen are thinner than those of the periphery; each block is approximately 1.618 times larger than the previous block. The two rows of blocks are each 4.2% of the height of the screen and are positioned slightly lower than the center of the screen. The small height and positioning of the retina concentrates the visual data on the position of the enemy.

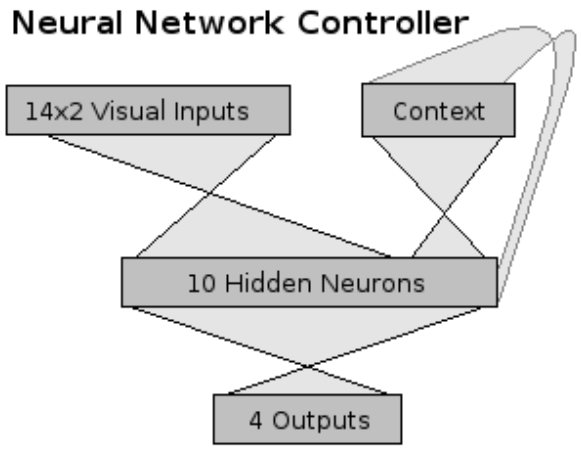

Fig. 4. The neural network control is a standard recurrent network with a hidden/context layer.

The neural network used in the neuro-visual controller is a simple recurrent network [13] with 28 inputs, 10 hidden/context units, and 4 output neurons (figure 4). A bias

value of 1.0 is input to the hidden layer and to the output layer. The weights of the network are floating point values that begin in the range of $[-2.0, 2.0]$, but exceed that range because of the mutation as the evolution continues. The 28 inputs are from the retinal blocks and are grayscale values in the range $[0, 1]$. After the inputs to any neuron are multiplied by their corresponding weights, they are summed and their values are squashed using the $tanh$ function. The 4 outputs of the neural network control the behavior of the agent; one output controls the speed of turning left or right, one controls left or right movement speed, one controls forward or back movement speed, and one controls whether or not to shoot. The movement and turning controls are real values and can be variable speed, but the output to control shooting is a binary value that is true if positive and false if negative.

V. Hand-Coded Bot for Backpropagation

The backpropagation of the neural network used for Lamarckian adaptation needs to imitate some supervising behavior. We could have attempted to imitate a human player, but humans are very inconsistent in their actions so learning would be difficult. Instead, we hand-coded a controller that performs the desired behavior; rather than using visual input like the neural network, it directly accesses the location coordinates of the enemy as relative to the agent and outputs the actions that the agent should do for every frame. The goal of this experiment is to have a controller that uses only visual input, but we can let the hand-coded controller "cheat" and access the direct coordinates because the neural network will learn to imitate this behavior using only the visual input.

In order to hand-code a controller that can be used with backpropagation for a neurovisual network, one must consider the capabilities of the visual field. The hand-coded bot must be designed to react only to situations that are clearly visible in the visual field; if it does react to non-visible elements, then the visual controller will become confused because duplicate visual situations will require different control outputs. For instance, if our Quake II bot always turned directly toward the enemy, even if the enemy is not currently on screen, then the visual controller will be trained to turn both left and right when it sees no enemy on screen, because sometimes the enemy will be offscreen and closer to the left, and other times offscreen and closer to the right. Because of this, our hand-coded bot assumes that there is no enemy until the enemy is somewhat centered in the view of the agent; only then does it follow its aim directly towards the enemy. When the enemy is not near the center, or not even on screen, then the bot always spins to the left, as if searching for the enemy. When the enemy is centered, then the hand-coded bot shoots at the enemy, vibrating its turning slightly to scatter the shots. When the enemy is killed it begins its death animation, and by observing the enemy's animation frames we can tell when the enemy has fallen out of view of the visual controller's retina and then proceed to search for the enemy again by turning left. Our controller performs no action inconsistent with the visual field, which is essential to successful backpropagation of the visual controller.

Like the neural network, the hand-coded controller outputs 4 values, corresponding to turning, moving left or right, moving forward or back, and shooting or not shooting. For the real valued movement, we can simply output the desired movement values. For the binary shooting output, which will shoot if positive and not if negative, we output the most extreme values of -1.0 and 1.0, which seems to work well for training.

VI. Training

To evolve the weights of the neural network, 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 store the population of chromosomes; the oldest individuals are at the tail of the queue and the youngest are at the head. Whenever a new individual chromosome is needed, two parents are taken from the queue using stochastic roulette wheel selection, according to fitness; the parents are combined into a new individual using crossover. After any new individual is finished being tested and is assigned a fitness, the individual is returned to the QGA and placed at the head of the queue, and the oldest individual is dropped off of the queue's tail. This technique provides evolution very similar to a regular genetic algorithm, and allows for easy distribution of the evolution. We can run a central QGA server and run several QGA clients on several different computers over the network, each running multiple fitness evaluation tests. Because we run Quake II in real-time at normal game speed, distributing the evolution is essential to completing the tests in acceptable time.

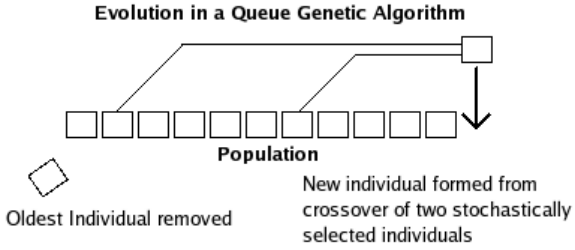

Fig. 5. The Queue Genetic Algorithm (QGA). New individuals are bred from parents chosen from the current population by roulette wheel selection according to fitness. After each new individual is evaluated it is enqueued and the oldest individual is dequeued and discarded.

For this experiment we used a population size (queue length) of 128 chromosomes, each consisting of 434 genes, 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. The mutations are calculated by drawing a delta from a sharply peaked random distribution, $\frac{log(n)}{10} * random(-1 or 1)$, where n is a random number in the range $[0, 1]$. This function has a high probability of generating small deltas and a low probability of generating high deltas.

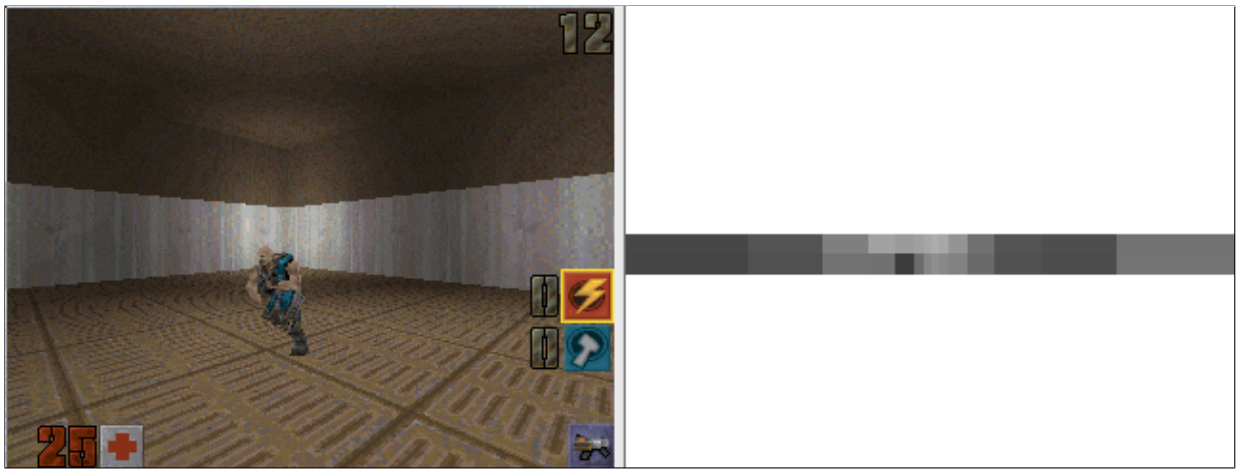

Fig. 6. An in-game screenshot of an agent looking at the enemy opponent in the map. The left side of the screen shows the Quake II game screen and the right side shows what the agent actually sees through its retina.

In one test we use only neuroevolution for the learning; in the other, we use Lamarkian neuroevolution with backpropagation for adaptation. The setup of the two tests are exactly the same, except that each individual in the latter test is given 12 seconds to backpropagate against the hand-coded controller before their fitnesses are tested. During these twelve seconds the hand-coded controller completely controls the agent. The 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 hand-coded control. The 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 returned to the QGA.

Each individual chromosome is tested according to this process:

1) The learning agent appears in the room and drops to the floor.
2) If the agent is using backpropagation, it imitates the hand-coded controller 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.

The fitness is according to the number of kills achieved in the 24 second testing period; in order to increase selection pressure, we use $(5n)^2$, where n is the number of kills. Because there is some time delay between the death of the enemy and its reappearance, the maximum number of kills possible in the 24 second period is about 12.

In this research as well as in our previous research we shape the enemy's speed. At the beginning of the evolution, the enemy stands completely still; after the average fitness of the population reaches a certain level, the speed increases. This shaping allows the controllers to incrementally learn to attack a fast moving opponent, aiding their ability to learn.

VII. Results

We tested the two learning schemes for 500 generations each. The tests that learned using backpropagation for Lamarckian neuroevolution were much more successful than the test that used only neuroevolution. Figure 9 shows the average fitness of the six populations that used backpropagation mixed with neuroevolution. In this graph we see that the fitness jumped up quickly and by the 75th generation the fitness was high enough that the enemy's speed began to be incrementally raised. By the end, the average enemy speed of the six tests is a little over half of the full speed capable by the enemy. Comparatively, figure 10 shows the average fitness for the six populations that used only neuroevolution. We see that the fitness does not reach nearly as high as in the previous test, and only by the end of the evolution does the movement of the enemy begin to increase. In five of the six neuroevolution-only tests the average fitnesses of the populations never reached high enough to increase the speed of the enemy past zero, whereas in all six of the backpropagation and neuroevolution combination tests the enemy's speed increased past zero. We can more clearly see the disparity of success between the two tests by looking at a graph that combines the average fitness of the populations with the average enemy movement speed: figure 9 shows the combined graph for the Lamarckian neuroevolution combination test, and figure 10 shows the combined graph for the neuroevolution-only test.

The dim lighting and varying shadows makes this problem extremely difficult for our simple retina controller. The 14x2 retina does not appear to provide adequate information for distinguishing the darkness of the enemy from the darkness of the shadows. Observance of the agents developed by the neuroevolution-only controller shows that the agents converge prematurely on a "sprinkler" solution: they turn around the room sprinkling it in a pattern that is somewhat likely to hit the enemy; the enemy itself is ignored and

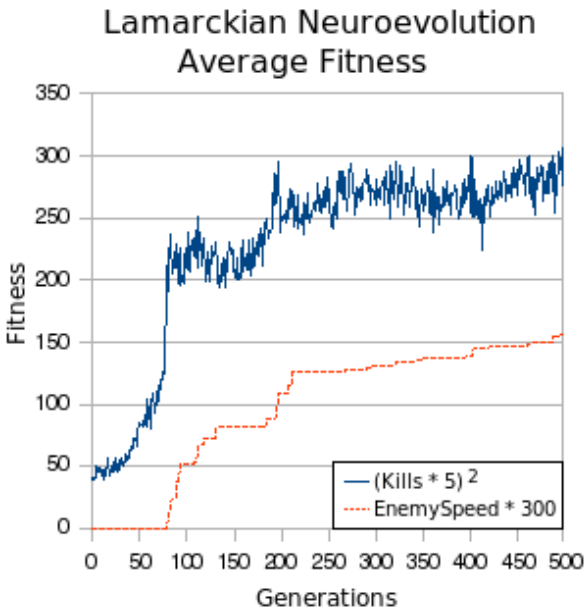

Fig. 7. Average of the average fitnesses of the six populations that used Lamarckian neuroevolution. The top dark line shows the fitness according to the number of kills, and the dashed line shows the enemy's speed, which increased whenever the fitness reached a certain point.

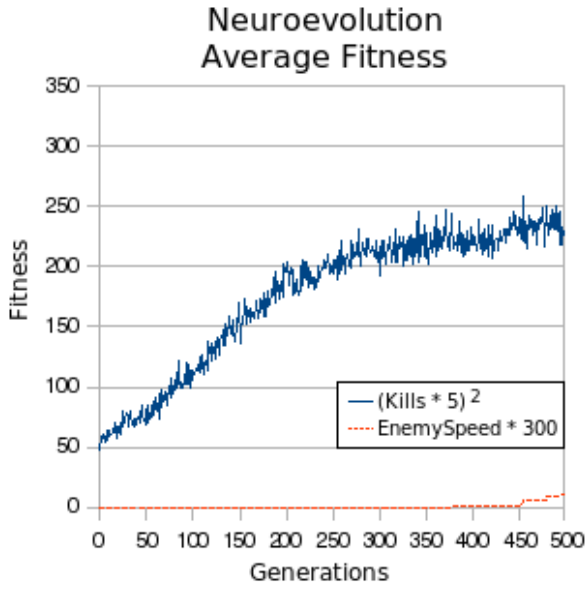

Fig. 8. Average of the average fitnesses of the six populations that used neuroevolution alone. The dark top line shows the fitness according to the number of kills, and the dashed line shows the enemy's speed, which increased whenever the average fitness reached a certain point.

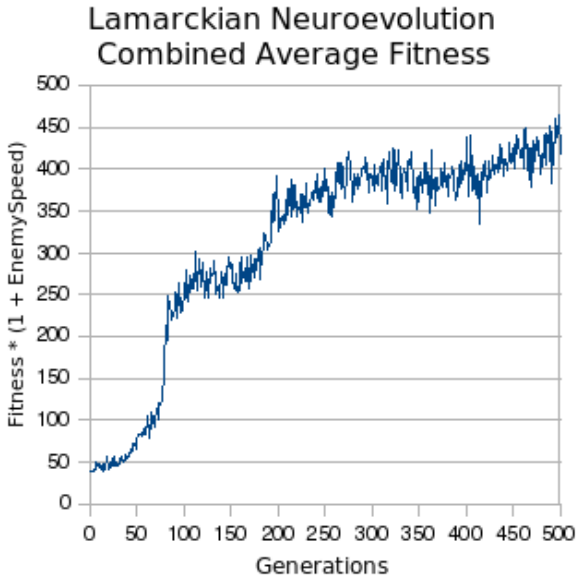

Fig. 9. This combination graph for the test using Lamarckian neuroevolution gives a clear representation of its success, which incorporates the average speed of the enemy with the average fitnesses of the test.

no special behavior is presented when the enemy appears in view. The test that used backpropagation as a learning aid, however, could not get stuck in this premature local optimum because the agents were always being influenced to imitate the hand-coded controller. After some generations of backpropagation and evolution, the controllers began to learn to react to the enemy's appearance. By the later generations, the agents behaved very similar to the hand-coded bot: they shot bursts of blaster fire at the enemy and would even track the enemy as it moved; they did not fire their blaster unless the enemy was in view, except at an occasional shadow. The fitnesses of the best individuals from the lamarckian controllers were comparable to the best fitnesses of the hand-coded bot, ranging between 7 and 9 kills per turn; the average fitness of the visual controller, however, was much lower at about 3.5 kills per turn, whereas the average fitness of the hand-coded controller was about 7 kills per turn. Since luck can play a large role in earning the "best" fitness, the average fitnesses are a more accurate representation of the abilities of the lamarckian visual versus hand-coded controllers.

One disadvantage of adding backpropagation to neuroevolution is that it requires extra time for the backpropagation training. In this experiment it added an extra 12 seconds of training to every individual; they each took a total of 36 seconds instead of 24 seconds, a 150% time increase. In the time that it would take for the neuroevolution-only test to finish its 500th generation, the test using backpropagation would only be at its 333rd generation. We can see by comparing the graphs that the fitness of the Lamarckian test is still much better at the 333rd generation than the neuroevolution-only test at the 500th generation.

Another disadvantage of the backpropagation is that the population will only become as successful as the hand-coded controller which it is imitating. Because the controller is designed by hand and is not using the visual field, it may not be an optimal controller for supervision. This problem can likely be solved by decreasing the effect or time of backpropagation as the evolution proceeds. For example, in the beginning longer periods of backpropagation could be used to quickly boost the behavioral fitness; then, as the fitness increases, the time used doing backpropagation could slowly drop down until sometime when there is no backpropagation, and the evolution can continue to perfect

the controller for visual control without any influence from the hand-coded controller.

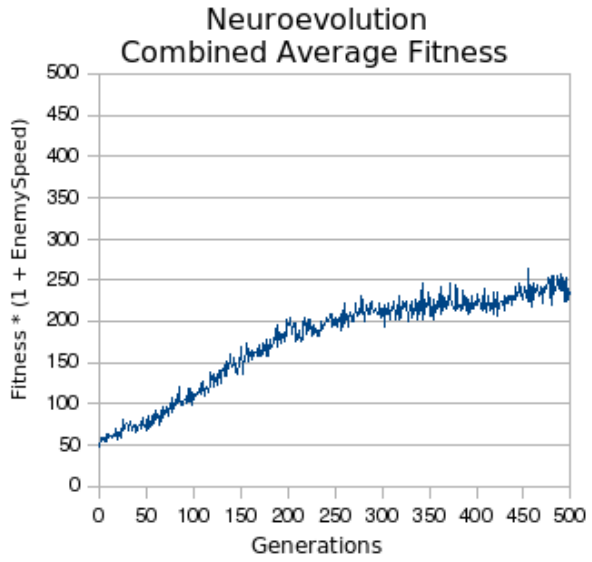

Fig. 10. This combination graph for the neuroevolution-only test combines the average fitness of the test with the enemy movement speed. By comparing this graph with Figure 9, it is clearly seen that neuroevolution-only was much less successful.

Using backpropagation by itself was an option for this research, but distributing backpropagation across multiple computers on a network is a difficult problem. If we were to match the running computational time of a population from the the neuroevolution-only test, running only a single backpropagating client, it would require over 1066 days of computation. There are some techniques to distribute backpropagation, such as was done by Chen et al. [23], but the techniques are complicated and do not exactly model pure backpropagation. Using the distribution of the QGA to evolve the controllers in combination proved to be an easy and fast solution that could utilize our large number of computers, and that also had the advantage of the incorporated evolutionary learning.

VIII. Conclusion

In this research we compared two learning methods: one used neuroevolution only and one used neuroevolution with backpropagation for Lamarckian adaptation. The controller used by both tests was a neural network with a visual retina input. The visual retina consisted of a 14x2 grid of blocks; each block consisted of the average grayscale values of a number of pixels. The blocks in the retina were thinner in the center and wider near the periphery so that the agent could see "clearer" near the center of the screen. The neural network was a standard recurrent network with a hidden/context layer. The network had four outputs: move forward/back, move right/left, turn right/left, and shoot. The neuroevolution-only test evolved the weights of the neuro-visual controller over 500 generations. The Lamarckian neuroevolution test evolved the weights of the network for 500 generations and also used backpropagation on every individual to persuade them to imitate a hand-coded controller. The hand-coded controller was programmed using non-visual game data, such as the enemy's exact location, and controlled the agent to shoot at the enemy. During the backpropagation, the hand-coded controller controlled the agent while neural network controller learned to imitate it.

The task learned was to shoot a moving enemy in a visually complex environment. The environment was a square room with dim lighting and varying shadows. The enemy's speed increased throughout the evolution, after the average fitness of a population reached a certain value. Fitness was awarded according to the number of kills during a 24 second test period. Results showed that using Lamarckian neuroevolution was much more successful than using neuroevolution alone. The neuroevolution-only tests learned a premature solution that ignored the appearance of the enemy and merely sprinkled bullets around the room in a semi-deadly pattern. The tests that used Lamarckian neuroevolution learned to imitate the hand-coded controller using only the visual inputs, and their fitnesses were accordingly more successful.

This research shows that backpropagation can be used with a hand-coded non-visual controller and mixed with neuroevolution to learn visual-only control in a visually complex environment. Moreover, this research emphasizes the general idea that an effective controller that uses high-level inputs can be used with backpropagation to teach a controller that uses lower-level inputs. This could also be useful to train non-visual agents, such as a robot that learns to use its infrared distance sensors to navigate by imitating a higher-level hand-coded controller which uses some higher-level triangulation of WiFi signals to determine its location; in the final product the controller might not be able to use WiFi to pinpoint its location, but to train the lower-level sensors it can "cheat" and use WiFi triangulation. In future research in visual control we will increase the complexity of the retina and of the environment, and learn more complex behaviors. We also will try to add color to the retina, rather than grayscale only. We also will eliminate the need to hand-code a controller by evolving, rather than coding, a non-visual supervising controller, thereby completely eliminating human supervision beyond a fitness function in the learning.

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] S. Priesterjahn, O. Kramer, A. Weimer, A. Goebels, "Evolution of Human-Competitive Agents in Modern Computer Games", *Proceedings of the 2006 IEEE Congress on Evolutionary Computation* (CEC 2006), Vancouver, BC, Canada, July 2006.
[11] 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
[12] "Q2 LNX stuff," Nov 14, 2005. http://icculus.org/quake2/
[13] J. L. Elman, "Finding structure in time", *Cognitive Science*, 14:179–211, 1990.
[14] 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.
[15] 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.
[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.
[23] Q. Chen, Y. Lai, and J. Han, "A Implementation for Distributed Backpropagation Using Corba Architecture", *Proceedings of the 2006 5th International Conference on Cognitive Informatics* (ICCI 2006), Beijing, China, July 2006.