# Neurovisual Control in the *Quake II* Environment

Matt Parker and Bobby D. Bryant, *Member, IEEE*

***Abstract*—A wide variety of tasks may be performed by humans using only visual data as input. Creating artificial intelligence that adequately uses visual data allows controllers to use single cameras for input and to interact with computer games by merely reading the screen render. In this research, we use the *Quake II* game environment to compare various techniques that train neural network (NN) controllers to perform a variety of behaviors using only raw visual input. First, it is found that a humanlike retina, which has greater acuity in the center and less in the periphery, is more useful than a uniform acuity retina, both having the same number of inputs and interfaced to the same NN structure, when learning to attack a moving opponent in a visually simple room. Next, we use the same humanlike retina and NN in a more visually complex room, but, finding it is unable to learn successfully, we use a Lamarckian learning algorithm with a nonvisual hand-coded controller as a supervisor to help train the visual controller via backpropagation. Last, we replace the hand-coded supervising nonvisual controller with an evolved nonvisual NN controller, eliminating the human aspect from the supervision, and it solves a problem for which a solution was not previously known.**

***Index Terms*— Artificial intelligence, computational intelligence, computer vision, evolutionary computation, neural networks.**

## I. Introduction

HUMANS with normal vision use their visual system to help them live and survive in the world. Even if all other senses, such as taste, touch, hearing, and smell, are removed, a human can still accomplish a large variety of tasks. For example, a human can remotely control a military drone using only a streaming 2-D camera image taken from the nose of the airplane; likewise, cars or other vehicles can be controlled with just visual input. Doctors are able to operate remotely on patients using a robotic scalpel system and a camera image. Computer games usually only require that a player can view the graphical data presented on a 2-D screen, yet there are computer games and simulations for almost every interesting human action, all of which can be performed by using the visual screen to collect the game information.

There are many possible real-time applications for artificial intelligence (AI), which use visual data for input. Rather than using expensive global positioning system (GPS), laser range finding, and sonar equipment, vehicles could be controlled entirely or in part by an AI looking through a single camera [1], [2]. AI agents in computer games could also be created that would see exactly what a human would see, so that the AI could not cheat by directly reading the game environment information. It is our goal in this research to improve and further realize the capabilities of AI for real-time visual control.

### A. Overview

In this research, we create AI controllers that are able to play a computer game using visual data from the game screen as input. Computer games are good testbeds for AI controllers because they provide cheap, robust, and well-tested simulations that are generally nondeterministic in nature [3], [4]. They often will allow for game customizability by changing models, maps, and game rules. Moreover, AI controllers used in games can easily provide a direct comparison to a human's ability to play the same game; often, as in the case of multiplayer games, by allowing humans to directly compete against the AI. For this research, we use the first-person shooter (FPS) computer game *Quake II* and train our AI controllers to hunt and kill an opponent using only visual data from the rendered game screen as input.

Imitating the abilities of the human visual system is difficult because it is very complex and processes visual data rapidly [5], [6]. The visual cortex's computational process is massively parallel [7], but an individual computer processor typically computes serially. Writing algorithms to decode visual data is difficult because they must attempt to do sequentially what the human cortex does seemingly instantaneously and in parallel; solving such problems sequentially is hard and computationally intensive [8]. Algorithms such as artificial neural networks (NNs) are designed to imitate in part the computational processes of the human brain. Their computational abilities for certain visual tasks have been shown to be much faster than conventional techniques [9], [10]. NNs are also able to take advantage of multiple processors, spreading out their computations in parallel much like the human visual system, and can even compute using the hundreds of cores in graphics processing units [11]. Our simulator *Quake II* runs in real time at 40 frames/s, so our visual computation algorithm must be fast in order to keep up with the gameplay. Because of our speed requirements and the proven abilities of NNs to process visual data quickly, we use them for the visual controllers in this research.

The research reported here is divided into three main experiments. First, we compare two different retinal layouts, one which loosely imitates a human retina in that it has greater acuity in the center of the retina and less to the sides, and another which has uniform acuity across the entire span of the retina [12]. We train NNs attached to each type of retina with neuroevolution, measuring fitness by the ability to control an agent

Manuscript received May 14, 2010; revised November 14, 2010, June 23, 2011, and November 01, 2011; accepted November 07, 2011. Date of publication January 12, 2012; date of current version March 13, 2012. This work was supported in part by the National Science Foundation (NSF) under EPSCoR Grant EPS-0447416.

The authors are with the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557-0208 USA (e-mail: mparker@cse.unr.edu; bdbryant@cse.unr.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2012.2184109

to shoot a randomly moving enemy in a visually simple room. In our second experiment, we increase the visual complexity of the room, adding darker textures and varying shadows. We find that the controller from the retina experiment is unable to learn a satisfactory solution when trained only with a genetic algorithm (GA). We introduce a Lamarckian learning system that, in conjunction with the normal evolution, trains the NNs via backpropagation using a hand-coded nonvisual supervising controller [13]. The new Lamarckian system is able to successfully train the same controller that failed when trained with only neuroevolution. In our third experiment, realizing that it may not be plausible to always hand code a nonvisual supervising controller, especially in situations where a good solution is not previously known, we evolve a supervising NN using nonvisual inputs, thus completely eliminating the human *a priori* knowledge from the supervising controller of the Lamarckian learning [14]. We test this system on a new problem for which we did not previously know a solution, involving a visually complex room with a large central pillar about which the agent must traverse to kill the enemy. In this paper, we have reconducted the first two experiments, quadrupling the number of tests to increase statistical significance, and have allowed the evolution to run twice as long, both of which help to solidify our findings. We also had some inconsistencies between our original three experiments, such as the retina's height and position, which we have unified for this report.

## II. Background

Some previous research has been done using NNs for real-time control using raw visual input. The most famous is Dean Pomerleau's Autonomous Land Vehicle in a Neural Network (ALVINN) driving system, which trained an NN to drive a van along a road using a $30 \times 32$ grayscale visual input taken from a camera [15]. He trained the NN in real time by backpropagation, with a human driver as supervisor. Baluja modified the experiment by training the weights of the NN using an evolutionary computational model [16]. This new method evolved controllers with more robust control, but had the drawback that it could only learn offline, using recordings of human drivers. Floreano *et al.*, instead of using a real car, trained a virtual car to race around a track in a simulator using an active vision system, which had a small moving retina that could zoom in and out anywhere in the visual data, as controlled by an NN. They were able to use this same system to train a real robot to move forward as much as possible while avoiding obstacles in a cluttered office [17]. Kohl *et al.* used a car racing simulator as well, but used a static $20 \times 14$ grayscale retinal input instead of active vision; they evolved a vehicle warning system which can warn a driver whenever there is danger of collision. They also used this technique on an actual robot with a mounted camera, and it was able to warn the robot of impending collisions [18]. These experiments show that NNs are quite capable of using raw visual data as input to control vehicles in simulators and in actual real-world environments. Our current work differs from these other neurovisual experiments mainly in that the controller must play an FPS, which is arguably a more complicated task than driving around a track or avoiding obstacles.

An FPS is a type of computer game in which the player takes control of a (usually) gun-wielding character who, for some reason or another, must kill many other characters in the game while maneuvering about in the environment. They are called "first person" because the game display is rendered from the perspective of the in-game character. FPSs are quite popular among game enthusiasts, and have also been used for AI research, though no other research has been done using the raw vision from the game display for AI, except that of the authors of this paper. Bauckhage *et al.* recorded many demos of human players fighting in a map in the FPS *Quake II*; they then took these demos and used them to train separate NNs, one which controlled the yaw and pitch aiming during combat, the other which learned to control the forward/back and right/left velocities during combat, and one which learned to navigate around the map when not in combat [19]. The NNs in his research used nonvisual egocentric inputs such as distance sensors and enemy coordinate locations. Similar research was conducted by Zanetti and El Rhalibi, who trained NNs with a GA to imitate prerecorded human player combat in the game *Quake III*. They also similarly divided the behavioral tasks into a few separate networks [20]. *Quake III* is the sequel to the game used in this research and provides superior graphics, but requires dedicated 3-D hardware. Graham *et al.* used *Quake II* to train NNs using non-visual egocentric inputs to learn path-finding strategies and obstacle avoidance [21]. Unlike the previously mentioned research using FPSs, Graham used reinforcement learning with a GA, and rather than learning to imitate human players, the controllers were awarded fitness depending on how well they followed certain rules. A mapping method called neural gas was used by Thurau *et al.* to imitate the waypoint navigation of prerecorded demos of real players in *Quake II* [22], a method unique among the others previously mentioned in that it did not use egocentric inputs. Priesterjahn *et al.* used egocentric behavioral grids in *Quake III* to train agents that could defeat the AI agents supplied by the game; mappings between grids representing the surrounding game environment and desired behavioral output were learned, and best matches to current environment situations were used to control the agent [23]. More recently, Van Hoorn *et al.* trained AI bots for the FPS Unreal Tournament using an incremental approach that first trained NNs to perform specialized actions, such as shooting or map traversal, and then evolved a controller to select which specialized action to perform at any game state [24]. Our research with *Quake II* is an attempt to learn the complex behaviors learned in all of these works using only visual raw input. A visual control system that worked well throughout the game would be a very robust solution because it would use such low-level inputs.

An option for AI in virtual environments is to use synthetic vision, which involves some preprocessing of the image using nonvisual inputs in order to help differentiate and enhance the visual data's important aspects. For example, in research done by Enrique *et al.*, which used a simulator much like an FPS, the screen was rerendered into two separate color-coded views: the first displayed the color of an object according to its velocity, and the other displayed an object's color according to its identification number and the wall color according to its angle. These enhanced visual data were then used to create a hand-coded controller that could walk around the map picking up health boxes [25]. In another example, Renault *et al.* enhanced pixel data by including a distance value along with the basic color values. They likewise used this enhanced visual information to create a hand-coded controller that could walk down a hallway and avoid obstacles [26]. We refrain from using synthetic vision in our own research because it requires the aid of nonvisual input to preprocess the visual data, and would also make it much more difficult to use our visual control systems with other games and with real-world robots.

Unlike our aforementioned research [12]–[14], we have previously reported on visual research using *Quake II* that did not use an NN as its controller. Instead, it used a controller in which the genetic chromosome represented a control program which included jump and action instructions [27]. The jump instructions determined whether to jump forward in the chromosome, and how far to jump, according to the grayscale value of a 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 very well in a visually simple environment, but that it did not scale well to environments with greater visual complexity.

## III. The *Quake II* Environment

We use the FPS game *Quake II* by Id Software, LLC (Richardson, TX) as the testing platform for this research. In *Quake II*, the player controls a space marine who must destroy an army of aliens on their hostile home planet using a variety of weapons and powerups. The graphics in *Quake II* are gritty and realistic, and the environment is dim and dreary (Fig. 1). Many aspects of the commercial release of *Quake II* are easily modified, such as maps, game rules, models, weapons, and sounds.

One of the most important features of *Quake II* in regards to its usability for research is that it is open source, released under the GNU General Public License (GPL), and compiles on many UNIX-like operating systems [28].[1] Although there are many open-source FPSs, some more recent and advanced than *Quake II*, we chose *Quake II* because it is the most recent engine that still has an option for software rendering. Hardware rendered games require the use of a video card, which means that only one copy of the game can run per card. With a software-rendered engine, several copies of the game can be run on the same computer, and it may even run headless on a cluster node that has no graphics card. This is essential for our research since we distribute the learning over several nodes and processors.

### A. Artificial Intelligence Interface

*Quake II* natively supports creation of server-side programmed bots. These bots run on the server rather the client, and have full access to game-world data. We could not use server-side bots for this research because they do not actually render a game display, but rather directly access environment variables to determine their actions. We instead use the client program, which normally interfaces between the server and a player. This client program renders the particular viewpoint of the player and also accepts inputs from the keyboard and the mouse. Since it is open source, we created an AI interface that allows a controller to read inputs, such as visual data from the rendered screen, coordinates and velocities of in-game entities, and wall locations, and to output player controls, such as forward/back movement, yaw/pitch changes, weapon selection and firing, etc. This setup gives us a versatile and lightweight simulator, which is excellent for testing visual AI controllers.

### B. Blaster Modification

Throughout this research the only weapon we use is the blaster, which normally shoots weak plasma bolts at a fast but noninstantaneous velocity, and at a steady fire rate. We modified the blaster's properties slightly to better suit our training purpose. First, we changed the plasma bolt's damage rate so that one shot kills any victim it happens to hit. Second, we wanted to encourage our learning agents to only shoot when necessary, so we added an energy reservoir to the blaster, such that if the reservoir is full the blaster is able to shoot plasma bolts in rapid succession until the energy is depleted. The reservoir is constantly recharging, so if the fire button is held down steadily, the gun will shoot about two bolts per second. The bursting capabilities of a full energy reservoir motivate the agents to wait to fire until it is necessary, because a burst is more likely to hit a moving opponent than a single shot.

## IV. Neurovisual Control

The basic idea of neurovisual control is to take the color information in a visual input such as a 3-D rendering for a game display and process it with an NN. The NN's output controls an agent's behavior according to the patterns of color information it receives as inputs. The objective is to train the NN to perform the proper behaviors on the basis of that visual input alone.

### A. Retina and Neural Network

*Quake II* renders the game display in a software buffer, and the color values for each pixel can be quickly accessed by direct query. The dimensions of the smallest game display in *Quake II* are $320 \times 240$ pixels, which would be too many pixel inputs for our NNs, if taken individually, both because of real-time speed limitations and the drastically increased learning time requirements. Thus, instead of using individual pixels for the inputs, we divide the display into rectangular blocks and use the average color value of each block as a single input. We can use different sized blocks at different locations of the screen; we refer to the particular arrangement of the sizes and shapes of these blocks as the *retinal layout*. As shown in Section V, the retinal layout is very important to the success of the controller.

Fig. 2 shows a diagram of the neurovisual controller. The retina accesses the game display, and the color information for each block is input into the NN, which is a simple recurrent NN with a hidden/context layer [29]. The outputs of the network

[1]Our modifications to the *Quake II* source code, including a general-purpose application programming interface (API) for client-based AI, are available from the Neuroevolution and Behavior Laboratory at the University of Nevada, Reno, http://nebl.cse.unr.edu/.

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

control the behavior of the agent in the game. The signal from the inputs is propagated through the network for each frame of scene rendering, at 40 frames/s in our version of *Quake II*. There are 28 rectangular input blocks, the hidden/context layer contains ten neurons, and there are four output neurons. We use tanh for the squashing function. The four outputs control forward/back movement, right/left "strafing" movement, turning, and shooting (Fig. 3). The first three outputs are continuous values in the range $(-1, 1)$, with the sign interpreted to indicate direction and the magnitude used to indicate speed. The shooting output, which in the game is a binary option, is interpreted as *true* only if its output is greater than or equal to zero. We use the same network architecture and the same number of retinal inputs throughout the research presented in this paper.

## B. *Learning*

Between the retinal input of the NN and its four outputs are 434 floating-point weights, which are parameters that determine the behavior of the network. We train the network by changing the weights until their values define a function which outputs suitable values for the inputs. Since the network is recurrent, the output values depend on the entire history of input values while in operation. It is not possible to explore all combinations of weight values, and it is essentially impossible to calculate appropriate values for this architecture and application. Thus, we set the weight values with a machine learning algorithm called *neuroevolution*.

*1) Neuroevolution:* Neuroevolution uses a GA to evolve the weights of the network. GAs imitate microevolutionary breeding that can be seen in reproductive organisms. In dog breeding, for example, dogs that exhibit certain desirable characteristics are mated together to produce dogs with different combinations of those traits. Similarly, a GA contains a set of simulated chromosomes which specify the traits. The genotypes for the chromosomes are tested to determine their fitness for a task, and are then "mated" by crossing the chromosomes of the most fit instances. When this process is performed over several generations, the outcome is a set of chromosomes that specify a network that performs its task well. Using this method to set the weights is a form of reinforcement learning, which means that no specific input/output combinations are required for training; it is only necessary to distinguish better and worse performances.

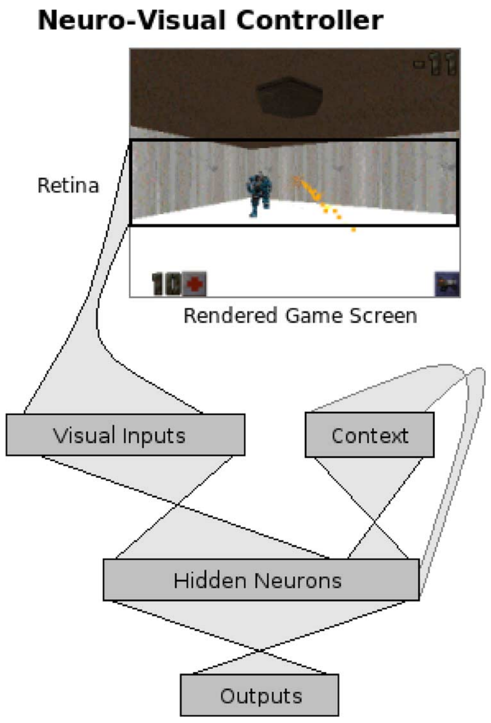

Fig. 2. The neurovisual controller used in this research. The retina reaps color information from the rendered screen image and feeds it as inputs to a simple recurrent NN, which in turn controls the behavior of the in-game agent.

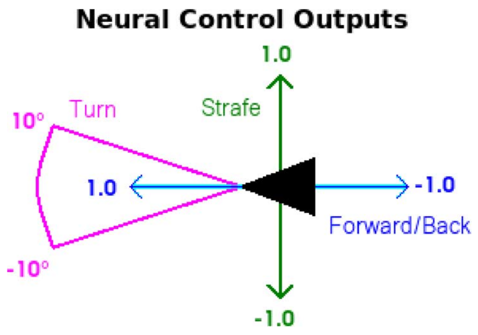

Fig. 3. The NN outputs four values in the range $[-1.0, 1.0]$. Three of the outputs control the movement: forward/back, right/left strafing, and turning. The fourth output controls whether to shoot, which is *true* if greater than or equal to zero, and *false* otherwise.

In this research, we use a queue genetic algorithm (QGA), which is a steady-state first-in–first-out GA that stores the population of chromosomes in a queue (Fig. 4) [30]. When a new individual is needed, a roulette-wheel-style selection, in which chromosomes with greater fitness have a higher chance of being selected, is used to select two parent chromosomes. These two chromosomes are mixed into one child by using uniform crossover, which selects each gene randomly with an even chance from either parent. The genes in this experiment are floating point numbers, which directly represent the weights of the NN, of which there are 434 total. Mutation is used to encourage diversity and to help prevent premature convergence on a suboptimal solution. Each gene has a 10% chance of being

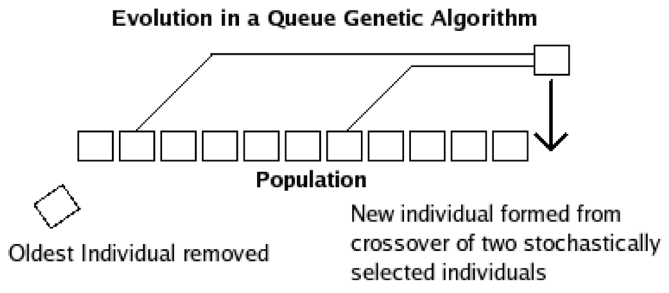
Evolution in a Queue Genetic Algorithm
Population
Oldest Individual removed
New individual formed from crossover of two stochastically selected individuals

Fig. 4. The 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.

mutated; the mutation is a random value added to the current gene value, taken from sharply peaked random distribution $(\log(n)/10) * \mathrm{random}(-1 \text{ or } 1)$, where $n$ is a random number in the range $(0,1)$, which has a high probability of producing low deltas and a low probability of producing high deltas. Because the QGA has a steady-state population, we measure a generation to be every interval that $p$ individuals have been tested, where $p$ is the size of the population.

The QGA is particularly useful for distributing the evolutionary learning over multiprocessor systems, networked systems, and clusters. *Quake II* simulators that are testing fitnesses of chromosomes can be run all over the network and connect to a single QGA server, drastically decreasing the clock time necessary for the evolutionary computation.

*2) Backpropagation:* Another learning method that is used to train the weights of NNs is backpropagation. In backpropagation, the NN calculates outputs for an input state; the actual output is compared to a target output for that particular input state, and the error is used to adjust the values of the weights, working backward from the outputs to the inputs. By training the network on a set of desired input/output pairs the NN can learn a generalized solution, which can later be used for input patterns not seen during training. Since backpropagation typically relies on a set of known input/output pairs, it is a form of supervised learning.

## V. Uniform Versus Graduated Density Retina

Our first experimental goal was to determine whether the retinal layout made much difference for learning and the resulting behavior. We compared two different layouts that both used the same number of input blocks, but arranged their widths differently. Our inspiration for one retinal layout was that of a human, which, instead of having a uniform density of rods and cones, contains a greater density in the center of the retina and less near the periphery [31]. We designed a graduated density retina loosely in imitation of this, with each block in the layout 1.618 times as wide as the next block toward the center, so that the retina has higher resolution toward the center and lower toward the periphery. We compared this retinal layout to a uniform density retina, in which all the blocks are the same width. Both retinal layouts used two short rows of 14 blocks across the center of the screen, with the only difference being the width of the blocks, as shown in Figs. 5 and 6. Both retinas were connected to the NN as described in the previous section.

### A. Test Setup

Our test for these two retinal layouts was to learn to control the agent to shoot a randomly moving opponent in a visually simple room with dimensions of about half a basketball court (Fig. 5). We used the QGA for the learning, with the fitness function being based entirely upon the number of kills achieved by the agent, which number is multiplied by five then squared in order to make it nonlinear, to increase fitness selection pressure. To help with the learning, we started training with a stationary enemy and gradually increased its speed whenever the average fitness of the genetic population reached a threshold. The enemy is harmless and never shoots back in any of this research.

The order in which the learning agents were tested is as follows:

1) the agent is inserted into the room and drops to the floor;
2) the agent is given 24 s to kill the enemy as many times as possible;
3) whenever the agent kills the enemy, the enemy immediately reappears at a random location;
4) at the end of the 24 s, the agent is removed, and a new agent with a new chromosome is inserted for evaluation.

### B. Results

We evolved 24 populations for each of the two retinal layouts, with 128 chromosomes per population. We evolved the populations for 1000 generations each, then graphed the average of the 24 populations' average fitnesses, as well as the average of the enemy movement speeds obtained during training. Fig. 7 shows the resulting graph for the uniform density retina, which on average did not allow reaching either the fitness or the enemy movement speed obtained when using the graduated density retina, shown in Fig. 8.

### C. Discussion

We see from the results of this experiment that retinal layouts make a difference in the behavior of the agents, even when only the widths of the retinal blocks are varied. Of the two retinal layout tests, the graduated density retina performs much better on this problem. The agents that use the graduated density retina are able to spot the enemy from across the room and shoot at him, whereas the agents using the uniform density retina must be somewhat closer to the enemy to notice it. The agents with the uniform retina learn to traverse around the perimeter of the room in order to find the enemy. In contrast, the agents with the graduated retina can just spin in a tight circle until they see the enemy, even if they are on opposite sides of the room, which saves time and allows them to kill the enemy more frequently in their allotted fitness evaluation time. Because of the advantages of the graduated density retina, we used it for the remaining two experiments presented in this research.

## VI. Lamarckian Neuroevolution

We decided next to test the graduated density retina controller in a more visually complex room. We darkened the textures

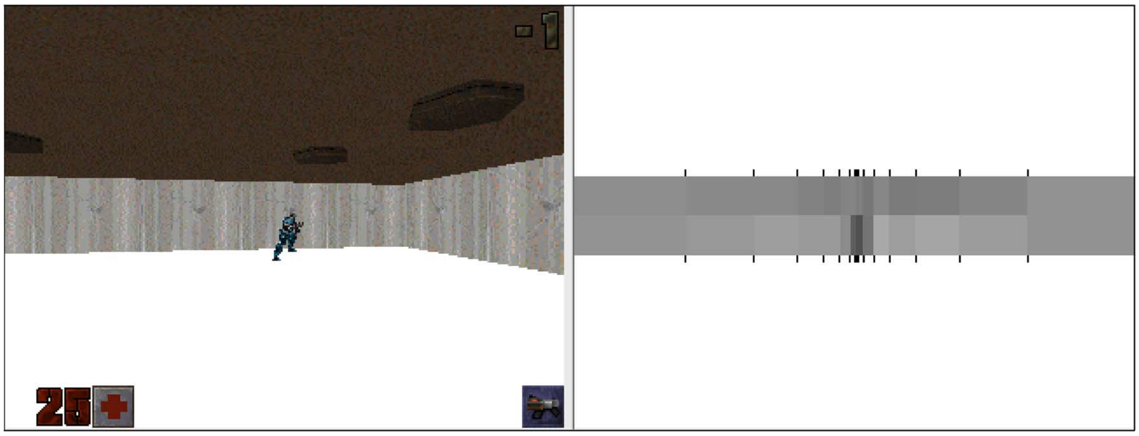
Fig. 5. (a) A scene as rendered by the game engine of the simple map used in the retina experiment. (b) The same scene as viewed via the graduated density retina. The black tics above and below the image indicate the retinal block widths.

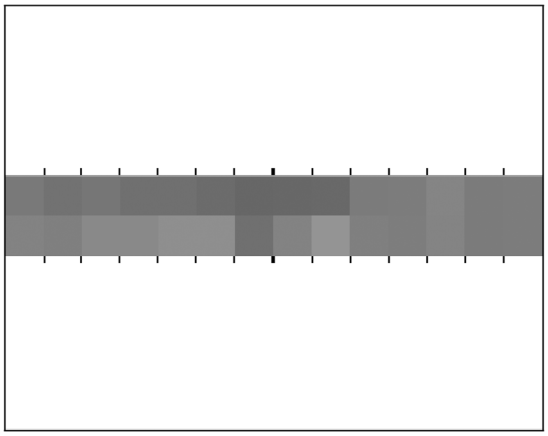
Fig. 6. A view via the uniform density retina. The enemy's location and distance are similar to the view in Fig. 5. The contrast between the enemy and the walls and floor are much less distinct in the uniform retina than in the graduated density retina, because of the increased area averaged into the visual block where it appears.

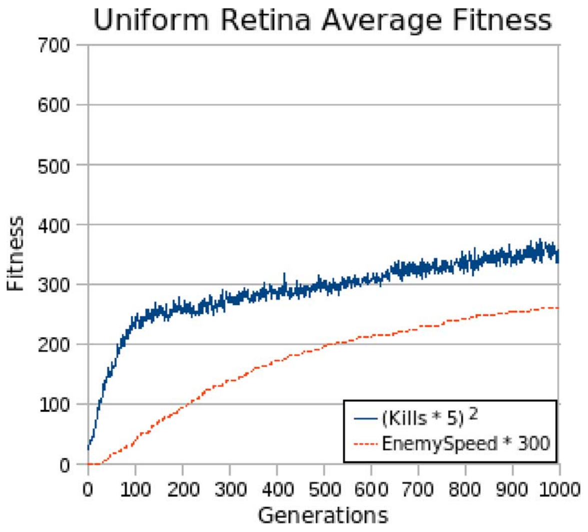

Fig. 7. The population fitness averaged over 24 tests, using the uniform retina (Fig. 6). The dashed line is the enemy's movement speed, where 300 is full speed.

and added shadows to make the environment more akin to what would be found in a real game of *Quake II* or even in real life

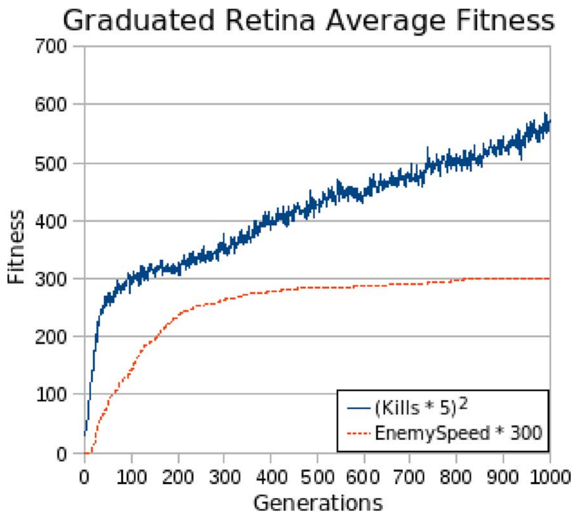

Fig. 8. The population fitness averaged over 24 tests, using the graduated density retina (Fig. 5). The dashed bottom line indicates the movement speed of the enemy, where 300 is full speed.

environments (Fig. 9). We evolved the graduated density neurovisual controller for 1000 generations in the new map and found that the controller was unable to learn satisfactorily. Instead of learning to shoot at the enemy, it learned a semirandom "sprinkler" pattern, which had a likelihood of killing the enemy. Unfortunately, all of our tested populations seemed to converge on this suboptimal solution. We believed that the retina and NN were capable of satisfactorily solving this problem, but that they needed something to push the learning toward the desired solution.

### A. *Supervising Controller*

We knew from the previous retinal layout experiment that a good solution is to spin around in circles until the enemy is centered, then spray shots randomly in his direction until he is felled. Because we knew a good solution, we could theoretically use supervised learning with backpropagation to train the NN to behave as we desired. As in some other experiments, we could use human examples to train the NN [19], [20], [22], but human play is often inconsistent and presents new learning difficulties.

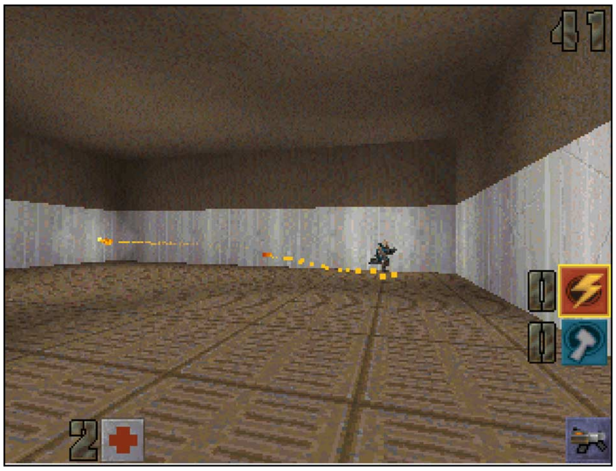

Fig. 9. An in-game screenshot of the environment for the Lamarckian neuroevolution experiment. The floor and ceilings are brown, the walls are gray, and the enemy is dark blue. The room is dimly lit with varying shadows. The visual complexity of this room is much greater than the room used in the retinal layout experiment (Fig. 5).

Also, a human would have a very difficult time spinning around in circles as smoothly and stopping as precisely as we would like. Our solution, then, was to make a hand-coded controller that behaved just as we wanted.

The hand-coded controller completely ignores the visual game screen and retrieves its inputs directly from the data structures in the game; namely, it accesses the enemy's location coordinates. Using this input, it performs a simple behavior of spinning around in circles to the left; as soon as the enemy is centered, it begins shooting a spray of shots and wiggling its aim randomly. As soon as the enemy falls, it continues spinning to the left. This hand-coded controller is an imitation of the behavior we observed in the graduated density controllers in the visually simple room of the retinal layout experiment. Since we are using it to train a visual-only controller, we had to make sure it did not do anything that a visual controller could not do, such as turn the shortest distance to the enemy when he is behind the agent's back, as would be quite possible to do with the nonvisual controller.

## B. Training

Once the nonvisual hand-coded controller was created, it was used to train the neurovisual controller via backpropagation. For this procedure, the nonvisual hand-coded controller controls the behavior of the in-game agent and the neurovisual controller looks at the screen for its inputs, then calculates its outputs. The outputs of both the hand-coded controller and the NN are compared and the difference in outputs is used to train the weights of the network using backpropagation.

Backpropagation by itself was considered as a solution to train the network, but we would need to distribute the learning over several computers to speed it up. Although there are a few solutions for distributing backpropagation, they are complicated and would be difficult to use with this problem [32]. Instead, we combine backpropagation with our already distributed neuroevolution into a Lamarckian-style evolution. Jean-Baptiste Lamarck first proposed in the early nineteenth century a

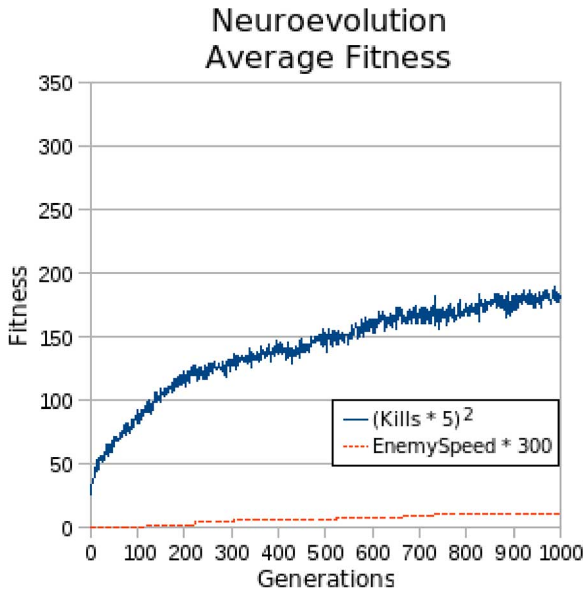

Fig. 10. Average of the average fitnesses of the 24 populations that used neuroevolution alone. The dark top line shows the fitness according to the number of kills, and the dashed line shows the enemy's speed, which increased whenever the average fitness reached a certain point.

theory which hypothesized that phenotype adaptations learned within an individual's lifetime were passed onto its offspring [33]. Lamarckian evolution can be employed in neuroevolution in several different ways [34]–[36]; one way is to use backpropagation to train individual chromosomes for a time before they are tested for fitness [37], [38]. The traits learned with backpropagation are permanent changes to the chromosome and are passed on to future offspring. This solution allows for an intuitive way to distribute backpropagation, and also retains the advantages of robust evolutionary reinforcement learning.

The order for testing and training the chromosomes using Lamarckian neuroevolution is much the same as the order used in the retinal layout experiment, except that the agents are given an extra 12 s of learning with backpropagation under the hand-coded supervisor before their 24-s fitness test begins. The maximum number of kills possible in 24 s is about 12.

## C. Results

We compared 24 populations using Lamarckian neuroevolution with 24 populations using neuroevolution only. Everything was the same in both tests except that each Lamarckian chromosome was trained for the extra 12 s with backpropagation. Fig. 10 shows the average of the average fitnesses for the 24 neuroevolution-only tests; on average they performed poorly in comparison to the 24 Lamarckian tests shown in Fig. 11.

Visual observation of the Lamarckian tests show that the agents are actually noticing the enemy and shooting at him whenever he is centered. Because the room is very dark in several places, the agents will occasionally shoot at the corners and shadows, but this in itself is a useful tactic because an enemy could be hiding in those places. The neuroevolution-only test, as previously stated, learned a sprinkler pattern and never shot specifically at the enemy.

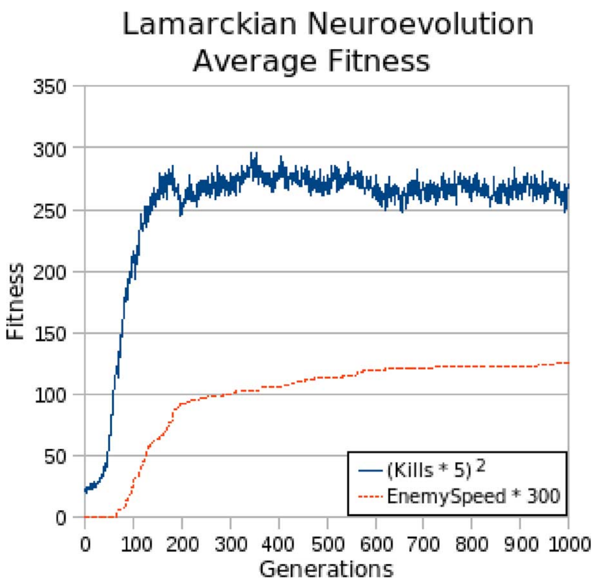

Fig. 11. Average of the average fitnesses of the 24 populations that used Lamarckian neuroevolution. The top dark line shows the fitness according to the number of kills, and the dashed line shows the enemy's speed, which increased whenever a population's average fitness reached a certain point.

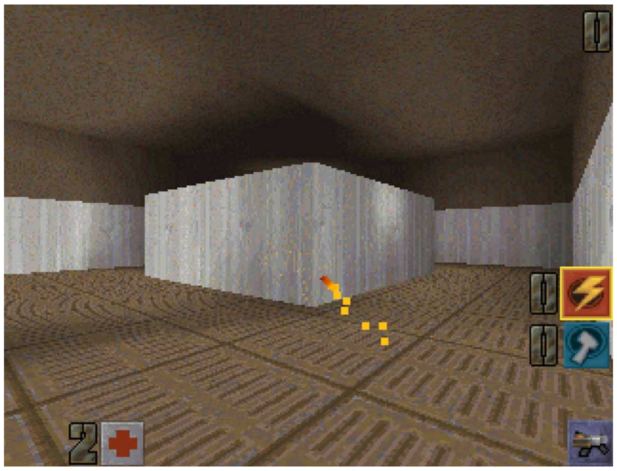

Fig. 12. A screenshot of the room with a large central pillar used in this experiment. The moving enemy always reappears after death on the opposite side of the pillar from the learning agent.

### D. Discussion

The neuroevolution-only tests were unable to find a good solution to this problem because they prematurely converged on a suboptimal behavior, but the Lamarckian neuroevolution tests learned to actually shoot at the enemy due to their backpropagation training, which pushed the population in the correct direction. This experiment shows that it can be very beneficial to train a low-level controller with a hand-coded high-level controller in a Lamarckian learning scheme, with the disadvantage that, in order to do so, a good solution must be previously known.

## VII. Removing the Human Element From Supervision

One of the drawbacks of the previous Lamarckian neuroevolution experiment is that knowledge of a good solution must be previously known, and a human must be able to hand code that solution. Programming a hand-coded controller, even with higher level inputs, can be quite difficult, especially for complex behaviors. Moreover, if a good solution to the problem is not already known, then programming it is just guesswork. Both of these problems can be solved by evolving, rather than hand coding, a higher level input controller, which can in turn be used as a supervisor to the lower level controller in Lamarckian learning.

### A. Room With Pillar

The new problem, for which we did not already know a solution, is for the agent to shoot and kill a moving opponent in a visually complex room that has a large central cube pillar (Fig. 12). The enemy always drops on the opposite side of the pillar from the location of the agent. The agent can no longer use its simple tactic of spinning in circles and shooting at the enemy when he is centered, simply because there will almost always be a pillar between him and the enemy. The obvious solution is to traverse around the pillar, looking for the enemy, but there are questions as to how near around the pillar to walk, what direction to point when walking, and the best way to go around corners, variables which can be optimized by evolving the nonvisual supervising controller.

### B. Nonvisual NN Supervisor

For a nonvisual controller network to be useful as a supervisor for a visual controller, it must not make decisions on the basis of inputs that are uncorrelated with the visual scene. Our supervising network uses a combination of wall sensors and measurements of the enemy location and velocity. The wall sensors point directly ahead and to 10°, 25°, and 60° on either side. The enemy's location is given in terms of distance and horizontal offset relative to the direction that the agent is pointed, and the enemy's velocity is the delta of these two values between frames. Whenever the enemy is killed or occluded behind the pillar, the distance and direction are reported as if he were extremely far away, so that the controller will not make decisions on the basis of nonvisible knowledge of his whereabouts.

The 11 nonvisual inputs are fed into a simple recurrent network with four outputs. The network was trained with neuroevolution using the QGA for 1000 generations. Fig. 13 shows the fitness of these controllers. At the 1000th generation, we take the best controller from each population and use them to supervise corresponding Lamarckian visual controllers.

### C. Training the Visual Controllers

We trained and compared the Lamarckian test to a regular neuroevolution-only test. The order of the test is the same as used in the previous Lamarckian experiment, except that the hand-coded controller is replaced by the nonvisual NN controller, which trains the chromosomes for 12 s by backpropagation.

In order to accurately compare the Lamarckian and neuroevolution-only tests, we note that the Lamarckian tests take extra computational time for evolution of the nonvisual supervising network, as well as the 12 s extra needed per chromosome for the backpropagation. In the time it takes to train the supervising

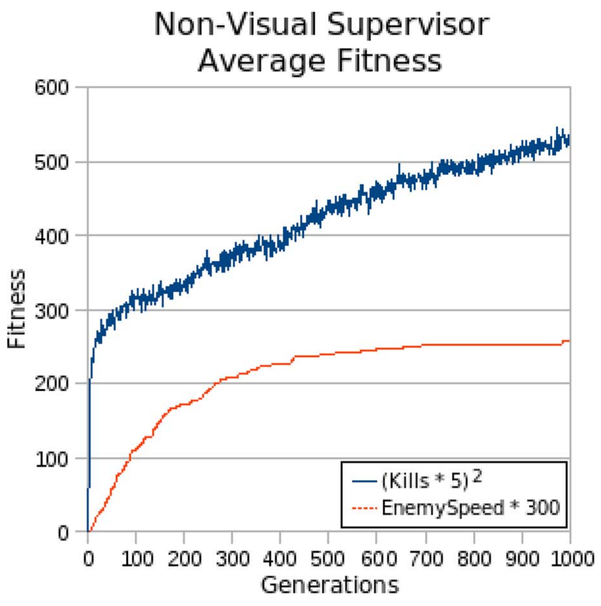

Fig. 13. A graph showing the average of the average fitnesses of the 25 nonvisual NN supervisor tests. It performs much better than the visual controller it supervised because it is easier to perform this problem with nonvisual input.

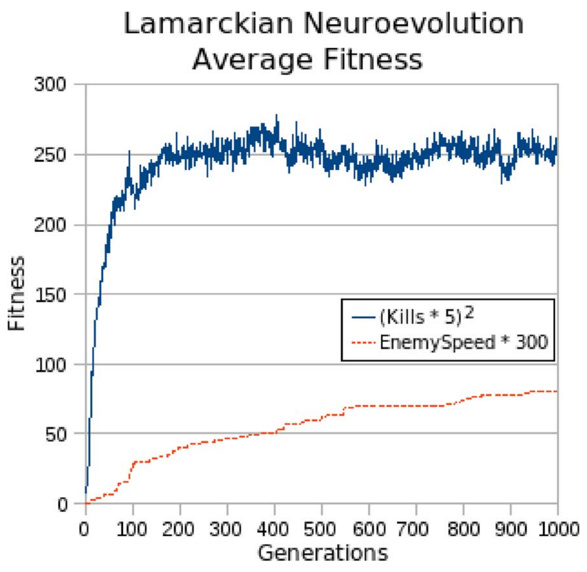

Fig. 15. A graph of the average of the average fitnesses for 25 Lamarckian tests, using the nonvisual NN as supervisor. The fitness is not as high as that of the nonvisual supervisor (Fig. 13), but it is higher than the fitness of the neuroevolution-only tests (Fig. 14).

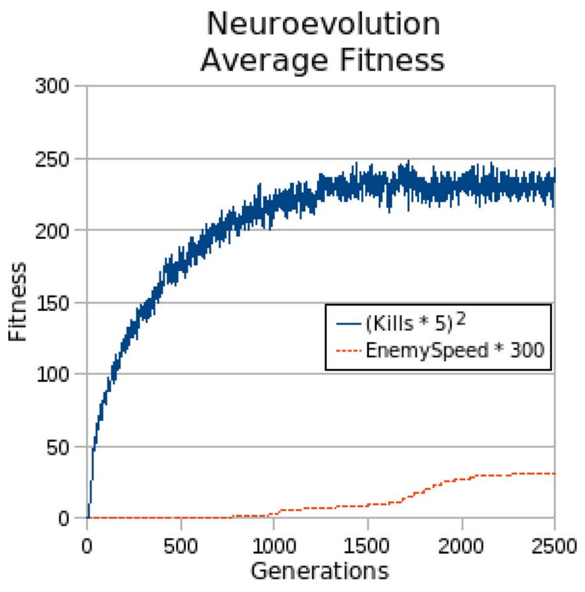

Fig. 14. The average of the average fitnesses of the 25 neuroevolution-only tests. (Notice that both axes differ from Fig. 13.)

network and then the Lamarckian controller for 1000 generations each, it is possible to train a neuroevolution-only controller for 2500 generations. Thus, to balance the computational effort used by the two learning procedures, we compared 1000 generations of Lamarckian learning to 2500 generations of neuroevolution-only learning.

## D. Results

We trained 25 populations each for the Lamarckian test and the neuroevolution-only test. Fig. 14 shows the graph of the average of the average fitnesses for the neuroevolution-only tests. It is clear that the Lamarckian tests (Fig. 15) learn faster and achieve higher fitness than the neuroevolution-only tests.

In all tests, the agent learned to traverse around the pillar facing forward. As in the previous experiment, the neuroevolution-only controller learned a sprinkler pattern and did not shoot specifically at the enemy. The sprinkler pattern worked better here than in the previous experiment, since there was a greater likelihood of hitting the enemy in the narrow hallway around the pillar. In contrast, the Lamarckian tests that were supervised by the nonvisual NN controller learned to shoot directly at the enemy, only occasionally shooting also at corners or dark shadows.

## E. Discussion

The results of this experiment show that it can be useful to evolve an NN controller that uses high-level inputs to supervise a controller that uses lower level inputs, such as visual inputs. Particularly, it was useful in this pillar problem, for which we did not know an optimal solution. It also meant that we were able to completely remove the human element from the supervision, and hence needed no *a priori* knowledge of the solution.

## VIII. Conclusion

In this research, we explored various techniques for learning visual control in the *Quake II* environment. We used NNs that read the visual data directly from the rendered game display. We trained the NNs with a GA or with a GA in combination with backpropagation.

In our first experiment, we tested two different retinal layouts, one uniform and one humanlike with greater acuity in the center. Both retinas used the same number of input blocks and learned using the same NN structure. Using neuroevolution, we trained the networks as controllers to try to shoot an enemy moving about in a visually simple room. The results showed that controllers that used the biologically inspired graduated-density retina outperformed controllers with the uniform density retina, most likely because in *Quake II* the most important game information tends to be in the center of the screen, where the gun aims.

In our second experiment, we added shadows and darker textures to the simple room and tried to evolve a controller that used the graduated-density retina layout. Neuroevolution was unable to produce a satisfactory solution; rather, most of the controllers ignored the enemy's appearance and moved around the room shooting in a sprinkler pattern that would occasionally hit the enemy. In order to help push the agents to our desired solution, we hand coded a controller which used nonvisual inputs. We used this nonvisual controller to supervise the visual controllers through backpropagation in a Lamarckian evolution scheme, in which each controller learned by backpropagation, then was tested for evolutionary fitness. We ended up with pure visual controllers that exhibited the desired behavior and also had higher fitness than the neuroevolution-only visual controllers.

A limitation of the second experiment was that we had to hand code the nonvisual supervisory controller. If the ideal solution were unknown or too complicated, then hand coding a supervising controller might be too difficult. For the third experiment, we created a map with a large obstructing central pillar, around which the enemy would appear and walk randomly. Instead of hand coding a supervisory controller using *a priori* knowledge, we used neuroevolution to train a nonvisual controller, which was then used as a supervisor to train the visual controllers using Lamarckian evolution. The Lamarckian controllers had much higher fitness than the neuroevolution-only controllers.

Currently, in many computer games, the AI controllers use high-level game information, allowing the AI to "cheat" by using knowledge hidden to the player. A pure vision-based controller would help limit the AI's knowledge to that of the human player, perhaps aiding the illusion of realism. In addition, vision-based AI controllers would allow for new player strategies, such as using camouflage and hiding in shadows, potentially making the game more fun.

The use of lower level inputs allows for a more generalized solution, while higher level inputs limit a controller's use to areas where those high-level inputs are available. In robotics, for example, a robot that uses range finders to navigate will work almost anywhere, but a robot that uses GPS location coordinates and a map will only be able to navigate in a known mapped area where GPS is available. Hand coding for low-level inputs is difficult, and, as seen in our Lamarckian experiments, evolving them is sometimes insufficient. Moreover, the training environment for a controller, especially when it is a virtual simulation, often has access to higher level symbolic inputs. In those situations, the Lamarckian learning method presented in this paper could be used: a high-level input controller could be evolved, which would be used as a supervisor to help evolve the low-level controller.

One limitation of the Lamarckian method is that we had to manually limit the high-level inputs to use only information available to the visual inputs. For instance, we would not allow the high-level controller to "see" the enemy when there was a wall in the way because the visual controller could only see the wall. Because our problem was so simple, limiting the high-level inputs was not particularly difficult. However, in other problems, especially in real-world robotics applications, where there may be multiple high- and low-level inputs, limiting the high-level inputs by hand may be impossible. A method for automatically limiting the high-level supervisory inputs would be necessary, which could perhaps be done by observing random samples of environment changes and their effects on the low-level inputs. Otherwise, it may be possible to coevolve supervisory controllers with the low-level controllers, where the success of the "students" has some effect on the evolution of the "teacher."

The experiments presented in this research are foundational discoveries that will be used for further exploration in the area of neurovisual control. The final behaviors exhibited by our agents were far from that of competitive and fun AI opponents, but we have purposely kept our controllers and tasks simple in order to isolate the strategic requirements, so that we could see exactly which techniques work better when training the visual agents. In future work, we intend to increase the size and resolution of the retina, so that the controller can see more detail in complex environments. We also would like to explore the use of active vision with a roaming retina, and further expand the Lamarckian experiments to simultaneously evolve the visual controller and the nonvisual supervisor as described above. We will gradually increase the difficulty of the problems solved so that hopefully we will eventually have a fun and competitive AI that uses only visual input.

## References

[1] I. Masaki, "Vision-based vehicle guidance," in *Proc. Int. Conf. Power Electron. Motion Control*, Nov 1992, vol. 2, pp. 862–867.

[2] T. Jochem, D. Pomerleau, and C. Thorpe, "Vision-based neural network road and intersection detection and traversal," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Aug 1995, vol. 3, pp. 344–349.

[3] J. Laird and M. Van Lent, "Human-level AIs killer application," *AI Mag.*, vol. 22, no. 2, Summer, 2001.

[4] J. Laird, "Research in human-level AI using computer games," *Commun. ACM*, vol. 45, no. 1, Jan. 2002, DOI: 10.1145/502269.502290.

[5] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, pp. 520–522, Jun. 1996.

[6] K. Fukushima, "A neural network for visual pattern recognition," *Computer*, vol. 21, no. 3, pp. 65–75, Mar. 1998.

[7] K. Grill-Spector and R. Malach, "The human visual cortex," *Annu. Rev. Neurosci.*, vol. 27, pp. 649–677, Jul. 1998.

[8] D. Ballard and G. Hinton, "Parallel visual computation," *Nature*, vol. 306, pp. 21–26, Nov. 1983.

[9] L. Zhao and C. Thorpe, "Stereo- and neural network-based pedestrian detection," *IEEE Trans. Intell. Transp. Syst.*, vol. 1, no. 3, pp. 148–154, Sep 2000.

[10] R. Feraund, O. Bernier, J. Viallet, and M. Collobert, "A fast and accurate face detector based on neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 1, pp. 42–53, Jan. 2001.

[11] K.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognit.*, vol. 37, no. 6, pp. 1311–1314, Jun. 2004.

[12] M. Parker and B. Bryant, "Neuro-visual control in the Quake II game engine," in *Proc. Int. Joint Conf. Neural Netw.*, Hong Kong, Jun. 2008, pp. 3828–3833.

[13] M. Parker and B. Bryant, "Lamarckian neuroevolution for visual control in the Quake II environment," in *Proc. IEEE Congr. Evol. Comput.*, Trondheim, Norway, May 2009, pp. 2630–2637.

[14] M. Parker and B. Bryant, "Backpropagation without human supervision for visual control in Quake II," in *Proc. IEEE Symp. Comput. Intell. Games*, Milano, Italy, May 2009, pp. 287–293.

[15] D. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Comput.*, vol. 3, no. 1, pp. 88–97, 1991.

[16] S. Baluja, "Evolution of an artificial neural network based autonomous land vehicle controller," *IEEE Trans. Syst. Man Cybern.*, vol. 26, no. 3, pp. 450–463, Jun. 1996.

[17] D. Floreano, T. Kato, D. Marocco, and E. Sauser, "Coevolution of active vision and feature selection," *Biol. Cybern.*, vol. 90, no. 3, pp. 218–228, 2004.

[18] N. Kohl, K. Stanley, R. Miikkulainen, M. Samples, and R. Sherony, "Evolving a real-world vehicle warning system," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2006, pp. 1681–1688.

[19] C. Bauckhage, C. Thurau, and G. Sagerer, "Learning human-like opponent behavior for interactive computer games," in *Pattern Recognition*, ser. Lecture Notes in Computer Science, B. Michaelis and G. Krell, Eds. Berlin, Germany: Springer-Verlag, 2003, vol. 2781, pp. 148–155.

[20] S. Zanetti and A. El Rhalibi, "Machine learning techniques for FPS in Q3," in *Proc. Int. Conf. Adv. Comput. Entertain. Technol.*, Singapore, Jun. 3–5, 2004, DOI: 10.1145/1067343.1067374.

[21] R. Graham, H. McCabe, and S. Sheridan, "Neural pathways for real time dynamic computer games," in *Proc. 6th Eurograph. Ireland Chapter Workshop*, Jun. 2005, vol. 4, pp. 13–16, ISSN: 1649-1807.

[22] C. Thurau, C. Bauckhage, and G. Sagerer, "Learning human-like movement behavior for computer games," in *Proc. Int. Conf. Simul. Adapt. Behav.*, 2004, pp. 315–323.

[23] S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels, "Evolution of human-competitive agents in modern computer games," in *Proc. IEEE Congr. Evol. Comput.*, Vancouver, BC, Canada, Jul. 2006, pp. 777–784.

[24] N. van Hoorn, J. Togelius, and J. Schmidhuber, "Hierarchical controller learning in a first-person shooter," in *Proc. IEEE Symp. Comput. Intell. Games*, Milano, Italy, May 2009, pp. 294–301.

[25] S. Enrique, A. Watt, F. Policarpo, and S. Maddock, "Using synthetic vision for autonomous non-player characters in computer games," in *Proc. 4th Argentine Symp. Artif. Intell.*, Santa Fe, Argentina, 2002.

[26] O. Renault, N. Magnenat-Thalmann, and D. Thalmann, "A vision-based approach to behavioural animation," *J. Vis. Comput. Animat.*, vol. 1, no. 1, pp. 18–21, 1990.

[27] M. Parker and B. Bryant, "Visual control in Quake II with a cyclic controller," in *Proc. IEEE Symp. Comput. Intell. Games*, Perth, Australia, Dec. 2008, pp. 151–158.

[28] "Q2 LNX Stuff," Nov. 14, 2005 [Online]. Available: http://icculus.org/quake2/

[29] J. L. Elman, "Finding structure in time," *Cogn. Sci.*, vol. 14, pp. 179–211, 1990.

[30] M. Parker and G. Parker, "Using a queue genetic algorithm to evolve Xpilot control strategies on a distributed system," in *Proc. IEEE Congr. Evol. Comput.*, Vancouver, BC, Canada, Jul. 2006, pp. 1202–1207.

[31] J. Jonas, U. Schneider, and G. Naumann, "Count and density of human retinal photoreceptors," in *Graefe's Archive for Clinical and Experimental Ophthalmology*. New York: Springer-Verlag, 1992.

[32] Q. Chen, Y. Lai, and J. Han, "A implementation for distributed backpropagation using corba architecture," in *Proc. 5th Int. Conf. Cogn. Inf.*, Beijing, China, Jul. 2006, pp. 830–834.

[33] J.-B. Lamarck, *Philosophie Zoologique*. Oxford, U.K.: Oxford Univ. Press, 1809.

[34] J. Grefenstette, "Lamarckian learning in multi-agent environments," in *Proc. 4th Int. Conf. Genetic Algorithms*, San Mateo, CA, 1991, pp. 303–310.

[35] D. Whitley, S. Dominic, R. Das, and C. W. Anderson, "Genetic reinforcement learning for neurocontrol problems," *Mach. Learn.*, vol. 13, pp. 259–284, 1993.

[36] K. Ku, M. Mak, and W. Sui, "A study of the Lamarckian evolution of recurrent neural networks," *IEEE Trans. Evol. Comput.*, vol. 4, pp. 31–42, 2000.

[37] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[38] B. Bryant and R. Miikkulainen, "Acquiring visibly intelligent behavior with example-guided neuroevolution," in *Proc. 22nd Nat. Conf. Artif. Intell.*, 2007, pp. 801–808.

**Matt Parker** received the M.S. degree in computer science from the University of Nevada, Reno, in 2009.

He is currently an entrepreneur developing an online educational tool for learning foreign languages.

**Bobby D. Bryant** (S'03–M'06) received the B.A. degree in classical studies from the University of Houston, Houston, TX, in 1995 and the M.S.C.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin, in 1999 and 2006, respectively.

He is currently an Assistant Professor in the Department of Computer Science and Engineering, University of Nevada, Reno. His research focuses on controllers for autonomous intelligent agents, inductive agent modeling, artificial neural networks, and applications of biologically realistic neural networks to machine intelligence.

Prof. Bryant is a member of the Association for Computing Machinery (ACM).