Neuro-visual Control in the Quake II Game Engine

Matt Parker and Bobby D. Bryant

Abstract— **The first-person-shooter Quake II is used as a platform to test neuro-visual control and retina input layouts. Agents are trained to shoot a moving enemy as quickly as possible in a visually simple environment, using a neural network controller with evolved weights. Two retina layouts are tested, each with the same number of inputs: first, a graduated density retina which focuses near the center of the screen and blurs outward; second, a uniform retina which focuses evenly across the screen. Results show that the graduated density retina learns more successfully than the uniform retina.**

I. Introduction

RAW visual input is important for many applications in robotics and simulation. A large amount of information about the surrounding world can be collected merely from a simple camera image. Very complex behaviors can be performed by a human using only raw vision as input. Humans can remotely control camera-mounted vehicles and play a large variety of complex computer games using a simple two-dimensional image as visual input. Despite its usefulness, many real-time Artificial Intelligence (AI) controllers do not use visual input, but rather use a large array of other sensors, such as GPS, range-finding, and light sensors. This is due largely to the complexity of processing raw visual input as useful data.

There have been many neural network applications that use visual input for static image processing, but less that use it for real-time dynamic processing. In research by Pomerleau, a neural network was trained using back-propagation to drive a car using a 30x32 grayscale visual input [1]. This experiment worked well, and could train in real-time. Shumeet Baluja extended the experiment to use an evolutionary computational model for learning, rather than back-propagation, with the results that the system learned more robust control, but with the drawback that it learned offline rather than in real-time [2]. Floreano et al. used a genetic algorithm to train a simple recurrent network that used active vision to race a car around a track in a realistic driving simulator [3].

Experiments have also tested using vision for controlling agents in a virtual world. Many of these experiments use Synthetic Vision, which processes the visual input before sending it to the controller. This preprocessing might include extra-dimensional data so that every pixel contains a color value, a distance-from-agent value, and an entity identifier value, as in the case of the behavioral animation experiment by Renault et al., which made a hand-coded agent that could navigate down corridors and avoid obstacles [4]. In research by Enrique et al., the visual input was re-rendered as two separate color-coded views: one which color coded polygons according to their angles, and color-coded entities by their object i.d.; and the other which displayed a color according to the object's velocity [5]. While Synthetic Vision may be useful, it requires extra computational time and information that might not be ascertainable in a real world environment; it also loses the simplicity of having a single camera as input. In an experiment by Kohl et al., a controller in an auto-racing simulator used only a raw visual gray-scale input array of 20x14 pixels and NeuroEvolution of Augmenting Topologies (NEAT) to evolve a vehicle warning system [6]. They were then able to successfully transfer this technique to a real robot with a mounted camera.

Much AI research has been performed using first person shooters (FPS), which are 3-Dimensional games where the player is usually given the task to survive in a hostile environment and shoot all the enemies. Zanetti and El Rhalibi trained neural networks for Quake III, using neuroevolution and the location coordinates, angles, and weapon information from pre-recorded demos of human players to train the agent to maneuver through the map, perform combat, and collect items [7]. Similar research was performed by Bauckhage et al. for Quake II [8]. Thurau et al. used a Neural Gas method in Quake II to train agent way-point navigation, also by observing movements of pre-recorded human players [9]. Graham et al., used Quake II and neuroevolution to evolve path-finding strategies and obstacle avoidance [10]. Vision is a particularly realistic input for first person shooters that, if used with a proper controller, could provide robust and human-like character behavior.

In the experiment reported in this paper, we research the use of raw vision as the only input to a neural network, which is trained by neuroevolution, to perform simple combat in a first person shooter. Moreover, rather than use the recorded data of a human player or controller, as in the above-mentioned vision and FPS experiments, we use a fitness function that awards agents that perform well in the game, without regard to imitation of human behavior. We test and compare two different visual input controllers, as well as incremental evolution of the difficulty of the task.

II. The Quake II Environment

We use the Quake II engine by Id Software, Inc. as our platform for research. Quake II is a first person shooter which puts the player into an alien world as a space marine who must fight his way to freedom. Multiplayer games are supported over a network or the internet, up to 256 players on a single server (if hardware is capable). The graphics in the

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.

game are generally dark and dreary, but very realistic. Quake II was at one time very popular, and many maps, models, and tools to create custom content are freely available.

There are a few key reasons we chose to use Quake II as a research platform, instead of other FPSs. First, it is open-source, released on the General Public License (GPL); there is a UNIX version available which installs on many platforms [11], and the layout of the code is very modularized and friendly for modification. Another important feature is that the game is able to soft-render. Most newer first person shooters can only render using hardware acceleration, which only allows for one copy of the game to run on a computer at any time. Because Quake II can render in software, we are able to run several copies of the game on one machine, which is essential for quick evolution.

While most Quake II AI-controlled characters (bots) are run on the game server, we have chosen to use the client for our agent control. We modified the source code of the client so that we can access the pixels of the display and fully control the agent in the game. We access the pixels through the video framebuffer, which returns the 24-bit color information for any pixel in the game window, as specified by x, y coordinates. The agent can be controlled to shoot, move forward or backward, move right or left ("strafe", in gaming jargon), turn right or left, look up or down, and jump or crouch. The client can also read messages from the server in order to see player chat or information about who killed who, and it can read the agent's health, ammunition, armor, and number-of-kills figures.

III. The Experiment

As previously stated, the Quake II game world is dark and dreary. Because of this, there is usually not much contrast between enemies and walls, between walls and ceilings, or walls and floors. While human players are usually able to cope with this environment, we decided to start out on something simpler for our first vision-based controller.

We created a simple room map that is square in shape and quite small; relative to humans, about as big as half a basketball court. The ceiling is a dark brown texture, the walls are a gray texture, and the floor is white. The enemy is dark-blue and black, and contrasts very well against the lighter walls and floors, even when converted to grayscale (figure 1). In regular Quake II multiplayer, the players enter the game through spawn portals. We raised the ceilings and put the spawn portals above the play area, so that they would not be in the way of the agents. Whenever an agent or enemy dies, it reappears at a random spawn portal and drops to the floor. In order to lessen the computational burden, we also disabled the display of dead bodies and their scattered body-parts that otherwise would litter the floor.

The goal for the learning agent in this experiment is simply to kill an enemy as many times as possible in the allotted time. The enemy does not shoot, but does move around the room in random sequences. The learning agent is equipped with a blaster weapon, which we have removed

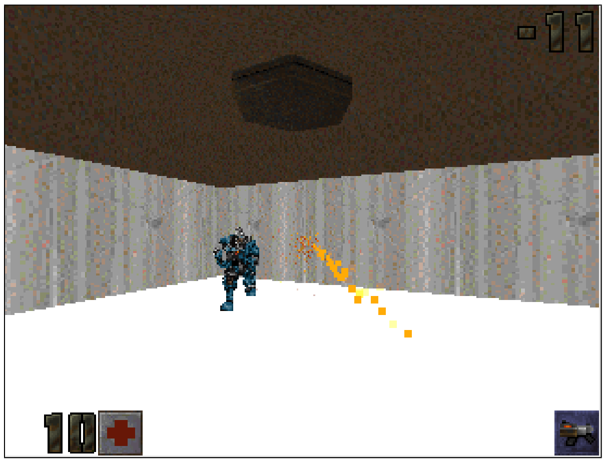

Fig. 1. A screenshot of the simple room used for this experiment. The ceilings are a dark brown textures, the walls are a gray texture, and the floors are white. The enemy is dark-blue and contrasts the light colored walls and floors. The trail of dots indicates the bolts from the learning agent's blaster.

graphically from the screen so as not to interfere with the visual environment. We modified the blaster weapon from the original game to inflict more damage so that one shot kills the enemy, and we made it capable of shooting in bursts. Normally, if the learning agent is constantly shooting, the frequency of shots is about one every half second. However, if the agent refrains from shooting for a time, the gun will charge and will be capable of firing up to 5 shots in a quick-succession burst, depending on how long it has charged. We made this modification so that agents would have a reason to avoid shooting constantly, and only when needed.

IV. The Neuro-Visual Controller

We tested two different visual schemes for our controllers. We could not simply input the entire pixel buffer of Quake II's lowest default screen resolution, 320x240, into a neural network because the network would need 76,800 weights for each hidden-layer neuron, and would be too slow in computation and take too long to evolve. Also, the agent does not need so much information to perform such a simple task, so we reduce the resolution by averaging pixel values together into larger blocks. Also, since the agent is only fighting one enemy, and the enemy appears much darker than the walls and floor, color is not important, so we use gray-scale input.

Because the map is entirely flat, the agent never needs to look up or down. The most important visual information for the agent appears in a horizontal band that runs across the center of the screen. Thus, in order to keep the controller simple and pertinent, we read only a 14x2 band of gray-scale pixel blocks across the center of the screen (figures 2, 3).

The first type of block input layout is inspired by the human retina, which is able to capture visual data of higher resolution in the center of the retina than in the periphery. To make such a graduated density retina, we used a 14x2 band of blocks and made the blocks 23 pixels in height, as with a uniform retina, but the blocks nearer to the center

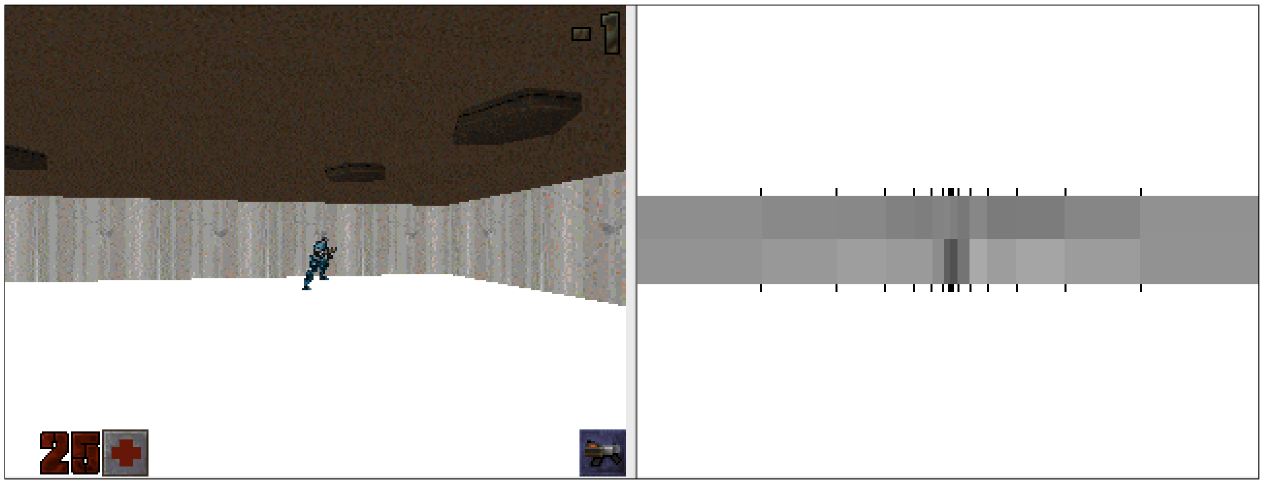

Fig. 2. *Left:* A scene as rendered by the game engine. *Right:* The same scene as viewed via the graduated density retina. The black tics above and below the image indicate the retinal block widths.

encompass smaller amounts of pixels averaged than the blocks nearer to the outer edge; each block away from the center is approximately 1.618 times larger than the previous block. With this system the center views much finer detail than the blocks near the outer edge (figure 2).

The second type of input layout that we tested used evenly spaced blocks that averaged a 23x23 grid of grayscale pixel values into one value for each block (figure 3). Both the band and uniform layouts used 28 blocks, so that each used the same number of inputs.

We rescale the integer grayscale value of the blocks in the 14x2 input array onto floating point numbers in the range $[0, 1]$ for use as inputs to the controller network (figure 4). The controller is a simple recurrent network [12] with 28 inputs, 10 hidden/context units, and 4 output neurons. We use a "flat" hidden layer, with no attempt to capture the input geometry in the network architecture. The neurons are squashed by a scaled logistic function, $S(x) = 2 * \frac{1}{1+e^{-6*x}} - 1$, which scales their outputs to the range $(-1, 1)$. The four output nodes control turning, forward and backward velocity, lateral velocity (strafing), and shooting. For turning, the output is scaled onto the maximum possible per-frame turn rate (10°), and the sign determines the direction of the turn. For the longitudinal and lateral velocity controls, the output activations represent the desired fraction of full speed, with the signs indicating the choices of direction. No bias units were used for the current experiments.

V. TRAINING

For the evolution, we use a Queue Genetic Algorithm (QGA), which is a steady state first-in-first-out algorithm that allows for easy distribution of fitness evaluations over multiple instances of the simulation [13]. A QGA uses a queue to store the population; a new individual is created from stochastically selecting two individuals in the population; once the new individual is tested, it is added to the front of the queue and the oldest individual in the population is removed. In this experiment we use a queue size of

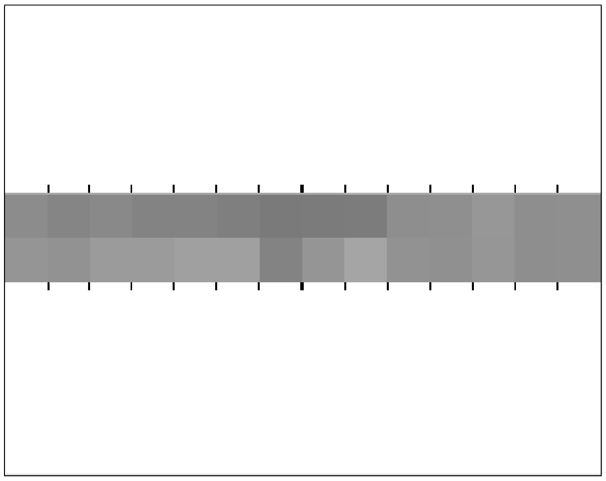

Fig. 3. A view via the uniform density retina. The enemy's location and distance are similar to the view in figure 2. The contrast between the enemy and the walls and floor are much less distinct in the uniform retina than in the graduated density retina, because of the increased area averaged into the visual block where it appears.

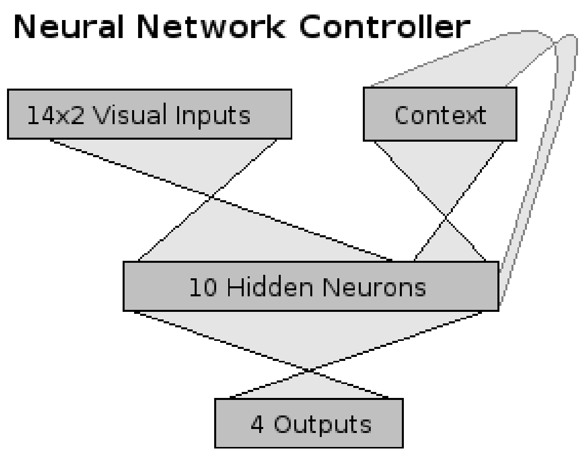

Fig. 4. Diagram of the controller network, with 28 visual inputs, 10 recurrent hidden layer neurons, and 4 outputs.

128 individuals. The 420 weights of the controller network are stored as floating point numbers, which are genes in a chromosome for each individual. To initialize the population, individuals are created with random genes with values in the range $[-1,1]$. Once selection begins, each new individual is formed by roulette wheel selection of two parents from the population, where there is higher chance that individuals with higher fitness will be selected over lower fitness individuals. Crossover is uniform between the two parents, per gene, with equal chance for each parent that its gene will be selected. Mutation occurs with a 10% chance per gene. The mutations are generally very small and are calculated by adding a delta from a sharply peaked distribution function $\frac{\log(n)}{10} * random(-1 or 1)$ to the current value of a gene, where n is some random number in the range $[0,1]$.

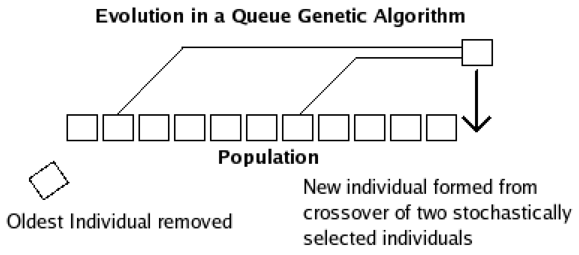
Evolution in a Queue Genetic Algorithm
Population
Oldest Individual removed
New individual formed from crossover of two stochastically selected individuals

Fig. 5. The Queue Genetic Algorithm (QGA). New individuals are bred from parents chosen from the current population by roulette wheel selection. After the new individual is tested, it is placed on the beginning of the queue and the oldest individual is discarded.

Each individual is tested in the following manner:

1) The agent appears in room and drops to the floor.
2) The agent is given 24 seconds 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 seconds, the agent is removed, and a new agent with a new chromosome appears for evaluation.

Fitness is awarded solely by the number of times the agent has killed the enemy. The time-limit of 24 seconds is long enough to allow for skilled individuals to distinguish themselves from lucky individuals, and short enough to complete the evolution in a reasonable time. In order to increase the bias toward the more fit genomes in roulette wheel selection, the number of kills is multiplied by 5 and then squared.

We found it too difficult for a random population to learn to shoot a quickly moving enemy, so we increment the difficulty of the enemy as the evolution proceeds. To do this we first evolved a population against a completely stationary enemy and marked at approximately what average fitness of the population the agents appeared to be actually aiming at the enemy. We set the fitness bar at that fitness, and, starting from new populations, whenever the average fitness of the population reached the fitness bar, the difficulty is increased by increasing the speed of the enemy. After every sixth individual has been evaluated for fitness the average fitness is tested again, and the difficulty level is incremented again if that average has risen back above the current fitness bar. Difficulty is increased by adding 0.01 to the enemy's speed multiplier, which begins at 0.0. The difficulty must be increased 100 times before the enemy reaches its maximum possible speed.

VI. Results

The graduated density retina performed better than the uniform density retina (figures 6, 7). The main behavior learned by controllers that used the graduated density retina is to spin in circles, moving forward, until the enemy appears as a dark dot in the center of its retina, then to stop spinning, or spin in the reverse direction, move toward the enemy, and shoot. Some populations learned to hold their fire until the enemy appeared in their center, and then fire a burst of shots. In the first stages of the evolution, the agents appear to only pay much attention to the center blocks of the retina. Since the agents only compute at 40 frames per second, maximum, sometimes they turn so quickly that the dark spot of the enemy agent skips over the center of the retina, and the agent does not react. However, as the agent becomes more advanced it learns to use the information in the periphery as well.

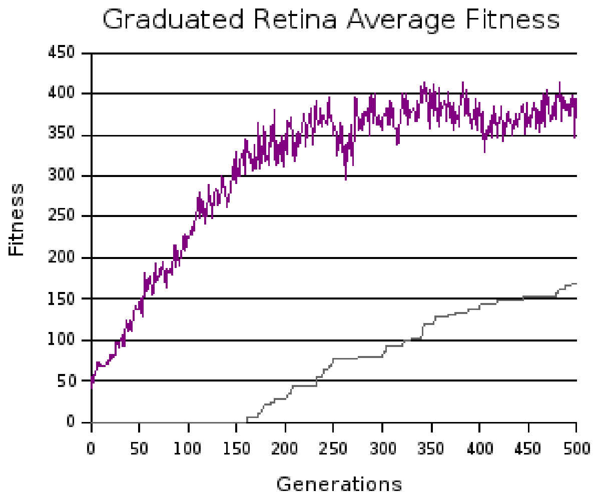

Fig. 6. The population fitness averaged over six independent runs, tested using the graduated density retina (figure 2). The lighter bottom line indicates the increased movement of the enemy, where 300 is the maximum possible enemy movement speed.

The uniform density retina controller performed poorly compared to the graduated density retina controller (figures 6, 7), likely due to its inability to fine-tune its aim. When the enemy is far away the retina's input blocks viewing the enemy turn only slightly darker due to the relatively large visual area being averaged, unlike the blocks of the graduated density retina, which clearly show the enemy as dark when centered, even at a great distance. Moreover, controllers using the uniform retina can not accurately aim because the large block sizes do not indicate the position of the enemy with

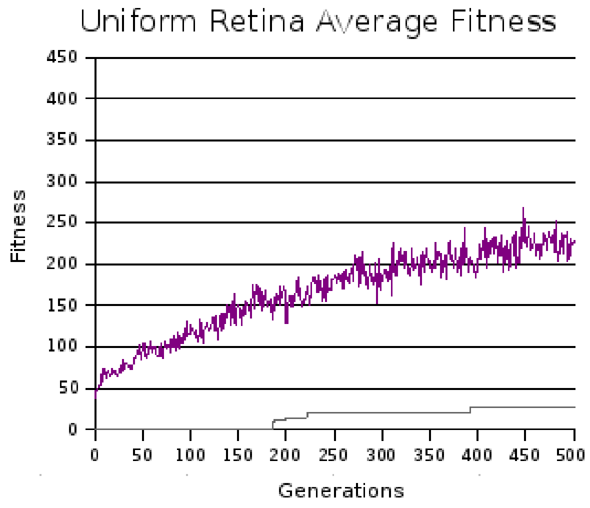

Fig. 7. The population fitness averaged over six independent runs, tested using the uniform retina (figure 3). One of the six tests was about as successful as the average of the dense retina tests, but the other 5 were unsuccessful, never reaching the fitness bar, but hovering at fitnesses between about 200 to 300.

much precision. Thus the training tended to produce agents that shot constantly and moved about the room in patterns that were likely to kill the enemy, without seeming to pay much regard to the actual retinal inputs. In only one of the six tests was a successful behavior learned, which utilized the strategy used by all the dense retina tests: spinning until the enemy came into view, then slowing the spin or reversing the spin and shooting. This successful test was still limited by the large retinal block size, and the agent could not spot the enemy from far away. To counteract this, rather than spinning in tight circles, the bot circled the entire perimeter of the room in a wide spin, to increase its chances of walking near the enemy.

VII. Conclusion

In this experiment, the first-person-shooter Quake II was used to test two different retina layouts as inputs to an evolving neural network to accomplish a simple task. Quake II was modified to appear visually simpler, so that the ceilings, walls, floors, and enemy were all distinct. A genetic algorithm was used to evolve the weights in the neural networks. To test the fitness of the agents, they were placed into a simple single-room map and given 24 seconds to shoot an enemy as many times as possible, given fitness for each kill. At the start of the test, the enemy did not move, but after the agent population reached a certain average fitness bar, the enemy began to increment its random movement speed, and continued to increment whenever the average fitness rose above the fitness bar. Two different retina layouts were tested, each with the same number of block inputs: first, a graduated density retina that is more focused in the center, with smaller blocks that become wider as they near the periphery; second, a uniform retina which uses equal block widths across the width of the view. The results of the tests showed that the graduated density retina learned faster and with more success on average than did the uniform retina. Five out of six of the uniform retinas never reached the fitness bar within the 500 generations tested, while all of the graduated density tests reached the fitness bar several times, and learned to successfully fight against a moving enemy.

This experiment shows that it is beneficial to focus on particular sections of information in the visual field. With the same amount of input information, a focused retina can greatly improve behavior. In our experiment, the focused retina displayed the enemy as a distinctly darker block, even at great distance, and it enhanced the agent's ability to accurately aim at the enemy. In this experiment, setting the focus to the center of the screen was so successful because the agent's gun aims at the center of the screen. However, in vehicle-based road-following experiments, or in other aspects of first-person shooters, it may not be best to focus in the center of the screen, but rather on the side of the road, or on likely pathways. Future work will include evolving where to focus the retina for particular tasks, as well as evolving controllers which dynamically change the focus in real-time. Our modified version of Quake II will be a test-bed for other future first-person-shooter vision-based artificial intelligence research, such as path-finding, team-based combat with a color-sensitive retina, and resource gathering or capture the flag. Such research will expand the realism of artificial intelligence in 3D games, and be expanded to camera-based robotic behavior control in a real environment.

The source code of the Quake II AI software used in this experiment is available at website of the Neuroevolution and Behavior Laboratory at the University of Reno, Nevada: http://nebl.cse.unr.edu/

VIII. 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.

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] 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.

[5] 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.

[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] 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.

[8] 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.

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

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

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

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

[13] M. Parker, and 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.