Learning Control for Xpilot Agents in the Core

Matt Parker, *Computer Science, Indiana University,* matparke@indiana.edu
Gary B. Parker, *Computer Science, Connecticut College,* parker@conncoll.edu

***Abstract*— Xpilot, a network game where agents engage in space combat, has been shown to be a good test bed for controller learning systems. In this paper, we introduce the Core, an Xpilot learning environment where a population of learning agents interact locally through tournament selection, crossover, and mutation to produce offspring in the evolution of controllers. The system does not require the researcher to develop a fitness function or suitable agents to engage with the evolving agent. Instead, it employs a form of co-evolution where the environment, made up of the population of agents, evolves to continually challenge individual agents evolving within it. Tests show its successful use in evolving controllers for combat agents in Xpilot.**

I. Introduction

LEARNING controllers for autonomous agents is a difficult task, but important in the development of adaptive autonomous robots. Using simulated environments for testing learning algorithms significantly helps in their development, provided that the environments have some of the complexity and non-deterministic qualities of real world environments. A computer game such as Xpilot can be used since it offers an environment that is challenging and models relevant characteristics of the physical world. In addition, it offers several levels of complexity while requiring minimal in-game graphics. Xpilot is a 2D network game that is in wide use with several hosts sponsoring ongoing games. The user with internet access and an installed free client program can join a selection of arenas set with a variety of objectives and environmental conditions. The user controls a spaceship (represented as a triangle) that can engage in combat (through shooting bullets) with other ships while attempting to complete some other objective. Although many actions exist, our current research concentrates on control for close combat. Learning the correct behaviors to successfully beat an opponent in this space simulation is very challenging. We hope to have our autonomous agents learn to compete in the world of Xpilot, which is presently occupied by human players with a variety of skill levels and experience.

Games have been used by several researchers to test learning systems based on evolutionary computation. Most of this work has been done on board games [1,2] and on problems like the prisoner's dilemma problem [3]. More related to our work is research that has been done on learning controllers for interactive computer games such as Counter-Strike [4], a first person shooter game, Pac-Man [5], and Tron [6]. In earlier work [7], we introduced the use of Xpilot for the learning of complex control behaviors for autonomous agents. Xpilot was modified to allow researchers to write programs that obtain information about the environment and reply with control responses generated by an artificial intelligent agent. The system was tested by using a standard genetic algorithm (GA) to learn a simple neural network controller [7] and a rule based controller [8].

To make our Xpilot learning environment more usable, we made modifications to the original system, written in C, by creating a Scheme interface, which we call *Xpilot-AI*. This interface allows programs written in Scheme to provide control decisions, which results in easier development and rapid testing of learning algorithms. As a further enhancement, we wanted a training environment where the agents could be tested in parallel. Xpilot sessions involve periods of combat that require time enough for the opponents to interact in positioning, firing bullets, and evading bullets. Although we determined a way to speed up the simulation to seven times its normal speed, any additional speedup could cause lost frames. This would result in controllers that were fit at the high frame rates, but that may not be suited for actual play. We needed the capability of running several combat episodes in parallel to significantly increase the speed of evolution. In addition, a system where we could distribute the clients among several computers would provide additional speed increases in fitness computation. To maintain the desired parallelism, we wanted the evolution (selection, crossover, mutation) to be decentralized. This would allow the system to be asynchronous, which is important since some trials can take ten times longer than others.

In addition to speed considerations, we required reasonable agents for opponents. They needed to be progressively better as our learning agents developed. Co-evolution between two populations was an option, but would require additional computation time to evolve the second population. The system that we developed, which we refer to as *the Core*, addresses this issue with no additional computation time by creating an environment where the other agents in the learning population are opponents who are constantly increasing in their capabilities. In this way, the environment as a whole is co-evolving with the individual agents who are learning in it. The Core also allows us to avoid developing a fitness function to achieve the desired behavior since fitness is determined solely by

how well the agent fights in the environment. These factors combine to make a unique learning environment that produces excellent results.

In this paper, we report the advancement of our previous work through the use of the Core to learn controllers for Xpilot agents. The development of this a unique parallel learning environment was in response to the difficulties we found in using the standard system for evolution. This paper will present details of the Core's implementation and will show the results of a test conducted where a cyclic control program for an autonomous combat agent was evolved.

II. THE CORE

A. Description

The Core is a large virtual arena in which a population of Xpilot agents can learn combat controllers. It is a parallel system that accommodates several participating computers. Because Xpilot servers are designed to allow for a wide variety of capabilities, the client computers do not have to be equivalent in capabilities. A client computer's capabilities merely dictate the number of clients that can be run concurrently on that computer. However, since each client is an autonomous entity in itself, no coordination in the number of clients run on each computer is required. It is plausible, with a fast connection, that computers anywhere in the world could be contributing clients to the Core. The computer acting as the server can also have a limitation. This limitation is placed on the system by the hardware and operating system on which the Xpilot server runs. However, as hardware becomes faster, more clients will be able to concurrently connect. The Core has a spatially-distributed population. Parameters within the system can be adjusted to keep newly produced offspring in the local area or disperse them anywhere in the environment. The Core's type of parallelism would most closely identify it with a fine-grained parallel GA that is asynchronous [9,10].

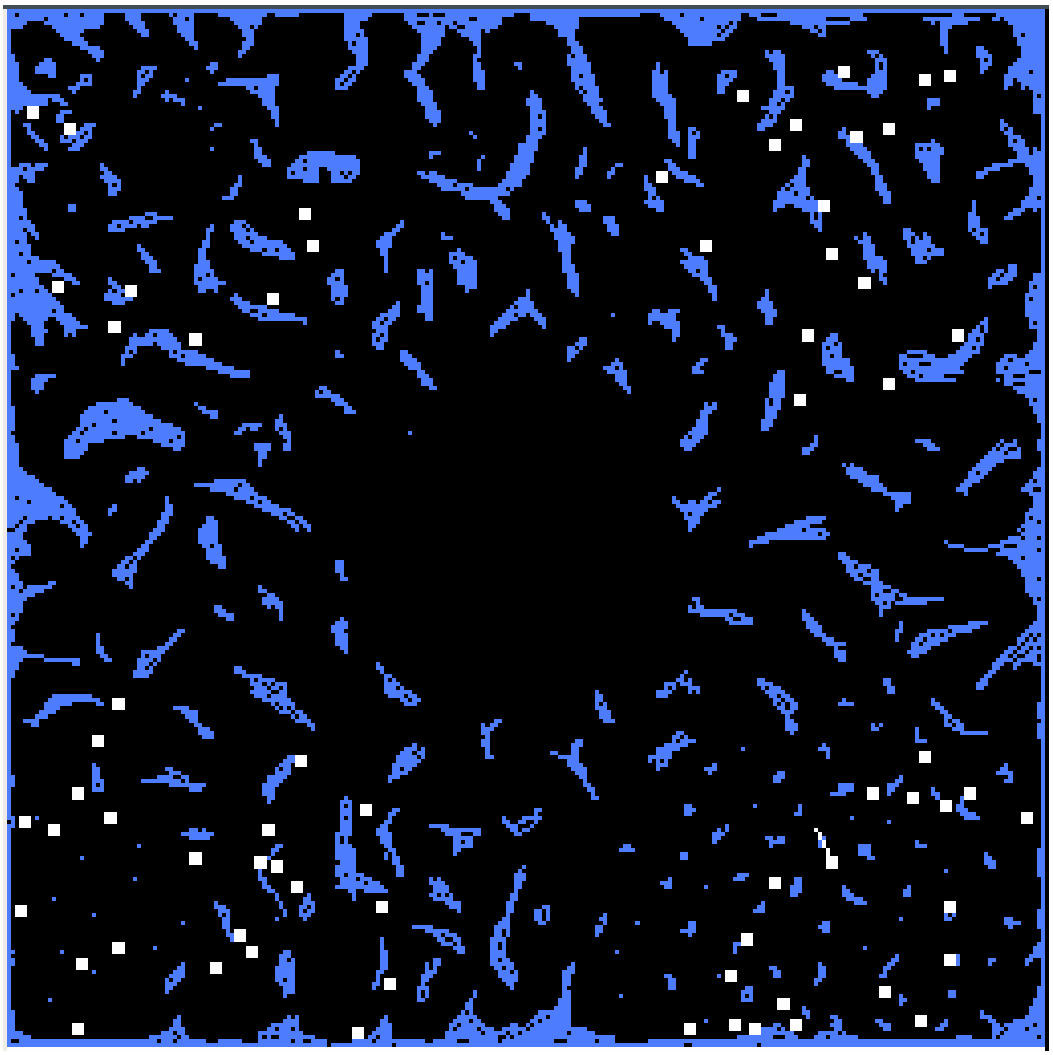

Fig. 1. A radar screenshot of the populated Core. The white dots are individual agents; most of which are engaged in combat. Many more agents are dead and waiting to appear, so do not show on the map.

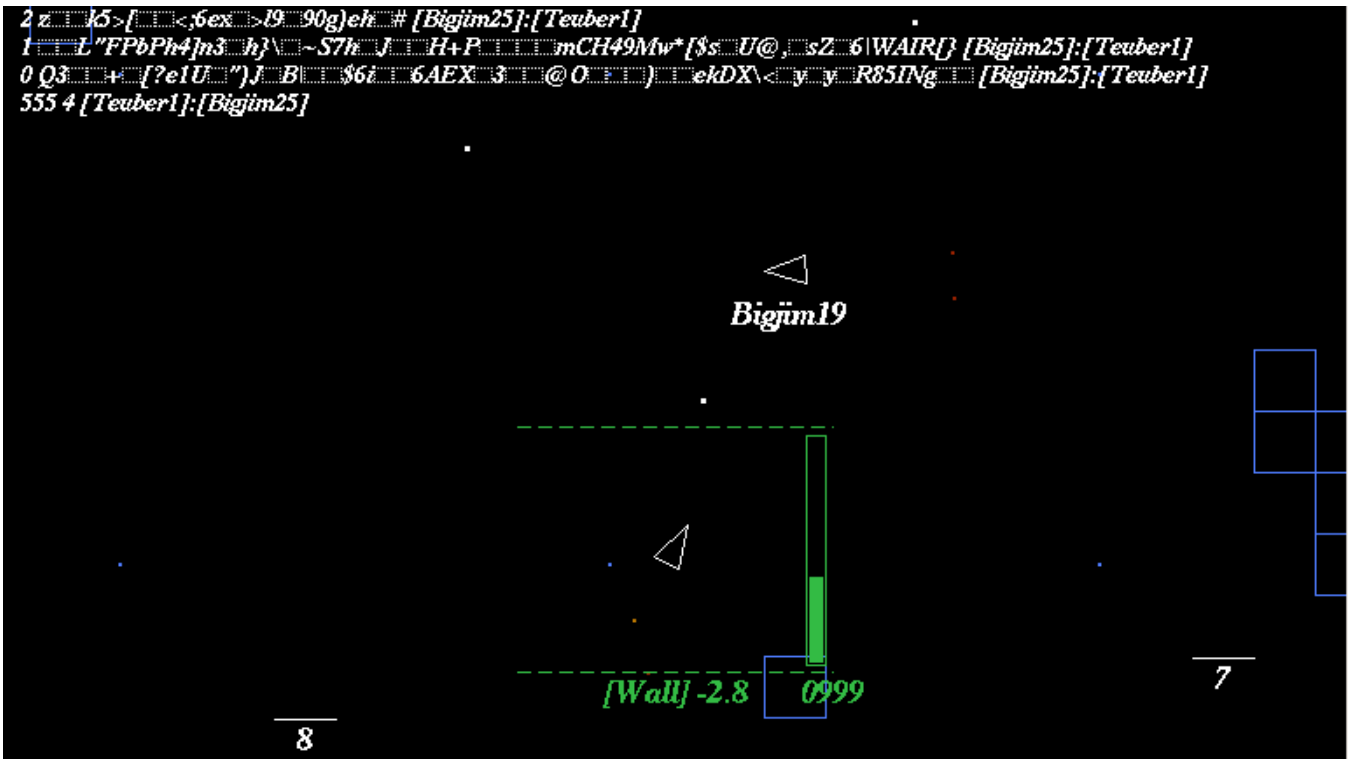

Fig. 2. A view of the action in the Core. Here Teuber1 is firing at Bigjim19. The strange characters atop are the encoded chromosome of Bigjim25, who recently killed Teuber1, being passed over the player-to-player messaging system.

The Core area for competition is a large 256x256 tiled map (Figure 1). Each tile is about 3 times the area of a ship. Each of the four quadrants of the map has a unique terrain in order to represent multiple combat environments. The top left quadrant has fingers of walls with an open space in the center, the top right quadrant has fingers with no large space, the bottom left quadrant is cluttered with small asteroid-like chunks of walls, and the bottom right quadrant has just a few dots of walls and mostly open space. Bases, where a player or agent may appear upon a new life, are scattered in the center of each quadrant. The four quadrants collect towards the center where there is a large circle of empty space. Combatants within each quadrant interact mostly with agents within that quadrant, but also interact with the agents in adjacent quadrants at the borders and the central empty space.

When a client first joins the Core, the client generates a random chromosome to control the agent. Therefore, when the Core is first loaded the entire population is made of agents with random chromosomes. Whenever an agent dies, its ship disappears from the map for about 32 frames, then it reappears in a new starting location, and the agent lives again. In our current experiments, this new starting position is randomly generated. This is intended to train for a generalized solution. However, it is possible to keep the agent in its local area. One possibility is to have it always reappear within its starting quadrant. This allows for a very natural method of niching.

There are four ways that an agent can die in our current system: collision with a wall at sufficient speed, collision with another agent, killed by its own shot, or shot by another agent. Of these four deaths, the first three simply result in the agent's death and reappearance in another location; its chromosome is unchanged. However, when it is shot by another agent, the dead agent sends a request to the killer agent, the killer sends its chromosome to the dead agent, and the dead agent performs crossover and mutation between its own chromosome and the chromosome of its killer. The messages requesting the chromosome and the chromosome itself are coded into ASCII and sent directly between the agents using the player-to-player text messaging system native to Xpilot (Figure 2). In this way, the agents are not dependent on a central system for recombination.

Using this form of selection and recombination, the agents who are more capable of killing other agents spread their genes to a larger number of agents than do those agents who are less capable of killing. In turn, the weaker agents who are killed, slowly evolve to become as strong as their killers, increasing the abilities of the entire population. By crossover and mutation new traits are formed in agents, and the traits that make the agent stronger are slowly spread across the entire population. In this way, the agents

continually increase their individual abilities while the entire population continues to evolve to be a more competitive learning environment.

B. Setup

With modifications, a dedicated Xpilot server is capable of hosting a very large number of Xpilot clients (we typically run 120). The normal Xpilot-NG server, which accepts a maximum of 32 clients, was modified to accept a virtually limitless number of clients. In addition, the number of informative messages that the server sends to its clients, such as those about who killed who, or about who switched teams, etc. was reduced to a minimum. These messages are not important for the agents in the Core and they use extra network bandwidth that is needed to pass chromosome information.

The Xpilot server needs to be on a reasonably fast computer running a reliable operating system. We tested several computer / operating system setups to find one appropriate for the server. A Sun AMD Opteron 64 bit and a PC with a Pentium IV, both with over 512MB of memory were found to be suitable computers. A Sparc Sunblade 1500, however, was apparently too slow to handle the large number of clients. The operating system for the Xpilot server needs to be reliable when handling many concurrent network connections. We tested FreeBSD, Linux, and Solaris. FreeBSD is the most reliable of the three, able to host around 130 Xpilot-AI clients at 32 frames per second. At the same frame rate Solaris hosts about 115 reliably, and Linux, using a 2.6.x kernel with Fedora, could host 122 clients. For the tests reported in this paper, we ran the Xpilot-AI clients on dual processor Xeon computers, each with 1-3 Gigabytes of memory. The graphics of the clients were run in a low color-depth X Video Frambuffer (Xvfb) rather than on the graphics display, to reduce the strain on the CPU. With these computers and settings we ran about 25 clients on each computer. Four or five computers were sufficient to populate the Core.

C. Learning

The Core can accommodate most any type of evolutionary computation for the agent controller learning system. For the research reported in this paper, we used a multi-loop cyclic genetic algorithm (CGA). The original CGA [11] was developed to learn a single loop control program for locomotion in hexapod robots. It is a modification of the standard GA in that the genes (logical groupings of bits) represent tasks to be completed in a set amount of time as opposed to traits of the solution. Using a CGA, single-loop control programs with variable lengths could be evolved. In further research [12], the CGA was expanded to be capable of evolving multiple loop programs (allowing conditionals) and was employed to learn the control program for a robot with light and touch sensors.

The CGA used for the research reported in this paper is unique. The chromosome is made of 16 loops, with 8 genes per loop and 8 bits per gene. Each loop has a set of conditionals directing the program execution and a set of instructions directing the agent's actions (Figure 3). The last five genes in each loop are devoted to controlling the actions of the ship. While the agent's control program is in that particular loop, it will cycle through these five action genes frame by frame, performing the actions as instructed by the genes. The first bit in the action gene instructs the ship whether or not to thrust. The second bit, whether or not to shoot. Three bits instruct the ship's quantity of turn and three bits correspond with where to turn, such as from the nearest enemy or towards the nearest wall.

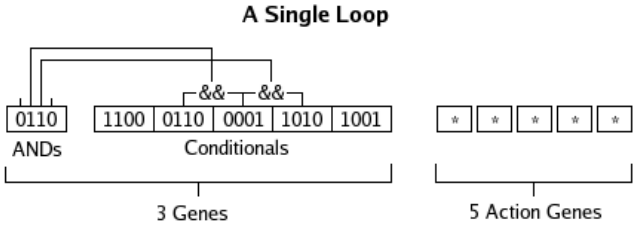

Fig. 3. Diagram of an example single loop of the 16 total. The first three genes are devoted to conditionals, which if true, results in a jump corresponding to that conditional. The 5 action genes are cycled through frame-by-frame to control the ship.

The first three genes of each loop are devoted conditionals which instruct the agent's program if it should jump out of the current loop into a new loop. Each conditional consists of four bits (half a gene) and corresponds to one of sixteen conditions about the environment, such as "agent velocity > 10" or "enemy distance < 100". Each of the conditionals has a corresponding loop, which the program control will jump to if the condition is true. Between every frame, before the agent's control program executes an action, it checks all the conditionals for whatever loop it is in and jumps to the corresponding loop of the first conditional that is true, or if none are true, continues with the next action gene in its current loop. To give the conditionals more versatility, we turn the first four bits of the first gene into AND bits. The first AND bit will act as an AND between the first two conditionals. If all the AND bits are enabled, then all five conditionals will be AND'ed together and the program will not jump out of that loop unless all are true. If an entire series of AND's are true, then the program will jump to the loop that corresponds with the last conditional.

D. Adjustments

The goal is to evolve controllers that equip the Xpilot agents to excel at general combat. Genetic algorithms often find ways to exploit the fitness function so that good fitnesses result without the agent achieving the intended behavior. The Core does not have a "fitness function", but the setup of the map and rules of play have an enormous impact on the evolved behavior of the agents. For example,

our first tests of the Core gave the agents almost no reason to avoid crashing into the walls, so thrusting toward a wall was actually a good strategy to avoid being shot. Being shot resulted in a chromosome change, whereas dying in a wall crash did not. Consequently, the entire population soon became fast-flying agents that were not concerned with dying in wall collisions.

For the tests reported in this paper, we handled the learning agents' propensity for wall-collision deaths by penalizing them upon being reborn after a wall collision. If an agent collides with a wall, that agent then reappears, like normal, at a random starting location, but instead of being allowed to fly immediately, it must sit motionless for a certain amount of time. During this penalty time the agent is defenseless against any enemy opponents and makes easy prey for any nearby ships. This effectively weeds out from the population those agents that carelessly collide with walls.

E. Previous Work Relevant to the Core

Although the Core has only one population, we consider it to be competitive co-evolution. Instead of two distinct populations, it has an environment that is constantly evolving to be more hostile to the individual agents within it. Significant work has been done in the field of competitive co-evolution by several researchers [13,14,15,16,17]. One of the important issues with competitive co-evolution is the problem of having one population dominate the other [13,17] resulting in stalled evolution. This is not an issue with the Core. The body of agents making up the environment for testing individual agents cannot evolve to be constantly superior to the individual agents since it is made up of those individuals. On the other hand, individual agents that evolve to be superior pass their genes on to defeated foes and their traits eventually spread throughout the population resulting in a more challenging environment.

Using a fitness function that is a property of the environment as opposed to a predefined fitness function has been used in Alife simulations [18,19]. This is also a property of *embodied evolution* [20] used in evolutionary robotics. In this system, robots attain a virtual energy level that corresponds to their performance in completing an assigned task. They transmit their chromosomes with a rate proportional to their energy level and receiving robots accept it at a rate inversely proportional to their energy level. Upon acceptance, the receiving robot crosses it with its own chromosome producing a new controller, which overwrites its current controller. Tournament selection using only two opponents with the least fit being replaced by the recombination of the two was found to be a successful strategy (Microbial method) by Harvey [21]. We use a similar method in the Core. When an agent kills another agent, its chromosome is passed to that agent's client. The client crosses the victor's chromosome with that of the defeated agent's chromosome, producing a new agent that is sent back into the Core for continued learning.

III. TESTING THE CORE

Recording the fitness growth of individuals in the Core is a unique issue. Since there is no fitness function, it is difficult to measure the success of the evolution, even though one can visually observe that the agents have significantly improved in their development of effective strategies. Their success cannot be measured by their time alive or by their number of kills in the Core. An individual agent's competitors increase their ability to kill and avoid being killed as the agent increases its ability to do the same. The fitness of the agents from the Core must therefore be judged by their ability to fight some standard opponent, whose behavior remains the same over time. This can be accomplished in one of two ways: introduce standard opponents into the Core, or remove agents from the Core to fight standard agents in a new arena. For the tests reported in this paper we have recreated a smaller version of the Core, approximately one fourth the size, and populated it with 12 enemy bots. Rather than sticking one particular type of enemy bot into the test map, we use controllers saved from random agents pulled from the Core at equal intervals of time. The enemy bots in the miniature Core do not evolve or learn, so they effectively reproduce the average conditions experienced by the agents throughout their lives in the Core.

The Core was allowed to run for 24 hours. The clients saved each agent's chromosome to file every 30 minutes, so 48 chromosomes were saved for each agent in the Core. We randomly picked five individuals from each saved population and tested each in a separate miniature Core. In this way, we tested every chromosome of the 48 saved for each individual. Each chromosome was allowed control of a ship until it had been killed by an enemy ship ten times. As it was in the Core, if the agent crashed into a wall, it was penalized upon its next life, being forced to sit motionless for an extent of time, but a wall collision did not count as one of the ten deaths. We recorded the number of times the agent killed the enemy and how many times the agent crashed into a wall over the span of its ten lives. This gave us a clear idea of the general improvement in behavior and efficiency of killing for the agents in the Core over time.

IV. RESULTS

The agents in the Core clearly show visible improvement over the 24 hour test. The number of kills per ten lives (Figure 4) improved over time in the miniature Core simulation. The number of times the agents crashed into the walls over time (Figure 5) also was reduced as the agents learned more conservative thrusting patterns.

Behavior in the Core was never completely homogenous; the average of the agents grew better over time, but there were always some with superior controllers and others with poor controllers. Some agents became good aimers, others perfected a style of wall-crashing, and still others had an inferior strategy because of a bad mutation or detrimental crossover. Since the 5 agents tested in the miniature Core were a random sampling taken every 30 minutes, some of the chromosomes sampled at the 30 minute intervals were less or more intelligent than the average. For this reason, the graphs spike up and down. However, they show overall improvement, mirroring the general improvement of the population.

The improvement in behavior of the agents can also be visibly seen by watching them fight in the Core. When first started, the majority of the population flew wildly about the map, some shooting, some turning, and mostly all dying very quickly by a wall collision. After a time, those who were flying less quickly, and those who had some ability to aim at the enemy, began to kill more than the others, and their genes began to spread across the map. Soon everyone was flying more slowly and aiming more. Eventually the Core was dominated mostly by ships that would merely aim and shoot at the nearest enemy, not attempting to avoid an oncoming wall or dodge an enemy bullet. Very rarely one of the agents appeared to be consistently thrusting away from oncoming walls, but after being killed by a less advanced agent, it would lose that ability, and apparently it did not spread its genes successfully enough while still alive for the entire population to learn the trait. By the end of the Core test, many of the agents had learned to slowly thrust while they were aiming and shooting at the nearest ship. Their acceleration was only about double the backwards acceleration caused by firing bullets, so their flight was slow enough to put them in no major danger of dying from crashing into a wall and fast enough to make them a more difficult target to shoot.

Just as the entire population visibly evolves over time, a new random agent placed into a mature Core also visibly evolves over a span of a few minutes. When it first joins, it appears to fly in unintelligent flight patterns, usually not heeding the enemy opponents. Then, after it is shot by an enemy, it changes into a strange mix of its previous wild behavior combined with a faint ability to aim at the enemy. After a few more deaths from the more advanced enemies, it becomes nearly indistinguishable from the other agents in the Core.

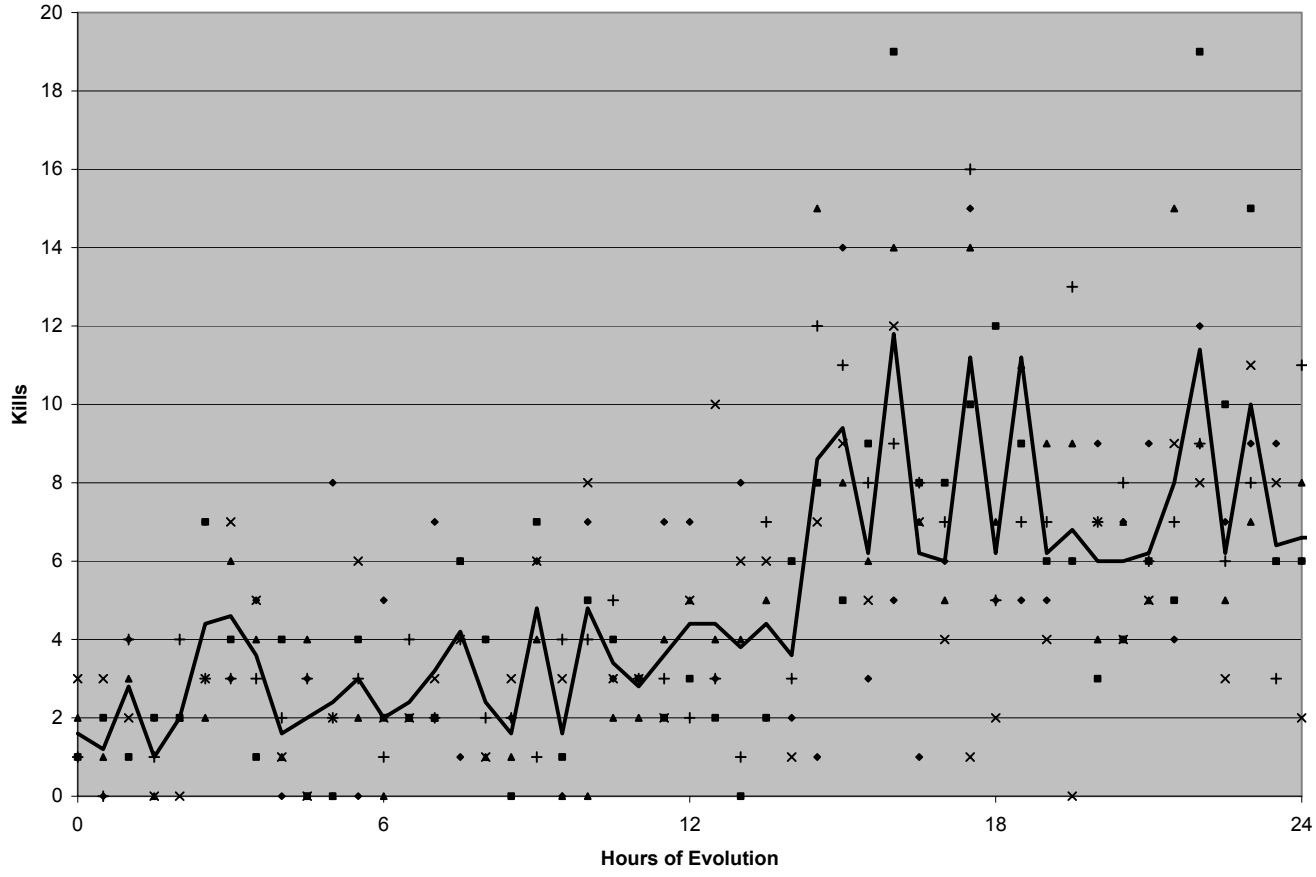

Fig. 4. Results of 5 tests showing enemy kills per 10 lives. All 5 tests are shown as points; the average is shown as a solid line.

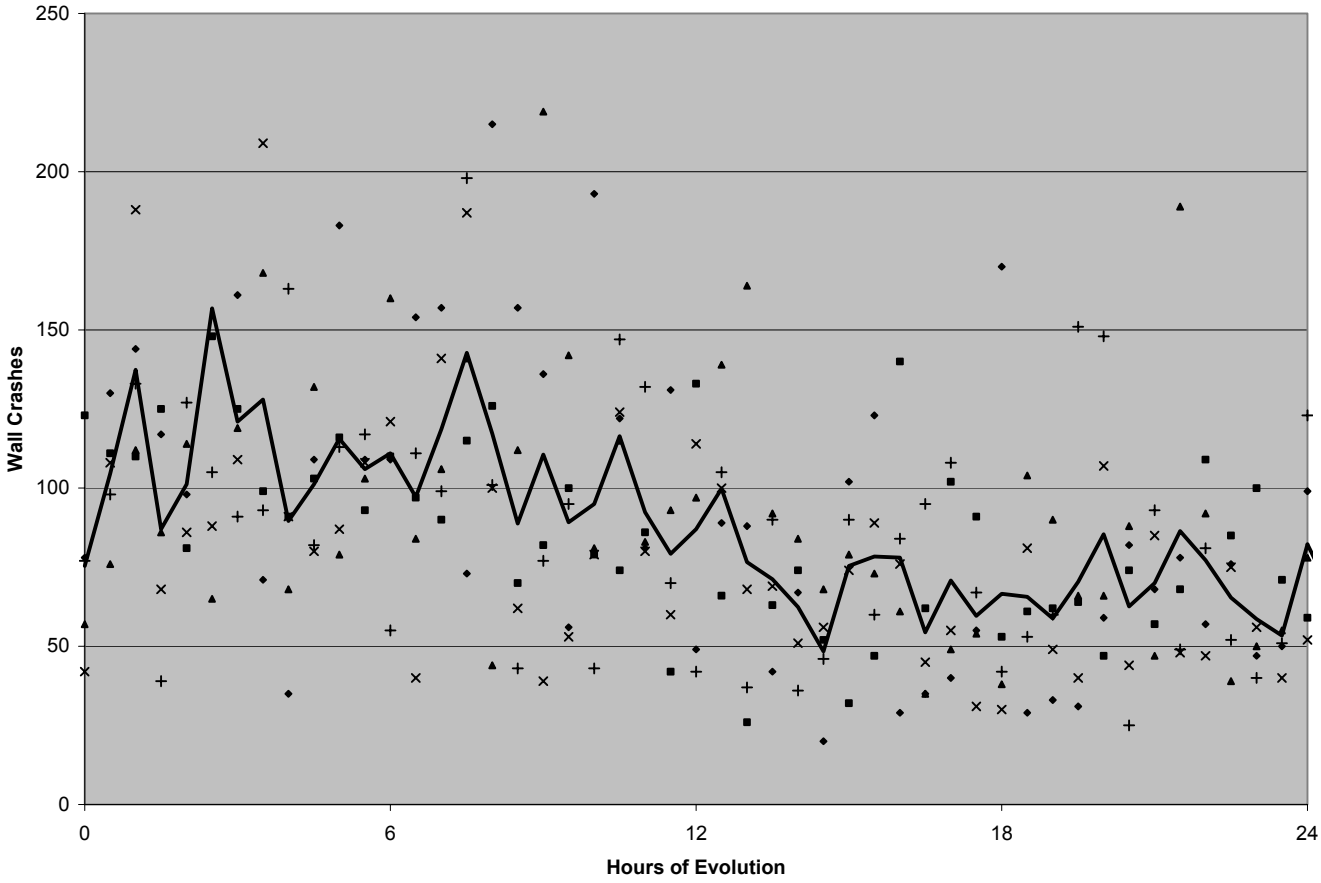

Fig. 5. Results of 5 tests showing fatal wall crashes per 10 lives. All 5 tests are shown as points; the average is shown as a solid line.

V. Conclusions

Xpilot offers a challenging environment for testing learning systems for autonomous agents. The Core allows the researcher the flexibility of evolving a controller that will be useful in the wide variety of available combat environments. It avoids the need for the creation of a fitness function that adequately tests the large number of tasks in general combat and an environment populated with hand-coded opponents that reliably test the agent. The Core is highly parallel, employs simple single population co-evolution, and can be run on several dissimilar computers. Tests show that it is effective for evolving robust combat agents in the Xpilot environment.

There are several modifications that can be made to the Core which change its characteristics. The rules of play, the map, and the clients can be altered to change the nature of the evolved controllers. In future work, we will investigate where multiple species of agents compete against one another, fighting to become the dominant population in the Core. The idea of the Core could have general applicability to the evolution of competitive agents, including the evolution of non-combatant agents such as controllers for racing cars. In a Core like environment, the dominant cars can send their chromosome to the less successful cars as they are lapped.

Tests show that the Core is a suitable method for evolving combat agents in Xpilot. Future work will expand its use with other forms of evolutionary computation and test its general applicability in the evolution of controllers for competitive agents.

References

[1] Fogel, D. *Blondie24: Playing at the Edge of AI*, Morgan Kaufmann Publishers, Inc., San Francisco, CA., 2002.

[2] Konidaris, G., Shell, D., and Oren, N. "Evolving Neural Networks for the Capture Game," *Proceedings of the SAICSIT Postgraduate Symposium*, Port Elizabeth, South Africa, September 2002.

[3] Hingston, P. and Kendall, G. "Learning versus Evolution in Iterated Prisoner's Dilemma," *Proceedings of the International Congress on Evolutionary Computation 2004 (CEC'04),* Portland, Oregon, 20-23 June 2004, pp 364-372.

[4] Cole, N., Louis, S., and Miles, C. "Using a Genetic Algorithm to Tune First-Person Shooter Bots," *Proceedings of the International Congress on Evolutionary Computation 2004 (CEC'04), Portland, Oregon*, 2004, pp 139–145.

[5] Yannakakis, G. and Hallam, J. "Evolving Opponents for Interesting Interactive Computer Games," *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04); From Animals to Animats 8*, 2004, pp 499-508.

[6] Funes, P. and Pollack, J. “Measuring Progress in Coevolutionary Competition,” From Animals to Animats 6: *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*. 2000, pp 450-459.

[7] Parker, G., Parker, M., and Johnson, S. “Evolving Autonomous Agent Control in the Xpilot Environment,” *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005),* Edinburgh, UK., September 2005.

[8] Parker, G., Doherty, T., and Parker, M. “Evolution and Prioritization of Survival Strategies for a Simulated Robot in Xpilot,” *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK., September 2005.

[9] Cantu-Paz, E. “A Survey of Parallel Genetic Algorithms,” Technical Report 97003, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, Illinois, 1997.

[10] Tomassini, T. “Parallel and Distributed Evolutionary Algorithms: A Review,” *Evolutionary Algorithms in Engineering and Computer Science*, pages 113–133. J. Wiley and Sons, Chichester,1999.

[11] Parker, G. and Rawlins, G. “Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots**,”** *Proceedings of the World Automation Congress (WAC '96), Volume 3, Robotic and Manufacturing Systems*. May 1996.

[12] Parker, G. and Georgescu, R. **“**Using Cyclic Genetic Algorithms to Evolve Multi-Loop Control Programs**,”** *The 2005 IEEE International Conference on Mechatronics and Automation (ICMA 2005),* Niagara Falls, Ontario, Canada, July 2005.

[13] Bongard J. and Lipson, H. "Nonlinear System Identification using Co-Evolution of Models and Tests," *IEEE Transactions on Evolutionary Computation*, 2004.

[14] de Jong, E. “The Maxsolve Algorithm for Coevolution.” Proceedings of the Genetic and Evolutionary Computation Conference, 2005.

[15] Popovici, E. and De Jong, K. “Relationships between Internal and External Metrics in Co-evolution,” *Proceedings of the Congress on Evolutionary Computation -- CEC-2005*.

[16] Rosin, C. and Belew, R. "New Methods for Competitive Coevolution." *Evolutionary Computation*, 1997, 5(1):1–29.

[17] Williams, N. and Mitchell, M. “Investigating the Success of Spatial Coevolutionary Learning,” *Proceedings of the 2005 Genetic and Evolutionary Computation Conference, GECCO-2005* , New York: ACM Press, 523-530, 2005.

[18] Ventrella, J. “Attractiveness vs Efficiency (How Mate Preference Affects Locomotion in the Evolution of Artificial Swimming Organisms),” *Artificial Life VI*, MIT Press. 178-186, 1998.

[19] Werner, G. and Dyer, M. “Evolution of communication in artificial organisms.” *Artificial Life II*, Addison-Wesley. 659-687, 1991.

[20] Watson R., Ficici S., and Pollack J., “Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots”. *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 335-342, 1999.

[21] Harvey I. “Artificial Evolution: A Continuing SAGA,” In *Evolutionary Robotics: From Intelligent Robots to Artificial Life*, Takashi Gomi (ed.), *Proceedings of the 8th International Symposium on Evolutionary Robotics (ER2001)*. 2001.