10-2012

Evolving Expert Agent Parameters for Capture the Flag Agent in Xpilot

Gary Parker
Connecticut College, parker@conncoll.edu

Sarah Penrose
Connecticut College, spenrose@alumni.conncoll.edu

Recommended Citation

Parker, G.; Penrose, S., "Evolving expert agent parameters for capture the flag agent in Xpilot," Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on , vol., no., pp.791,796, 14-17 Oct. 2012 doi: 10.1109/ICSMC.2012.6377824

Evolving Expert Agent Parameters for Capture the Flag Agent in Xpilot

Keywords
Xpilot-AI; evolutionary computation; autonomous agent; genetic algorithm

DOI10.1109/ICSMC.2012.6377824

Evolving Expert Agent Parameters for Capture the Flag Agent in Xpilot

Gary Parker and Sarah Penrose
Department of Computer Science
Connecticut College
New London, CT, USA
parker@conncoll.edu, spenrose@conncoll.edu

***Abstract*—Xpilot is an open source, 2d space combat game. Xpilot-AI allows a programmer to write scripts that control an agent playing a game of Xpilot. It provides a reasonable environment for testing learning systems for autonomous agents, both video game agents and robots. In previous work, a wide range of techniques have been used to develop controllers that are focused on the combat skills for an Xpilot agent. In this research, a Genetic Algorithm (GA) was used to evolve the parameters for an expert agent solving the more challenging problem of capture the flag.**

Keywords – Xpilot-AI; evolutionary computation; autonomous agent; genetic algorithm

I. Introduction

Video games are popular for entertainment and many can be used to test autonomous agent control programs. It is desirable that these agents are competitive so that the human player is challenged. Creating competitive non-player agents can be difficult, especially if a diversity of controller or agents that can adapt are desired. One method of learning that can create distinct competitors and has the potential for adapting to different players is Evolutionary Computation (EC). Video games can make viable test beds for researching different methods for evolving control programs for autonomous agents including actual robots. The video game used for this research, Xpilot, is an excellent environment due to its low computational requirements and the Xpilot-AI add-on that allows a user to create an agent to control ships during the game [1].

Many researchers choose to use video games to test various artificial intelligence techniques because they offer a large dynamic environment. These environments make the learning that is required for an evolved agent to compete seriously in these games much more challenging. The resultant evolved solutions for video game agents are also often more versatile, because they are not hard-coded, and may develop strategies a human programmer would never consider.

Research with learning control for video game agents has a long history. In the past, researchers have evolved agents for the video game Quake3, a 3-dimensional multiplayer game [2]. They used genetic algorithms to evolve an agent that preformed better than the bot provided with the game, and then went on to co-evolve opponents to use instead of the provided bot. In other work, the game Pac-Man was used to consider a predator-prey scenario [3]. In this case, the connection weights for a neural network were evolved by a GA to serve as the control for the predators' team behavior. Another example is the development of an agent to play a real-time strategy game known as DEF-CON through a combination of artificial intelligence techniques such as simulated annealing, decision tree learning, and case-based reasoning [4].

The goal of the research reported in this paper is to create a competitive agent for Xpilot, a 2D combat-based video game. Many experiments to evolve agents for the Xpilot environment have been completed in the past, although all have concentrated on the development of agents for a combat role. The parameters for a combat expert agent were successfully evolved [5], using a standard GA. Other artificial intelligence strategies, such as multi-layer neural networks [6] and reinforcement learning [7], have also been used.These approaches have all been successful strategies to make a better Xpilot combat agent. In most of these works, the agent developed was far superior to those provided by the game.

The research reported in this paper differs from previous work in Xpilot because it explores the capture the flag game mode rather than the traditional combat scenario. The typical combat style of game play in the Xpilot environment has agents that may be controlled by either a computer script or a human player. In the capture the flag mode, which has yet to be researched, agents are still controlled by scripts or humans in a combat environment. However, the goal is no longer to destroy their opponents, but instead to retrieve the opposing team's flag, a large ball, and drag it to a goal to receive points. The control for this type of play is more difficult, as humans playing the game can easily attest. The ball is massive and difficult to drag by a ship, which has much less mass. In this paper we discuss the evolution of an agent whose goal is to move toward the ball, pick it up, and transport it to the goal. An expert agent was written and a GA was used to evolve the parameters for that agent.

II. Xpilot-AI

Xpilot-AI is a modification of the computer game Xpilot, a 2-dimensional space game (Figure 1), most often played in a combat scenario, but with the capabilities for capture the flag and racing. Xpilot has server and client and client components, with the server handling the game-wide configuration and is the central source for all information about each ship and object in the game. The client gets this information from the server, accepts the commands of the user and relays them to the server. This allows the user to thrust, shoot, and pick up objects. Xpilot-AI works in between the client and the server, so that the information about the game is available to a programmer. A typical Xpilot server runs at 16 frames per second. Between each frame, the server and client exchange information, and Xpilot-AI intercepts the communications to obtain necessary information for the user. The programmer can write scripts that get info from functions Xpilot-AI provides, use this information to determine what actions to take depending on the situation, and set variables describing this action that the client sends back to the server.

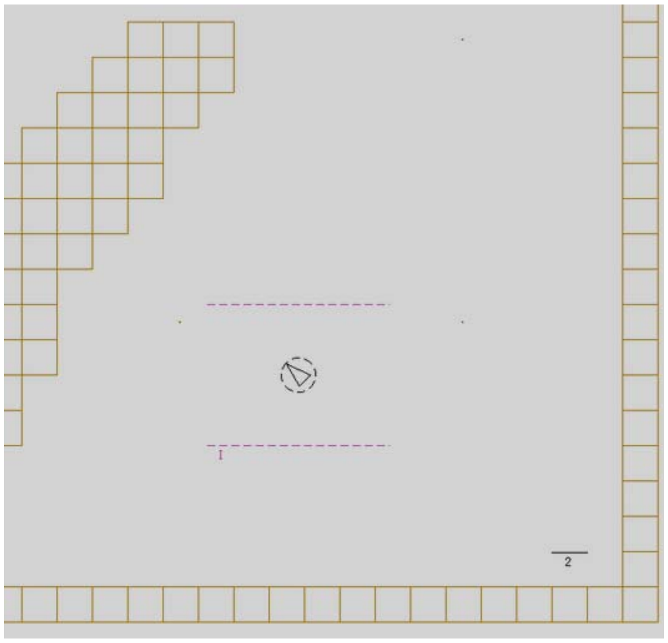

Figure 1. The Xpilot Environment. The ship is shown in the center in black, and is surrounded by a shield. The green blocks along the edges and in the upper left are walls, and in the lower right hand corner there is a base (the two signifies that it is a base for team two). The background is typically black, with a white ship and blue walls, but the colors have been inverted to save printer ink and so that the details can be seen.

Xpilot-AI provides many functions to the programmer that give information about the current state of the game. For example, the speed of the ship and its location are available, as well as the speeds and locations of any other ships currently playing in the map. The programmer also has the ability to get the location and heading of the bullets that are in play, and the ability to determine if there is a wall between two points. The programmer can make use of this information in a control program that determines when to determine when to turn, shoot, and thrust. These control programs can take many forms, but one means of writing a control program is to write a rule-based system, using rules such as "if there is an enemy close by, then turn toward the enemy and shoot at the enemy." This rule would check the functions provided by Xpilot-AI to determine if the agent is close and then call functions that cause the agent to turn and to shoot.

III. Genetic Algorithms

A Genetic Algorithm is a form of computational intelligence (a subset of artificial intelligence) that attempts to mimic natural selection and heredity [8]. In the natural world, a species' traits can shift over generations of time in response to changes in the environment. The individuals that are best suited to deal with those changes are more likely to reproduce, and thus those traits are passed to the next generation. A genetic algorithm attempts to replicate this phenomenon to find the solution to a problem.

To initialize, a population of random chromosomes is created. A chromosome represents a possible solution to the problem, and is typically represented as a string of binary digits. Each chromosome, or individual, of the population is evaluated to determine how well it solves the problem. The next population is generated by recombining individuals in the current population. To accomplish this, two parents are chosen from the current generation. This selection process has a random element, but chromosomes that preformed better during evaluation have a greater chance of being selected. A common method of selection is called roulette wheel selection, in which each member of the population is assigned a portion of a roulette wheel (circle), with the size of the portion dependent on the fitness. A random point on the circle is determined, and whichever chromosome has that point of the circle inside of its designated area is selected. Two distinct parents are selected in this way and are recombined together to create a new individual that will be placed into the next generation.

A standard method of recombination, known as crossover, is done by choosing a random point from within the chromosome. From the first chromosome, everything is taken up to that point, while the second chromosome supplies the latter part of the new chromosome. These two sections are then joined together to create a new chromosome. Once the chromosome has been generated, there is a chance for it to undergo mutation. Mutation is included because it occurs in natural evolution, even though it is typically detrimental. In biology, mutation rarely produces a more viable offspring because there are typically many more opportunities for the mutation to cause harm rather than better the child. Because mutation is not often beneficial there is a low probability, for example, one in one three hundred, of it happening. Each gene in the chromosome, each one or zero, is considered in turn and has a chance of being flipped. A mutation could turn the chromosome '101' into '100' if the last bit happened to have been selected.

Although mutation is typically bad, in some rare instances it can create a better or different solution. Consider the case when a population of chromosomes has the identical bits in one spot, i.e. every chromosome has a zero for the fifth bit. When crossover is done in this population, there is no possibility of getting the zero in the fifth bit to change into a one. However, through mutation, this gene can be switched and potentially improve that chromosome.

Genetic algorithms are applicable in situations where brute force algorithms are not possible and there are many possible solutions that are less than optimal. In addition, a genetic algorithm is best suited to a problem where there can be multiple solutions that are not necessarily on the path to the best solution. Gradient assent hill climbing, another method of evolutionary computation, involves using mutation alone and selecting the best to continue searching for the solution. Because a genetic algorithm has both crossover and mutation, it is much less likely to get stuck on an ineffective solution. Having crossover allows for jumps in the search space that is not possible with mutation alone.

IV. Capture The Flag

In typical Xpilot play players fight each other in one-on-one combat or in a team combat situation. In the capture the flag game in Xpilot agents are divided into teams, and each team has a ball that they must protect. The goal is to protect their own ball from being taken and to steal the other teams' ball and bring it to their own goal for points (Figure 2 shows an agent dragging a ball). The combat part of Xpilot is still in effect, so players can shoot at each other in their pursuit or defense of a ball. The unique challenges associated with capture the flag are effectively controlling the ball and protecting the ball from the other team. In addition, the agent will die if it touches the ball or the goal, so it has to take care to maintain a safe distance.

This research is concentrating on the development of controllers for an agent in the capture the flag scenario. The developed agent is focused solely acquisition, control, and delivery of the ball, without other agents in the game. In a capture the flag game in Xpilot, an agent attempting to put the ball in the goal can be in one of four situations. The first situation is when the agent is travelling to the ball before the ball has been acquired. At this stage the ball is stationary because the ball does not move before an agent touches it. The second situation occurs when the agent has picked up the ball and is dragging it to the goal. In this stage, the ball is being dragged behind the agent, making it difficult to maneuver. Maneuvering in Xpilot is already hard because there is no friction in space, and the ball is several times heavier than the agent, which adds to the difficulty. The third scenario is the release of the ball and timing it correctly so that the ball ends up in the goal. The fourth scenario occurs when the user attempts to pick up the ball while the ball is travelling throughout the map. Ideally, the fourth scenario would not be needed, but there can be times when the user has dropped the ball, either by accident or in a failed attempt to deposit it in the goal. If the ball has been dropped, it continues its movement and only changes direction if it bounces off a wall or another object.

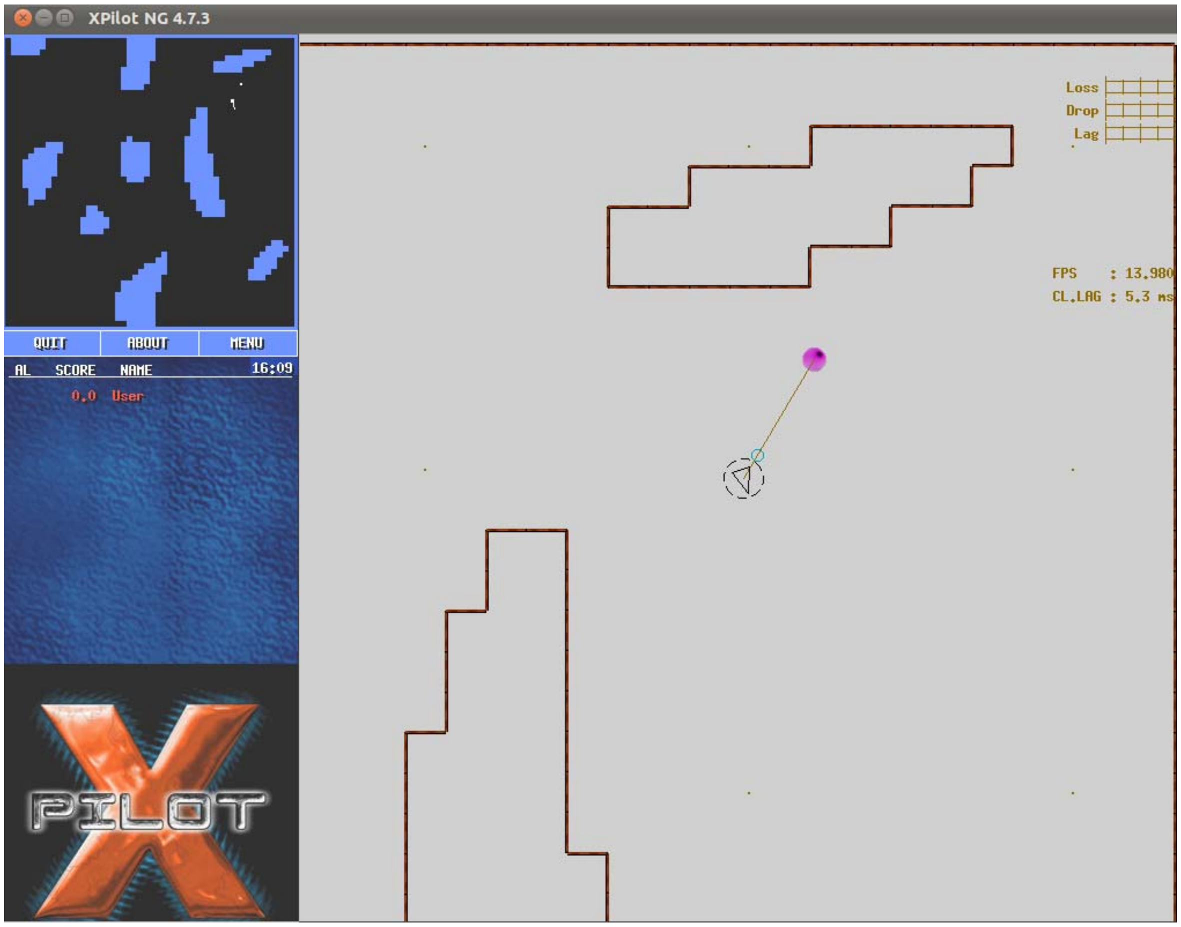

Figure 2. Capture the Flag play in Xpilot. The agent has acquired the ball by pressing a key when close to the ball. Once acquired, it is connected by a tether. If the agent pulls too hard, or turns too sharply, the tether will break. Since the ball has much more mass than the ship, it's difficult to deal with its momentum. The background is usually black, with blue walls (seen here in dark brown), a white ship, and a green ball (seen here in purple). The colors have been inverted to save printer ink and so details can be seen. The left hand panal has not been changed.

In this research, we concentrated on the first three situations. That is, the agent would attempt to move toward the ball, pick it up, drag it to the goal, and deposit it there. While the expert agent did have the capability to move towards a moving ball, most of the tests did not cover this situation because the expert agent did not typically drop the ball.

V. Evolving the Controller

For this research, a GA was used to learn the parameters of a rule-based system. This method of learning was selected because it has been the most successful method used in previous research in learning Xpilot combat agent controllers.

A rule-based system to serve as an expert agent was written to move toward the ball, pick it up, and place it in the goal. This rule-based system contained a number of rules that dictate what actions to take in each scenario. These rules were written by the researchers, but were far from optimal. There are a number of numeric parameters that are used for control, though their optimal value is not known. For example, the agent will turn to avoid a wall if it is less than *x* distance away and traveling at more than *y* speed. These parameters can be determined by the programmer through hours of trial and error, or can be learned by a GA.

Parameter Name	**Description**
Max angle, ball, screen, no wall	Max angle between agent and ball where the agent is considered to be pointing to the ball when the ball is on screen and there are no walls
Min speed, ball, screen, no wall	If the agent is going slower than this speed when the ball is on the screen and there no walls between the agent and the ball the agent will thrust
Max angle, ball, screen, wall	Max angle between agent and ball where the agent is considered to be pointing to the ball when the ball is on screen and there are walls
Min speed, ball, screen, wall	If the agent is going slower than this speed when the ball is on the screen and there are walls between the agent and the ball the agent will thrust
Max angle, ball, radar, no wall	Max angle between agent and ball where the agent is considered to be pointing to the ball when the ball is on radar and there are no walls
Min speed, ball, radar, no wall	If the agent is going slower than this speed when the ball is on the radar and there no walls between the agent and the ball the agent will thrust
Max angle, ball, radar, wall	Max angle between agent and ball where the agent is considered to be pointing to the ball when the ball is on radar and there are walls
Min speed, ball, radar, wall	If the agent is going slower than this speed when the ball is on the radar and there no walls between the agent and the ball the agent will thrust
Max angle, ball, not visible, no wall	Max angle between agent and ball where the agent is considered to be pointing to the ball when the ball is not visible and there are no walls
Min speed, ball, not visible, no wall	If the agent is going slower than this speed when the ball is not visible and there no walls between the agent and the ball the agent will thrust
Max angle, ball, not visible, wall	Max angle between agent and ball where the agent is considered to be pointing to the ball when the ball is not visible and there are walls
Min speed, ball, not visible, wall	If the agent is going slower than this speed when the ball is not visible and there no walls between the agent and the ball the agent will thrust
Max angle, goal, no wall	Max angle between agent and goal where the agent is considered to be pointing to the goal when there are no walls
Min speed, goal, no wall	If the agent is going slower than this speed toward the goal and there are no walls between the agent and the goal the agent will thrust
Max angle, goal, wall	Max angle between agent and goal where the agent is considered to be pointing to the goal when there are walls
Min speed, goal, wall	If the agent is going slower than this speed toward the goal and there are walls between the agent and the goal the agent will thrust
Wall feeler length, reactive	Distance away the agent checks for walls when the agent gets too close
Wall feeler length, proactive	Distance away the agent checks for walls when planning path around the wall to the ball or goal
Velocity multiplier	Number multiplied by speed to determine if thrust should be used to get by a wall
Wall feeler angle	Angle at which the agent checks for walls
Max speed	Above this speed the agent will slow down
Spin distance	Distance away from the goal the agent will begin to spin

Figure 3. Table describing the 22 parameters with descriptions for each

Another example of a parameter that needed to be learned is the angle parameter that determines whether or not the agent is pointing towards the goal. That is, if the agent is 5 degrees off from pointing to the goal, does that count as pointing towards the goal? Using an angle of zero difference would be unreasonable because this rule would be very unlikely to ever fire. On the other hand, if the difference between the agent's heading and the angle towards the goal was too large then the agent would have trouble getting to the goal because it would determine that it was pointing toward the goal even when it wasn't close, and would therefore go in the wrong direction. The parameter needs to be fine tuned to be between the extremes, something that the GA can do very well.

In total, 22 parameters were evolved (Figure 3). The first 16 represent whether or not the agent is going slow enough to thrust and whether the agent is headed in the right direction. When the agent is moving towards the ball, it either can see the ball on the screen, can see the ball on radar, or cannot see the ball, in which case the agent goes towards the last known position. In each of these three cases, the agent can either have walls between itself and the ball, or it can be empty space between the agent and the ball. These three scenarios based on wall position are therefore made into six scenarios based on whether there is a direct path or not. The agent always knows the exact location of the goal, but it again could have an obstacle in the way. Therefore, in total, there are eight cases that the agent can be in while playing: 6 based on the knowledge about the ball and wall location, and two based on wall position between the agent and the goal. Each of these scenarios needs two parameters each, one based on speed and one based on the angle difference allowed for the agent to be considered to be pointing at its current target. These parameters were allowed to be different based on the current game scenario so that, for example, the agent could move faster if it is far from the ball (ball on radar not on screen) and if there were no walls between the agent in the ball.

These 16 parameters described were combined with six others: two parameters relating to the distance that the agent should check for walls, a multiplier for the velocity of the agent to help determine if a wall was close enough to be a threat, the angle at which the agent looks for walls, a speed at which the agent is determined to be going too fast, and the distance away from the goal that the agent should start its "spin." Because the agent cannot touch the goal, it must stay far enough away and cause the ball to swing into the goal. In order to do this, when the agent is within a certain radius of the goal it tries to move away from the goal, and when it gets back outside the radius it goes back to trying to move toward the goal. This back and forth movement causes the ball to swing around and hopefully get into the goal. This "spin distance" is the distance of the radius of the circle that the agent should swing around the goal.

These twenty-two parameters were combined into a single chromosome to be learned by the GA. In order to allow the GA to learn the optimal parameter from a wide range of possible values, each parameter was assigned four or six bits, making the entire chromosome ninety-four bits long. Some of the parameters needed a larger range than the straight binary conversion, so some of the parameters were multiplied by a factor to increase their maximum value.

The 8 angle difference parameters were all 4 bits (a range of 0-15) and were multiplied by 2 making the range 0-30 with increments of 2. The 8 speed parameters are also 4 bits giving them a range of 0-15. The wall feeler parameters were given 6 bits (making the range 0-64) and were multiplied by 10 so that the range was from 0-640 with increments of 10. The parameter to determine if the agent is moving too fast and the speed multiplier are given 4 bits (0-15 range). Finally, the spin distance parameter was assigned four bits giving it a range of 0-15.

A standard GA, as described in section 4, was used to learn these parameters. The population size was chosen to be 128, roulette wheel selection was used, and standard crossover was employed. The mutation rate was one in 50. A typical server runs at around 16 frames per second (FPS), but this research uses 64 FPS to increase the speed in which the agent can learn.

Each individual was evaluated three separate times and the fitness assigned to that chromosome was the average of those three. The fitness function used for this agent was based mainly on the Euclidian distance of the agent to the ball or to the goal. If the agent had not yet reached the goal then its fitness was the 1000-x where x was the closest distance the agent ever was to the ball. In this way, agents that went towards the ball were rewarded more than agents who did not. If the agent reached the ball and picked it up the agent's fitness was then equal to 10,000-x, where x was the closest distance the agent ever was to the goal. Finally, if the ball was put into the goal then fitness awarded was 20,000.

VI. RESULTS

Six trials with randomly generated unique initial starting populations were run for 245 generations. At each generation, the fitness of every individual from the population was determined by averaging three runs, each from different starting locations.

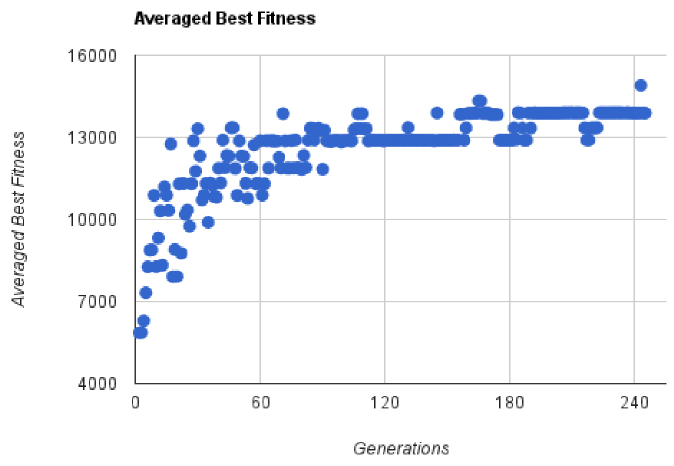

Figure 4. Best fitness for six trials averaged together. At generation 60 lowest best fitness of the 6 trials was 7877 and the highest best was 13890. At generation 120 the lowest best fitness of the six trials was 7887 and the highest best was 13951. At generation 180 the lowest best fitness was 7877 and the highest best was 13970. At generation 240 the lowest best fitness was 13866 and the highest was 13970.

Figure 4 shows the average of the best individual from each of the six populations for every generation from 0 to 245. The figure shows that in most cases the agent has learned to go to the ball, and bring the ball to the goal. The maximum fitness is 20,000, and that occurs when the agent brings the ball all the way to the goal.

In all six trials, the average fitness of the best individual was around 13,000 (Figure 4). The results for each map were also looked at separately, and it was found that this 13,000 was typically reached because the agent reached the ball and goal in two of the maps (achieving around 20,000) and did very poorly in the other map, attaining a fitness of less than 2000. The reason for this is due to the fact that the base where the agent starts is much closer to a wall than in the other two maps. This resulted in much higher rate of collisions before the ball could be reached. Although the agent typically did not reach the ball and goal on this harder map, it was capable of doing it, just with a much lower frequency. The parameters needed for the harder map are likely different than the parameters needed for the other two maps, so it was difficult to learn the optimal parameters for all three maps. Nevertheless, there were individuals in some of the trials that did reach the goal in all three runs. This can be seen in generation 241 where one individual from the six reached 20,000 and the other five were around 14,000.

A look at the average fitnesses of all the individuals of the six populations (Figure 5) also shows a steep learning curve in the beginning, with continual slower learning after that, which is typical in genetic algorithms.

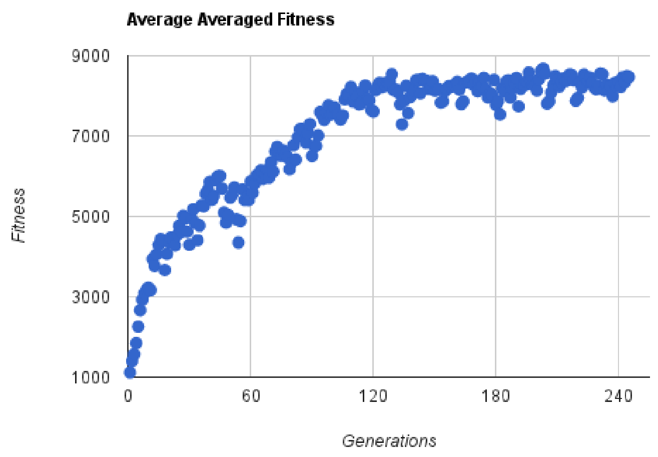

Figure 5. The average fitness for the six trials was averaged and the results are displayed above. At generation 60 the lowest average fitness was 4251 and the highest was 9356. At generation 120 the lowest average fitness was 5622 and the highest was 9419. At generation 180 the lowest average fitness was 5648 and the highest was 8727. At generation 240 the lowest average fitness was 6049 and the highest was 9234.

Observations of the agents in action reveal that as the agent makes its way toward the ball it often spins, which slows the agent down because it thrusts when it is pointing away from where it is heading, and then turns back around and continues towards the ball. This is a benefit since when the agent is going too fast it is more likely to die when it hits a wall and it is more likely to have difficulty turning away from the wall in time. Therefore, this spinning and slowing down helps to prevent deaths. Once it attaches to the ball, the agent spins much less frequently because the agent does not reach speeds as high due to the heavy weight of the ball. When the agent towing the ball gets close to the goal it starts to spin again so that the ball will swing into the goal while the agent is kept a safe distance away.

VII. Conclusions and Future Work

The results of this experiment indicate that the agent can and has learned how to handle the ball and bring it to the goal in capture the flag play. Viewing better agents from later generations shows that they have a reasonable technique in accomplishing performing this difficult task and are more skilled at this task than the researchers in the lab. This is the first learned control program capable of controlling an agent performing capture the flag in Xpilot.

The maps used in this experiment were relatively small, as the agent can only know where the ball is when it is on screen or on radar. Future research could include having the agent search the map until the ball is in radar contact using some sort of AI search method. In addition, it would be interesting to explore control learning for capture the flag while the agents are engaging in one-on-one combat. In further work, team play can be learned; this could involve heterogeneous agents where one is the tow agent and others are escorts.

References

[1] G. Parker and D. Arroyo, "The Xpilot-AI Environment," Proceedings of the 2010 World Automation Congress International Symposium on Intelligent Automation and Control (ISIAC 2010), September 2010, Kobe, Japan.

[2] S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels, "Evolution of human-competitive agents in modern computer games," Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, BC, Canada, July 2006.

[3] G. Yannakakis and J. Hallam, "Evolving opponents for interesting interactive computer games," Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB 2004), pp. 499 – 508, 2004.

[4] R. Baumgarten, S. Colton, and M. Morris, "Combining AI methods for learning bots in a real-time strategy game," International Journal of Computer Games Technology, vol. 2009.

[5] G. Parker and M. Parker, "Evolving parameters for Xpilot combat agents," Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Games (CIG 2007), Honolulu, HI, April 2007.

[6] G. Parker and M. Parker, "The evolution of multi-layer neural networks for the control of Xpilot agents," Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Games (CIG 2007), Honolulu, HI, April 2007.

[7] M. Allen, K. Dirmaier, and G. Parker, "Real-time AI in Xpilot using reinforcement learning," Proceedings of the 2010 World Automation Congress International Symposium on Intelligent Automation and Control (ISIAC 2010), Kobe, Japan, September 2010.

[8] Mitchell, Melanie. An Introduction to Genetic Algorithms. 8th ed. MIT, 1996.