The Evolution of a Multi-Layered Neural Network for Choosing a Fuzzy Controller for Xpilot Agents

Andrew Godwin[1] and Austin DiMartino[2]

Abstract— **In competitions in the two-dimensional space combat game, Xpilot, it is challenging to create a controller that tells an autonomous agent to activate a certain behavior given the current status of the agent and the game. A genetic algorithm (GA) learning the weights on a multi-layered artificial neural network (NN) that chooses between a passive and aggressive fuzzy and rule-based controller is used as a solution to this problem, along with the problem of maximizing the overall chance of winning for an Xpilot agent. The agent using a GA to evolve weights on a NN choosing between a passive and aggressive controller revealed an upward trend in average fitness over 30 generations, but remains inconclusive.**

I. INTRODUCTION

In this paper, we use a genetic algorithm (GA) to learn the weights on a multi-layered neural network (NN) that will evolve the behavior of an agent in the two-dimensional space combat game, Xpilot. The GA was used to evolve the weights on the NN that would determine an output telling the agent to be either passive or aggressive based on the status of the game in a one-on-one match. The output of the NN determines whether the agent should use a passive or aggressive controller, which both incorporate fuzzy logic for thrusting, and rule-based expert systems for turning and shooting.

Prior to this research, many other computer scientists have implemented GAs to evolve the behaviors of Xpilot agents and other robots in a simulated environment. For example, Parker, Doherty, and Parker implemented a GA to evolve the behavior of an Xpilot agent in order to improve the agents survival rate. Parker, Doherty, and Parker (2005) used the GA to evolve multiple survival strategies for the Xpilot agent, while teaching the agent the prioritization of such strategies based on the dynamic obstacles of the game. In this experiment, this team used 16 rules that they found to be necessary for reasonable gameplay, and used the GA to learn the numerical parameters of these rules in order to optimize survival strategies. They also incorporated a prioritization mechanism to allow the agent to learn which strategies were more important in particular situations. This experiment differs from ours as this team did not use a NN, but rather learned parameters for the Xpilot agents behavior.

In other research, Gary Parker and Matt Parker implemented a GA to choose the inputs, node thresholds, and weights on a NN to evolve a strong Xpilot agent. Parker and Parker (2007) use a GA to evolve the NN that outputs whether the bot should turn left, turn right, or not turn, thrust or not thrust, and shoot or not shoot. This teams work differs to ours as they are using the GA to learn the weights, inputs, and node thresholds within the NN that outputs certain specific behaviors for the agent. In our case, we are using the GA to learn only the weights on the NN that will tell the bot whether to use the passive or aggressive controller.

A third research paper by Hong and Cho (2004) outlines the use of a GA to learn emergent behaviors for a tank-simulation game in Robocode. In the game, two tanks have a gun, radar, and certain energy level, and the goal of each tank is to navigate around the battlefield, avoid shots from the enemy, and kill the enemy. The tank takes in different inputs like the coordinates of itself and enemies, the heading direction of itself and enemies (also for the radar and gun), and the specifications of the battlefield. Each tank has six different strategies, each with anywhere between two and eight primitive behaviors. The goal of this research was to find optimal combinations using one behavior from each of the six strategies to defeat different enemies. Each gene represented the encoding of one primitive behavior per strategy. This team wanted to come up with the best behaviors to defeat certain enemies using the GA. This paper relates to our research as the GA is learning certain behaviors against different enemies. Though they do not use a NN, they are learning combinations of behavioral components, which parallels our work as we are learning a combination of passive and aggressive behaviors over the course of a game that will help the agent win.

II. THE PROBLEM

A. Xpilot

Xpilot is a two-dimensional spacecraft flight simulation game that is open source and written in C. Within the game, two or more robots can battle against each other, earning points for shooting an opponent, and losing points for being hit by a bullet, colliding with the opponent, or colliding into a wall. A robot in Xpilot can turn left, turn right, thrust, and shoot a bullet. This game incorporates a client and a server, allowing multiple players to compete. This experiment was completed on the lifeless map in Xpilot (See Figure 1).

In one-on-one Xpilot competitions, since the agents typically learn one behavior to use throughout a match, it would be beneficial to teach an agent to choose a specific behavior based on the current scenario of the game. When agents are winning by a large margin, it would be beneficial to them

*This work was not supported by any organization

[1]A. Godwin is a student at Connecticut College, Computer Science, New London, CT 06320, USA

[2]A. DiMartino is a student at Connecticut College, New London, CT 06320, USA

to use a more reserved behavior that allows the other agent to waste time seeking it. On the contrary, when the time is running out on the clock and the agent is losing, it would be beneficial to use a more aggressive behavior, since there is nothing to lose and this would increase the agents chance of getting a kill and earning more points. Since only one behavior is learned for a particular game, the agent cannot alter its behavior based on certain scenarios with various score differentials and times left.

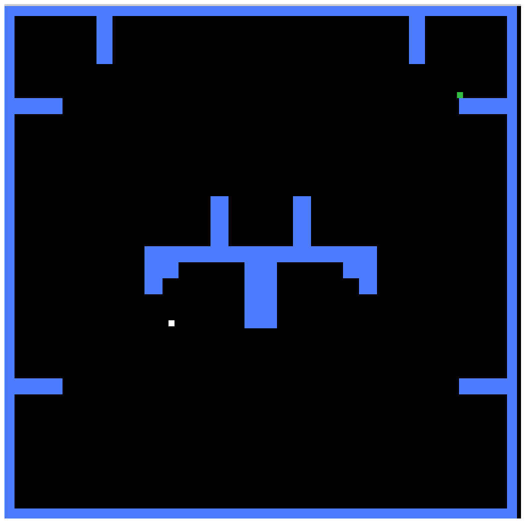

Fig. 1. Lifeless map in Xpilot

B. *Computational Intelligence Methods*

In order to solve this issue, a GA and NN were both implemented, along with controllers constructed with the use of both fuzzy logic and rule-based systems.

A GA is a branch of evolutionary computation that simulates evolution in nature through the use of crossover, mutation, and natural selection. Chromosomes are represented using a binary string as a potential solution that we will manipulate over many generations. The typical procedure of a GA is as follows: represent the problem as a fixed-length chromosome and initialize the population, evaluate the fitness of each chromosome in the population, create a new population based on the fitness of the individuals in the initial population, perform crossover and mutation, then repeat these steps (except for step 1) until the number of pre-specified generations is satisfied. GAs are commonly used in optimization and maximization problems, as they allow for the evolution of chromosome to encode the evolution of a solution. In this scenario, we are evolving the weights on a NN to create the best autonomous Xpilot agent.

An artificial NN simulates neural networks in the brain because it consists of neurons connected by weighted links that pass signals from one neuron to another. Each neuron has multiple input signals, but only one output signal. In order for learning to happen, the numerical weights on these links must change. An artificial neural network is made up of a hierarchy of levels of neurons. Each neuron gets input signals from the input links, computes a new activation level, then sends out an output signal. Input signals can come from actual inputs or from other neurons. Also, an output signal can either be the official output solution or be inputs for other neurons. Each neuron computes the weighted sum of the input signals and compares it to the threshold value. In a sign function, if the weighted sum is less than the threshold, the output is -1, and if the weighted sum is greater than the threshold, the output is 1. The step function, sigmoid function, and linear functions are also common within neurons.

Finally, Fuzzy logic is the theory of fuzzy sets, where fuzzy sets are sets that convey ambiguous or vague information. We use fuzzy logic to emulate how humans think. For example, when someone is defining something that is not necessarily binary, or black and white, we can use fuzzy logic and representation using degrees of membership. While crisp logic theory uses only two values, true or false, fuzzy logic theory says that an element is part of a fuzzy set with a certain degree of membership. This logic is applied to our thrusting rules for both the passive and aggressive controllers, while the turning rules use a rule-based expert system.

III. IMPLEMENTATION

A. *Neural Network Evolution*

Using a GA, the weights on a three-layer NN were learned. First, two separate controllers using a rule-based expert system for turning rules, and a fuzzy system for thrusting rules were created, enabling both a passive and aggressive agent. The linguistic variables used in both controllers were wall distance with fuzzy sets of close, medium, and far, and speed with fuzzy sets of slow and fast. These variables were used to compute backwall and sidewall risk, which were then incorporated into our thrust rules. The passive controller allowed the agent to remain stationary until approached by the enemy, then would become defensive, avoiding shots and shooting bullets where necessary. Contrarily, the aggressive agent was more active, flying around the map at a higher pace until coming in contact with the enemy, in which case it would locate and thrust towards the enemy while shooting continuously. The passive and aggressive controllers were both modeled after a pre-constructed fuzzy controller to ensure that neither of them would outperform the other significantly. If one were to considerably outperform the other, our NN would learn an output that would favor the better controller each life.

The NN used in this scenario was implemented to output a value between 0 and 1, where any value greater than or equal to 0.5 would choose the aggressive controller, and any value below 0.5 would choose the passive controller. The five inputs to the NN were a dummy variable revealing whether or not the agent was winning in the game (0 if the agent is losing, 1 if the agent is winning), proportion of deaths by colliding with the enemy, proportion of deaths by a bullet from the enemy, the proportion of deaths by colliding into a wall, and the number of frames elapsed. These inputs would update for each time there was a death, regardless of which

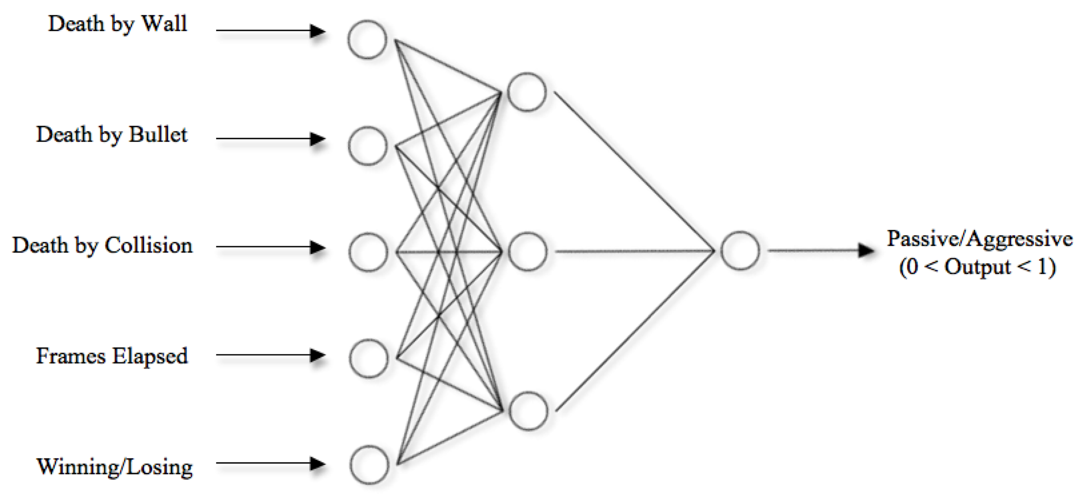

Fig. 2. Multi-layered neural network with five numerical inputs and three hidden nodes

bot died. This NN had one hidden layer of three nodes, and one output (See Figure 2). Each nodes input, except for the input values, was squashed using the sigmoid function (See Figure 3).

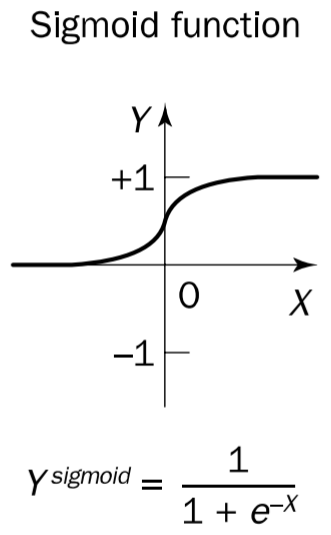

Fig. 3. Sigmoid squashing function used in the neural network

B. Fitness

The weights on the NN were evolved using a GA. For our fitness function, we were primarily concerned with whether or not our agent was winning, and whether it was using the behaviors we thought were most beneficial to its success at a certain point in the game. For that reason, the fitness function uses a binary variable, either 0 or 1, if the agent loses or wins. If the robot loses in a game of 2520 frames (which is roughly 3 minutes), then the fitness will be 0. However, if the agent wins the game, we increased the fitness to 100 as an additional bonus to emphasize the importance of winning.

Additionally, since we felt that the agent should be passive and attempt to run out the clock when it is winning with a lot of time left, we decided to add bonuses accordingly. These bonuses were assessed each time one of the robots was killed. If the agent was winning by 20 or more points and running the passive controller, we added a 20-point fitness bonus. Then, if the agent was winning by at least 20-points with at most 2000 frames elapsed (roughly 2 minutes into the game) and it is using the passive strategy, we gave the fitness an additional 20-point bonus. In this scenario, we want to make sure that, when the agent is winning by a fairly large margin with a lot of time on the clock, it is staying relaxed and not get itself into unnecessary trouble.

We also felt that the agent should be aggressive and attempt to find and kill the enemy when it is losing and there is not a lot of time left. So, we added bonuses according to these theories as well. As with the passive bonuses, these bonuses were assessed each time one of the robots was killed. If the agent was losing by 20 or more points and using the aggressive controller, we added a 20-point fitness bonus. Then, if the agent was losing by at least 20-points with at least 2000 frames elapsed and it is using the aggressive strategy, we gave the fitness an additional 20-point bonus. In this case, we want to make sure that, when the agent is losing by a fairly large margin with little time on the clock, it is trying to seek out and kill the enemy to get as many points as quickly as possible, without worrying too much about being shot or running into walls.

The population size was 64 individuals. Each chromosome consisted of 704 bits, containing twenty-two 32-bit genes, since there are 18 total weights and 4 threshold values on the NN. Each gene encoded a float value using an IEEE 32-bit float technique. Individuals were selected using roulette wheel selection. Two methods of crossover were used, and there was a 50% chance that either was chosen. The first method picks either one parent or the other to be the child. The second method randomly selects each bit from either parent to create the child. There was a 1/300 chance of mutation for each bit.

IV. Results

We ran two tests, one to 20 generations and one to 30 generations (see Figure 4). One test was inconclusive after completing 20 generations due to a time constraint. Though it did not learn to use the correct strategy in certain scenarios as we had anticipated, over 30 generations, the agent revealed an upward-sloping trend in average fitness. This overall improvement exhibits signs of learning in early stages of the evolution process. We can expect that, if the agent had run for the 100 generations we had planned on, that it would exhibit better fitness levels after each additional generation. Since we had a time constraint, we could not see the agent fully develop to a point where it was alternating behavioral strategies throughout a single 3-minute game. However, the agent did seem to exhibit learning to a small extent, as the number of losses seemed to decrease over the 30 generations.

Due to this time constraint, our research is currently inconclusive until it can be picked up again at a later date. However, as mentioned previously, the agent did show signs of learning through the evolution of the weights on the NN choosing between a passive and aggressive controller.

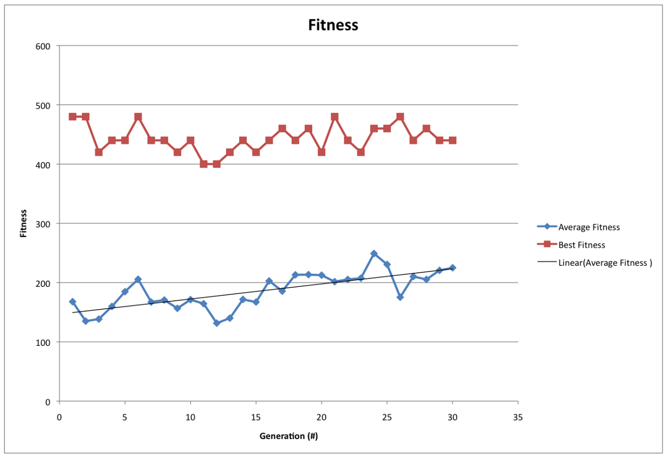

Fig. 4. Performance graph for 20 generations of 64 chromosomes. This shows the average fitness and maximum fitness value from each generation. A line of best fit is included to show the upward slope of the average fitness.

V. Conclusions

In the Xpilot environment, it is challenging to train an agent to learn multiple behaviors to play over the course of one game. When the agent learns one behavior for an entire game, it may be successful in some situations, but not in others. For example, an aggressive bot could start off great, finding and killing the enemy and increasing the score differential, but the bot may find itself flying into walls and losing points as the game goes on, ultimately leading to its demise. If the bot knew to play a defensive strategy and not put itself at risk, it could take time off the clock and give itself the best chance of winning the game. In this experiment, we evolved the weights on a neural net that chooses between a passive and aggressive controller based on meta-game information in Xpilot using a genetic algorithm and artificial multi-layered neural network.

We found that our research is inconclusive as our time restraint hindered the number of generations we could achieve for this bot to learn. Because we saw an upward trend in the average fitness of the agent over 30 generations, we can infer that we may have seen promising results given a greater time frame. At the end of 30 generations we found a considerably greater amount of wins compared to losses from generation 1. Thus, our bot seemed to partially pick behaviors in the correct situations that gave it a greater probability of winning over the 30 generations.

There are a variety of areas for future work. In this experiment, we trained a neural network that would choose whether to use an aggressive or passive behavior depending on the situation in the game using a genetic algorithm. To further develop this idea, using a genetic algorithm to learn the parameters for aggressive and passive bots could be possible using a neural network once again. As a result, the neural network could evolve to output a scale of aggressiveness based on the situation. The agent could then evolve to learn to be more or less aggressive or passive depending on the situation.

Further development of our hard-coded controllers could also be implemented. Using the same structure as with this experiment, a genetic algorithm could be used to learn new behaviors for the aggressive and passive controllers. For example, the genetic algorithm could be implemented to evolve a seeking behavior for the aggressive controller in which the agent uses the coordinates of the enemy to locate it and kill it. In the passive case, the agent could be trained to use an escape behavior, paired with an improved bullet-dodging behavior that would increase its survival rate and make it more defensive in a situation in which it is winning and needs to kill time off the clock.

This research could be applied to sporting events or any real-world competitions with a score and a time limit. The purpose of this research was to determine which strategies are best in certain scenarios. In the case of Xpilot, we believed that, in an instance where the robot is losing and there is little time left on the clock, the best strategy would be to play with nothing to lose and be aggressive to try to acquire as many points as possible. Contrarily, when the robot is winning by a large margin and there is plenty of time left in the round, the best strategy would be to play it safe and hold back without putting itself in danger. In sporting scenarios such as boxing, a winning team or player may apply these kinds of strategies to give themselves an optimal chance of winning. Future work with this field could promote new applications for this work and new techniques for implementation.

ACKNOWLEDGMENT

We would like to thank Gary Parker and Evan Gray for their continued expertise and support throughout the research process and semester. We would also like to thank the COM407 class for their input on our project, which helped us make great strides in our research.

REFERENCES

[1] 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.

[2] 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.

[3] 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 (CEC04), Portland, Oregon, 2004, pp 139145.

[4] J. Hong and S. Cho, Evolution of emergent behaviors for shooting game characters in Robocode, Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC 2004), Portland, OR, June 2004.

[5] Negnevitsky, *Artificial Intelligence-A Guide to Intelligent Systems* (Second Edition). Harlow, England: Biddles Ltd, Kings Lynn, 2005.