# Evolution and Prioritization of Survival Strategies for a Simulated Robot in Xpilot

**Gary B. Parker**
Computer Science
Connecticut College
New London, CT 06320
parker@conncoll.edu

**Timothy S. Doherty**
Computer Science
Connecticut College
New London, CT 06320
tsdoh@conncoll.edu

**Matt Parker**
Computer Science Department
Indiana University
Bloomington, IN 47405
matparke@indiana.edu

**Abstract- Simulated evolution by the use of Genetic Algorithms (GA) is presented as the solution to a two-faceted problem: the challenge for an autonomous agent to learn the reactive component of multiple survival strategies, while simultaneously determining the relative importance of these strategies as the agent encounters changing multivariate obstacles. The agent's ultimate purpose is to prolong its survival; it must learn to navigate its space avoiding obstacles while engaged in combat with an opposing agent. The GA learned rule-based controller significantly improved the agent's survivability in the hostile Xpilot environment.**

## 1 Introduction

In this paper we present the implementation of a Genetic Algorithm (GA) to evolve behavior for a simulated robot (agent) in the flight game Xpilot. Specifically, the algorithm was used to evolve the behavior of the robot when encountering such obstacles as walls, enemy craft and incoming enemy fire. The ability to simulate the learning of autonomous agents is important for the development of independent robots.

Several researchers have used simulations to evolve navigation behaviors for robots. One example is Allen Schultz from the Naval Research Lab. Schulz (1994) used a GA to evolve behavior for a simulated robot to navigate a course of obstacles to within a certain radius of a given goal. The Xpilot project differs in that there is no physical goal, but rather the goal is to navigate without hitting an obstacle or falling to a "predator robot" for as long as possible. Unlike Schultz's experiment, any contact with an obstacle will result in the termination of the controlling chromosome's "life" in this experiment. In an earlier work, Schultz (1991) used a GA to develop a set of decisions for navigating a simulated autonomous underwater vehicle (AUV) through a mine field. Simulation of an AUV is closely related to the experiment described in this paper, as contact with a mine by the AUV would result in a failure. Though both the AUV and the Xpilot robot must avoid collisions, the experiments differ as the Xpilot robot must also evade or eliminate an aggressive enemy robot in this experiment, while the AUV is constrained by a time limit and a geographic goal that it must reach.

Several researchers have used evolutionary computation for learning games. Much of this work was in developing strategies for board games such as checkers (Fogel, 2002) and go (Konidaris, Shell, & Oren, 2002) or for the prisoner's dilemma problem (Hingston & Kendall, 2004).

Researchers have also used computer games to evolve the control of autonomous agents. One example is the work done by Yannakakis and Hallam (2004), who explored learning predator strategies for agents in a version of "Pac-Man." While they concentrated on evolving the behavior of a group of 4 predators, the research presented in this paper aims at learning reactionary behavior for one agent under constant attack from a single enemy. Furthermore, Yannakakis and Hallam were interested foremost in what characteristics of a video game opponent would increase excitement and entertainment value. Evolutionary computation has also been used to allow an agent to decide upon weaponry and learn behaviors in the first person shooter Counter-Strike (Cole, Louis & Miles, 2004) and how to move a light-cycle in the game of Tron (Fumes & Pollack, 2000).

An unpublished online paper was the only previous research found which evolved behavior for robots in Xpilot (Funes). This heuristic system evolved to ultimately one behavior of six possible: firing at the nearest enemy. For our research, we sought to allow the robot to learn multiple behaviors for various situations which may overlap and determine the relative importance of the situations.

The hope is for this project to advance the use of computational intelligence, and specifically GAs, in areas of adaptive robotics requiring multiple strategies to be learned simultaneously. The robot used in this research is capable of learning to react in numerous situations involving obstacles, enemy fire and enemy craft. This paper also highlights the capabilities of evolutionary computation for video game and simulation technology.

## 2 The Problem

### 2.1 Xpilot

Xpilot is a 2-dimensional flight simulation game written in the C programming language in which multiple robots can compete (See Figures 1 & 2). The game is open source, which makes it possible to modify the program to introduce a learning component. Xpilot is playable over the internet, and players control their ships using the keyboard or mouse. The design uses a client and a server to support multiple players, and has many variables for weapon and ship upgrades, map settings, team play, etc. By parsing variables used by Xpilot such as velocity and position, and through the addition of a GA, principles of evolution can be used to learn the consequents for the rules of a rule based system to control the behaviors of robots in the game. The rule based system uses the AI interface to simulate keystrokes, and thus to control the agent.


Figure 1: The Robot evading the predator "Clyde" during a trial run. The specks in the vicinity of both agents indicate emissions from their thrusters.

### 2.2 The Arena

An Xpilot robot was placed in a specific scenario within the game, where it was forced to learn reactions to possible conditions in order to survive. The robot was placed in a square "arena" enclosed by 60 blocks on each side (Figure 2). Each square was comprised of 35 pixels on a side. In addition to the surrounding walls, the arena contained a horizontal bar 1 block high and 17 blocks in length across the middle of the grid (Figure 1), 32 blocks from the bottom boundary. The robot shared this space with a "predator" robot. This enemy robot was controlled by the computer intelligence native to Xpilot. The evolving robot started each run near the bottom left-hand corner of the map. The predator started near the bottom right-hand corner. Whenever one of the robots crashed or was shot, it was regenerated at its starting location. In addition to evading the opposing agent, the learning robot needed to evolve its flight, avoiding the external walls and the single internal wall.

The central horizontal wall was used so that the robot could learn to "hide" if it so desired, and to provide further challenges in learning navigation. The robot thus had the possibility of learning to stalk or evade its enemy. The problem was to have the robot's "intelligence," which was imbedded in the chromosomes, evolve to allow the robot to survive as long as possible without crashing or being shot. This was measured in the number of frames the robot survived. To shorten run time for the

GA, all runs were performed in servers running at the maximum rate of 100 frames per second (fps), which is about six times faster than normal play.

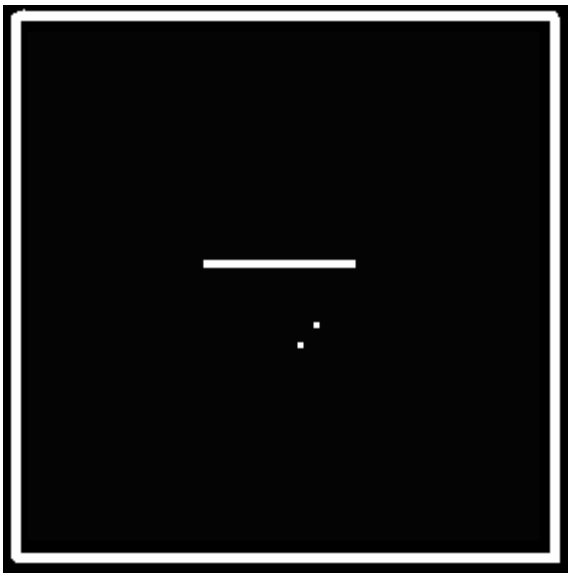

Figure 2: The mini-map inset to Xpilot, showing the relative positions of the robots from Figure 1.

## 3 Learning the Controller

A set of 16 conditions important for reasonable play were developed. These included conditions dealing with the ship's position relative to walls, the enemy ship, and hostile fire. A binary coding was developed to represent the possible responses that were to be learned by the GA. In addition, the GA learned the pointers of theses different rules to resolve conflicts if more than one fired at the same time.

A population of 128 random chromosomes was generated initially. The robot is represented by chromosomes of 16 genes. Each gene is represented as a segment of 14 bits, with a value of either 0 or 1; thus, a chromosome is comprised of 224 bits (See Figure 3). Each 14-bit gene contains information for how the robot should behave in each of 16 different cases which serve as antecedents for a rule based system.

### 3.1 The Rules

The 16 cases were isolated as critical situations for the robot to learn reactions for. When the criteria for the antecedent case are reached, the rule fires and the corresponding gene representing the consequent component of the system is accessible. The nature of the 16 rules allows for more than 1 rule to fire in each frame. For this reason, a priority system is used to allow the agent to decide between rules. To allow the robot to learn an optimal balance of rule priorities, the priority for each rule is embedded in the corresponding gene. The first 5 bits of each gene encode its priority from 0 – 31. When multiple rules fire, the gene with the highest priority will be used to control the robot. In case of ties, one of the tied rules will be chosen at random, but progression of the GA learning should limit this possibility in critical cases. The 16 rule antecedents are:

1. Agent is approaching wall and is very close to wall.
2. Agent is approaching wall and somewhat close to wall.
3. Agent is approaching wall and at a moderate distance from wall.
4. Agent is approaching wall at any distance and a bullet is incoming.
5. Agent is approaching wall at any distance and an enemy is close and closing.
6. Agent is approaching wall at any distance and an enemy is close but not closing.
7. Bullet is incoming and very close to striking.
8. Enemy is close and incoming bullet is close.
9. Enemy is at a medium to far range and incoming bullet is close.
10. Enemy is detected medium to close.
11. Enemy is detected medium to far.
12. Agent is entering a corner clockwise.
13. Agent is entering a corner counterclockwise.
14. Agent detects no walls, bullets or enemy agents.
15. Wall detected 90° to left of agent.
16. Wall detected 90° to right of agent.

All distance values such as "close" and "medium" are not fuzzy sets, but discrete values to the robot, within the range of 0 to 150 pixels from the robot. Cases 12 and 13 are somewhat different in nature from many of the other antecedents. They are accessed anytime two different walls are detected in consecutive frames, indicating that the robot is flying into a corner. The robot can detect one wall in a frame, and in the following frame detect either the wall clockwise or counterclockwise from the first wall; thus there are only 2 cases. The final 2 antecedents fire when a wall is detected directly to the right or left of the robot and are present to allow the robot to consider its surroundings before evading an enemy craft or bullet.

### 3.2 Xpilot Chromosome

The chromosome is made up of 16 genes, each having 14 bits. The first 5 bits of the gene determine the rule's priority as discussed in the previous section. The next 4 bits of a gene encode the number of degrees the robot should turn, from 0 to 15. In some cases, the robot is allowed the option to make its turn relative to an enemy or bullet and in these cases it may also use these bits to adjust within 0 to 15 degrees of the target direction. The $10^{th}$ and $11^{th}$ bits determine whether or not the robot fires its weapon and whether or not it uses its thruster, respectively. The final 3 bits are not all used in all the cases, and vary somewhat in their function. For example, in the first three rules the bits allow the agent to choose between turning based on the turn bits, turning 90 degrees to the wall either direction, or 180 degrees from the wall. The robot can only turn 15 degrees in one frame, but will continue to turn at this maximum rate each frame as long

as that rule continues to fire. In other rules, the bits allow the agent to turn directly towards or away from a bullet or enemy (limited to a maximum turn rate of 15), or decide between a left and right turn.

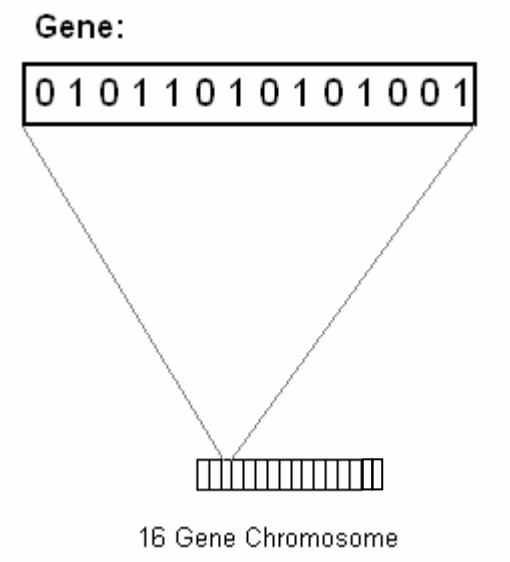


Figure 3: Sample chromosome used to encode the priorities (first 5 bits) and reactionary part (remaining 9 bits) of rule system.

In the cases when no enemy or bullet is present and the robot is approaching a wall, it is able to discern the turn it can make with the greater angle perpendicular to the wall; i.e. it recognizes the "easier turn" (See Figure 4). It uses one of the final 3 bits of the gene to decide whether or not to make the turn with the least chance of striking the wall. If it does not use this ability, it accesses another of the final 3 bits which is dedicated to deciding which way to turn.

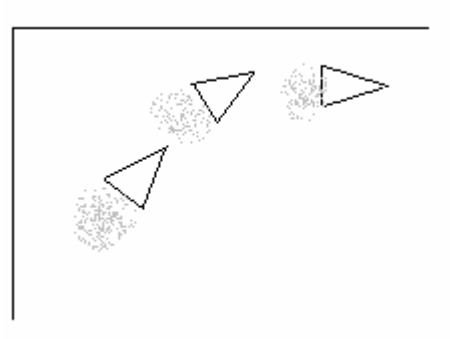

Figure 4: Agent making easier turn based on angle of approach to wall.

### 3.3 Genetic Operators

Each chromosome is tested separately, and the number of frames it survives is stored. The number of frames each chromosome survives is squared and this number is stored as the individual's fitness. The total fitness of the population is stored so that it can be accessed up to any point in the trial of that population. Roulette wheel selection as described by Goldberg is used to choose parents for crossover (Goldberg 1989).

Two forms of crossover are employed to generate a new population. When two individuals are chosen for crossover, there is an even chance that they will be combined using either inter-gene crossover or intra-gene crossover. If inter-gene crossover is utilized, then for each 14-bit gene, the gene from one of the two parents is chosen at random and copied to the offspring. This allows optimal genes for specific scenarios to be kept intact. If intra-gene crossover is selected, then for each bit of each gene, the bit at that spot is copied at random from one of the two parents. This allows for greater diversification. Each bit also has a chance of mutation; there is a one in three hundred chance of each bit being flipped, regardless of the type of crossover used.

The best chromosome from each generation is saved to a data file, along with the number of frames it lived and its fitness. Additionally, the average fitness of every 5$^{th}$ generation is stored for analysis of fitness growth.

Although the default frame rate for Xpilot is 16 frames/sec, the trials are conducted at the maximum rate of 100 frames/sec to decrease the length of the runs. Despite the increase in frame rate, each trial takes several days to complete.

The enemy robot does not effectively search the entire arena for the evolving robot. As this experiment aims to evolve the behavior of an agent in constant conflict with another agent, the evolving Xpilot robot is forced to thrust any time that its velocity is equal to zero. In effect, the robot will drift slowly about the arena if no rule fires that will force it to move more rapidly. This increases the possibility of interaction, as the agent cannot merely rest in a corner, but it allows the agent to more often behave in reaction to its enemy. The ability of the agent to return fire at its enemy serves to balance its forced movement, plus it has the ability to retreat when it detects its enemy.

## 4 Results

The results of the trials were highly successful. Five test runs that were started with randomly generated populations were performed for 145 generations. The average survival frames (fitness) for every fifth generation of each run is shown in Figure 5.

The enemy-harried robot evolved from an average life of 82.61 frames to a maximum average lifetime of 265.50 frames in the 130$^{th}$ generation, averaged across all runs. The best individual chromosome from the 5 runs lasted 2507 frames in the 117$^{th}$ generation of run 4. At the normal Xpilot frame rate of 16 frames per second, the

maximum average life would correspond to 16.59 seconds, with the best chromosome lasting 156.69 seconds.

In all five runs, the robot evolved wall-avoidance behavior that appears near-optimal. The robot coasts through its environment until it approaches a wall. When it first detects the wall, it will begin to turn slightly, as dictated by gene 3. It continues to turn as it nears the wall, and when the robot is close enough to the wall to fall under the jurisdiction of gene 1, it turns as much as possible away from the wall, and uses a short burst of thrust. In this way the robot appears to "bounce" off the wall. This burst propels it away from the wall, and it again allows itself to drift.

The robot thus showed marked improvement in its ability to successfully navigate the arena. The robot exhibits a variety of behaviors when encountering its enemy. In some chromosomes the robot continues to drift but pivots to face the enemy craft so that it might fire upon the enemy robot. In others it thrusts to engage the enemy toe-to-toe, or circles the enemy while firing.

In one run, the robot learned more successful behavior. The robot learned to constantly thrust, flying in quick counterclockwise circles around the arena. As it moved, it would continually adjust its aim to point towards the center of the arena. When the enemy would be detected, the robot would continue its motion but adjusts its aim to fire at the enemy agent. This behavior was very successful for evading enemy attacks while returning fire. The usual cause of the agent's death in this run was that after the agent would kill the enemy agent, the enemy would restart in the corner behind the circling robot. If the swooping robot were to pass by as the enemy restarted, it was unable to change its motion or turn fast enough to return fire.

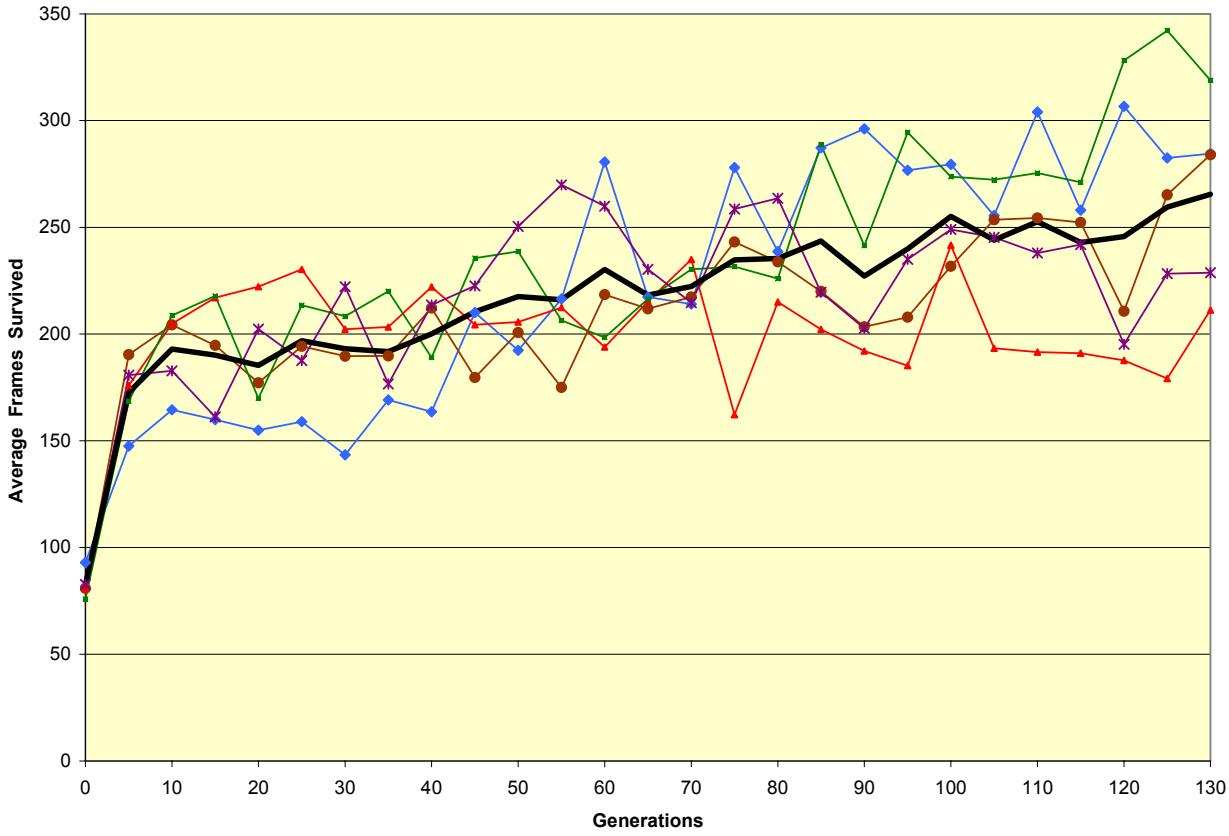


Figure 5: The results of evolving rule consequents and priorities for Xpilot simulated robot controllers. The average population fitness is shown for each of 5 test runs with the average of the 5 runs shown in bold.

## 5 Conclusions

Based upon the results of the experiment, a GA can be used to simultaneously evolve multiple survival strategies and their relative priorities for an autonomous robot operating in the Xpilot environment. In the later portions of the runs, the GA driven robot was more successful at dispatching its enemy than was the enemy robot. Survival attempts were somewhat frustrated by the enemy robot's willingness to sacrifice its own life and crash into the GA driven robot. If the robot was allowed to remain stationary it might live longer, though it is probable that it would be unable to accelerate sufficiently to avoid enemy fire. If given a larger number of possible behaviors, particularly for attacking its enemy, and the ability to

more finely adjust its motion, we speculate that the robot could survive significantly longer in the arena. The robot has the capability to predict where the enemy will be within a few frames and aim for that point, but if it were given more genes involving an enemy robot it might be able to evolve more complex attack patterns. Currently, the robot will not seek out its enemy; it will engage it only as a defensive maneuver as the enemy enters within a certain range. Adding a "fitness bonus" to chromosomes that successfully defeat the enemy robot might encourage that trend in evolution, and ultimately lead to longer average lifespans. More complex attack patterns and the option to be the aggressor might allow the intelligent robot to be a successful predator, rather than a prey relying on evasion and self-defense.

Several possibilities exist for future research. Although the enemy was generally the cause of death for the evolving robot in the battle arena, it was not always directly so; at times the fleeing robot would fly into a wall. The robot might perform more successfully if it were granted the ability to map the entire arena, rather than merely detecting obstacles in its flight path or off its flanks. Better results might also be achieved if navigation genes were evolved in an environment free of enemies, and then placed in the chromosomes of a robot in a battle arena. This same concept could be used to evolve the robot in an arena with stationary enemies, to allow the robot to more highly evolve its ability to evade shots. A scenario in which several robots evolved simultaneously would open up many possibilities, such as evolution of predator/prey roles, and the possibility for evolving coordinated attacks as a "pack."

## Bibliography


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

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

Funes, P. "XPILOT: A differential game and a learning game bot." http://www.cs.brandeis.edu/~pablo/xpilot2.ps

Funes, P. and Pollack, J. (2000). "Measuring Progress in Coevolutionary Competition," *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior.*

Luke, S. and Spector, L. (1996) "Evolving Teamwork and Coordination with Genetic Programming. *Proceedings of the Genetic Programmin Conference, Stanford University, 1996.* (GP96)

Goldberg, D. (1989). *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley, Reading MA.

Hingston, P. and Kendall, G. (2004). "Learning versus Evolution in Iterated Prisoner's Dilemma," *Proceedings of the International Congress on Evolutionary Computation 2004* (CEC'04).

Konidaris, G., Shell, D., and Oren, N. (2002). "Evolving Neural Networks for the Capture Game," *Proceedings of the SAICSIT Postgraduate Symposium*.

Schultz. A. C. (1991). "Using a genetic algorithm to learn strategies for collision avoidance and local navigation." *Proceedings of the Seventh International Symposium on Unmanned, Untethered Submersible Technology*.

Schultz, A. C. (1994). "Learning robot behavior using genetic algorithms." *Intelligent Automation and Soft Computing: Trends in Research, Development, and Applications*.

Yannakakis, G. and Hallam, J. (2004) "Evolving opponents for interesting interactive computer games." *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior* (SAB'04).