Abstract—First-person shooter robot controllers (bots) are generally rule-based expert systems written in C/C++. As such, many of the rules are parameterized with values, which are set by the software designer and finalized at compile time. The effectiveness of parameter values is dependent on the knowledge the programmer has about the game. Furthermore, parameters are non-linearly dependent on each other. This paper presents an efficient method for using a genetic algorithm to evolve a set of parameters for bots which play as well as parameters tuned by a human with expert knowledge about the game’s strategy. This indicates genetic algorithms as being a potentially useful method for tuning bots.

I. INTRODUCTION

Commercial game developers are faced with the challenge of creating realistic, human-like artificial intelligence robot controllers (game AI) within tight hardware constraints [9]. In terms of first-person shooters, rule-based expert system robot controllers which play the game are called “bots.” Due to computational constraints, most bots are written in C/C++, taking the form of state-based machines [15].

In order to save both computation and the programmer’s time, the game AI uses many hard-coded parameters to complete the bot’s logic. Authors of bots spend an enormous amount of time setting parameters. Parameters can be thought of as values which act as thresholds in the bot’s rule-based logic. Figure 1 shows an example of such parameters in terms of Counter Strike gameplay. Counter Strike is a popular first-person shooter game [3].

By adding more parameterized rules, the bots become more realistic. Consequently, the development time increases since the programmer has more parameters to tune using trial-and-error. The tuning of these parameters becomes increasingly complicated even if the programmer is an expert in the game’s strategy.

While there exists work in applying genetic algorithms to board games such as checkers and game theoretic problems like the iterated prisoner’s dilemma, little work has been done within the scientific community in applying a genetic algorithm to a popular, 3-D, first-person shooter game like Counter Strike [6], [2].

We propose a technique which applies a genetic algorithm to the task of tuning these parameters, such as the bold face ones in Figure 1. This will help game developers write game AI that is efficient, realistic, and easy to develop.

```c
if( enemy.distance \leq 5 )
{
    ATTACK-WITH-KNIFE()
}
else if( enemy.distance \geq 5 \text{ AND } enemy.distance \leq 30 )
{
    ATTACK-WITH-SUBMACHINE-GUN()
}
else
{
    ATTACK-WITH-RIFLE()
}
```

Fig. 1. An example of parameterized rule-based AI. The parameters we are concerned with appear in **bold face**.

The process of finding an acceptable set of parameters is referred to as tweaking the bot. In Counter Strike, this would be the time spent tuning the bot’s weapon preference, initial aggressivity, path preference, and style of gameplay. Weapon preferences are set as relative probabilities of selection. If a bot selected the same weapon round after round, then opponents would easily exploit the bot since no single weapon is perfect for every situation. Therefore, programmers give each weapon a probability of being selected. This leads to a biased randomness in the bot’s weapon selection behavior. Table I shows an example of the relative probabilities for weapon selection.

Choosing a correct set of parameters is not always a straightforward process and requires a great deal of trial-and-error testing. An acceptable parameter set can be thought of as the code to a safe, and in the case of Counter Strike bots, there are many combinations which will unlock the safe. In terms of search space, there are a number of acceptable parameter optima that will lead to good gameplay. Determining the correct set of parameters is a tedious and time-consuming one, since a slight change to one parameter will often have a negative impact on other parameters, i.e. they are non-linearly dependent. This is our motivation for applying a genetic algorithm to find a good set of these rule-based parameters. We ask, “Is it possible to use a genetic algorithm to tune the
TABLE I
BOT WEAPON PREFERENCES

<table>
<thead>
<tr>
<th>Weapon</th>
<th>Selection Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steyr Scout</td>
<td>10</td>
</tr>
<tr>
<td>Benelli XM1014</td>
<td>5</td>
</tr>
<tr>
<td>Ingram Mac-10</td>
<td>15</td>
</tr>
<tr>
<td>Steyr Aug</td>
<td>30</td>
</tr>
<tr>
<td>H&amp;K UMP 45</td>
<td>10</td>
</tr>
<tr>
<td>Sig SG-550 Sniper</td>
<td>5</td>
</tr>
<tr>
<td>AI Arctic Warfare Magnum</td>
<td>25</td>
</tr>
<tr>
<td>H&amp;K MP5-Navy</td>
<td>30</td>
</tr>
<tr>
<td>FN M249 Para</td>
<td>5</td>
</tr>
<tr>
<td>Benelli M3 Super-90</td>
<td>15</td>
</tr>
<tr>
<td>Colt M4A1 Carbine</td>
<td>20</td>
</tr>
<tr>
<td>Steyr Tactical Machine Pistol</td>
<td>10</td>
</tr>
<tr>
<td>H&amp;K G3/SG-I</td>
<td>3</td>
</tr>
<tr>
<td>Sig SG-552 Commando</td>
<td>30</td>
</tr>
<tr>
<td>AK-47</td>
<td>20</td>
</tr>
<tr>
<td>FN P90</td>
<td>5</td>
</tr>
</tbody>
</table>

Recently, the scientific community has taken an interest in using commercial 3-D engines such as Quake, Unreal, and Half-Life as a testbed for advanced AI research [9], [1], [11], [12]. We also believe Counter Strike, a game which runs inside the Half-Life engine, to be a good testbed for AI research. John Laird uses the Soar AI Engine, a rule-based expert system, to design bots that play Quake II, another popular first-person shooter. Laird’s Soar engine is re-useable from game to game within a specific genre, requiring only changes to the engine calls and not to the AI logic inside of the Soar Engine [15]. Rogelio Adobatti et al. have developed a framework inside the Unreal Tournament engine which allows them to study AI behavior within the virtual environment provided by the engine [1]. Adobatti et al. have also developed a non-violent game in order to attract more researchers, who would otherwise be turned away by the extreme violence found in most first-person shooters.

Realizing the fact that computer game AI is constrained by hardware limitations, Khoo et al. proposed inexpensive yet effective methods for improving game AI [9]. Their work included adding an Elisa-based chat program to an existing Counter Strike bot in order to make the human players believe they were not playing against bots but rather other humans [9].

Work by Fogel with Blondie24 shows that coevolutionary computation can lead to a ranked AI checker player [6]. Axelrod’s work with the iterated prisoner’s dilemma problem has also shown that coevolution techniques can lead to the discovery of successful game strategies [2]. Since Counter Strike is not a turn-based game like most board and card games, it may not lend itself well to currently established coevolution techniques. Since coevolution may not be directly applicable, we are simply applying a genetic algorithm to the tuning of parameters to ascertain how effective evolutionary computation techniques are at first-person shooter games. If the genetic algorithm can make progress, then we will continue with our plans to coevolve Counter Strike bots. Our eventual goal is to investigate evolutionary computing techniques for knowledge acquisition, human modelling, and teamplay based on this platform. We chose Counter Strike because it is extremely popular. Last month, players spent over 1.5 billion minutes playing Counter Strike online [14]. Gamers connect from across the globe. This popularity will make it easier to collect more data for human modelling from human players as they connect to our server.

A. Genetic Algorithms

Genetic algorithms (GAs) are stochastic, parallel search algorithms based on the mechanics of natural selection and evolution [8], [7]. GAs were designed to efficiently search large, non-linear, poorly-understood search spaces where expert knowledge is scarce or difficult to encode and where traditional optimization techniques fail. Robust and flexible, GAs exhibit the adaptiveness of biological systems. As such, GAs appear well-suited for searching the large, poorly-understood spaces that arise in tuning problems, specifically, tuning parameterized rule-based bots for Counter Strike.

B. Counter Strike

Counter Strike is a popular first-person shooter game in which counter terrorists try to neutralize terrorists. The game has a strong emphasis on tactics, decision-making, and teamplay, and we believe it serves as a good testbed for our work. Figure 2 shows an in-game screenshot.

![In-game screenshot of Counter Strike](image)

There are a number of variant types of gameplay for Counter Strike, but we focused on one subset of gameplay inside Counter Strike called the defuse mission. In the defuse mission, counter terrorists are tasked with preventing the terrorists from planting a bomb at one of two locations on the map. A map can be thought of as an environment in which the game takes place. Figure 3 shows an overhead view of a typical Counter Strike map. The counter terrorists may win by defusing a planted bomb. The terrorists may win by planting a bomb and protecting it from defusal until its detonation. Either side may win by eliminating all members of the opposite team, since either side’s goal could be trivially accomplished without interference from the other team.

To constrain things further, each round of gameplay is limited to five minutes with a six-second planning phase called
freeze time. During freeze time, each side may purchase new weapons and equipment. Each item has an associated cost. Table II shows the cost of each primary weapon. A primary weapon, as the name suggests, is the player’s weapon of choice. Secondary weapons, pistols, can also be purchased in case the primary weapon fails.

<table>
<thead>
<tr>
<th>Name</th>
<th>Cost</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bennell M3 Super90</td>
<td>$1,700</td>
<td>Shotgun</td>
</tr>
<tr>
<td>Bennell XM1014</td>
<td>$3,000</td>
<td>Automatic Shotgun</td>
</tr>
<tr>
<td>Heckler and Koch MP5-Navy</td>
<td>$1,500</td>
<td>Sub-Machine Gun</td>
</tr>
<tr>
<td>Steyr Tactical</td>
<td>$1,250</td>
<td>Machine-Pistol</td>
</tr>
<tr>
<td>FN P90</td>
<td>$2,350</td>
<td>Sub-Machine Gun</td>
</tr>
<tr>
<td>Ingram MAC-10</td>
<td>$1,400</td>
<td>Sub-Machine Gun</td>
</tr>
<tr>
<td>Heckler and Koch UMP</td>
<td>$1,700</td>
<td>Sub-Machine Gun</td>
</tr>
<tr>
<td>AK-47</td>
<td>$2,500</td>
<td>Assault Rifle</td>
</tr>
<tr>
<td>Colt M4A1 Carbine</td>
<td>$3,100</td>
<td>Assault Rifle</td>
</tr>
<tr>
<td>Steyr AUG</td>
<td>$3,500</td>
<td>Assault Rifle</td>
</tr>
<tr>
<td>Sig SG-552 Commando</td>
<td>$3,500</td>
<td>Assault Rifle</td>
</tr>
<tr>
<td>Steyr Scout</td>
<td>$2,750</td>
<td>Sniper Rifle</td>
</tr>
<tr>
<td>AI Arctic Warfare/Magnum</td>
<td>$4,750</td>
<td>Sniper Rifle</td>
</tr>
<tr>
<td>Heckler and Koch G3/SIG-1</td>
<td>$5,000</td>
<td>Sniper Rifle</td>
</tr>
<tr>
<td>Sig SG-550 Sniper</td>
<td>$4,200</td>
<td>Sniper Rifle</td>
</tr>
<tr>
<td>FN M249 Para</td>
<td>$5,750</td>
<td>Full-sized Machine Gun</td>
</tr>
</tbody>
</table>

The money to purchase these items is earned by the result of the previous round. There are penalties and rewards for certain actions during the round as shown in Table III. The Payoff/Fine table will serve as a useful gradient for our GA in determining fitness.

During this planning time, teammates communicate with one another to determine which path they want to take to neutralize the opposite team. Planning is a key element of gameplay as each player should purchase a weapon which is effective for the chosen path. For example, if a player has decided to take a path which follows a narrow series of hallways and air vents, it is desirable to use a close-quarters automatic weapon such as the MP5-Navy. On the other hand, if the player decides to head towards a large, open plaza, then perhaps a long-range, bolt-action sniper rifle such as the Steyr Scout would be more desirable. The individual tactics of the player is paramount to the weapon selection. Long-range, bolt-action rifles can only be effective when fired from a fixed position where the player is not moving and has a clear line of sight towards the enemy. An aggressivity parameter determines the bot’s style of play. A less aggressive player tends to locate easily-defended positions and wait for the enemy to enter...
their kill-zone, whereas an aggressive player relies on the
effectiveness of a fast-attack to try and catch the enemy off-
guard. A bot should have an aggressivity parameter that is
appropriate for its weapon selection. Guarding a large open
plaza from long distance is nearly impossible with a shotgun.
Likewise, charging into a room with a sniper rifle that needs
significant time to aim properly is not a prudent tactic. By
the time the bot can aim the weapon, its opponents will have
already eliminated it from the round.

The remainder of this paper will discuss our architecture,
methodology, results, and, finally, our conclusions and future
work. For convenience, we shall now refer to bots which use
genetic algorithm chosen parameters as GAABs which stands
for Genetic Algorithm Assisted Bot.

II. ARCHITECTURE

Working with a commercial 3-D engine is not an easy task,
and Counter Strike is a game which runs inside Half-Life, a
popular commercial 3-D engine. The Half-Life engine only
makes a Software Developer Kit (SDK) available for public
use. The SDK allows for engine calls to be made, such as
requesting the number of kills a certain player has, but the
SDK does not allow one to view or modify the actual code
inside the engine. Half-Life, like many other commercial 3-
D engines, is frame-driven. This means if a task takes up
too much processing time, the frame-rate will drop, clients
will lose their connections, and the engine will become non-
responsive.

The Half-Life engine itself handles physics, rendering 3-D
geometry, drawing textures, playing sound effects, and man-
aging client connections. Counter Strike has its own Dynamic
Link Library (DLL) to manage Counter Strike specific tasks,
which includes weapon profiles (rate of fire and reload time),
user interface, and gameplay code (win/loss conditions). Since
Counter Strike does not have any of its own AI code, it
became necessary to employ the use of yet another DLL
which contained our bot code. Figure 4 shows how the DLLs
and the engine interact. All components enclosed inside of
the Half-Life Dedicated Server communicate by means of
function calls. As mentioned previously, we wanted to keep
the frame rate inside of Half-Life up, so we developed a GA
Server which manages all GA-related operations outside of the
Half-Life engine. The GA Server communicates directly with
the Bot DLL via TCP/IP. The bots receive new parameters
from the GA Server each round. At the end of the round,
they report their score, which becomes the fitness for that
individual (parameter set). The GA Server is written in Java, so
it is platform independent. Since the Half-Life engine is real-
time dependent for its physics calculations, it is infeasible to
speed up the engine artificially, while maintaining a reliable
simulation. Realizing this, we are working towards upgrading
the GA Server so it can manage multiple Half-Life Dedicated
Servers, all of which are running simulations of our bots in
parallel. This will significantly reduce our simulation time.
Currently, it takes over two hours to run 50 generations with
a population size of 30.

Ideally, commercial 3-D engines could be more modular. At
the moment, the physics, the graphics, and multiplayer logic,
run interdependently. For the purpose of tuning bots, it is only
necessary for us to have information about the physics and
geometry of the world--rendering graphics for a bot is wasteful.

III. METHODOLOGY

In order to have a genetic algorithm tune the parameterized
rule-based AI, we had to do: (1) select parameters to tune, (2)
allow the GA to evolve parameters values, and (3) pit the
GAABs against bots which were tuned by us. As we have
many years of Counter Strike playing experience as well as
a solid understanding of the elements of which a good bot is
comprised, the bots we tuned are challenging to most veteran
Counter Strike players.

Although many commercial bots cheat to play well, our
bots, however, do not cheat. Cheating game AI destroys the
game experience [10]. For example, imagine a human player
is hiding behind a car, cheating game AI would be able to
detect this player just as easily as if they were standing in the
middle of the street. Our bots, however, use sensor information
gathered from their environment much like human players. For
example, if they detect another player it is only because the
bot has a line of sight to the player, or it has heard the player’s
footsteps.

The rest of this section will describe the selection of para-
eters, encoding, the evaluation function, the GA’s parameters,
and Counter Strike game settings.

A. Parameter Selection

First, we identified the parameters to optimize. The two
sets of parameters we focused on were: (1) weapon selection
parameters and (2) aggressivity parameters (which ultimately
affects path preference). The weapon selection and aggressiv-
ity of the bot are closely-related in playing Counter Strike.
Previously, we described a bot which used a sniper rifle to
play more defensively, waiting for the enemy to come to
it. Also, a bot that uses small automatic weapons should be
highly aggressive to be effective against its enemies because
its weapons have limited range. There exists no one correct
strategy to Counter Strike, but it is generally accepted that
following these styles of play will lead to a better score. Since
weapon selection and aggressivity are somewhat dependent
sets of parameters, we choose to allow the GA to optimize
these bot parameter sets in the hope that the GA will find a
player who fits either one of these predominant styles. It is
also quite possible that the GA will arrive at an strategy that
is not easily understood yet effective.

B. Encoding

The encoding was straightforward. Each parameter was
encoded into a binary string consisting of 178 bits. Figure 5
shows the chromosome’s layout.
The Half-Life engine’s architecture relies on communication between DLLs to operate the game. This figure shows how the various DLLs interact with each other inside of the Half-Life Dedicated Server (HLDS) and how our GA Server communicates with the HLDS.

Fig. 4. The Half-Life engine’s architecture relies on communication between DLLs to operate the game. This figure shows how the various DLLs interact with each other inside of the Half-Life Dedicated Server (HLDS) and how our GA Server communicates with the HLDS.

Fig. 5. Basic layout a of chromosome. Each parameter was represented in binary.

C. Evaluation Function

The evaluation function was an approximation of the standard Counter Strike Payoff/Fine table found in Table III and provided a straightforward method for measuring fitness. For example, if a bot is tuned with parameters that would lead to excessive friendly fire (such as high aggressivity and a strong weapon preference for automatic shotguns), then it would receive a low fitness per the Payoff/Fine table since team killing is severely punished.

D. GA Parameters

The GA used the parameters in Table IV during the training phase for the bots. Since we considered our search space rather large, we wanted the GA to move quickly away from poor parameter selections. We chose CHC as our selection method [5]. During crossover, CHC doubles the population to $2n$. Then, the best $n$ individuals are chosen from the parents and offspring. CHC’s elitist selection allowed the best individuals to remain in the population after crossover. Since our crossover probability was so high, elitism was necessary to prevent the high fitness individuals from being destroyed during crossover.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generations</td>
<td>50</td>
</tr>
<tr>
<td>Individuals</td>
<td>30</td>
</tr>
<tr>
<td>Crossover Points</td>
<td>2</td>
</tr>
<tr>
<td>Probability of Crossover</td>
<td>0.95</td>
</tr>
<tr>
<td>Probability of Mutation</td>
<td>0.1</td>
</tr>
<tr>
<td>Selection Method</td>
<td>CHC-GA</td>
</tr>
<tr>
<td>Chromosome Length</td>
<td>178</td>
</tr>
</tbody>
</table>

E. Game Settings

Counter Strike has settings which determine the gameplay constraints and style. We changed the round time from its default 5:00 minutes per round to 3:00 minutes. This reduced the evaluation time per generation by a minimum of 2:00 minutes, which was a significant gain. The teams were even, 15 counter terrorists versus 15 terrorists. Each team was randomly composed of both GAABs and bots tuned by us. The map chosen for the match was "de-dust2," a well-rounded map, which offers a good mix of both indoor and outdoor combat as well as long-range and short-range combat. An overhead view of "de-dust2" can be seen in Figure 3. It is also popular among Counter Strike players.

IV. RESULTS

The results of the GA can be seen in Figure 6. The GA made steady progress during the 50 generations.
The best individual’s phenotype from generation 50 was saved to file. Then, this phenotype was shared among 15 GAABs. These 15 GAABs then played over 100 rounds of Counter Strike against bots whose parameters we tuned. The average time per round was 2 minutes and 4 seconds. The results of the match can be found in Table V. The first column shows the names of the two teams. The second column shows the standard deviation of each team’s average skill. Finally, the fourth column shows the median skill on each team.

<table>
<thead>
<tr>
<th>Team</th>
<th>Average Skill</th>
<th>Standard Deviation</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAABs</td>
<td>1005</td>
<td>55.2</td>
<td>987</td>
</tr>
<tr>
<td>Bots tuned by us</td>
<td>999</td>
<td>40.3</td>
<td>991</td>
</tr>
</tbody>
</table>

The statistics found in Table V are based on standard tournament ranking of Counter Strike players [13]. The skill of a bot is a calculated by the following formula:

\[ \text{NewSkill} = \text{Skill} + \frac{K}{1 + 10^{(k-v)/1000}} \]

Skill is calculated using the ELO Method, a standard chess player rating system created by Arpad Elo [4]. The system takes into consideration two important factors when rewarding skill points to a player: (1) the difficulty of the kill and (2) the experience of the current player [13]. Each player begins with skill 1000 by default. When someone is killed, the resulting skill of each player is calculated by the formula above, where \( K \) represents the experience of the player and \( k \) and \( v \) represent the skills of the killer and victim respectively. \( K \) is a simple coefficient which begins at 20 and after the player has 100 kills or more, \( K \) is reduced to 15 [13].

When observing the bots in game, the GAABs and the bots tuned by us play at first glance the same. This is expected since they share the same rule-based logic. We noticed that the GAABs would generally begin with no particular preference for a single weapon or even single type of weapon (shotgun, sub-machine gun, sniper rifle, or assault rifle). By the last few generations, they would converge strongly on at least one weapon from each of the various types. In some cases, however, when the GAABs only converged on a single weapon preference, they chose one inexpensive weapon and set for high aggressivity. This is an unexpected strategy but not unheard of. The GAABs could manage to inflict enough damage before dying to earn just barely enough money to purchase another inexpensive sub-machine gun for next round to do it over again.

Our results indicate that GAABs play statistically the same as bots tuned by us. This shows that a genetic algorithm tunes a bot’s parameters as well as a human could. With game engine or compiler support, this method would be far superior to tuning the bots manually. Programmers would only need to define a range of values for each parameter. The rest of the work would be performed by the GA. Depending on the constraints the GA evaluation function exerts on the bots, the AI could be universally shaped to favor a certain play style or tactic. A good mix of different types of bots makes a game more interesting. Different styles of bots could be created by simply changing the evaluation function.

V. CONCLUSIONS AND FUTURE WORK

This paper presented a method for tuning the parameterized rules of rule-based, expert system robot controllers (bots) using a genetic algorithm. Our results show that using this method results in bots that play as well as bots tuned by a human with expert knowledge about the game. We believe this method is generalizable not only to other first-person shooter bots but to other games. This method significantly reduces the time to develop such systems, moreover, this method, with compiler or game engine support, could be completely automated, allowing the programmer to focus on other tasks. Also, the programmer is not required to be an expert in the game’s strategy. We believe this method could lead to more sophisticated bots since more rules can be added to the bot’s logic without the consequence of tuning a growing number of parameters using trial-and-error.

This work, like Khoo and Zubek’s work in employing computationally inexpensive methods to improve AI in computer games, is useful in that all of the extra computation performed to determine the correct parameters is done during a training phase which happens before human players ever meet the bots in game [9]. This saves developer and testing time while yielding similar bots.
For future work, we would like to move toward having people evaluate the bots as they play them to determine their fitness. Play-testing is nothing new for game development. It would be interesting to see the results of play-testers evaluating how human-like a bot is and then using their evaluation as the individual’s fitness. This would hopefully move the bots from being more efficient to playing more like human players.

Finally, we will look at using a genetic algorithm to determine the “optimal” configuration for a team. Bots can be coded with a personality which drives their behavior. A Leader bot will give radio commands and signals to other bots instead of trying to hunt down opponents. On the other hand, a Psycho bot will simply try to score the most kills in the round regardless of how it affects the team and their accomplishing of the goal. A genetic algorithm could be used to determine the optimal configuration of personalities, given some time to train. This work represents a start in a promising new area of research.

ACKNOWLEDGEMENT

This material is based in part upon work supported by the Office of Naval Research under contract number N00014-03-1-0104. We would like to thank Jeffrey “Botman” Broome, Johannes Lampel, and the rest of the Half-Life coding community.

REFERENCES