# Agent Smith: a Real-Time Game-Playing Agent for Interactive Dynamic Games

Ryan Small
Department of Computer Science
University of Southern Maine
P.O. Box 9300
Portland, ME 04104
small@cs.usm.maine.edu

## ABSTRACT

The goal of this project is to develop an agent capable of learning and behaving autonomously and making decisions quickly in a dynamic environment. The agent's environment is a fast-paced interactive game known as Unreal Tournament 2004. Unreal allows for a spectator to watch the agent as it performs its tasks and even to enter the game and challenge the agent.

The agent's behavior is controlled by a rule-based system, which looks at multiple high-level conditions, such as whether the agent is weak, and determines single high-level actions, such as whether to head for the nearest known healing source. Using an evolutionary computation approach, in which the behavior is evolved over a number of generations, the agent learns increasingly better strategies for its environment.

Through the work in this project, we are exploring several research questions, including the development of successful vocabulary of high-level conditions and actions for the rule set, the challenges of rapid decision making, and the trade-offs between hand coding a rule set and using the evolutionary process to hone a rule set.

## Categories and Subject Descriptors

I.2.1 [**Applications and Expert Systems**]: Games.

## General Terms

Algorithms, Experimentation

## Keywords

Learning Classifier System (LCS), Games, Agents, Evolutionary Computation, Rule-Based System

## 1. INTRODUCTION

Over the last few years the video game industry has exploded into a multi-million dollar business. These newer games are becoming increasingly dynamic. Among some of these games is a genre known as First Person Shooter or FPS. These types of games are typically fast paced, forcing the player to react quickly to changes within the environment. Over time a player may develop a sense of when to execute a particular action for the current scenario. For instance when a weak and poorly equipped player meets an opposing player, the more experienced player may retreat while an new player may engage. This is the type of learning that will be mimicked through Agent Smith.

### 1.1 Primary Goals

The initial goal of this project is to develop a learning classifier system (LCS) capable of evolving a rule set that would allow a bot to compete in a fast-paced dynamic game. The bot would be able to rely on this evolved rule set to provide the appropriate response for a given scenario, thus maximizing the bot's score. It can be expected that the agent will initially perform poorly; however, over time this rule set will evolve into a more complete and sophisticated set of rules. Unlike a human, the bot will not be evolving its technical playing skills, but instead, the ability to react strategically to any given situation.

## 2. Background

### 2.1 First Person Shooter

Unreal Tournament 2004 was the game of choice for the project. This game was selected because it provides a Java API and the plug-in, Pogamut, that is available to help develop the bot. As with most FPS games the objective is to obtain a higher score than any of your opponents. A high score is achieved by killing your enemies; however, you are penalized for each death. The game also allows for humans to connect to the game and participate, which could allow a human to challenge the bot. The bot will have some natural advantages over any human players. Since the bot is fed information about the environment within its peripheral vision and doesn't need to do any visual processing, players would be unable to blend in with their surroundings, as humans would try to do against other humans. The other advantage for the bot is its ability to be precise when shooting its weapon since every shot is made by passing in the location of the enemy.

### 2.2 Learning Classifier System

A learning classifier system is comprised of two parts. The first part of an LCS is the rule-based system. This rule-based system uses a rule set which contains multiple rules. Each rule has an array of conditions and a single action. In these rule-based systems the environment is surveyed periodically, and then a check is made on each rule to see if the conditions match the current environment. When a matching set of conditions is found the associated action is executed.

The second part of the system is the evolutionary process known as a genetic algorithm. During the evolutionary process a rule set is modified using a survival of the fittest scheme. Traits that made a particular rule set effective are likely to carry on into future generations, while bad traits are dropped from the evolutionary process.

## 3. System Design

This section will describe the four main components of this system, including the Unreal game environment that the bot plays in, the Pogamut plug-in that allows external bots to connect to an Unreal game, the Chimera LCS that will be used to evolve the rule sets used by the bot, and the Agent Smith bot that controls the avatar within the Unreal game.

## 3.1 Unreal

The game is set up in such that the bot will initially be challenged by another bot. This second bot is static and its behavior will always be the same given a particular state. The game server does not have any terminating conditions and allows the bot to continue to evolve without any human interaction. Because Smith adapts to its environment only a single map is used as a different map may find different rule sets to be effective.

## 3.2 Pogamut

The Pogamut plug-in was developed for the Netbeans IDE. Pogamut was specifically created for the development of bots in Unreal Tournament 2004. Pogamut takes care of connecting to the game and handling any messages given by the server. Fortunately, the plug-in also has developed packages that simplify finding the location of items, way-points, and other information that may be useful to the bot. This tool provides example bots. One of the examples provided will be the static challenger in this project.

## 3.3 Chimera

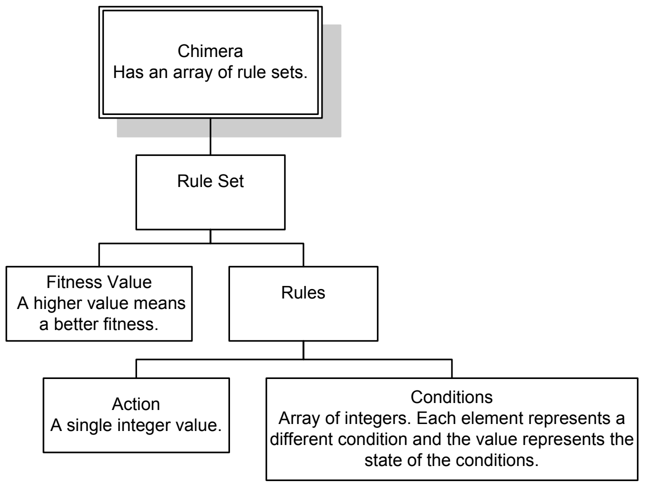
For this project, the Agent Smith bot will use Chimera, a LCS to evolve its rule sets. Chimera was developed from scratch using Java. This encoding scheme is further explained in section 3.4.1. Chimera uses a collection of solutions or rule sets; these rule sets are evolved over time. Ideally, the average fitness of the population will increase with each generation. The individual rule sets themselves will become more complex. As more rules are introduced into a rule set, the specificity increases, allowing for the fine tuning to control the appropriate action.

### *3.3.1 Rule sets*

The rule sets used contain a variable number of rules. The number of rules a rule set will carry depends on the fitness of its parents. The rule size depends on the fitness of both parents, and is relative to the size of the longer rule size between both parents. If both parents are above average fitness, then there will be one additional rule. If both parents are below average fitness the rule size will be one less. If each parent is on a different size of the average fitness the rule length will equal the length of the longer rule set.

Each rule is made of an array of integers representing the conditions and a single action. The array of integers is a fixed length. This forces each element in the array to be specified with a value. Before Chimera begins the length of the array may be changed, but not during evolution. Smith will only be concerned with four conditions, thus the array will be a length of four.

During initialization all of the rule sets will be created randomly. This approach introduces plenty of diversity. Diversity is maintained through the crossover method. During the crossover each rule has a chance for a mutation. Although the chance is small, the mutation chance is calculated individually for each rule. The diversity is important to prevent the population of rule sets to converge on a single solution.



Chimera
Has an array of rule sets.
Rule Set
Fitness Value
A higher value means a better fitness.
Rules
Action
A single integer value.
Conditions
Array of integers. Each element represents a different condition and the value represents the state of the conditions.

**Figure 1. Chimera Architecture**

### *3.3.2 Fitness Function*

Chimera delegates the responsibility of evaluation a rule set to a user defined function. In Smith's case, the fitness function is the most time-consuming part of the evolutionary process, as Smith will need to spend time playing Unreal to evaluate the rule set. The function used in this project will focus on creating a rule set that is primarily aggressive, but in some rare situations will retreat. By awarding more points for a kill, the bot will develop a more aggressive rule set.

### *3.3.3 Evolution*

Chimera uses a four stage process. These four stages are known as the initialization, evaluation, selection, and reproduction. The last three stages are repeated for the number of generations specified.

During the initialization phase the population is randomly generated. The initialization will only happen once and is the first stage in the process. There will be no changes to the rule set once they are initialized.

The second phase utilizes the fitness function created specifically for its application. In this case, the bot will be using the rule set to compete in the game. At the end of the round the fitness is determined by the bot's success.

In the selection phase rule sets are randomly selected based on the fitness value assigned at the end of the evaluation phase. This would be best viewed as a pie chart; the greater the fitness value, the greater the chance of being selected. Using this selection process, the rule sets that are selected and are passed into the fourth stage.

During the final phase the two selected rule sets create a new rule set that contains parts from both parents. This is the stage where mutations may occur. At the end of this stage a new generation will have been created and the evaluation process will begin again.

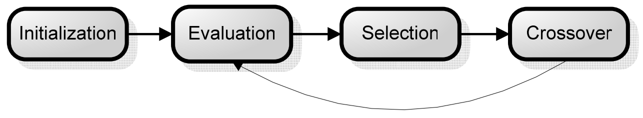


**Figure 3. Evolutionary Process**

The process will repeat for fifty generations, using a population size of ten. With ten minute rounds, the entire experiment runs for about eighty-three hours. These rule sets can be saved and further evolution may continue at a later time.

## 3.4 Agent Smith

Agent Smith is the interface between Chimera and the Unreal game. In its basic form, Smith is a java based program, which merges the evolutionary process of Chimera with the Unreal game. Smith is primarily the fitness function for Chimera. Agent Smith will initialize Chimera. When Chimera needs a rule set evaluated it will call one of Smith's methods. Once in the method, the round is considered to have begun. During the round, the method will continue to reassess the current state of the environment, then looking for the best matching rule within rule set. Once a rule has been found, the associated action will be executed within the Unreal game using the API provided by Pogamut. After ten minutes the round is considered complete and a fitness value is returned based on the bot's success. The entire process is repeated for each rule set within the population for each generation.

### *3.4.1 Smith's Encoding*

In order for Chimera to remain generic it uses an encoding scheme that requires everything to be converted to integers. Smith will use three alleles to represent the values of the conditions. If the value of the condition is zero then the condition is considered *false*, if the value is one then the condition is *true*, a value of two is a "don't care" and means the condition can be *true* or *false*. Since each condition is represented as an element within the integer array, the length of the array will be equal to the number of conditions used. This experiment looks at four conditions; the health, armor, weapon, and visibility of the enemy. To determine the value of a condition like health, a limit is used. If the health of the bot goes below the limit, then the value is considered *true*. The same method is used for armor. To determine the value of the weapon condition, a list of weapons is used by the bot. If the weapon the bot is currently using is on the list, then the value can be considered true. The values for whether the bot can see an enemy is considered *true*, if Smith can see an enemy.

The evolutionary process may specify multiple rules that use a single action if they desire. Using multiple rules to describe a single action produces the fine tuning that may be needed for some situations, at the cost of time. If the max rule size is set to high, the array of rules could become large enough to cause undesirable delays when trying to find a rule that matches the current state of the environment.

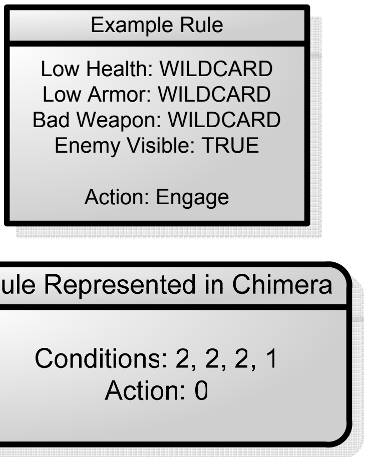


**Figure 4. Example of a Rule Represented in Chimera**

## 4. Results

In the experiments conducted here, the primary goal of the project was achieved. The bot showed progressive improvements over time. Although Agent Smith continued to evolve and became more efficient, it was still unable to dominate the challenger on a consistent basis. Smith's rules became more complex with each generation, providing more specificity for more scenarios.

The rule sets used in the early generations scored very low and seemed unintelligent. Allowing Smith to start based on hand-coded rule sets would save a lot of time during the evolutionary process and provide it a jump start. As an example, some of the rule sets gave the instruction to engage only when the enemy was not visible. Since these rule sets scored poorly, they were not carried on into future generations.

## 5. Future Work

The initial stage of development demonstrates that evolutionary computation is a promising approach to working with interactive games. However, there is obviously much room to extend this project.

### *5.1.1 Condition Limits*

This initial work limited the conditions to four criteria that appeared to be the most relevant, and four possible actions. When playing Unreal, not only do I perform more than four different actions, but my actions are also based on more than four conditions. It would be nice to see Agent Smith become more flexible by adding more complex actions. More conditions could be added to help make more informed decisions. Some of these

### *5.1.2 Technical Evolution*

Evolution could also be incorporated into the technical aspect of the game play, techniques such as strafing and camping could help excel the bots efficiency. During this experiment actions such as healing lead to the bot randomly grabbing healing items. Allowing the bot to develop a sense of priority over these various healing items may allow the bot to spend less time in a healing state and more time in an engage state, thus increasing its score.

### 5.1.3 Team Evolution

The development of a team of Smiths could force the progression of each individual by allowing the team to share some of their best rule sets. This would allow for both a team and the individual bots to evolve. A system of communication could be used to increase efficiency when engaging the enemy. Although, the primary focus of this experiment would be the communication protocol that was able to adapt. This could potentially force the bots to play different roles. One bot may act as a scout, alerting its teammates of enemy positions, while other bots may escort the weakest bot to a healing location.

## 6. Acknowledgements

I would like to thank all of my friends in the Unix Lab for all of the Unreal advice, and a special thanks to Clare Bates Congdon for the artificial intelligence classes and all of her help with this project.

## 7. References

[1] Unreal Tournament 2004: www.unrealtournament2003.com/ut2004/index.html

[2] Pogamut: https://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php

[3] Netbeans: www.netbeans.org

[4] Jones, Tim M. AI Application Programming. 2nd ed. Boston: Charles River Media, 2005. 229-261