UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FRANCISCO JOSÉ PRATES ALEGRETTI

ChicuxBot – Genetic Algorithm Configured Behavior Network Multi-Agent for Quake II

Thesis presented in partial fulfillment of the requirements for the degree of Master in Computer Science

Prof. Dr. Dante Augusto Couto Barone
Advisor

Porto Alegre, September 2006.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Alegretti, Francisco José Prates

ChicuxBot – Genetic Algorithm Configured Behavior Network Multi-Agent for Quake II / Francisco José Prates Alegretti – Porto Alegre: Programa de Pós-Graduação em Computação, 2006.

64 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2006. Orientador: Dante Augusto Couto Barone.

1.Redes de Comportamento. 2.Algoritmos Genéticos 3.QuakeII. 4.Inteligência Artificial. I. Barone, Dante Augusto Couto. II. Titulo

ACKNOWLEDGEMENTS

To my father, who never allowed me to give up.

TABLE OF CONTENTS

LIST OF ABREVIATIONS AND INITIALS

3D	Three-Dimensional
AI	Artificial Intelligence
BN	Behavior Networks
CD	Compact Disc
DNA	Deoxyribonucleic Acid
GA	Genetic Algorithms
GUI	Graphical User Interface
IP	Internet Protocol
LAN	Local Area Network
MIT	Massachusetts Institute of Technology
PC	Personal Computer
UFRGS	Universidade Federal do Rio Grande do Sul

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This work describes the implementation of a multi-agent system using Behavior Networks configured by Genetic Algorithms. The system uses the computer game Quake II as the simulated environment for the agents. Behavior Networks are used as the decision making mechanism. The Genetic Algorithm is used to configure the parameters of the Behavior Network. Each agent of the system is an independent program that connects to the game server to perform tasks and to exchange genetic material in order to evolve. The results obtained indicate a dynamically configured multi-agent system that can evolve and adapt accordingly throughout the course of the game.

Key Words: Behavior Networks, Genetic Algorithm, Quake II, Multi-Agent, Artificial Intelligence.

ChicuxBot – Sistema Multi Agente de Rede de Comportamento Configurado por Algoritmo Genético para Quake II

RESUMO

Este trabalho descreve a implementação de um sistema multi agente usando Redes de Comportamento configurada por Algoritmos Genéticos. O sistema utiliza o jogo de computador Quake II como o ambiente simulado para os agentes. Redes de Comportamento são utilizadas como o mecanismo de tomada de decisão. Um Algoritmo Genético é utilizado para configurar os parâmetros da Rede de Comportamento. Cada agente é um programa independente que se conecta ao servidor do jogo para realizar tarefas e trocar material genético a fim de evoluir. Os resultados obtidos mostram um ambiente multi agente dinamicamente configurado capaz de evoluir e se adaptar apropriadamente conforme o andamento do jogo.

Palavras-Chave: Redes de Comportamento, Algoritmo Genético, Quake II, Multi Agente, Inteligência Artificial.

1 INTRODUCTION

This text describes a solution to the problem of configuring the global parameters for a Behavior Network (MAES, 1989b). The solution developed in this work uses Genetic Algorithms (HOLLAND, 1975) to treat the problem of global parameter configuration. Behavior Networks (BN) are typically used to control the behavior of a multi-agent system. In order to develop the solution and verify that Genetic Algorithms (GA) can be successfully used to configure the global parameters of a BN, the well-known computer game Quake2 (IDSOFTWARE, 2006) has been chosen as the testing platform for the experiments.

Games constitute a good domain for the exploration of Artificial Intelligence (AI). In general, games have well defined rules, which make it easy to measure success and failure (RICH, 1988). A game can be used as a research platform (LAIRD, 2001) to investigate, develop and test AI algorithms (ADOBBATI, 2001). In this research, the computer game Quake2 is used as the simulated environment for the experiments. The game has a complex and dynamic environment (JACOBS, 2005) that runs in real time. The software developed for this research implements a multi-agent system. The actions of an agent is selected by a Behavior Network algorithm. The Behavior Networks are configured dynamically by Genetic Algorithms.

The result is an automated, computer-controlled player for the *deathmatch* game mode of Quake2. This automated player, named ChicuxBot, can adapt to different opponents during the course of the game, dynamically changing its own behavior through the configuration of the Behavior Network's parameters by the Genetic Algorithm. The evolution of the ChicuxBot shown in lab experiments indicate that Genetic Algorithms can indeed be successfully used to solve the global parameters configuration problem.

This chapter introduces Quake2 and the ChicuxBot. The definition of the problem engaged by this work is presented. The objective of the research is defined. The importance of the problem is discussed. And criteria for the success of the solution are established.

The remainder of this document is organized as follows. Chapter 2 describes Behavior Networks, showing the problem of configuring the global parameters of the network. The following chapter discusses Genetic Algorithms. Chapter 4 shows how the implementation of the ChicuxBot was done. The next chapter presents a startup guide for running the ChicuxBot. Chapter 6 describes the tests conducted and their results. And finally, chapter 7 contains the conclusions.

1.1 Quake II

Quake2 is a first-person-shooter (WOOD, 2004). The game is entirely three-dimensional. The players (human and computer controlled alike) have no global view of the world, but only the perception of their immediate surroundings (LENT, 1999). The game levels (or maps) are typically in the form of mazes with both open spaces and enclosed quarters. There are corridors, stairs, elevators, bridges and even water. Since the player does not have a global view of the map, he must learn to navigate through the levels and search for items and opponents. The player has a health meter that is decreased every time he is injured, like being shot by another player, falling from an elevated height or even staying under water for too long. The game player can also pick up weapons, ammunition and medical kits that heal injuries, restoring the health meter. Quake2 has been around for quite a while and therefore it can run smoothly on a very modest computer by today's standards. The game was released on December 6, 1997. It requires only a 90 MHz processor and 16 MB RAM to run. Quake2 has a 3D user interface, implemented in OpenGL (WRIGHT, 2000). The game server itself has no graphical interface and runs on text mode. This means that more processing power can be made available to the AI system when needed. Figure 1.1 shows a screenshot of the Quake2 client Graphical User Interface (GUI). Additionally, the game has open source code (NORLING, 2001) and there are versions for both Windows and Linux. Therefore, the game is platform independent and can be customized if required. To simply put it, this computer game is perhaps one of the best readily available simulation platforms to develop software agents and it has already been used to do so (LAIRD, 2000).

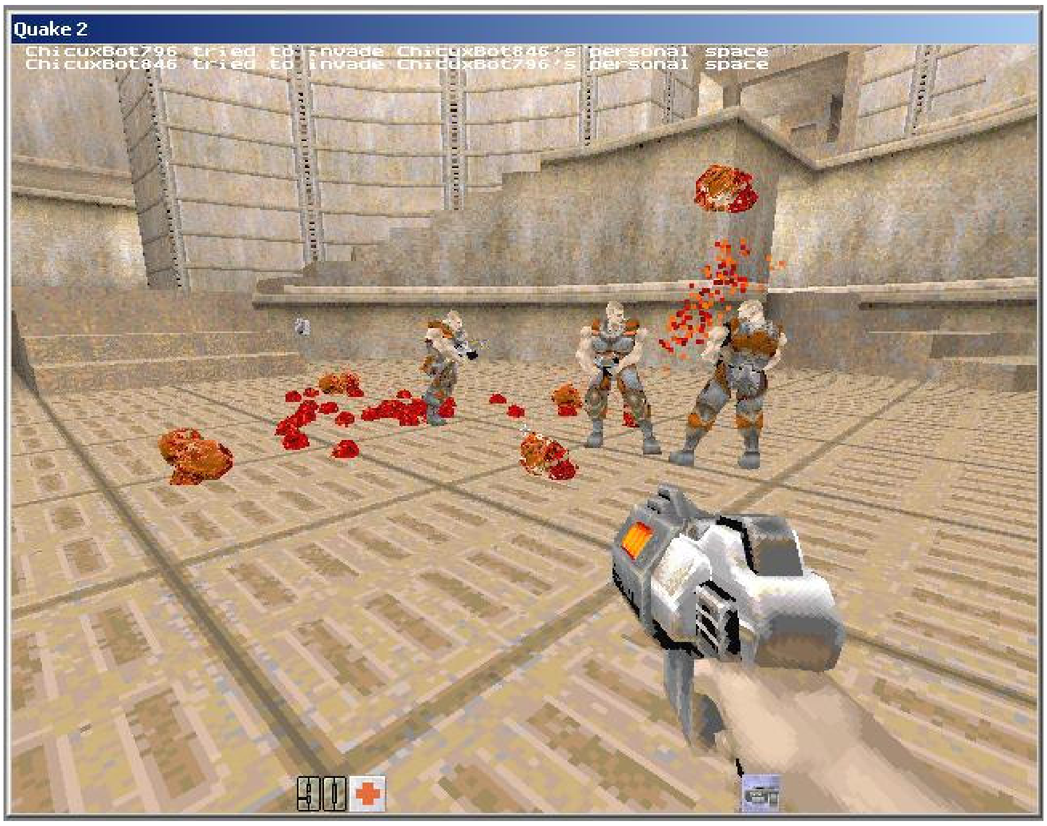

Quake 2
ChicuxBot796 tried to invade ChicuxBot846's personal space
ChicuxBot846 tried to invade ChicuxBot796's personal space
90

Figure 1.1: Screenshot of the Quake2 client user interface.

In the 3D first-person-shooter computer game jargon a "*bot*" is an automated player (AHN, 2006). *Bots* are used in multiplayer games to play against human players or even against other bots. The most popular of these types of games is called a "*deathmatch*", where the objective is to simply kill all the other players and, of course, not get killed. The winner is the player with the most kills (or "*frags*"). In order to do so, the player faces conflicting tasks, like deciding whether to attack an enemy or run away for more ammunition. Such decision making problems, in which conflicting objectives are involved, are ideal to be handled with Behavior Networks.

Behavior Networks is an action selection algorithm for autonomous agents (MAES, 1989b). In order to work properly, the Behavior Network must have a number of different parameters configured. However, the BN algorithm itself does not define how these parameters should be configured. It is up to the user (or programmer) to configure these settings. In large Behavior Networks this becomes a problem, because it is a complex, difficult and tedious job to be done by hand. That is why it becomes interesting to use Genetic Algorithms to automatically configure the parameters of the Behavior Network. There have been extensions of the BN proposed (RHODES, 1995). Extended Behavior Networks have already been used in Robot-Soccer (DORER, 1999)(NEBEL, 2003) and in other first-person-shooter games like Unreal Tournament (PINTO, 2005). However, so far, the global parameter configuration problem remains untreated.

1.2 ChicuxBot

ChicuxBot is a multi-agent system for Quake2. A *bot* is a computer-controlled player of Quake2. ChicuxBot uses Behavior Networks as its decision making mechanism which, in turn, is configured by Genetic Algorithms. The ChicuxBot multi-agent system runs several instances of single ChicuxBots. All bots have the same Behavior Network, but each one is configured differently, according to their genetic code. As the game runs, the bots increase their fitness points by destroying other bots. A mating season happens periodically in the Genetic Algorithm (SIMOES, 2000), when bots exchange genetic material. The genetic material carries the instructions on how to configure the Behavior Network. Since the best-fitted bots are most likely to pass on their genetic material to the next generation, the configuration of the ChicuxBot Behavior Network is improved through time.

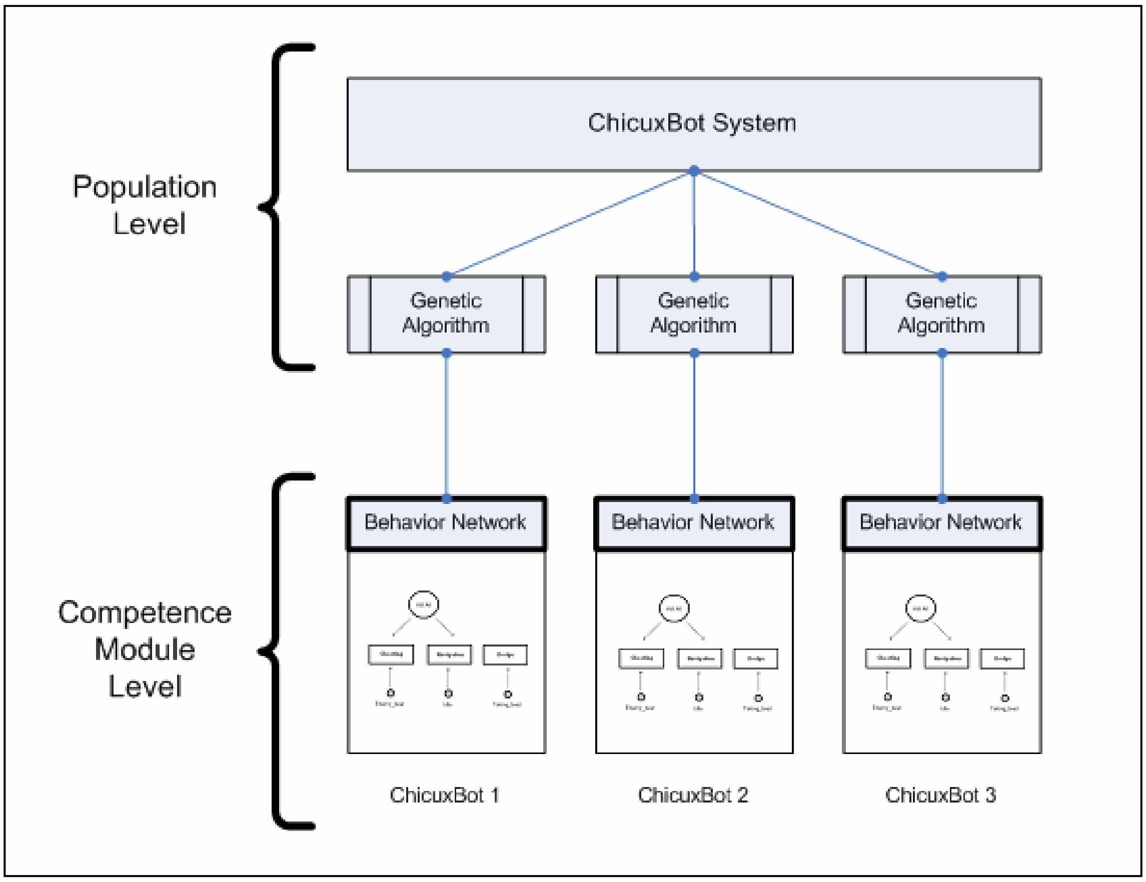


Figure 1.2: Levels of the ChicuxBot Architecture

The architecture of the ChicuxBot can be divided in two distinct levels. First, there is the *Competence Module* level that defines the abilities (or competences) that the *bot* has. This level is in the context of a single ChicuxBot. Several competence modules are combined in a Behavior Network to make a single ChicuxBot.

On the next level, Genetic Algorithms are used to configure the Behavior Network. In this level, multiple ChicuxBots form a population for the GA. The bots have to interact with the user for several cycles in order to "learn" the opponent's behavior and configure itself to play against that player's style. Figure 1.2 shows a diagram of the ChicuxBot architecture levels.

1.3 Definition of the Problem

Quake2 is a changing and unpredictable environment (GRAHAM, 2005). Behavior Networks is an algorithm for action selection fitted for such dynamic environments (MAES, 1989b) (RHODES, 1995). A BN is composed of *competence modules* and *links* between these modules. *Activation Energy* flows among the modules according to the configuration of the network's *global parameters*. Behavior Networks presents the problem of configuring its global parameters. Genetic Algorithms are an appropriate technique to solve the problem of configuring a Behavior Network. The objective of this work is to determine if Genetic Algorithms can be successfully used to configure the global parameters of a Behavior Network using the game Quake2 as a testing platform.

1.3.1 Importance of the Problem

The global parameters of a Behavior Network have to be properly configured in order for the technique to work effectively (MAES 1989a, 1991). Since the definition of Behavior Networks does not specify how this configuration can be done, it is often adjusted by hand. Or worse yet, the configuration of the global parameters is guessed by the programmer. Neither way of doing it can guarantee that the optimum configuration is found. And for large networks, the problem only gets harder. Therefore, without a proper solution for the configuration problem of the global parameters, the Behavior Networks algorithm may not reach its optimum efficiency.

1.3.2 Criteria of Success

The solution to the problem of configuring the global parameters of a Behavior Network will be successful if the performance of a BN configured by the proposed solution is better than the initial performance. In other words, if several agents with ad hoc configuration to their BN are put to compete with agents configured by the proposed solution, the performance of the latter should be significantly better.

1.4 Other Considerations

This section presents some theoretical concepts that are considered taken for granted in the rest of the text. The most important and complex of these is the concept of a Dynamic System, which is described in the following subsection.

1.4.1 Dynamic Systems

A dynamic system can be seen as a set of functions (rules and equations) that specify how variables change through time (WIKIPEDIA, 2005). A system of n dimensions is defined by a set of n differential equations of the first order.

The **state** of a dynamic system in a determined instant in time is represented by a point in the n-dimensional space by the variable values of the system x_1, x_2, …, x_n (coordinates) in that instant.

The set of all the possible states that can be achieved by a certain type of system defines a **phase space**. The sequence of states through time defines a curve in the phase space known as the **trajectory**. As time increases, the trajectories either occupy the whole phase space or converge to a set of minor dimension called the **attractor**.

Mathematically, a dynamic system is described as an initial condition problem. This means that functions, parameters values and the **initial condition** are necessary to evaluate how the system behaves. Opposed to the variables, the parameters do not change through time.

The dynamic system is **deterministic** if there exists, for every state, only one transition of state, that is, they are one-to-one. A dynamic system can also be **random** or **stochastic** if there exists more than one transition of state, that is, one-to-many.

A dynamic system can either be discrete or continuous in time. A **discrete** system is defined by a function, $z_1 = f(z_0)$, that originates the state z_1 in the instance of time immediately after the initial state z_0. The calculation process of the new state in a discrete system is called **iteration**. On the other hand, a **continuous** dynamic system is defined by a flux.

2 BEHAVIOR NETWORKS

This chapter introduces Behavior Networks. The problem of configuring values for weights and links of the network is explicitly shown. This particular problem is the main point of interest in this chapter. The text starts with a brief history of the Behavior Networks algorithm. Then, the basic ideas behind Behavior Networks and the main components of the algorithm are explained. The inner-workings of the algorithm is presented. Finally, we focus on the configuration problem of Behavior Networks.

2.1 Brief History

Behavior Networks was published in 1989 by Pattie Maes in an Artificial Intelligence memo of the Massachusetts Institute of Technology (MIT) (MAES, 1989b). The memo was entitled “How To Do The Right Thing” and, as the name suggested, presented a new algorithm to the problem of action selection for autonomous agents. The basic ideas behind Behavior Networks had already been explored by Maes in a more complex solution, but with less results, in another paper published in 1989: “The Dynamics of Action Selection” (MAES, 1989a). Just a couple of years later, Maes would publish another paper about Behavior Networks, exploring even further her ideas (MAES, 1991).

Behavior Networks can be traced back to the context of 1986, when Marvin Minsky published the Society of the Mind (MINSKY, 1986). Minsky states that an intelligent system is composed by a society of interacting agents. Each mindless agent incorporates a competence module. Like a society of ants, agents have their specific competence tasks, like finding a sugar lump, breaking it down to pieces, moving the pieces around and so on; thus, the society emerges the intelligence of being able to find food and carry it back to the nest. No one would attribute this intelligence to a single ant, but the ant colony, however, seems to have some intelligence. Pattie Maes developed Behavior Networks to solve the problem of how action can be controlled in such a system.

2.2 Behavior Networks Algorithm

Behavior Networks is an action selection algorithm (MAES, 1989b). An agent based on Behavior Networks is composed of a set of *competence modules*. Each module represents a particular behavior of the agent. A competence module can also be referred to as *behavior* or *action*. The manner in which the competence modules are programmed is not specified by the Behavior Network algorithm, and can be implemented using any technique desired. In fact, a competence module can even be implemented by another Behavior Network. Each module implements a specific task to solve a determined problem. In addition to the instructions on how to implement a specific behavior, a competence module ***i*** is composed of:

- c_i – Precondition list
- a_i – Add list
- d_i – Delete list
- λ_i – Activation Level

A competence module can only be executed once its list of preconditions has become entirely true. The *add list* contains the items that become true after the behavior has been executed. A module also has a *delete list* of predicates that become false after the action is performed. Activation Level is the amount of activation energy that a competence module holds in a specific moment in time (MAES, 1989b).

The competence modules of the network are connected via links that spread *activation energy*. The links can connect the competence modules to objectives, environment states or other modules. A module executes once it reaches a certain activation energy threshold. There are three different types of links that can either activate or inhibit the system:

- Successor links
- Predecessor links
- Conflicter links

A successor link connects two actions that should be taken in a particular sequence. For instance, the *bot* should first find an enemy and only then shoot at it. Formally, given the competence modules *x* and *y*, there is a successor link from *x* to *y* if every proposition in the add list of *x* is also a precondition of *y*. A predecessor link works the other way around, that is, it connects a module to another behavior that makes a precondition true. For example, the ChicuxBot will only dodge if a fire is shoot at it. Formally, there is a predecessor link from *x to y* if every precondition of *x* is also in the add list of *y*. Finally, there is a conflicter link between behaviors that conflict with one another, that is, an action that would make the precondition of another competence module false. There is a conflicter link from *x* to *y* for every precondition of *x* that is also in the delete list of *y*.

Activation energy flows through the Behavior Network. The energy comes from two major sources: the goals of the system and the state of the environment. If a Behavior Network is configured properly, the activation energy will accumulate in the competence module that holds the best action to be taken in order to achieve the goals of the system. The Activation Energy Flow can be categorized in the following:

- Activation by State
- Activation by Goals
- Inhibition by Protected Goals
- Activation of Successors
- Activation of Predecessors

- Inhibition of Conflicters

The state of the environment can inject activation energy into the system. For example, a state that represents an opponent player nearby might send activation energy to the competence module that shoots at the enemy. Goals can also increase the amount of activation energy in the system. For instance, the goal to kill the enemy can stimulate the navigation module to execute, so that the ChicuxBot will run around the map in search of enemies to destroy. Protected goals, on the other hand, can inhibit the system. Inhibition is the removal of activation energy from modules that would undo the achieved goal that is to be protected. If the *bot* has its aim locked at an enemy player and is ready to fire, it would be unwise to turn around and pursue another objective (like looking for ammo or health kits) before actually killing the enemy.

As mentioned before, Behavior Networks have global parameters that are used to configure the action selection behavior of the network (MAES, 1989b). The global parameters are:

- θ Activation Threshold
- φ Proposition Energy
- γ Goal Energy
- δ Protected Goal Energy

Activation Threshold is the amount of activation energy a module must achieve in order to execute. Preposition Energy is the quantity of activation energy a state injects into the network when its proposition is true. Goal Energy is the amount of activation energy a goal feeds into the system. Protected Goals, on the other hand, take activation energy away from the network.

2.3 The Configuration Problem

Behavior Networks has the problem of configuring values for the weights and links of the network. As Maes points out in her paper (MAES, 1989b), Behavior Networks "provide global parameters, which one can use to tune the action selection behavior to the characteristics of the task environment". Indeed, it does. But how exactly do we configure the global parameters? Maes paper does not specify how. For small Behavior Networks, with few links and modules, parameter configuration can be done by hand. But for larger networks, this task becomes extremely difficult and inefficient to be done manually.

Configuring the global parameters of a Behavior Network requires finding the optimal weight value of each link in the network. And this kind of problem is efficiently solved by Genetic Algorithms (GOLDBERG, 1989) (MARDLE, 1999) (SINGH, 2005). As is shown by the experiments and tests conducted with the ChicuxBot, Genetic Algorithms are appropriate to handle the exact kind of problem that the configuration of a Behavior Network parameters presents. That is, Genetic Algorithms provide an efficient solution to the problem of finding the optimum value in a wide-base search.

3 GENETIC ALGORITHMS

Genetic Algorithms is a huge topic and can easily fill up a whole book on its own. It is assumed that the reader is familiar with the concept of Genetic Algorithms. A detailed tutorial of the inner-workings of Genetic Algorithms escapes the scope of this text and, therefore, will not be shown here. What this chapter does present is a study of the theory and main characteristics behind Genetic Algorithms.

Rather than providing a minute description in exact details of every aspect in each component of a Genetic Algorithm, the following sections describe the ***reasoning*** behind the possible choices for each configurable item of the algorithm. The text examines the effects and consequences of the different configuration options for a Genetic Algorithm. What follows is not an introduction to the concepts of Genetic Algorithms, but an analysis on how to use the mechanisms made available by the technique.

The text starts with an introduction to Genetic Algorithms, presenting an overview of the algorithm. The following section studies the possible configuration options and development choices for the mechanisms. The final section briefly presents an overview on how the Genetic Algorithm of the ChicuxBot was implemented. The details of the particular implementation of the Genetic Algorithm used by the ChicuxBot to configure its Behavior Network are specified in chapter 4.

3.1 Genetic Algorithms Overview

Genetic Algorithms, as the name suggests, were inspired by Darwin's Theory of Evolution. They were proposed by John Holland, in the work entitled "Adaptation in Natural and Artificial Systems", published in 1975 (HOLLAND, 1975). The basic idea of Genetic Algorithms is to use the same mechanism of evolution found in Nature.

The population of the Genetic Algorithm represents a set of solutions to a particular problem. Each solution is represented by a chromosome. The structure or the encoding of the chromosome depend on the specific problem being handled by the Genetic Algorithm. Therefore, the encoding of the chromosomes vary dramatically from one problem to another.

Each new generation of the population should be better than the older one, that is, the Genetic Algorithm should be closer to finding the optimum solution to the problem. Human-beings are the proof that the evolution system works. The basic structure of a Genetic Algorithm is presented in figure 3.1.

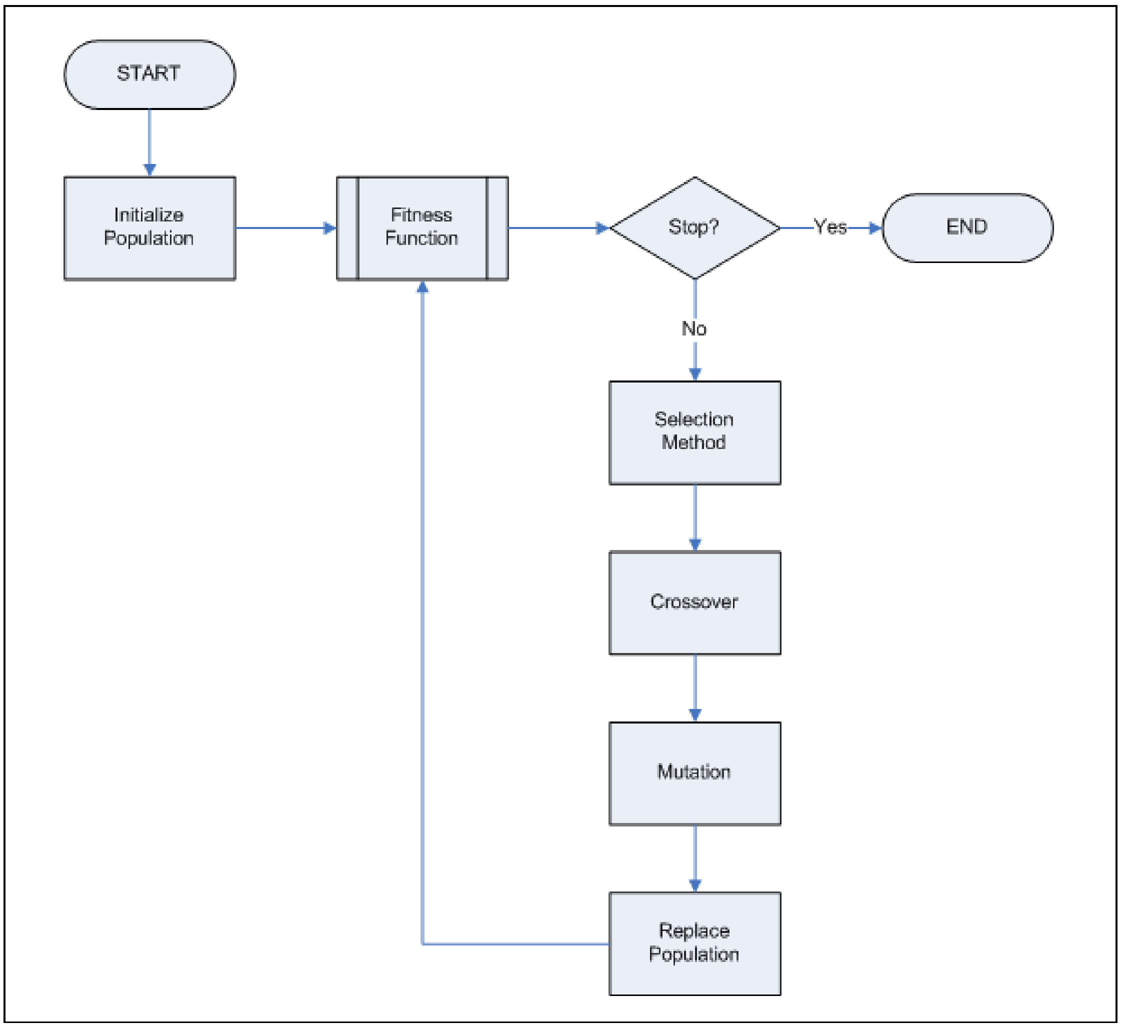


Figure 3.1: Basic Structure of a Genetic Algorithm

The Genetic Algorithm starts by initializing the population. Then it executes the fitness function for all individuals of the population. The fitness function evaluates each individual of the population, giving it a score according to its fitness. Once the whole population has been evaluated, the Genetic Algorithm checks if it has a good enough solution according to a predefined criteria. If the desired criteria has not been met, the algorithm starts the reproduction process. This process creates a new generation of individuals. The first step is to select the individuals of the population that will pass their genes along to the next generation. The manner in which the selection can be done is analyzed in the next section of this chapter. The selected individuals have their chromosomes recombined in the crossover process. The crossover produces the new generation and can either replace the old population partially or in its entirety. During the crossover, mutation of the genes can occur with a certain degree of probability, according to the configuration of the Genetic Algorithm.

3.2 Configuration Options

This section contains a description of the main parts of a Genetic Algorithm and is structured as follows. The two most important elements of a GA are analyzed first.

Namely, they are the chromosome structure and the fitness function. Then, the other remaining major parts of a Genetic Algorithm are described one by one. The configuration options of a Genetic Algorithm (GOLDBERG, 1989) are:

- Chromosome Structure
- Fitness Function
- Population Initialization
- Population Size
- Stop Criteria
- Selection Method
- Crossover
- Elitism
- Mutation
- Predation

The text starts with the two most important configuration options of any Genetic Algorithm: encoding of the population and the manner in which the individuals are evaluated. These two items are the most difficult parts to implement, and they are the major factors in determining the performance and efficiency of the algorithm. The other mechanisms of the Genetic Algorithm analyzed are: the initialization of the population, the definition of the population size, the stop criteria for the algorithm, the selection method used for reproduction and the crossover process executed to combine the genetic material of the selected individuals.

3.2.1 Chromosome Structure

The encoding of the population is directly related to the problem being treated. The structure of the chromosome can change radically from one problem to another. The definition of the chromosome is one of the two most important factors in determining the ability of the Genetic Algorithm to find the optimal solution for the problem. If the population is not encoded properly for the problem at hand, the Genetic Algorithm might not work at all.

It is up to the programmer to define the structure of the chromosomes according to the problem. There is no certified method to encode the population. It has to be done by hand. Each case is specific and every problem is different. Hence, defining the structure of the chromosomes is one of the most critical and difficult parts in the implementation of a Genetic Algorithm. It is generally agreed upon that half of the work in implementing a Genetic Algorithm is encoding the chromosome. The remaining half of the work to be done is defining the Fitness function.

3.2.2 Fitness Function

The fitness function determines how good (or fitted) an individual of the population is. The fitness function receives a chromosome as its input parameter and returns a score value for that genetic material. The closer that particular individual is to the optimum solution, the better it should score in the fitness evaluation.

Along with encoding the population, the fitness function is the other most important part in the implementation of a Genetic Algorithm. And just like the definition of the structure of a chromosome, there is no standard method to implement the fitness function. Each problem is a different case and will require a different solution. The programmer is in charge of developing an efficient fitness function.

3.2.3 Population Initialization

The initialization of the population is typically random. The value of each chromosome for a single individual is randomly chosen via some probabilistic function. The more different gene values that a random initialization function produces, the more genetic options the Genetic Algorithm will have to work with.

According to the particular needs of the problem being handled by the Genetic Algorithm, the initialization function can perform validations on the random data values. For instance, a problem might require that there are no repeated genes or that the values are within a certain range. With randomly initialized data, a validation check is a good practice to be performed.

The initialization function can also load the genome of a previously saved population. The Genetic Algorithm can then continue the evolutionary process for that population with new generations. The population can also be started with a seed, for a more directed search. And finally, if there is a good guess of where the optimal solution is in the search space, the population could be initialized by hand (or again, from a seed).

3.2.4 Population Size

The population size of a Genetic Algorithm can either be fixed or vary through time. In a population of fixed size, the number of individuals remains constant throughout the entire execution of the algorithm. On the other hand, a population of variable size can increase or decrease the number of individuals in the population. In this case, a control function is needed to manage the size of the population (MICHALEWICZ, 1994).

The purpose of having a population of variable size is to optimize the performance of the Genetic Algorithm. The size of the population directly affects the time a GA takes to execute. The bigger the population, the more processing power the algorithm will require. Therefore, the larger the population gets, the longer each generation will take to be processed.

However, as the population increases, so does the search space of the algorithm. This means that the GA will have better changes of finding the solution with fewer generations. So, although the time each iteration will take to execute with bigger populations increases, the algorithm can find the solution with less iterations. Changing the size of the population affects the time it will take for the Genetic Algorithm to find the solution.

Besides that, the need for a large or a small population during the execution of the algorithm can vary through time. In some situations, a big population might be very useful. For example, when the algorithm needs to broaden its search space, in order to avoid or get out of local extremes and continue its evolution. In other situations, however, a large population would only be a waste of time and processing power. For instance, when there is no need for a large search space and the algorithm is not using all of the genetic variety it has at its disposal to continue the search.

Varying the population size is a configuration that has to be carefully analyzed and fine-tuned. It is a trade-off between the time each iteration of the algorithm takes to execute and the total number of iterations needed to be performed in order to find the desired solution. A well adjusted population size variation control can certainly bring performance benefits to the Genetic Algorithm.

3.2.5 Stop Criteria

Most Genetic Algorithms specify a stop criteria to determine when the computation should be terminated. The stop criteria could be the number of iterations the algorithm has executed (in other words, the number of generations spawned), the time elapsed since the algorithm started running (establishment of a time limit) or an acceptable threshold for a good-enough solution, just to name a few.

If the stop criteria is poorly defined and it interrupts the computation prematurely, the Genetic Algorithm will not be able to find the best solution. On the other hand, if the stop criteria allows the evolution process to run for too long after the solution is already found, the Genetic Algorithm will take more time than is necessary to finish its processing. Or worst yet, if the computation goes on for too long, there is the possibility that the solution provided by the Genetic Algorithm will downgrade. This could happen, for example, due to an unfortunate mutation of a key individual or the whole population can simply start to go down an evolutionary path that will produce a solution that is worst than the one that is currently available.

3.2.6 Selection Method

The selection method determines how the algorithm chooses which individuals of the population will take part in the reproduction process and pass their genes along to the next generation. There are many types of selection methods. This section will analyze two of the most popular ones: the roulette wheel and the tournament methods.

The *Roulette Wheel* is probably the most popular technique used as the selection method for Genetic Algorithms. In this method, the entire population is represented by a segmented wheel (WHITLEY, 1994). The total number of segments in the wheel correspond to the number of individuals in the population. Each individual is represented by a segment according to its fitness value. The most fit an individual is, the bigger its segment in the wheel will be. To select an individual, the wheel is rolled. When the wheel stops rolling, the individual who's segment is facing the marker will be the winner. In this manner, the most fit individuals will have better chances of passing their genes along to the next generations. Poorly fitted individuals are unlikely to have many chances of passing along their genes.

In the *Tournament* selection method (GOLDBERG, 1990), *n* individuals are randomly selected from the population. The most fit individual from this group will have its genes passed along to the next generation via the crossover procedure. This process is repeated until enough individuals have been selected to reproduce and create the new generation.

3.2.7 Crossover

Crossover is the process of reassembling the genetic material of two chromosomes, producing a new individual. There is an enormous variety of ways in which this process can be done. However, they can all be classified in to just two categories, according to the way the chromosome is divided for the crossover: one point and multi-point.

In the *one point* crossover method, the chromosome is broken up in two parts (WHITLEY, 1994). The exact location where the chromosome is divided yields different categories of one point crossover methods: the single separation point could be placed exact in the middle of the chromosome, separating it into equal parts; or the chromosome could be divided into uneven halves. Furthermore, the point of division could be selected from a myriad of choices, varying each time for every crossover.

The *multi-point* crossover establishes several points along the chromosome for it to be divided (SPEARS, 1991). The number of divisions and the size of each segment can vary according to the configuration of the particular multi-point crossover implementation. If compared to the one point method, the multi-point crossover blends the genetic material more rapidly. Although this widens the chance of creating a better combination of genes, it also increases the chances of breaking up a good one.

3.2.8 Elitism

Elitism is the preservation of the most fitted individual from one generation to the next. The genetic code of the selected individual from the previous generation is copied gene by gene to the chromosome of the new individual in the next generation. In other words, a clone of the elite individual is made. One or more individuals can be cloned during the reproduction process, according to a predefine parameter.

The goal of elitism is to preserve the best solutions found in the previous generation (COSTA, 2004). However, it can have a side effect: elitism can hold or push the population back to a local maxima.

3.2.9 Mutation

Mutation is the alteration of the genetic code of a particular individual of the population. This mechanism introduces diversity or, in other words, fresh new genetic material to the population. This can be particularly useful, for instance, if the evolution of the GA is stuck at a local maximum and the population does not have enough diverse genetic material to get out of that local maxima. Mutation of the genetic material can introduce the gene needed to get the population out of jammed point. And with a little luck, mutation can provide a shortcut to finding the optimum solution.

On the other hand, mutation can also be disruptive to the evolution process (WHITLEY, 1994). If the mutation rate is set too high, then the Genetic Algorithm will not be able to focus on an evolutionary path and will never evolve to nothing. With a mutation rate too high, the Genetic Algorithm will practically perform a random search. Mutation can also harm the evolution if it affects a highly fit individual with a change that downgrades its performance or fitness value. This last case can happen whatever the rate of mutation is set to (unless it is set to zero, which would be disabled, of course). Therefore, even with restrained rate settings, mutation can be harmful to the evolution process. But it can also be the solution to escaping local maxima.

3.2.10 Predation

As it happens to all types of animals in Nature, the population of a Genetic Algorithm can also have predators (XIAODONG, 2003). A predator eliminates an individual and it’s respective genetic material from the population. Intuitively, one might think that the predator would always hunt and kill the least fitted individuals of a population. This maybe true for most of the cases. However this intuitive notion can be misleading. Typically, the predator will end up catching the slowest, least skilled, inexperienced or injured prey in the herd. Take, for example, a lion hunting down a deer. The predator will most likely attack the infants first. Then it would probably go for an injured adult next. The lion will, probably, go for the easiest ones to catch, that is, the least fitted ones.

3.3 Genetic Algorithm of the ChicuxBot

As will be described in details in the next chapter (Implementation of the ChicuxBot), the Genetic Algorithm implemented can work with a population of any size. The system can start with a population of fixed size and, throughout the course of the game, more agents can be added to the population. In fact, the ChicuxBot can even work as a single agent. In this peculiar configuration however, the only evolution of the system, if any, would come from mutation. The ChicuxBot can also work with only two agents. In this case, all the features of the Genetic Algorithm would be working, but the speed of the evolution would be slow. The more agents added to the system, the faster the improvement can happen.

The reproduction process of the Genetic Algorithm replaces the entire population. Partner selection of the ChicuxBot is done through the *roulette* technique. The roulette can be imagined as a pie chart, where the size of each slice of the pie corresponds to the value of the *fitness* of each *bot*. A slice of the pie is randomly chosen in the mating phase of the Genetic Algorithm. The bigger slices are more likely to be chosen than the smaller ones. Therefore, the higher the fitness of a *bot*, the more chances it will have of passing on its genetic code to the next generation.

As mentioned before, the Genetic Algorithm of the ChicuxBot uses *mutation*. This feature is specially useful in cases where a small population of agents gets stuck at a local maximum. Mutation provides a mechanism to break free of the local maximum and start climbing up the hill again.

4 IMPLEMENTATION OF THE CHICUX BOT

This section details implementation and functional aspects of the developed software. First, it is described how the system is structured and how it interacts with the Quake2 game server. Then the detailed implementation of the Behavior Network used by the software agents is shown, explaining the function of each of its components. Finally, the exact inner workings of the Genetic Algorithm is detailed. The software was programmed in C++ with Object Oriented methodology.

4.1 Software Structure

ChicuxBot is a multi-agent system. Each agent is composed of a Behavior Network configured by a Genetic Algorithm. All agents have the same Behavior Network, that is, every ChicuxBot in the system is running with the exact same competence modules. The difference between the agents is in the genetic code that configures the behavior of the network. The agents are controlled by Behavior Networks which, in turn, are dynamically configured by Genetic Algorithms in real-time during the course of the game.

The ChicuxBot multi-agent system uses the Quake2 game as its environment. Each agent in the system is an independent program. The agents connect to the Quake2 game server via sockets (STALLINGS, 2000). This enables each agent to run on a different machine and interact through a computer network. The whole system can also run on a single computer. All communications among the bots happen through the Quake2 server. The bots send messages and exchange genetic material through the console interface of the Quake2 server. It is the same interface that human players use to send text messages to one another. In fact, the messages exchanged by the ChicuxBots are string messages, and can be seen by the other players during the course of the game.

There are different ways to implement a *bot* for Quake2 (CHAMPANDARD, 2005). One can modify the game itself by inserting new code. This requires the re-compilation of the entire game. Another way is to develop a standalone program that connects to the game server. However, both methods require some knowledge of the inner workings of the source code from the game server. Since the main motivation for using Quake2 was saving time and getting an easy-to-use, off-the-shelf environment, it was chosen to use an interface called Q2 Bot Core (SWARTZLANDER, 2005). This interface does all the work to communicate with the Quake2 server. Although not as fast nor powerful as the other two possible approaches, the interface is efficient and quick to use.

Each instance of the ChicuxBot uses the Q2 Bot Core interface to connect to the Quake2 game server. Figure 4.1 shows an overview of all the parts involved in the system and how they connect. The Quake2 Server is in the center of the figure because it centralizes all communications between the rest of the parts in the system. ChicuxBots and human game players alike, connect in the same fashion to Quake2. As long as the Quake2 server is concerned, there is no difference between a ChicuxBot and a regular human player. The figure also shows the Quake2 Viewer, an instance of the Quake2 graphical client that provides an overview of the game. An observer can use the Quake2 Viewer to hover around the game maze without being seen or interacting with the other players. This is useful to get an overview of the match and see what is happening around in the game.

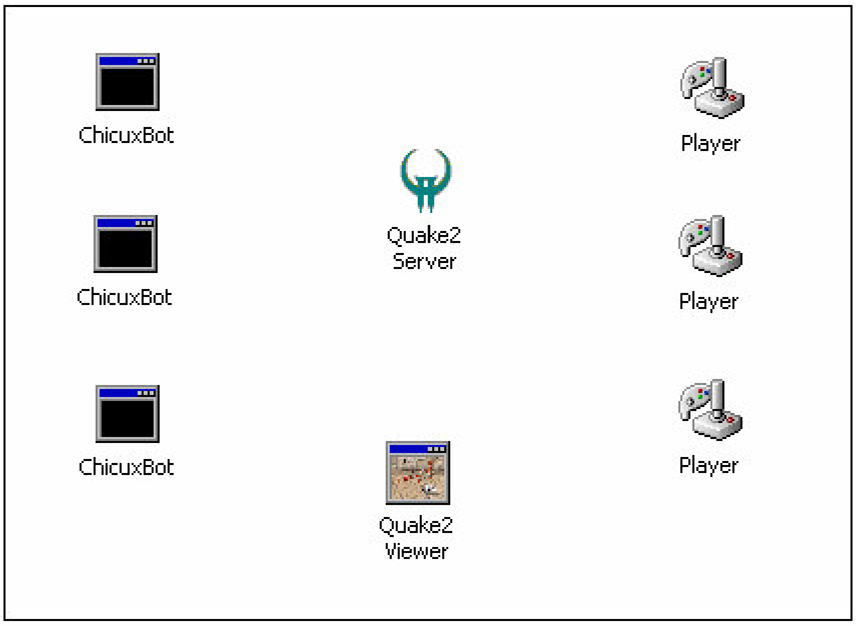


Figure 4.1: Overview of the ChicuxBot Multi-Agent System

4.2 Behavior Network of ChicuxBot

The Behavior Network implemented in the system is shown in the figure bellow. The same BN is used for all the agents of the system. The Behavior Net contains three competence modules that implement all the behavioral functions of the agent. There is a module for Shooting, one for Navigation and another one for Dodging. Basically, all a *bot* has to do is shoot opponent players, navigate around the map, and dodge enemy fire. Additionally, there are three environment states, represented in figure 4.2 by the gray filled circles. Each state could be viewed as an environment sensor, that continually scans the world; once the sensor detects a particular property, the state becomes true. Finally, there is the overall objective, that is to Kill All enemy players. This objective is also linked to competence modules. Thus, although the resulting Behavior Network is very simple, it contains all the necessary functions a Quake2 *bot* needs to play the game

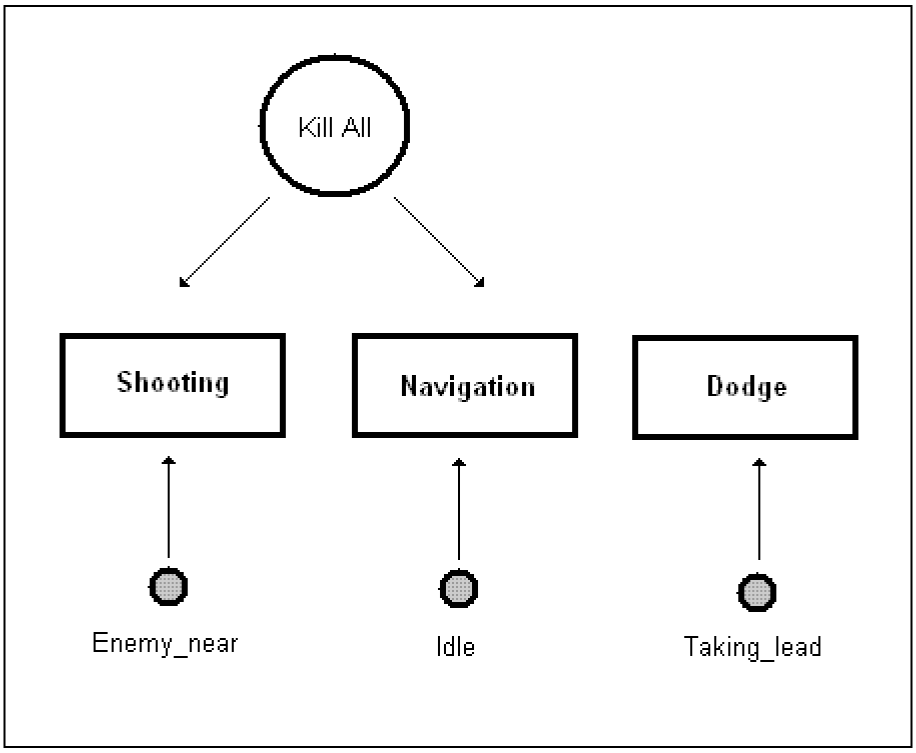


Figure 4.2: Behavior Network used in ChicuxBot.

Each link of the Behavior Network has an associated parameter that defines the amount of activation energy it carries. In addition to its specific task, each competence module has a variable that informs the amount of accumulated activation energy. The modules also have an energy threshold. Once the variable reaches this limit, the module is executed. As explained before, in Maes original Behavior Networks each module has an explicit add list of preconditions that have to become true in order for the module to execute (refer to section 2.2). Since the particular Behavior Network used in this research would not be too complex, the precondition lists of the competence modules are implicit in the environment states; that is, if a state has become true, it is because the precondition has also been fulfilled. The Behavior Network has been implemented in this peculiar way because it is designed to function in a real-time environment, where an explicit precondition list is not necessary. The competence modules can be composed of a different number of links to other modules, objectives or environment states.

To understand the exact inner working of the Behavior Network, consider the following example: when an enemy player enters the *bot's* line of sight, the *Enemy_near* environment state will become true; this will send the amount of activation energy specified by the link from the *Enemy_near* state to the Shooting competence module. For every cycle (determined by the Quake2 server and the socket connection speed), the state will send the same amount of activation energy through the link. The module will add up activation energy until it reaches the threshold, causing the behavior to be executed. In this example, the *bot* will shoot. Notice that the agent will shoot only once, and then the activation energy variable of the Shooting module will be reset; the whole process will have to repeat itself for the *bot* to shoot again.

Depending on the amount of activation energy a link transmits and on the threshold of each competence module, the agent can show distinct behaviors. For instance, if a *bot* has a high threshold on the Navigation module and the Idle state transmits a low amount of activation energy, this particular agent will tend to stand around for some time until it decides to start moving. On the other hand, if a *bot* has a low threshold and a high energy link, it will always be running around the map. These kinds of behaviors are determined by the activation energy transmission capacity of the links and the threshold of the modules which, in turn, are configured by the Genetic Algorithm. Therefore, the Genetic Algorithm directly configures the behavior of the *bot*.

Both the shooting and navigation modules involve difficult and fairly complex problems that could greatly benefit from sophisticated solutions. However, since the main objective of this research is to develop the overall genetically configured Behavior Network system, simple ad hoc solutions have been used on these modules. Nevertheless, it is highlighted here that these modules can be easily replaced by more advanced algorithms later on.

4.2.1 Shooting Module

The *shooting module* chooses the closest visible enemy to fire upon. The algorithm developed uses simple trigonometric functions in order to compute the correct angles the *bot* must turn in order to precisely shoot the target (SIMONS, 1985). The shooting module also causes the *bot* to chase its target if it tries to run away. The algorithm used by the shooting module is shown in figure 4.3.

The first step of the shooting module is to determine if an enemy is near. Only when the enemy is in sight that the module will perform the calculations to aim and fire. If there is an enemy near, the algorithm calculates the two-dimensional distance between the ChicuxBot and the target. The distance is used to calculate the angle the aim should turn in order to target the enemy. The angle calculation is performed by the following equation:

$$\alpha = \frac{T_x - P_x}{\sqrt{(P_x - T_x)^2 + (P_y - T_y)^2}} \qquad (4.1)$$

In Equation 4.1, *Px* and *Py* are the coordinates of the ChicuxBot; *Tx* and *Ty* are the coordinates of the target. The arccosine function returns the angles in radians. The final result value will be used to adjust the aim to hit the target.

The next step of the algorithm is to check if the enemy is coming from behind. If it is, the ChicuxBot should turn around to face the enemy. Then the shooting module adjusts the aim vertically. The vertical adjustment of the aim is done with pretty much the same formula of equation 4.1, except that the *z* axis is used instead of the *y* axis. In other words, the same formula is applied to a different plane in the three-dimensional space of the game. Once the aim is adjusted in both horizontal and vertical planes, the ChicuxBot can fire. As the flux diagram shows, if an enemy is near, the ChicuxBot will always shoot.

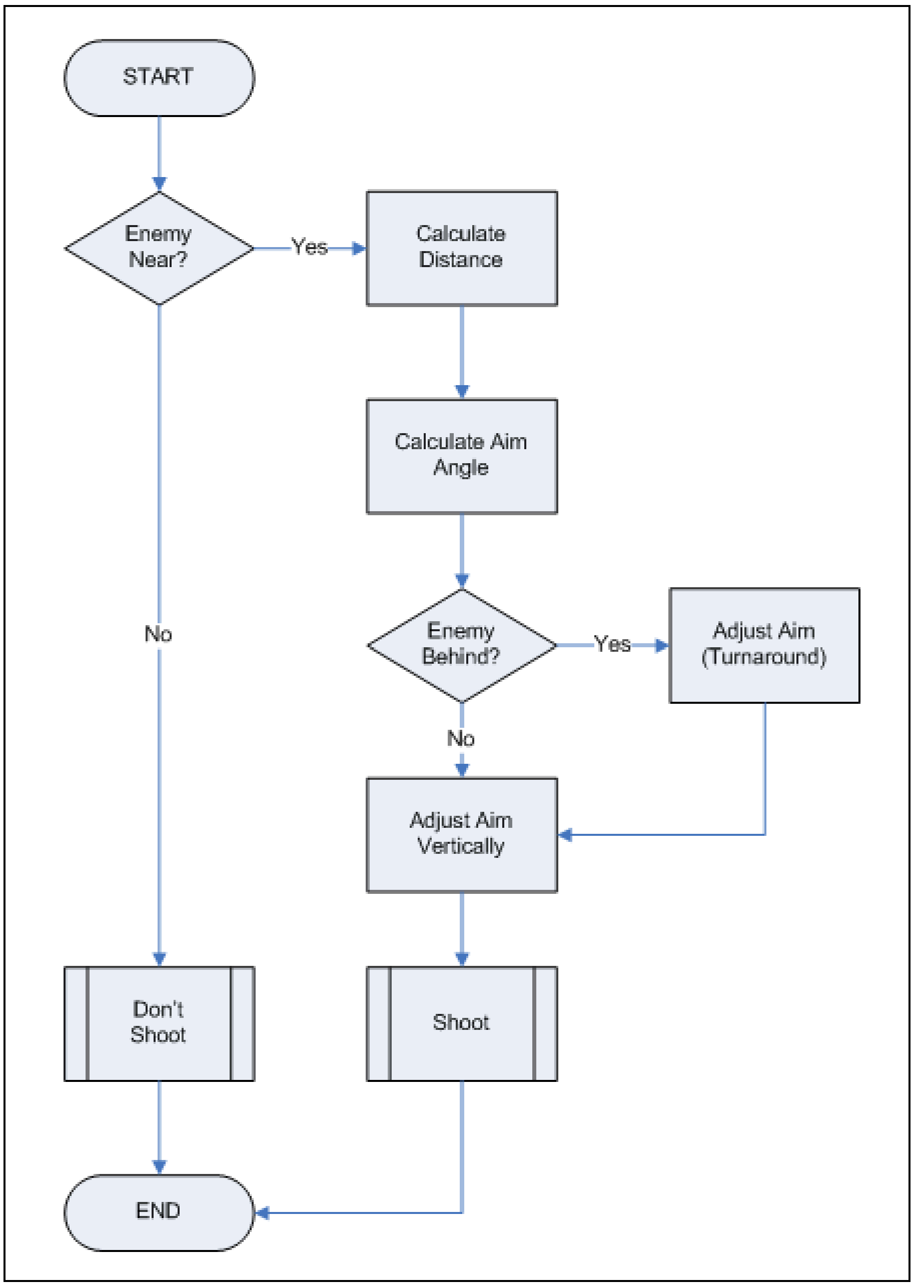


Figure 4.3: Algorithm used by the Shooting Module

4.2.2 Navigation Module

The *navigation module* works with predefined trails for each map. The trails are stored in a text file on disc, composed of a number of three dimension coordinates. Each file must have a main trail and any number of auxiliary trails needed for a particular map. The *bot* walks through the main trail while the module is active. If it ever gets off the main trail (for example, when the shooting module becomes active because of an approaching enemy), the navigation module will find the closest auxiliary trail which will lead the *bot* back to the main trail. Figure 4.4 shows the logic of the navigation module.

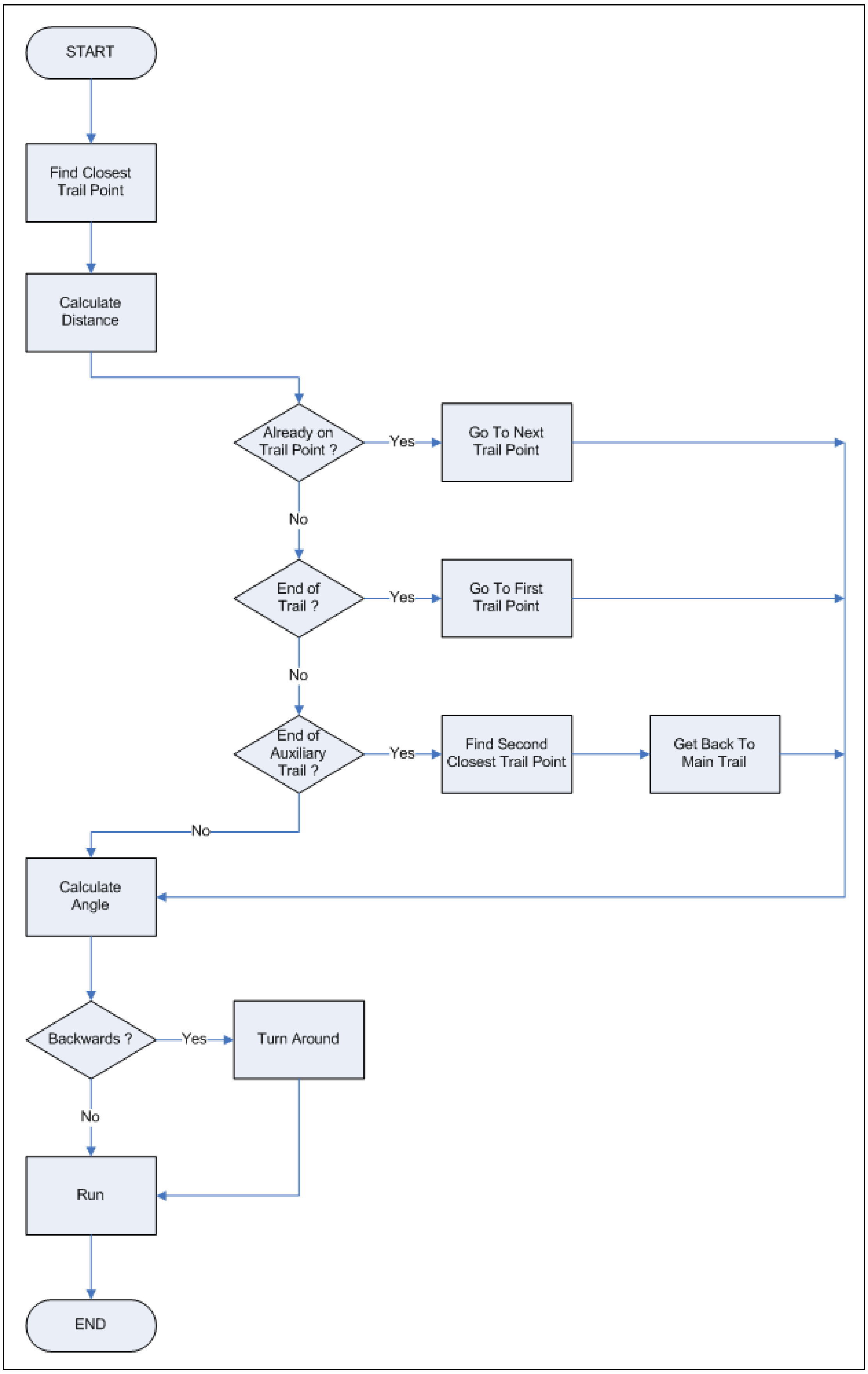


Figure 4.4: Algorithm used by the Navigation Module

The navigation module starts by finding the trail point the ChicuxBot is closest to. The trail points for each map are previously loaded on an array of user-defined *structs* at runtime. At the beginning of each game, while the level map is being loaded on the Quake 2 server, the ChicuxBot also loads the particular trail points for the specified game map.

The closest trail point to the ChicuxBot is found through an iterative search. The program searches the entire trail array, selecting the trail point that has the smallest distance to the ChicuxBot. Once the navigation module has found the closest trail point, it calculates the three-dimensional distance of that point to the current position of the ChicuxBot. The formula used to calculate the 3D distance is posted bellow:

$$d = \sqrt{(P_x - T_x)^2 + (P_y - T_y)^2 + (P_z - T_z)^2} \qquad (4.2)$$

As is shown above, the formula simply calculates the distance between two points in a three-dimensional space. In equation 4.2, (*Px, Py, Pz*) are the coordinate points of the ChicuxBot and (*Tx, Ty, Tz*) are the coordinates of the trail point.

Knowing the distance of the ChicuxBot to the trail point, the navigation module can determine if that point has already been reached. If so, then the ChicuxBot should go to the next trail point. If not, the navigation module checks if the selected point is not the end of the trail. If the ChicuxBot has reached the end of the trail, then it should go to the first trail point and start running the trail all over again. If it is not the end of the main trail, then the navigation module also checks if it is not the end of an auxiliary trail. If indeed it is the end of an auxiliary trail, then the ChicuxBot should get back to the main trail. This is done by finding the second closest point in the trail array to the ChicuxBot.

Having done all the checks and their respective adjustments, the navigation module has the coordinates for the target trail point the ChicuxBot should move to. It then calculates the angle that the ChicuxBot has to turn in order to face the destination trail point coordinates. The calculation of the angle is done with the same basic formula of equation 4.1 from the shooting module. If the ChicuxBot is facing the trail point backwards, it turns around to face the point straight ahead. Finally, the ChicuxBot moves forward.

4.2.3 Dodge Module

Last, but not least, the *dodge module* makes the *bot* jump in an attempt to avoid getting shot. The dodging algorithm simply detects when the ChicuxBot is being damaged and loosing health. When this is happening, the *bot* jumps.

4.3 Genetic Algorithm of ChicuxBot

All agents in the system have the same Behavior Network. This means that different instances of the ChicuxBot have the same competence modules with the same connections. But each agent has its own DNA. So every distinct agent in the system has its own set of parameters that configure the links between the competence modules of the Behavior Network. The parameters configured by the Genetic Algorithm include the activation energy threshold for a given competence module and the amount of energy that is carried by each one of its links. The DNA of each agent store the configuration values of its Behavior Network.

The following sections describe the characteristics of the Genetic Algorithm implemented in the ChicuxBot. The most important part of any GA is the encoding of the population and the fitness function that evaluates it. Therefore, those two components of the Genetic Algorithm are described first. The population's initialization and size are analyzed next. The selection method, crossover and mutation characteristics of the Genetic Algorithm of the ChicuxBot are also described in details.

4.3.1 Chromosome Structure

The structure of the agent's chromosome was implemented as an array of integers. The software was programmed with object-oriented methodology, so any changes in the Behavior Net are automatically adapted by the system. There is no need to adjust or re-program functions or structures by hand in the GA or any other part of the software; all modifications are automatically handled by the objects. This makes it easy to expand the BN, adding new links and competence modules, or modifying existing ones. On its default configuration, as shown in figure 4.2, the BN has three competence modules and three links between them to the world states. In this case, the chromosome array will have 6 positions: three for the threshold of each competence module, and three for the amount of activation energy carried by each link. Figure 4.5 illustrates an agent's chromosome. The chromosome values of each agent is initialized randomly.

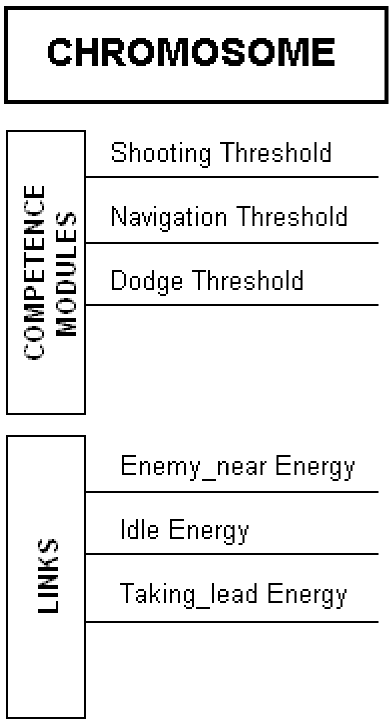


Figure 4.5: Structure of a chromosome.

4.3.2 Fitness Function

In general, the fitness function is one of the most difficult parts to implement in a Genetic Algorithm (WIKIPEDIA, 2006). But ChicuxBot uses Quake 2 as its development platform. Quake 2 is a game. Games have well defined rules. In such an environment, it is easy to measure success or failure. The success of a player in Quake 2 is directly related to its score in the game. Therefore, the fitness function for the GA of a game is simply the measure of the player's score. Having a simple and efficient fitness function is one of the main benefits of using a game as the agents environment.

What measures the quality and efficiency of a player in Quake 2 is the number of enemies it has killed during the game. The best players are the ones that killed more and died less. This is exactly what should be measured by the fitness function of the ChicuxBot.

In Quake 2, every time a player kills an enemy, his *frag* score is incremented. Each time the player kills himself (which happens quite frequently with human players by the way), the *frag* score is decreased. The fitness function is a count of the numbers of *frags* the ChicuxBot has accumulated during the match so far.

4.3.3 Population Initialization

The population of ChicuxBots are initialized randomly. There are no validations that have to be done on the randomly generated values of the chromosomes. The ChicuxBot uses a function that returns a pseudorandom number. The seed used for the pseudorandom number generator is the system time (in seconds) in which the function is called. The value of each gene in the chromosome is set one by one with a separate call to the random number generator function. Each individual of the population is initialized with a different seed.

4.3.4 Population Size

The total population size can vary, as new bots enter the game and others leave. The system will work with any number of bots, from 1 to n. The total number of bots in the system is limited by the number of bots supported by Quake 2.

In the case of only one bot present in the game, its DNA will tend to remain unaltered, because the reproduction process will always result in a clone of the bot. The only changes to the chromosome in this case, if any at all, will have to come from a mutation. As more bots are added to the population, the genetic variety increases.

The number of players influences directly the quality of the game and the skills required to win. If there are only a few players in the match, they can take too much time to find each other and the game can easily become tedious. This scenario favors the more skilled player, that knows how to position himself in the map, takes good aim and shoots precisely upon finding the enemy. On the other hand, if there are too many players in the map, the game can become crowded. In this case, sheer speed in shooting and brute force in the choice of weapons becomes more appropriate.

The ideal number of players depends on which map the match is being fought. For the map that is used by default in the ChicuxBot (q2dm1), the appropriate number for a good match is from eight to twelve players. The available map levels have different sizes and characteristics. There are close quarters maps and there are also levels with wide open areas. This influences in the choice of weapon to be used and the tactics to be employed.

4.3.5 Selection Method

The ChicuxBot uses the Roulette Wheel selection method. In this technique, each individual of the population is assigned to a number of segments in the roulette wheel (WHITLEY, 1994). The total number of segments in the roulette wheel equal to the sum of the fitness points from the whole population. In other words, the size of the roulette wheel will equal to the total number of *frags* in the game. The number of *frags* each ChicuxBot has is the number of consecutive segments in the roulette wheel it will have. Therefore, the percentage of the roulette wheel each individual has will be proportional to its fitness score.

The spinning of the roulette wheel is implemented by drawing a random number. The segments of the wheel are numbered from zero to the total number of *frags* in the game. The randomly picked number will determine the selected segment. The most fit individuals will have better chances of being selected and passing along their genes to the next generation, because they will be assigned to more segments of the roulette wheel. The more segments one individual has, the more chances it will have of being selected.

Each instance of the ChicuxBot executes its own roulette wheel. There is no central system that runs the process. This means that each *bot* decides which partner it will mate with, instead of being assigned a partner by a central process. This allows for a ChicuxBot to select itself to reproduce with. This will result in the cloning of the particular chromosome of that ChicuxBot.

4.3.6 Crossover

The crossover implemented in the ChicuxBot is of the multi-point type (DEJONG, 1991) (DEJONG, 1992). The algorithm goes through all the genes in the chromosome. For each gene of the new chromosome, a draw is made to determine if the value will come from one parent or the other. There is equal probability for the gene value to come from parent A or parent B. In other words, the process is done by “flipping a coin” to determine which genes will be passed along to the offspring.

There is a flip for each gene, that is, if gene 1 will come from progenitor A or B, if gene 2 will come from A or B and so on. In this manner, the crossover may produce an individual that is an exact copy (or clone) of either of its progenitors, or any mixture of the two. This allows for the maximum number of combination possibilities.

4.3.7 Mutation

Mutation (TATE, 1993) (CULBERSON, 1994) is the process that changes the value of a single gene in the chromosome of an individual of the population. In the ChicuxBot, mutation happens during the mating season. Specifically, mutation happens in the crossover process. Mutation is configured at a rate of 2% for each gene.

During the reproduction process, every gene is susceptible to suffer mutation. A specific gene in the chromosome can undergo mutation with the probability determined by the mutation rate. Therefore, a single chromosome can have multiple genes altered during the matting process. This means that an agent can have multiple genes altered by mutation from one generation to the next. Mutation is useful because it introduces new genetic material in the population. This can help the algorithm from getting stuck in a local maximum.

4.3.8 Elitism

The ChicuxBot software was implemented in a way that “naturally” allows elitism (CHAKRABORTY, 2003) to happen, as verified by testing. During the reproduction process, the selection of the partner is done by the roulette wheel technique. Since the most fitted individuals (the elite) will have bigger segments of the roulette wheel, it is very likely that the most fit individual will choose itself to reproduce with. When this occurs, what is in fact happening is the preservation of the best individual and, therefore, elitism. As shown by testing, this happens quite often, but not necessarily every time with the same individual, or with the best and most fitted one.

4.3.9 Mating Season

The Genetic Algorithm of the ChicuxBot uses the concept of the *mating season* (SIMÕES, 2000). This concept was introduced by the researcher Eduardo do Valle Simões in his PhD thesis, published in the year 2000. This mechanism is needed because the ChicuxBot runs on a real-time environment. The matting season is triggered by the user. In order for the reproduction process to start, the command *mateSeason* has to be issued by the *Chicuxbot* player. The command can be typed from the game’s command prompt. The *mateSeason* command is treated as regular text message by the Quake 2 server, but it is recognized and executed by the instances of the ChicuxBot connected to the game. Once the command is issued, the bots start executing the DNA Exchange Protocol (refer to the next subsection for details of the protocol).

The reproduction process of the Genetic Algorithm replaces the entire population. All the agents take active place in the process. Each *bot* has a three digit identification number that is incremented by one to identify its offspring. Since all the individuals are replaced by their offspring, the population size remains constant (as long as no new bots are added to the game, which may be done freely at any time).

4.3.10 DNA Exchange Protocol

In order to evolve and find the optimum configuration for the Behavior Net, the GA must exchange genetic material with other agents. This is done through the Quake2 server, that allows text messages to be sent from one player to all the others. The bots have unique identification numbers that are used to distinguish messages in the reproduction process. All messages are exchanged through the game server, thus making the program of each agent completely independent from the others.

The numbers of *frags* a bot has accumulated during a match serves as its fitness. Every time a matting season begins (defined by the user), each agent announces its fitness in the game. Once the agent has the fitness of all the others, it then chooses its mate by the roulette technique. Next, the bot sends its DNA to all the other agents in the game and, at the same time, searches incoming messages for the DNA of its selected partner. With the complete reception of the desired DNA, the crossover process executed.

4.3.11 Summary of the Genetic Algorithm

The table bellow presents a summary of the main characteristics of the Genetic Algorithm for the ChicuxBot.

Table 4.1: Characteristics Summary of the Genetic Algorithm

Configuration	Value
Population Size:	variable
Population Initialization:	random
Fitness Function:	number of *frags*
Selection Method:	roulette wheel
Crossover:	variable multi-point
Elitism:	yes
Mutation:	2%

4.4 Navigation Map Trails File

The navigation competence module from the Behavior Network of the ChicuxBot uses a file stored on the software's directory. The file contains a set of three-dimensional coordinates that is used by the ChicuxBot to navigate around the game map. This file is saved in text mode and can be edited by hand. There should be a corresponding navigation trails file for each different game map (or maze) that is to be used with the ChicuxBot.

The file contains navigation information in the form of trails. As the name suggests, a trail is a path for the *bot* to walkthrough the mazes. The navigation map file contains one main trail and any number of auxiliary trails. The main trail is a never ending loop around the maze. ChicuxBot uses the main trail to run through the map in search of enemies. When an opponent is found, the ChicuxBot will start chasing the enemy player. The chase will most likely lead the ChicuxBot out of the main trail. Every time the ChicuxBot wonders off from the main trail, it uses auxiliary trails to get back on track. All auxiliary trails must lead to the main trail.

The structure of the navigation map trails file is shown in table 4.2. The file starts with a header that contains two integer values: the number of nodes of the main trail (in other words, the size of the main trail) and the total number of nodes in the file. Note that the difference between these two values will result in the number of nodes of the auxiliary trails.

The main trail is the next item in the structure of the file. The main trail can be followed by zero or more auxiliary trails. Both main trail and auxiliary trails are composed of nodes. Each node contains three values that represent 3D coordinates in the game maze. The nodes in the file can be separated by blank lines for better visualization.

Table 4.2: Navigation Map Trails File

File Structure	**Description**	**Field Type**
Header	Size of Main Trail	Integer
	Total number of Nodes	Integer
Main Trail Nodes	Each node contains: • Coordinate *x* • Coordinate *y* • Coordinate *z*	3D Coordinates (one value per line)
Auxiliary Trail Nodes	Each node contains: • Coordinate *x* • Coordinate *y* • Coordinate *z*	3D Coordinates (one value per line)

Altering the navigation trails would directly change the behavior of the ChicuxBot. Therefore, during all the tests performed in this experiment, all bots used the same map trails. The default navigation map trails file provided with the ChicuxBot is for the Q2DM1 map of Quake2, which is the first *deathmatch* maze used by the game server. The ChicuxBot can work on other maps, so long as the appropriate navigation map trails file is provided. The system automatically retrieves the maze information from the game server and loads the corresponding file. The naming convention for the file ChicuxBot uses is "*map_name.txt*". For example, the name for the first level of deathmatch is Q2DM1.txt.

5 RUNNING THE CHICUXBOT

This guide shows how to start up the ChicuxBot multi-agent system and connect it to the Quake2 game server. This chapter is intended for those readers who are not familiar with Quake2. If you already know how to set up a multiplayer game in Quake2 and connect other players to it, you can safely skip to the next chapter.

5.1 Background

The ChicuxBot is an independent program that can execute by itself. However, to actually use the ChicuxBot, the Quake2 game must be installed in the system. The game is self-contained, in the sense that all the files it needs to run are contained inside its own directory. Quake2 does not install files in any other system directory. So, if the game isn't already installed, simply copying the game directory from a CD or another hard drive will work just fine.

The first step is to start a Quake2 server. The easiest way to do it, is to start the server through Quake's graphical user interface. It is easy because the user can start the server and play the game directly through a single interface, without having to type any text commands. In this mode, however, both server and client are running at the same time, on the same machine. This can lead to lag, or poor performance of the game server. Running the game server itself does not consume too much system resources. In fact, the Quake2 game server runs on text mode only. The client interface, on the other hand, renders graphics, sounds and has to read user input. These tasks demand large amount of system resources. Therefore, the best way to set up a Quake2 multiplayer game is to start a dedicated server on a separate machine, and have other clients connect from different machines. But since not everyone has a LAN of PCs at their disposal, this guide will also be showing how to set up the system on a single machine. Running the system on a single computer doesn't even require an active network connection.

5.2 Quake2 Game Server

To start the Quake2 server using the game's graphical user interface just select the menu *Multiplayer* then *Start Network Server*. The game prompts the user to input the server configuration. Table 5.1 shows the proper configuration that should be set up on the game server.

Table 5.1: Quake2 multiplayer server GUI configuration

Configuration	**Value**
Initial Map	The Edge Q2DM1
Rules	Deathmatch
Time Limit	0
Frag Limit	0
Max Players	8
Hostname	q2Game

The Initial Map setting specifies the maze that will be used in the game. ChicuxBot needs the map information for its navigation competence module, in order to be able to run around the maze during the game. The default map for Quake2 (and also for the ChicuxBot) is Q2DM1. The actual map information that the game server uses resides inside the Quake2 directory. The ChicuxBot uses a separate navigation file conveniently named "q2dm1.txt", that has trails and navigation information. This file is exclusive to the ChicuxBot and is not part of the Quake2 installation. A different map can be chosen through the Quake2 menu. ChicuxBot will automatically search for the appropriate navigation file in its own directory. If a maze other than the ones supplied with ChicuxBot is selected, a navigation file should be created and copied to the ChicuxBot directory. Table 4.2 specifies the structure of the navigation file used by ChicuxBot.

Rules specify the type of game that will be played. There are two options: *deathmatch* or *cooperative*. In deathmatch mode, all players of the game are enemies and the rule is every player for itself. ChicuxBot was designed to play death matches. So the Rules configuration should be set to *Deathmatch.*

The Time Limit setting is the amount of minutes a match should last. Likewise, the Frag Limit is the maximum number of kills a single player can score. Once the time limit or the *frag* limit is reached, the game ends the current match and automatically starts a new one with a different map.

The maximum number of players during a match have to be set up from start and cannot change during the course of the game. This happens because the Quake2 server must know beforehand how many ports it should keep open and listening to, waiting for other players to join. The Max Players configuration sets the limit of players of the match. Not all players have to join for the game to start. In fact, the match can start with just one player. The recommended number of maximum players is eight. This is a good number if you are not running the dedicated server. More players can be added, but the server could start to lag. To solve this problem, a dedicated server should be used. More players can smoothly run on a dedicated server.

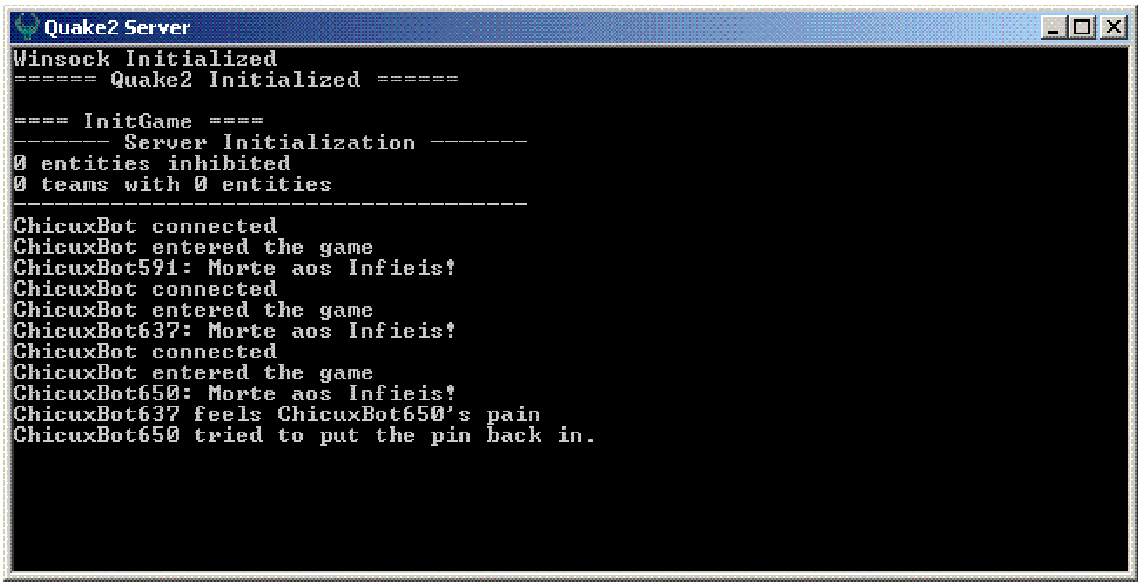
Finally, Hostname specifies the name of the match that will appear to other players. This value can be set to any name the user desires.

5.3 Dedicated Server

Setting up a dedicated server is more complex, but it results in much better performance. The dedicated server runs in text mode only. This uses much less system resources than the graphical mode. The freed up resources can then be used to actually serve the game. For the best performance, the Quake2 game server should be run on a separate and dedicated machine as well. To start a Quake2 dedicated server, the user has to type the following command:

```
quake2.exe +set dedicated 1 +set deathmatch 1 +set maxclients 32 +map q2dm1
```

This command calls the Quake2 executable and instructs it to run as a dedicated server. It also sets the rules of the game to deathmatch. The maximum number of players is configured to 32. And, finally, the initial map is configured as q2dm1. Figure 5.1 shows a screenshot of a Quake2 dedicated server.



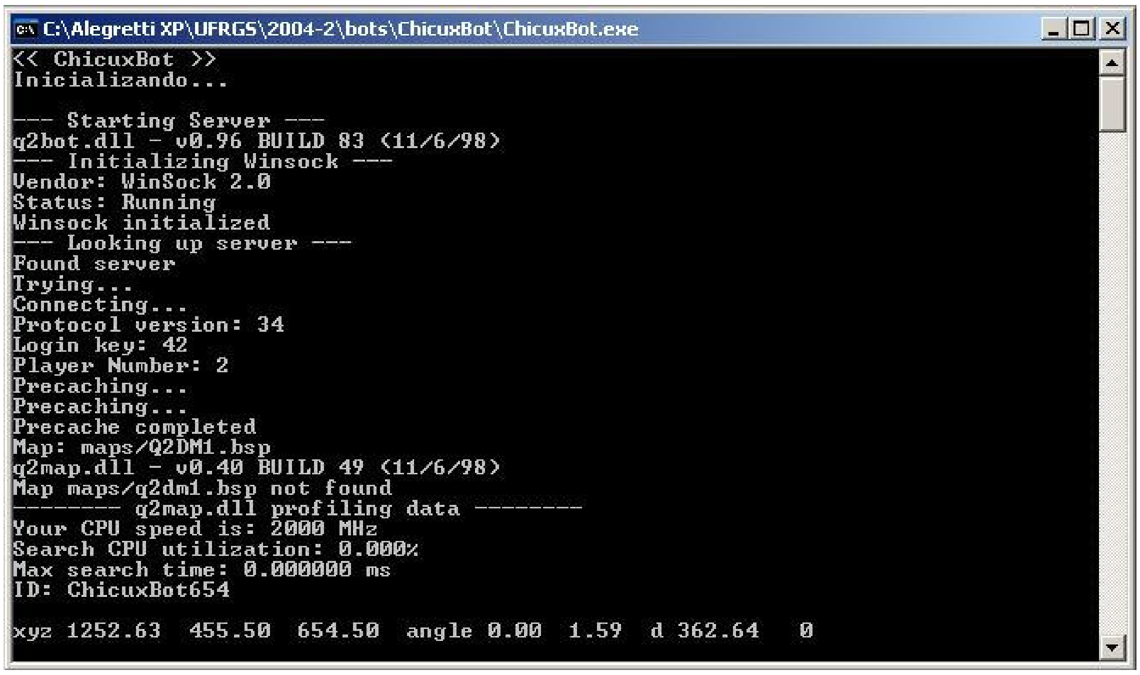
```
Winsock Initialized
====== Quake2 Initialized ======

==== InitGame ====
------- Server Initialization -------
0 entities inhibited
0 teams with 0 entities
-------------------------------------
ChicuxBot connected
ChicuxBot entered the game
ChicuxBot591: Morte aos Infieis!
ChicuxBot connected
ChicuxBot entered the game
ChicuxBot637: Morte aos Infieis!
ChicuxBot connected
ChicuxBot entered the game
ChicuxBot650: Morte aos Infieis!
ChicuxBot637 feels ChicuxBot650's pain
ChicuxBot650 tried to put the pin back in.
```

Figure 5.1: Quake2 Dedicated Server

5.4 Executing the ChicuxBot

Once the Quake2 server is running, instances of the ChicuxBot can start to enter the game. If the bots will be executing on the same machine as the game server and Quake2 is installed in the default directory (c:/games/quake2), then just double-clicking on the ChicuxBot icon will work. The program will be launched with its default configuration, which should work just fine in the described situation. Figure 5.2 shows a screenshot of the ChicuxBot software window.



```
C:\Alegretti XP\UFRGS\2004-2\bots\ChicuxBot\ChicuxBot.exe
<< ChicuxBot >>
Inicializando...

--- Starting Server ---
q2bot.dll - v0.96 BUILD 83 (11/6/98)
--- Initializing Winsock ---
Vendor: WinSock 2.0
Status: Running
Winsock initialized
--- Looking up server ---
Found server
Trying...
Connecting...
Protocol version: 34
Login key: 42
Player Number: 2
Precaching...
Precaching...
Precache completed
Map: maps/Q2DM1.bsp
q2map.dll - v0.40 BUILD 49 (11/6/98)
Map maps/q2dm1.bsp not found
-------- q2map.dll profiling data --------
Your CPU speed is: 2000 MHz
Search CPU utilization: 0.000%
Max search time: 0.000000 ms
ID: ChicuxBot654

xyz 1252.63  455.50  654.50  angle 0.00  1.59  d 362.64   0
```

Figure 5.2: ChicuxBot Program Screenshot

ChicuxBot provides all the mechanisms to work in a distributed environment, if a more elaborate and efficient configuration is desired. For instance, the Quake2 server can be running on a dedicated machine, and the various instances of the ChicuxBot can be distributed across multiple computers. The ChicuxBot software program can be configured in such cases through the use of command line parameters. The syntax is as follows.

Syntax: **ChicuxBot [-h hostname] [-dir quake2_directory]**

Exemple: **ChicuxBot -h 127.0.0.1 -dir c:/games/quake2**

The options that can be specified are the location of the Quake2 server and the game installation directory. The [-h] parameter sets the hostname or IP address of the Quake2 server. This option should be specified when the server is running on a different computer than the ChicuxBot. The [-dir] option configures the location of the game's installation directory. This information is need in order for the ChicuxBot to be able to find the Quake2 game. In the example provided, the ChicuxBot is being configure to look for the server on the *localhost* (an IP address is being used) and to look for the game installation in the c:/games/quake2 directory.

5.5 Using the Quake2 Viewer

A player can connect to a Quake2 game server that is being used by ChicuxBots in the same manner that he or she would connect to a normal multiplayer match. That is, just by double-clicking on the Quake2 icon and following the normal steps through the graphical menu the game provides. Using this path, however, the user will in fact become a player and interact with the game.

There is the possibility for a user to join the game as an observer. Observers do not interfere with the game. The other players cannot see the observers and are not affected by them. An observer can fly freely around the game map. The observer even pass through walls. This option can be very useful to provide an overview of the game match. To visualize the game as an observer, enter the following string at the command prompt:

```
quake2.exe +connect 127.0.0.1 +set spectator 1
```

This will launch the Quake2 client with full graphical user interface. Once the game interface is running, the arrow keys can be used to navigate through the map. The mouse can also be used to change the angle of view in the level.

6 TESTS

This chapter presents the tests conducted and the results they produced. Three distinct types of test were conducted. The ChicuxBot was tested against static bots, manually configured bots and human players. The text of this chapter is organized as follows. The first section starts by describing the environment in which the tests were conducted. Next, each different type of test is described. Then an example of a roulette configuration of the Genetic Algorithm is shown from an actual game match. The results obtained are presented. And finally, the possible applications for the work developed are discussed.

6.1 Test Environment

The ChicuxBot system was tested against three distinct types of opponents: randomly initialized bots, manually configured bots and, most importantly, human players. Both random and manually set bots used the same Behavior Network as the genetically evolved ChicuxBot. This means that, in essence, all bots used in the tests had the same basic abilities; what differed them was the manner in which their abilities were adjusted. It is highlighted that the objective of this experiment is to test if indeed a Genetic Algorithm can be used to configure a Behavior Network to play Quake 2; the efficiency of the specific modules of the Behavior Network is not being tested here. In fact, all modules of the Behavior Network of the ChicuxBot could be changed or replaced by more efficient ones that it would not affect the outcome of the tests. The experiment will show if the developed system can improve the configuration of a Behavior Network.

All tests conducted used Quake2's default multiplayer configuration on the game map q2dm1. During the tests, a new generation was spawned every five minutes. The human players that took part in this experiment were lab colleagues from the university. All of them were experienced Quake2 players.

6.2 ChicuxBot versus Static Bots

The first test placed five ChicuxBots to play against five static bots. Figure 6.1 presents graphic of the timeline (abscissas) by the number of *frags* of each bot (ordinates). This graphic illustrates the performance of the top three ChicuxBots and the three best static bots.

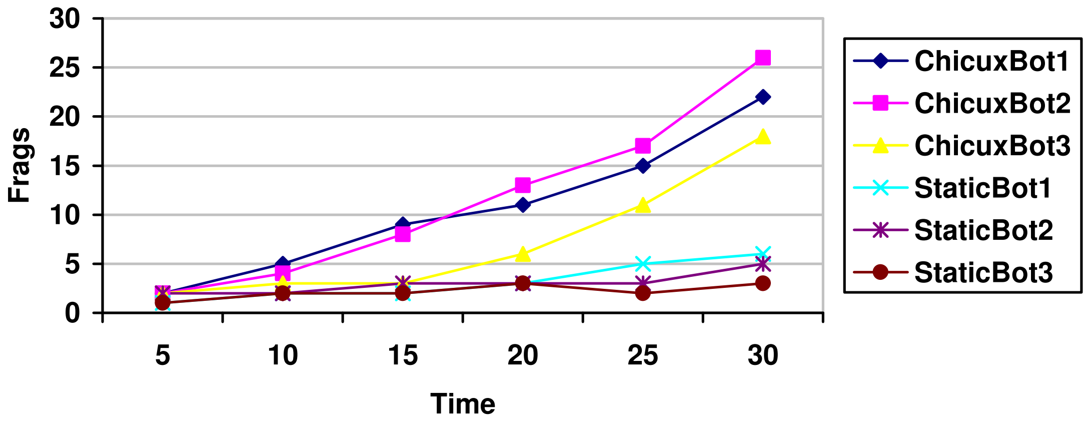


Figure 6.1: ChicuxBot versus Static Bots.

This test shows that when the dynamically configured GA bots play against static bots (with the same BN but no GA; the parameters of the BN are randomly set), after a certain amount of time, the first tend to have better performance. This happens because quite often randomly initialized static bots have poor parameter configuration on their Behavior Network. Although the genetically enhance bots are also initialized randomly, the Genetic Algorithm selects the best and most fit bots to exchange genetic material. This results in more adapted offspring with better performance.

Figure 6.1 clearly shows that the performance of the ChicuxBot improves over time. As new generations of the ChicuxBot were spawn, the Genetic Algorithm optimized the configuration of the Behavior Network. By the end of the game, the static bots had not improved their performance at all, while the ChicuxBots score much higher kills.

6.3 ChicuxBot versus Manual Bots

The next test placed ChicuxBot against manually configured bots. This scenario is shown in figure 6.2. When GA bots are put against bots hard coded by hand (again we have the same BN, but the parameters of the latter are set manually by the programmer) the result is always a tie. This happens because both types of agents are equally well configured and thus have practically the same performance. At first, the hand coded bots start better off; but as soon as the Genetic Algorithm tunes the Behavior Network's parameters, performance starts to rise.

We point out here that this kind of test had this particular result because the Behavior Network used is small and simple. In a larger, more complex BN, it would be very difficult to adjust the parameters by hand and, in that case, the Genetic Algorithm would probably produce a better result.

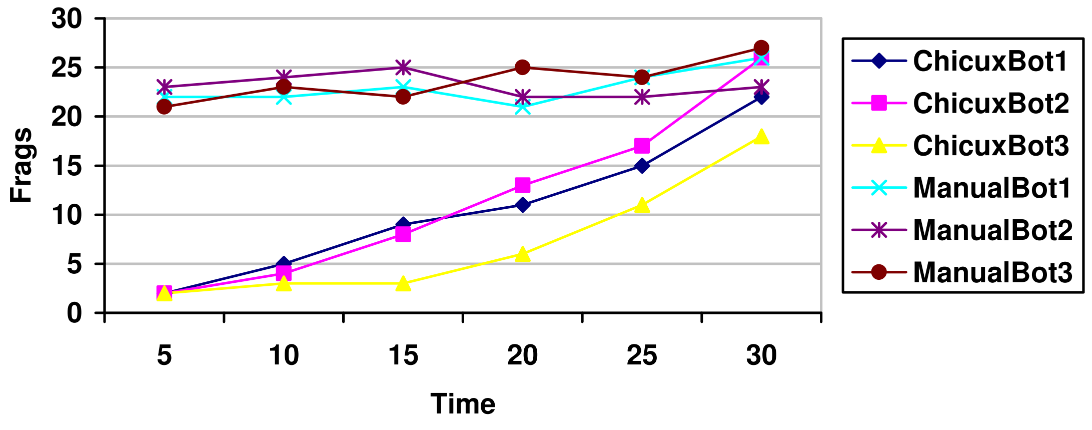


Figure 6.2: ChicuxBot versus Manually Configured Bots.

6.4 ChicuxBot versus Human Players

On our last series of tests, we put the ChicuxBot to play against human players. This match lasted 30 minutes and placed three human players up against seven ChicuxBots. Figure 6.3 shows the performance of the human players and the best three ChicuxBots. The ordinates indicate the number of *frags* each player scored and the abscissas show the time elapsed.

ChicuxBot is no match for a well-trained human being. However, this is due to the simplicity of the Behavior Network implemented (and to the low-end capabilities of the specific BN modules); the bad performance against good human players is not a fault of the ChicuxBot evolutionary configuration system. According to the human players subjected to the test, ChicuxBots did get better in the course of the game; as new generations evolved, the improvement in bot performance was clearly felt by the human players. The evolutionary configuration capability of the ChicuxBot is also confirmed by the test against randomly initialized bots shown earlier.

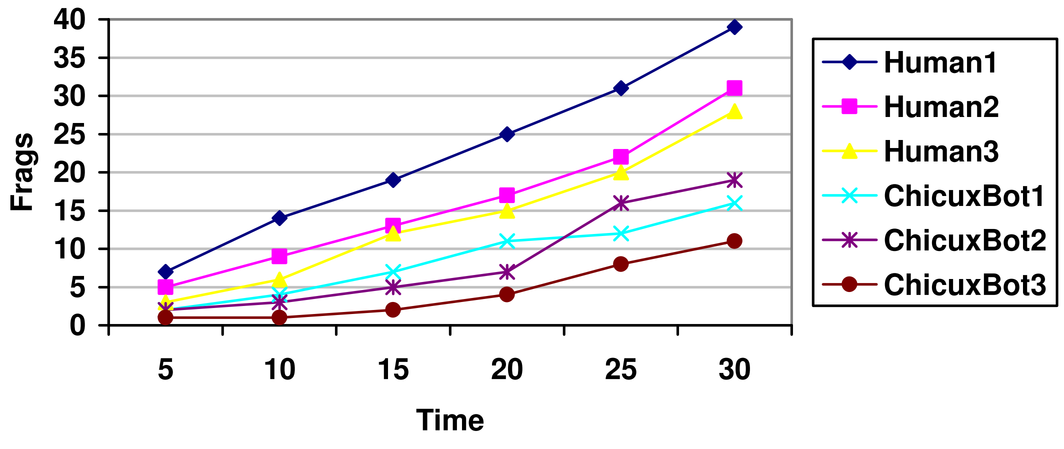


Figure 6.3: ChicuxBot versus Human Players.

6.5 Roulette Configuration

To illustrate how the selection mechanism of the Genetic Algorithm works, figure 6.4 shows a pie chart with the percentages of the roulette technique employed by the ChicuxBot to choose a mate. The figure shows the roulette used by the ChicuxBots during the fifth mating process (that is, 25 minutes of game time) of the ChicuxBot versus Humans match (shown on figure 6.3). The bigger the piece of each bot on the chart, the better the chances it will pass on its genes to the next generation.

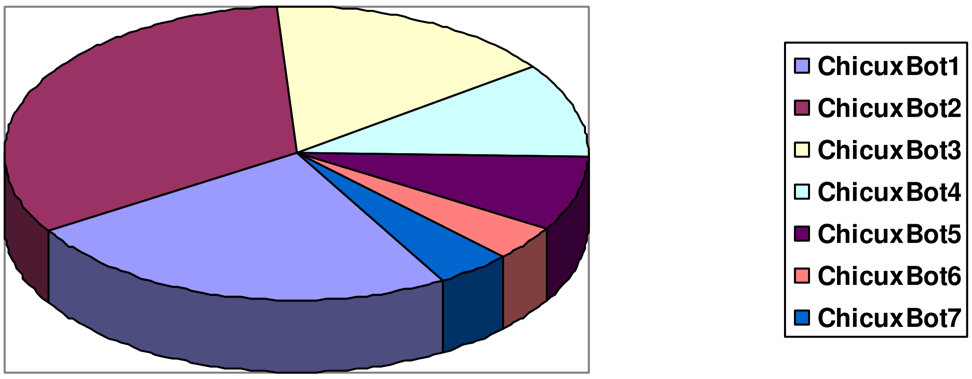


Figure 6.4: Example of Roulette Configuration.

6.6 Results

Tests show that according to its chromosomes, different agents will show distinct behavior. For example, an agent that has a lower threshold for the shooting module will fire more often. The same happens with an agent that has a stronger link for the *Enemy_near* world state. However, a bot will rather run than shoot if it has a low threshold for the navigation module. This demonstrates that the system has some flexibility to change its behavior and that these settings can indeed be done by the Genetic Algorithm.

Since the GA runs in real time during the course of the game, it can adapt according to changes in game play. This means that the ChicuxBot has the potential to adapt itself to different players and styles. The use of Genetic Algorithms to configure Behavior Networks adds adaptability to the solution.

Therefore, analyzing the results, what these tests show is that indeed the ChicuxBot can dynamically evolve from a random initial setting to an optimum configuration through the course of the game.

6.7 Applications

The ChicuxBot software itself can be directly be applied for entertainment, serving as what is popularly called the “Artificial Intelligence” of computer and video games.

For multiplayer games, the ChicuxBot naturally suits itself to be a deathmatch bot. But the ChicuxBot could also be applied to a single player game. This could be done in the following manner: several ChicuxBots could be used as the “soldiers” or normal enemies of the current game level. As the user explores the level’s map and interacts with the bots, the system is learning how the user plays. Then, at the end of the level, the “Boss” of the level would be the resulting optimized ChicuxBot, configured and fine-tuned as the player ran through the map during the game. In other words, the Genetic Algorithm configures the Behavior Network while the user plays through the level. When the player reaches the boss of the map, this final enemy would be controlled by the most evolved ChicuxBot of the population (that is, the fittest one).

But perhaps the most important application of this research will be the use of genetically-configured Behavior Networks in other problem domains. As the tests of this research indicate, the use of Genetic Algorithms to configure Behavior Networks not only provide an efficient solution to the global parameter configuration problem, but also add the functionality of adaptability in a dynamic environment. This could be used in other applications besides first-person-shooter games.

7 CONCLUSION

The use of Genetic Algorithms to configure a Behavior Network, as presented in this text, provides an efficient solution to the global parameter configuration problem. The tests conducted show that the implemented system outputs a configuration that is more efficient than the one it receives as input.

Using a game like Quake2 to implement Genetic Algorithms techniques is very interesting and especially fun because the agents of the population are literally fighting to survive. The use of Genetic Algorithms to configure Behavior Networks proves to be effective. Although the simple BN implemented in this experiment would not require such a sophisticated parameter setting mechanism (in this case, it could simply be hard-coded by hand), a larger, more complex BN would greatly benefit from the optimum solution search power of Genetic Algorithms.

The advantage of using Behavior Networks with Genetic Algorithms is that the resulting system can adapt itself to play with different types of players. Each person plays the game in their own particular way. A hard-coded solution that may work well against some people might not be as efficient for other kinds of players. With the Genetic Algorithm evolving and configuring a sufficiently complex Behavior Network in real time during the game, the *bot* would be capable of adapting itself to cope with different strategies from different players.

REFERENCES

ADOBATTI, R. et al. Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research. In: INTERNATIONAL WORKSHOP ON INFRASTRUCTURE FOR AGENTS, MAS, AND SCALABLE MAS, ACM AGENTS, 2., 2001. **Proceedings...** New York, NY: ACM Press, 2001.

AHN, L.; KEDIA, M.; BLUM, M. Verbosity: a game for collecting common-sense facts. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, CHI, 2006. **Proceedings...** [S.l.;s.n.], 2006. p. 75-78.

ANDREWS, G**. Foundations of Multithreaded, Parallel and Distributed Programming.** Reading: Addison-Wesley, 2000.

AXELROD, R. The Evolution of Strategies in the Iterated Prisioner's Dilemma. In: DAVIS, L. (Ed.) **Genetic Algorithms and Simulated Annealing.** London: Pitman, 1987.

BARONE, D. et al. **Sociedades Artificiais:** a Nova Fronteira da Inteligência nas Máquinas. Porto Alegre: Bookman, 2003.

BOULOS, P.; CAMARGO, I. **Geometria Analítica:** um Tratamento Vetorial. São Paulo: McGraw-Hill, 1986.

CHAKRABORTY, B.; CHAUDHURI, P. On The Use of Genetic Algorithm with Elitism in Robust and Nonparametric Multivariate Analysis. **Austrian Journal of Statistics**, [S.l.], v.32, n.1&2, p. 13–27, 2003.

CHAMPANDARD, A. **First Steps in Game AI:** Your Own Bot. Available at: <http://ai-depot.com/GameAI/Bot-Approaches.html>. Visited on: Apr. 2005.

COSTA, L.; OLIVEIRA, P. An elitist genetic algorithm for multi objective optimization. In: **Metaheuristics:** Computer Decision-making Archive. Norwell, MA: Kluwer Academic, 2004. p. 217-236.

CULBERSON, J. Mutation-Crossover Isomorphisms and the Construction of Discriminating Functions. **Evolutionary Computation,** [S.l.], v.2, n.3, p. 279-311, 1994.

DEJONG, K.; SPEARS, W. An Analysis of Multi-Point Crossover. In: **Foundations of Genetic Algorithms.** San Mateo, CA: Morgan Kaufmann, 1991. p. 301-315.

DEJONG, K.; SPEARS, W. A formal analysis of the role of multi-point crossover in genetic algorithms. **Annals of Mathematics and Artificial Intelligence**, [S.l.], v.5, n.1, 1992.

DORER, K. Behavior Networks for Continuous Domains Using Situation-Dependent Motivations. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI, 16., 1999. **Proceedings…** [S.l.]: Morgan Kaufmann, 1999.

GOLDBERG, D. **Genetic Algorithms in Search, Optimization, and Machine Learning.** New York, NY, USA: Addison-Wesley, 1989.

GOLDBERG, D. **A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-oriented Simulated Annealing.** 1990. TCGA Report 90003, Department of Engineering Mechanics, University of Alabama.

GRAHAM, R.; MCCABE, H.; SHERIDAN, S. Neural Pathways for Real-Time Dynamic Computer Games. In: IRISH WORKSHOP ON COMPUTER GRAPHICS, 6., 2005. **Proceedings…** [S.l.:s;n], 2005.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems.** Ann Arbor: University of Michigan Press, 1975.

ID SOFTWARE. "**Quake II".** Available at: <http://www.idsoftware.com/games/quake/quake2/>. Visited on: Aug. 2006.

JACOBS, S.; FERREIN A.; LAKEMEYER, G. Controlling Unreal Tournament 2004 Bots with the Logic-based Action Language GOLOG. In: ARTIFICIAL INTELLIGENCE AND INTERACTIVE DIGITAL ENTERTAINMENT CONFERENCE, 1., 2005. **Proceedings…** Menlo Park, CA: AAAI Press, 2005.

LAIRD, J. It Knows What You're Going To Do: Adding Anticipation to a Quakebot. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 5., 2000. **Proceedings…** Montreal, Canada: ACM Press, 2000. p. 385-392.

LAIRD, J. Using a computer game to develop advanced AI. **Computer**, New York, v.34, n.7, p. 70-75, July 2001.

LENT, M.; LAIRD, J. Developing an Artificial Intelligence Engine. In: GAME DEVELOPERS CONFERENCE, 1999. **Proceedings…** [S.l.:s.n], 1999.

MAES, P. The Dynamics of Action Selection. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI, 11., 1989. **Proceedings…** Detroit: Morgan Kaufmann, 1989a.

MAES, P. How To Do The Right Thing. **Connection Science Journal**, [S.l.], n.3, p. 291-323, 1989b.

MAES, P. A bottom-up Mechanism for Behavior Selection in an Artificial Creature. In: INTERNATIONAL CONFERENCE ON SIMULATION OF ADAPTIVE BEHAVIOR, 1., 1991. **Proceedings…** USA: MIT Press, 1991.

MARDEL, S.; PASCOE, S. An overview of genetic algorithms for the solution of optimization problems. **Computers in Higher Education Economics Review (CHEER)**, v.13, n.1, 1999.

MEYER, T. Building Cost-effective Research Platforms: Utilizing Free | Open-source Software in Research Projects. **Research Projects, Research Letters in the Information and Mathematical Sciences**, [S.l.], v.4, p. 91-99, 2003

MICHALEWICZ, Z.; ARABAS, J.; MULAWKA, J. GAVaPS - A Genetic Algorithm with Varying Population Size. In: IEEE CONFERENCE ON EVOLUTIONARY COMPUTATION, 1., 1994. **Proceedings…**Orlando, FL: IEEE Press, 1994. p.73 – 78

MICHALEWICZ, Z. **Genetic algorithms + data structures = evolution programs**. 3rd ed. Berlin: Springer-Verlag, 1999. 387 p.

MINSKY, M. **The Society of the Mind.** New York, NY, USA: Simon & Schuster, 1986.

MIZRAHI, V. V. **Treinamento em Linguagem C++ Módulo 1.** São Paulo: Makron Books, 1994.

MIZRAHI, V. V. **Treinamento em Linguagem C++ Módulo 2.** São Paulo: Makron Books, 1994.

NEBEL, B.; BABOVICH, Y. Goal-Converging Behavior Networks and Self-Solving Planning Domains, or: How to Become a Successful Soccer Player. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI, 18., 2003. **Proceedings…** San Francisco, CA: Morgan Kaufmann 2003.

NORLING, E. Learning to notice: Adaptive models of human operators. In: INTERNATIONAL WORKSHOP ON LEARNING AGENTS, 2., 2001. **Proceedings...** Montreal, Canada: [s.n], 2001.

NORTON, P. **A Bíblia do Programador.** Rio de Janeiro: Campus, 1994. 640 p.

PINTO, H. **Designing Autonomous Agents for Computer Games with Extended Behavior Networks:** An Investigation of Agent Performance, Character Modeling and Action Selection in Unreal Tournament. 2005. Dissertation (Master in Computer Science) - Instituto de Informática, UFRGS, Porto Alegre.

RHODES, B. **Pronomes in Behavior Nets.** Cambridge, MA: MIT Media Lab, Learning and Common Sense Section, 1995. (Technical Report # 95-01).

RICH, E. **Inteligência Artificial.** São Paulo: McGraw-Hill, 1988.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence:** A Modern Approach. New Jersey: Prentice Hall, 1995.

SIMMONS, G. F. **Cálculo com Geometria Analítica.** São Paulo: McGraw-Hill, 1985.

SIMÕES, E. V. **Development of an Embedded Evolutionary Controller to Enable Collision-Free Navigation of a Population of Autonomous Mobile Robots**. 2000.

PhD Thesis (Electronic Engineering for the degree of Doctor of Philosophy) - The University of Kent at Canterbury, England.

SINGH P. K.; JAIN S. C.; JAIN P. K. Comparative study of genetic algorithm and simulated annealing for optimal tolerance design formulated with discrete and continuous variables. **Journal of Engineering Manufacture,** [S.l.], v.219, n.10, p. 735-760, 2005.

SPEARS, W.; DEJONG, K. An Analysis of Multi-Point Crossover. In: **Foundations of Genetic Algorithms**. San Mateo, CA: Morgan-Kaufmann, 1991. p. 301-315.

STALLINGS, W**. Data and Computer Communications.** Upper Saddle River: Prentice Hall, 2000.

SWARTZLANDER, B. **The Quake 2 Bot Core Homepage.** Available at: <http://www.telefragged.com/Q2BotCore/ >. Visited on: Apr. 2005.

TANENBAUM, A. S**. Computer Networks.** Upper Saddle River, NJ, USA: Prentice Hall, 1996**.**

TATE, D.; SMITH, A. Expected Allele Coverage and the Role of Mutation in Genetic Algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 5., 1993. **Proceedings…** San Mateo, CA: Morgan Kaufmann, 1993. p. 31-37.

WIKIPEDIA. **Dynamical System**. Available at: <http://en.wikipedia.org/wiki/Dynamical_system>. Visited on: Apr. 2005.

WIKIPEDIA. **Genetic Algorithm**. Available at: <http://en.wikipedia.org/wiki/Genetic_algorithm>. Visited on: Apr. 2005.

WHITLEY, D. A Genetic Algorithm Tutorial. **Statistics and Computing Journal**, [S.l.], v.4, p. 65-85, 1994.

WOODS, O. **Autonomous Characters in Virtual Environments: The technologies involved in artificial life and their affects of perceived intelligence and playability of computer games.** 2004. 91 f. Dissertation (Master Degree in Computer Science) – Department of Computer Science, University of Durham.

WRIGHT, R.; SWEET, M. **OpenGL SuperBible.** 2nd ed. Indianopolis, USA: Waite Group, 2000.

XIAODONG, L. et al. A real-coded predator-prey genetic algorithm for multi-objective optimization. In: INTERNATIONAL CONFERENCE ON EVOLUTIONARY MULTI-CRITERION OPTIMIZATION, 2003, Faro, Portugal. **Proceedings…** [S.l.:s.n.], 2003. p. 207-221. (Lecture Notes in Computer Science, v.2632)

GLOSSARY

Frag – the score achieved by killing an enemy player.

Bot – a computer-controlled player in a network game.

Lag – the amount of time a multiplayer game is interrupted due to network stall or server overload.

Localhost – the computer that is currently running the application.

APPENDIX RESUMO EM PORTUGUÊS

Este capítulo contém um resumo, redigido em português, que apresenta os principais resultados da Dissertação. Salienta-se que o material deste capítulo é apenas um resumo e não substitui a leitura do texto completo da Dissertação. A primeira seção apresenta uma introdução sobre o trabalho desenvolvido. A seguir, descreve-se o sistema desenvolvido, detalhando a estrutura do software implementado. As tecnologias de Redes de Comportamento e o Algoritmos Genéticos utilizadas no desenvolvimento do presente trabalho também são descritas neste capítulo. Em seguida, são analisados os testes conduzidos e os resultados obtidos. Finalmente, é apresentada uma breve conclusão sobre o trabalho.

A.1 Introdução

Os jogos constituem um bom domínio para a exploração da Inteligência Artificial. Em geral, os jogos possuem regras bem definidas, onde é fácil medir o sucesso ou o fracasso (RICH, 1998). Um jogo pode ser utilizado como uma plataforma de pesquisa (LAIRD, 2001) para investigar, desenvolver e testar algoritmos de Inteligência Artificial (ADOBBATI, 2001). Talvez o melhor exemplo seja o jogo de xadrez, que catalisou a criação de algoritmos que foram utilizados em diversos outros problemas. Neste trabalho em particular foi utilizado o jogo Quake2, que possui um ambiente dinâmico e complexo que funciona em tempo real.

Quake2 é um jogo de primeira pessoa completamente tridimensional. Os jogadores (tanto humanos como os controlados por computador) não possuem visão global do mundo, mas somente a percepção do ambiente imediatamente à sua volta. As fases do jogo (ou mapas) apresentam-se tipicamente na forma de labirintos, tanto com espaços abertos, quanto com ambientes fechados. Os mapas possuem corredores, escadas, elevadores, pontes e até água. Uma vez que o jogador não possui visão global do mapa, ele deve aprender a navegar pelas fases e procurar por itens e oponentes. O jogador possui um medidor de vida que é decrementado cada vez que ele é ferido, seja por levar um tiro de outro jogados, cair de uma altura elevada ou até mesmo ficar embaixo d'água por muito tempo. O jogador também pode coletar armas, munição e kits médicos para curar os seus ferimentos, incrementando o seu medidor de vida. O jogo Quake2 está disponível há bastante tempo e, por isso, executa suavemente até mesmo em um computador bem modesto pelos padrões de hoje. O cliente do Quake2 possui uma interface gráfica 3D mas o servidor do jogo em si não possui interface gráfica e executa em modo texto. Adicionalmente, o jogo possui código fonte aberto e existem versões tanto para Windows quanto para Linux. Resumindo em poucas palavras, este jogo de computar é, talvez, uma das melhores plataformas de simulação disponíveis de imediato para desenvolver agentes de software.

Redes de Comportamento são um mecanismo de seleção de ação para agentes autônomos (MAES, 1989). Para funcionar corretamente, a Rede de Comportamento precisa que os seus diferentes parâmetros sejam configurados de forma adequada. No entanto, o algoritmo das Redes de Comportamento não define como esses parâmetros devem ser configurados. Cabe ao usuário (ou programador) ajustar esses parâmetros. Isso torna-se um problema em Redes de Comportamento grandes, pois a configuração de parâmetros é uma tarefa complexa, difícil e tediosa de ser feita manualmente. Por isso, torna-se interessante o uso de Algoritmos Genéticos para configurar automaticamente os parâmetros de uma Rede de Comportamento. O objetivo deste trabalho é verificar se é possível utilizar com sucesso Algoritmos Genéticos para configurar Redes de Comportamento, utilizando o jogo Quake2 como plataforma de testes.

A.2 Descrição do Sistema Multiagente

No jargão dos jogos de computadores em primeira pessoa, um "*bot*" é um jogador automático. *Bots* são utilizados em jogos *mutiplayer* para jogar contra jogadores humanos ou até mesmo contra outros *bots*. O tipo mais popular destes jogos é chamado de "*death match*" (ou "partida até a morte"), onde o objetivo é simplesmente matar todos os outros jogadores. O vencedor é o jogador que infligiu mais mortes (ou "*frags*"). A fim de alcançar este objetivo, o jogador enfrenta tarefas conflitantes, como decidir em atacar um inimigo ou fugir para conseguir mais munição. Tais problemas de tomada de decisões, em que estão envolvidos objetivos conflitantes, são ideais para serem tratados por Redes de Comportamento.

Esta seção detalha a implementação e os aspectos funcionais do software desenvolvido. Primeiro, descreve-se como o sistema está estruturado e como ele interage com o servidor de jogo do Quake2. A seguir, mostra-se a implementação da Rede de Comportamento utilizada pelos agentes, explicando-se a função de cada um dos seus componentes. Finalmente, descreve-se o funcionamento do Algoritmo Genético dentro do sistema.

A.2.1 Estrutura de Software

Existem diferentes maneiras para implementar um *bot* para Quake2. Pode-se modificar o próprio jogo através da inserção de código novo. Esta abordagem requer a re-compilação do jogo inteiro. Outra maneira é desenvolver um programa independente que se conecta ao servidor do jogo. Ambos os métodos, no entanto, exigem algum conhecimento do funcionamento interno do servidor do jogo. Uma vez que a maior motivação em utilizar o Quake2 era poupar tempo através do uso de um ambiente pronto e disponível de imediato, optou-se por utilizar uma interface conhecida como *Q2 Bot Core*. Esta interface realiza o trabalho de comunicação com o servidor do Quake2. Apesar de não ser tão rápida e poderosa quanto as outras duas abordagens possíveis, a interface é eficiente e fácil de usar.

Cada agente do sistema é um programa independente. Os agentes conectam-se ao servidor de jogo do Quake2 através de *sockets*. Assim, cada agente pode executar em uma máquina diferente e interagir através de uma rede de computadores. Mas o sistema todo também pode executar em uma única máquina, se assim desejado. Os agentes são controlados por uma Rede de Comportamentos que, por sua vez, é dinamicamente configurada por Algoritmos Genéticos em tempo real durante o decorrer do jogo. O software foi desenvolvido em C++ com a metodologia da Orientação de Objetos.

A.2.2 Rede de Comportamento

Um agente baseado em Redes de Comportamento é composto por um conjunto de *módulos de competência*. Cada módulo implementa uma tarefa específica para solucionar um determinado problema. A forma como esses módulos são programados não é especificada pela Rede de Comportamento e, inclusive, podem ser implementados por outra Rede de Comportamentos. Cada módulo representa um comportamento em particular. Os módulos de competência da rede são conectados via *links* que espalham a *energia de ativação*. Existem três tipos diferentes de *links* que podem ativar ou inibir o sistema. Os *links* podem conectar os módulos de competência a objetivos, estados do ambiente ou a outros módulos. Um módulo executa quando ele atinge um limiar de energia de ativação. Em sua definição formal original, um comportamento só pode ser executado uma vez que a sua lista de pré-condições torne-se completamente verdadeira. Um módulo também possui uma lista de predicados que tornam-se falsos após a execução de um determinado comportamento.

A Rede de Comportamento implementada pelo sistema é mostrada na figura abaixo. A mesma rede de comportamento é usada para todos os agentes do sistema. A Rede de Comportamento possui três módulos de competência que implementam todas as funções comportamentais do agente. Existe um módulo para *Ataque*, *Navegação* e *Desvio*. Basicamente, tudo o que um *bot* tem a fazer é atirar nos jogadores adversários, navegar ao redor do mapa e desviar do fogo inimigo. Adicionalmente, existem três estados de ambiente, representados na figura A.1 por círculos cinza. Cada estado pode ser visto como um sensor do ambiente, que continuamente sonda o mundo; quando o sensor detecta uma propriedade em particular, o estado torna-se verdadeiro. Finalmente, tem-se o objetivo geral que é de *Matar Todos* os jogadores inimigos. Este objetivo é ligado a todos os módulos de competência. Dessa forma, a Rede de Comportamentos resultante possui todas as funções necessárias para um *bot* de Quake2 jogar uma partida.

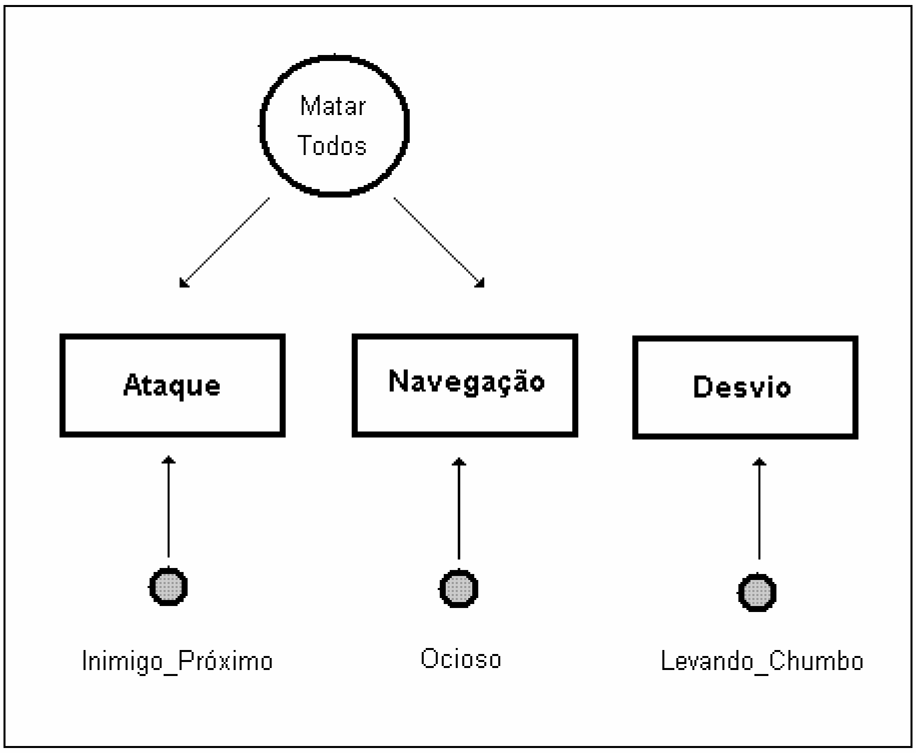


Figura A.1: Rede de Comportamento utilizada no ChicuxBot

Cada link da Rede de Comportamento é associado a um parâmetro que defino a quantidade de energia de ativação que ele transmite. Além de sua tarefa específica, cada módulo de competência possui uma variável que informa a quantidade de energia de ativação acumulada. Os módulos também possuem um limiar de energia. Quando a variável atinge esse limite, o módulo é executado. Conforme explicado anteriormente, na Rede de Comportamento original de Maes, cada módulo possui uma lista de pré-condições explícita que precisa tornar-se verdadeira para que o módulo possa executar. Na Rede de Comportamento implementada neste trabalho, a lista de pré-condições dos módulos está implícita nos estados de ambiente; isto é, se um estado tornou-se verdadeiro, é por que a sua lista de pré-condições também foi satisfeita. A Rede de Comportamento foi implementada nesta forma em particular porque foi projetada para funcionar em um ambiente de tempo real, onde uma lista de pré-condições explícita não é necessária. Os módulos de competência podem ser compostos por um número qualquer de *links* para outros módulos, objetivos ou estados de ambiente.

A fim de demonstrar o funcionamento exato de uma Rede de Comportamento, ilustra-se o seguinte exemplo: quando um jogador inimigo entra no campo de visão de um *bot*, o estado de ambiente *Inimigo_próximo* vai tornar-se verdadeiro; com isso, é transmitida a quantidade de energia de ativação especificada pelo link do estado *Inimigo_próximo* ao módulo de competência *Ataque*. Para cada ciclo (determinado pelo servidor do Quake2 e pela velocidade de conexão dos sockets), o estado vai enviar a mesma quantidade de energia de ativação através do link. O módulo vai acumular a energia de ativação até atingir o seu limiar, fazendo com que o comportamento seja executado. Neste exemplo, o *bot* vai atirar. Note-se que o agente vai atirar apenas uma única vez e, então, a variável da energia de ativação do módulo de *Ataque* vai ser zerada; todo o processo deve repetir-se novamente para que o *bot* atire outra vez.

Dependendo da quantidade de energia de ativação que um link transmite e no limiar de cada módulo de competência, o agente pode apresentar comportamentos distintos. Por exemplo, se um *bot* possui um limiar elevado para o módulo de *Navegação* e o estado *Ocioso* transmite uma quantidade baixa de energia de ativação, este agente em particular terá a tendência de ficar parando por um tempo antes de decidir começar a mover-se. Por outro lado, se um *bot* possui um limiar baixo e um link de ativação com bastante energia, o *bot* vai tender a correr o tempo todo pelo mapa.

Tanto o módulo de navegação quanto o módulo de ataque envolvem problemas difíceis e bastante complexos que beneficiar-se-iam muito de soluções sofisticadas. No entanto, uma vez que o objetivo principal desta pesquisa é desenvolver o sistema de Rede de Comportamento controlada por Algoritmo Genético, foram adotadas para estes módulos soluções *ad hoc* simples. Entretanto, é salientado que os módulos podem ser facilmente substituídos por algoritmos mais sofisticados posteriormente.

O *módulo de Ataque* escolhe o inimigo visível mais próximo para atirar. O algoritmo desenvolvido utiliza funções trigonométricas simples para computar os ângulos corretos que o *bot* deve virar para atingir precisamente o seu alvo. O módulo de ataque também faz com que o *bot* persiga o seu alvo, caso ele tente fugir.

O *módulo de Navegação* funciona com trilhas pré-definidas para cada mapa. As trilhas são armazenadas em um arquivo texto no disco, composto por um número determinado de coordenadas tridimensionais. Cada arquivo possui uma trilha principal e qualquer número adicional de trilhas auxiliares necessárias para um mapa em particular. O *bot* percorre a trilha principal enquanto o módulo de navegação estiver ativo. Se o *bot* sair da trilha principal (por exemplo, quando o módulo de ataque for ativado em função de um inimigo que se aproxima), o módulo de navegação encontrará a trilha auxiliar mais próxima para levar o *bot* de volta à trilha principal.

Finalmente, o *módulo de Desvio* faz com que o *bot* pule para esquivar-se do fogo inimigo.

A.2.3 Algoritmo Genético

Todos os agentes do sistema possuem a mesma Rede de Comportamento (ou seja, os mesmo módulos com as mesmas conexões), mas cada agente possui o seu própria DNA (isto é, o seu próprio conjunto de parâmetros que configura as ligações entre os módulos da Rede de Comportamentos). Os parâmetros configurados pelo Algoritmo Genético incluem o limiar de energia de ativação para um determinado módulo de competência, assim como a quantidade de energia que é transmitida por cada uma de suas ligações.

A estrutura dos cromossomos do agente foi implementada como um vetor de números inteiros. O software foi programado sob o paradigma da orientação a objetos, então qualquer mudança na Rede de Comportamento é automaticamente assimilada pelo sistema. Não é necessário re-programar manualmente as funções ou estruturas, nem qualquer outra parte do software; todas as modificações são automaticamente tratadas pelos objetos. Dessa forma, torna-se extremamente fácil expandir a Rede de Comportamento, incluindo novas ligações ou módulos de competência, ou modificando os módulos já existentes. Na sua configuração padrão, conforme mostrado na figura A.1, a Rede de Comportamento possui três módulos de competência e três ligações entre eles aos estados do mundo. Neste caso, o vetor do cromossomo possuirá 6 posições: três para o limiar de cada módulo de competência e mais três para a quantidade de energia de ativação transmitida por cada elo. A figura A.2 mostra a estrutura do cromossomo de um agente. Os valores do cromossomo de cada agente são inicializados randomicamente.

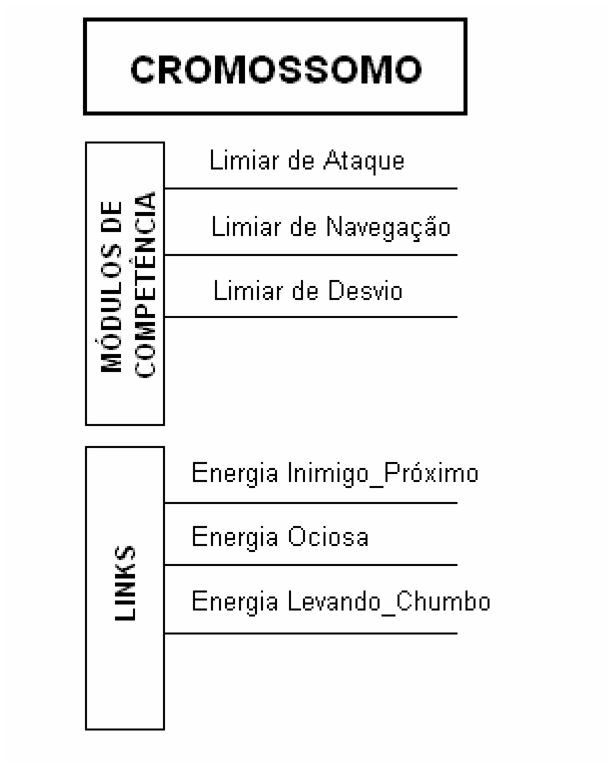


Figura A.2: Estrutura de um Cromossomo

O tamanho total da população pode variar, conforme novos *bots* entram no jogo e outros saem. O sistema funciona com qualquer número de *bots*, de 1 a n. No caso de apenas 1 bot estar presente no jogo, seu DNA vai tender a permanecer o mesmo, inalterado, pois o processo de reprodução vai sempre resultar em um clone do *bot*. As únicas mudanças ao cromossomo, neste caso, seriam advindas de mutação.

Para evoluir e encontrar a configuração ótima para a Rede de Comportamento, o Algoritmo Genético deve trocar o material genético com outros agentes. Este processo é feito através do servidor do Quake2, que permite a troca de mensagens de texto de um jogador para todos os demais. Os *bots* possuem um número de identificação único para a distinção das mensagens durante o processo de reprodução. Todas as mensagens são trocadas através do servidor do jogo, tornando o programa de cada agente independente de todos os demais.

O número de *frags* que um *bot* acumulou durante uma partida serve como a sua aptidão. Cada vez que o período de acasalamento é iniciado (definido pelo usuário), cada agente anuncia a sua aptidão no jogo. Quando o agente possui a aptidão de todos os demais, ele escolhe um parceiro através da técnica da roleta. A seguir, o *bot* envia o seu DNA para todos os demais agentes do jogo e, ao mesmo tempo, procura pelas mensagens com o DNA do parceiro escolhido. Uma vez que é recebido todo o DNA do parceiro desejado, o processo de cruzamento é realizado. A determinação de quais genes serão passados para a prole é feita através de sorteio, com a mesma probabilidade de transmissão dos genes de cada um dos progenitores. Dessa forma, o processo de reprodução pode produzir um indivíduo que é uma cópia exata de qualquer um dos deus progenitores, ou qualquer mistura dos dois.

Durante o processo de reprodução, um gene específico do cromossomo de um *bot* pode sofrer mutação. A mutação ocorre a uma taxa de 2% para cada gene. Um agente pode ter múltiplos genes alterados durante o processo de acasalamento. A mutação é útil pois introduz material genético novo na população, o que pode ajudar o algoritmo a sair de um máximo local.

O processo de reprodução do Algoritmo Genético substitui a população inteira. Todos os agentes têm participação ativa no processo. Cada *bot* possui um número de identificação de três dígitos que é incrementado por um para identificar a sua prole. Uma vez que todos os indivíduos são substituídos pela prole, o tamanho da população permanece constante (desde que nenhum *bot* novo seja adicionado ao jogo, o que pode ser feito livremente a qualquer momento). Durante o processo de reprodução, a seleção do parceiro é feita através da técnica da roleta. Nesta técnica, existe a probabilidade do indivíduo escolher a si mesmo para reproduzir-se. Quando isto ocorre, o que está de fato acontecendo é a preservação do melhor indivíduo da população e, portanto, elitismo. Testes demonstram que isso ocorre com certa freqüência, mas não necessariamente todas as vezes com o mesmo indivíduo, ou sequer com o melhor e mais apto.

A.3 Testes

O ChicuxBot foi testado com três tipos distintos de adversários: *bots* randomicamente inicializados, *bots* configurados manualmente e, principalmente, jogadores humanos. Tanto os *bots* configurados manualmente quanto os randômicos possuem a mesma Rede de Comportamentos utilizadas pelo ChicuxBot. Na sua essência, isto quer dizer que todos os *bots* utilizados nos testes possuem as mesmas habilidades básicas; o que os difere é a maneira com que as suas habilidades foram ajustadas. Salienta-se que o objetivos destes experimentos é verificar se, de fato, um Algoritmo Genético pode ser utilizado para configurar uma Rede de Comportamento para o jogo Quake2; a eficiência dos módulos específicos da Rede de Comportamento não está sendo coberta por estes testes. De fato, todos os módulos da Rede de Comportamento do ChicuxBot poderiam ser trocados ou substituídos por módulos mais eficientes que o resultado dos testes ainda seria o mesmo.

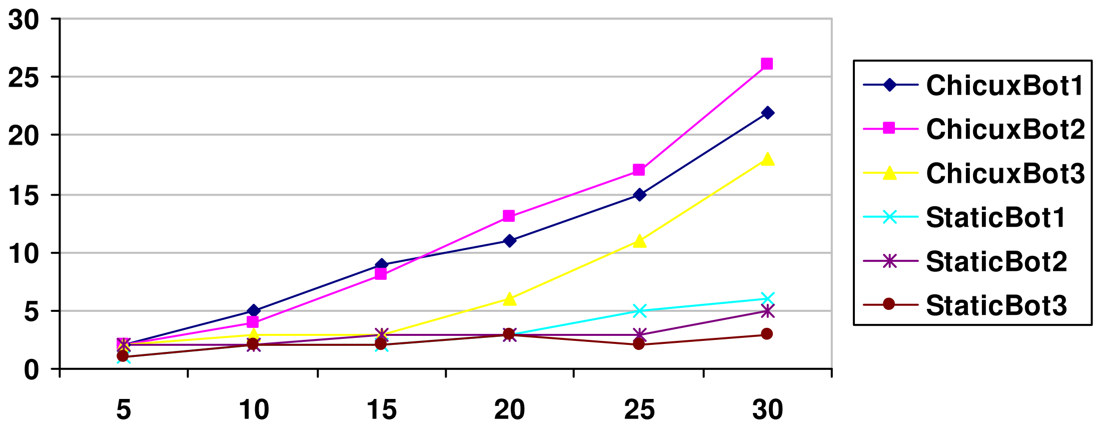


Figura A.3: ChicuxBot *versus* Bots Estáticos

O primeiro teste colocou cinco ChicuxBots para jogar contra cinco *bots* estáticos. A figura A.3 representa um gráfico com a linha do tempo (abscissas) pelo número de *frags* de cada *bot* (ordenadas). Este gráfico ilustra o desempenho dos três melhores colocados entre os ChicuxBots e os *bots* estáticos.

Este teste mostra que quando *bots* configurados dinamicamente por Algoritmo Genético são postos a jogar contra *bots* estáticos (com a mesma Rede de Comportamento, mas sem Algoritmo Genético; os parâmetros da Rede de Comportamento são ajustados randomicamente), após uma certa quantidade de tempo, os primeiros tendem a apresentar um desempenho melhor. Isto acontece porque freqüentemente *bots* estáticos, que foram inicializados randomicamente, apresentam uma configuração de parâmetros inadequada para a sua Rede de Comportamento. Apesar dos *bots* adaptados geneticamente também serem inicializados randomicamente, o Algoritmo Genético seleciona os *bots* mais aptos para trocar material genético, resultando em uma prole mais adaptada e com desempenho melhorado.

O teste seguinte colocou o ChicuxBot contra *bots* configurados manualmente. Este cenário é mostrado na figura A.4. Quando *bots* genéticos são colocados a jogar contra *bots* configurados manualmente (novamente os *bots* possuem a mesma Rede de Comportamento, mas os parâmetros desta são configurados manualmente pelo programador), o resultado é sempre um empate. Como ambos os tipos de agente são igualmente bem configurados, ambos apresentam praticamente o mesmo desempenho. No início, os *bots* configurados manualmente começam melhores; mas assim que o Algoritmo Genético consegue ajustar os parâmetros da Rede de Comportamento, o desempenho começa a melhorar. Salienta-se aqui que, este tipo de teste apresentou este resultado em particular porque foi utilizada uma Rede de Comportamento bastante simples; em uma rede maior e mais complexa, seria muito difícil ajustar os parâmetros manualmente e, neste caso, o Algoritmo Genético muito provavelmente produziria o melhor resultado.

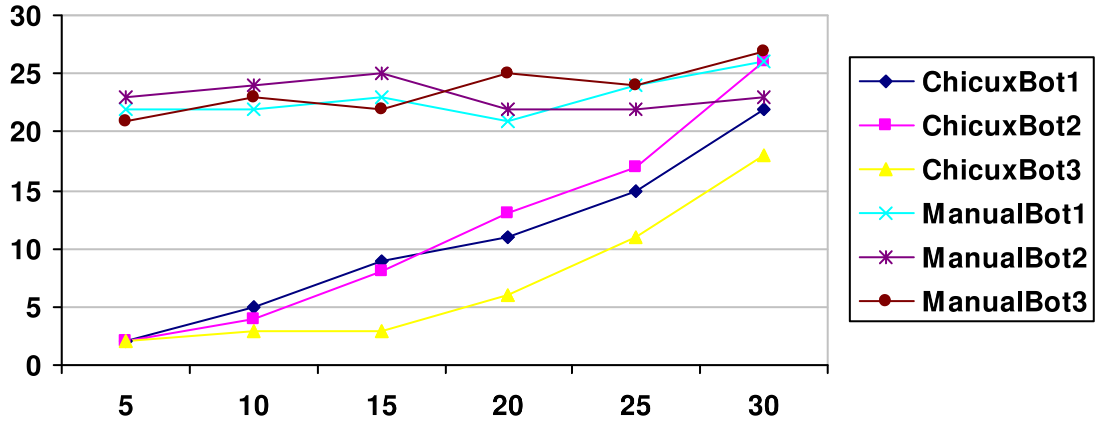


Figura A.4: ChicuxBot *versus* Bots Configurados Manualmente

Na última série de testes, o ChicuxBot foi colocado a jogar contra jogadores humanos. Esta partida durou trinta minutos e colocou três jogadores humanos contra sete ChicuxBots. A figura A.5 mostra o desempenho dos jogadores humanos contra os três melhores ChicuxBots. As ordenadas indicam o número de *frags* que cada jogador pontuou. As abscissas mostram o tempo decorrido.

O ChicuxBot não é páreo para um jogador humano bem treinado. No entanto, este resultado é devido à simplicidade dos módulos implementados para a Rede de Comportamento; o mau desempenho contra bons jogadores humanos não é falha do sistema de configuração evolucionária do ChicuxBot. De acordo com os jogadores humanos submetidos ao teste, o desempenho dos ChicuxBots de fato melhorou durante o decorrer do jogo; à medida que as novas gerações evoluíram, a melhora no desempenho do *bot* foi claramente sentido pelos jogadores humanos. A capacidade de configuração evolucionária do ChicuxBot também é confirmada pelo teste contra *bots* inicializados randomicamente demonstrado anteriormente.

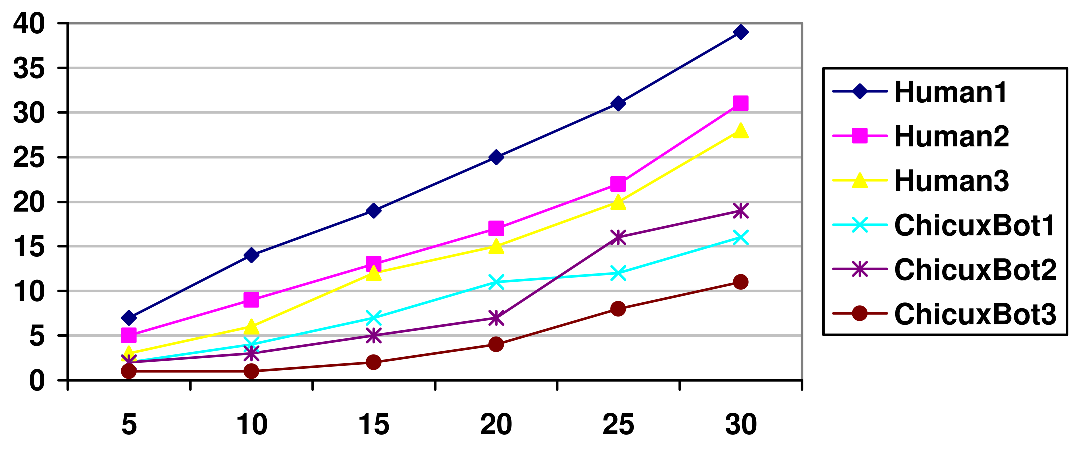


Figura A.5: ChicuxBot *versus* Jogadores Humanos

Os testes mostram que, conforme seus cromossomos, agentes diferentes apresentam comportamentos distintos. Por exemplo, um agente que tenha um limiar baixo no módulo de ataque vai atirar com mais freqüência. O mesmo acontece com um agente que possui um elo mais forte para o estado de ambiente *Inimigo_próximo*. No entanto, um *bot* vai preferir correr a atirar se tiver um limiar baixo para o módulo de navegação. Portanto, fica saliente que o sistema possui flexibilidade para mudar o seu comportamento e que esses ajustes podem, de fato, ser feitos pelo Algoritmo Genético. Uma vez que o Algoritmo Genético execute em tempo real durante o decorrer do jogo, ele pode adaptar-se a mudanças durante a partida.

Analisando os resultados, os testes mostram que, de fato, o ChicuxBot consegue evoluir dinamicamente de uma configuração inicial randômica para uma configuração melhor durante o decorrer do jogo.

A.4 Conclusão

O uso de Algoritmos Genéticos para configurar uma Rede de Comportamento prova-se efetivo. Apesar da Rede de Comportamento simples implementada nos experimentos não exigir um mecanismo de configuração tão avançado (neste caso, a configuração poderia ter sido configurada manualmente), uma Rede de Comportamento maior e mais complexa beneficiar-se-ia do poder de busca da solução ótima dos Algoritmos Genéticos.

A vantagem de utilizar Redes de Comportamento com Algoritmos Genéticos é que o sistema resultante pode adaptar-se para lutar contra diferente tipos de jogadores. Cada pessoa joga de seu próprio modo particular. Uma solução configurada rigidamente pode funcionar bem contra algumas pessoas, mas pode não ser tão eficiente contra outros jogadores. Com o Algoritmo Genético evoluindo e configurando em tempo real uma Rede de Comportamento suficientemente complexa, o *bot* seria capaz de adaptar-se a estratégias distintas de diversos jogadores.