Chapter 18

GAME IMPLEMENTATION: AN INTERESTING STRATEGY TO TEACH GENETIC ALGORITHMS

José M. Chaves-González, Noé Otero-Mateo, Miguel A. Vega-Rodríguez, Juan M. Sánchez-Pérez and Juan A. Gómez-Pulido
Dept. Informática, Univ. Extremadura, Escuela Politécnica, Campus Universitario s/n, 10071 Cáceres, Spain, {jm, noe, mavega, sanperez, jangomez}@unex.es, Fax: +34-927-257-202, http://arco.unex.es

Abstract: This chapter captures the experience acquired in the development of applications based on genetic algorithms. Specifically, we implemented two games that show an intelligent behaviour by executing genetic algorithms. They both show good results as well, because they are able to play successfully against human players. Moreover, the genetic algorithms parameters are user-configurable; so, the user can modify the number of individuals per generation, the number of generations, the mutation probability of each individual, the crossover function to generate new individuals, etc. This is very useful because the applications developed also generate statistical reports that show how individuals evolve in each generation. Therefore, the user can understand the evolution and analyze results easily. With this approach the user can test several combinations of parameters to study and compare them by analyzing their behaviour, speed, etc. In conclusion, as we are going to see in this chapter, the implementation of these two genetic games is an interesting strategy in order to teach and learn genetic algorithms.

Key words: Educational Strategy; Genetic Games; Genetic Algorithms; The Rabbit and the Dogs; Ten and a Half.

1. INTRODUCTION

Bio-inspired algorithms are a set of design methodologies based on how some natural or social systems work. They are vastly applied to several fields. Specifically, evolutionary algorithms are a sort of bio-inspired

B. Fernández-Manjón et al. (eds.), Computers and Education: E-learning, From Theory to Practice, 205–223.

algorithms based on the evolution mechanism. Species arise, then evolve and finally disappear. While they evolve, only some individuals survive. They are usually the individuals best adapted to the environment (Mitchell, 1998).

In evolutionary computing, genetic algorithms are one of the most complete and theoretically developed paradigms (Mitchell, 1998; Goldberg 1989), based on Neo-Darwinism and aimed to solve complex problems. Figure 18-1 shows the classical working schema for a genetic algorithm.

A genetic algorithm works over an initial population of individuals. Some of them, the best ranked by a fitness function, are selected to be parents for the next generation. This next generation is obtained by crossing and merging the parents' features that made them to be the best individuals. Some mutation probability is usually introduced to add new features in the new generation of individuals. These new features are not shown in parents, so the population evolves easily. The offspring is the new generation in the population, and the old individuals die (sometimes not all of them, but most). The fitness function will evaluate again each individual in the new generation, which is supposed to be better suited for the environment (new individuals are better problem solvers) than the old ones. Anyway, they are worse than individuals in the next generation, so the genetic algorithm can keep producing new generations until the stop criterion is reached.

If we select the individuals among the potential solutions for a problem (a move in a game, for example), only the best solutions of each generation are chosen to produce more individuals. For this reason, if there are generations enough, an optimal solution will be achieved at the end of the process.

On the other hand, it is known that the best way to understand, learn and study in depth a computational model is to build it in practice (Vega-Rodríguez et al, 2001; Granado-Criado et al, 2006). If students implement a model, they discover and learn fine details that the model has. Sometimes it is impossible to see these details if some complex concepts are only studied in a theoretical way.

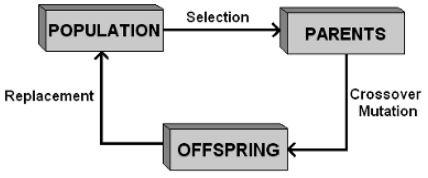

Figure 18-1. Genetic algorithm working schema.

Moreover, not only the practical application is relevant. It is also important that the topic which is going to be studied is attractive for the students (Gallego et al, 2006; Aznar et al, 2006; Overmars, 2004) because only if the students are motivated, their involvement will be increased and they will develop very good jobs (Garris et al, 2002; Martens et al, 2004). As a consequence, both students and the teacher will obtain a pleasant and nice sensation of a well-done job.

In this chapter, we present our experience in teaching and learning genetic algorithms by means of the development of two games:

- Sam, Coyote and Tazmanian Devil vs. the Road Runner - a funny variation of the game "the rabbit and the dogs" (Paredes-Juárez et al, 2006).
- The ten and a half game (a variation of the Spanish card game "Seven and a half" which is played with British cards).

In the following sections we explain in depth the implemented applications and their genetic components, as well as the results obtained with them and the educational conclusions that we have reached with the development and use of both games.

We want to emphasize that in both games the user can configure the different genetic parameters and then he/she can run the game with the chosen configuration. Both applications generate reports about the genetic behaviour of the games which the user can analyze and work with. For example, the reports are very useful if they are used to find an optimal genetic combination or to study the influence that the configuration of different parameters has on the artificial intelligence of the games: very important aspects from an educational point of view.

2. THE RABBIT AND THE DOGS GAME

The game "the rabbit and the dogs" (Paredes-Juárez et al, 2006) or the version developed for this work: "Sam, Coyote and Taz versus the Road Runner" is a board game that consist in 3 dogs (the characters of the Sheep Dog Sam, Wile E. Coyote and Tazmanian Devil in our case) that try to catch a rabbit (the Road Runner) in a board with a particular dimensions and characteristics. We can see in figure 18-2 the game board and the places which are taken by the different characters when the game starts.

2.1 Game description

The user can play both in the dogs' team and in the rabbit team. The application controls, using a genetic algorithm, the team which is not selected by the user. The goal of the dogs' team is to trap the Road Runner, or in other words, the dogs have to force the Road Runner to go to a square in the board where he can not do any movement. Only three cells in the board make this possible (see figure 18-3). On the other hand, the Road Runner goal is to get rid of the dogs. If the Road Runner gets to be on the left side of the three dogs, or the dogs move 17 times without catching the Road Runner, he wins the game.

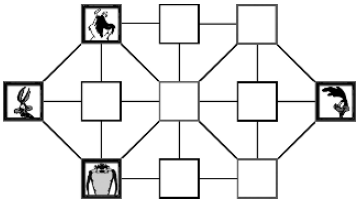

Figure 18-2. Game board for "Sam, Coyote and Taz vs. the Road Runner".

The game board has 11 positions and the connections which are drawn in figure 18-2. When the game starts, the dogs are on the left side of the board, one in each row, and the Road Runner is just on the opposite side, in the central row on the right-hand side. The game always starts with the persecution team (Taz, Coyote or Sam), and they swap the turn with the Road Runner alternatively until the match ends.

During the dogs turn, any of them can move, with neither restriction of repetition nor order among them. They only have some movement limitations: Taz, Coyote and Sam only can move towards empty adjacent cells which are connected with a line in the board and which are on the right,up or down to their position (dogs can not move back). On the other hand, the Road Runner can move to any empty adjacent cell (no matter if he goes up, goes down, goes forward or goes backward).

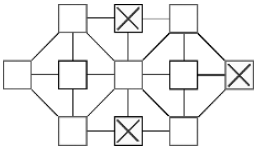

Figure 18-3. Places where the Road Runner can be trapped.

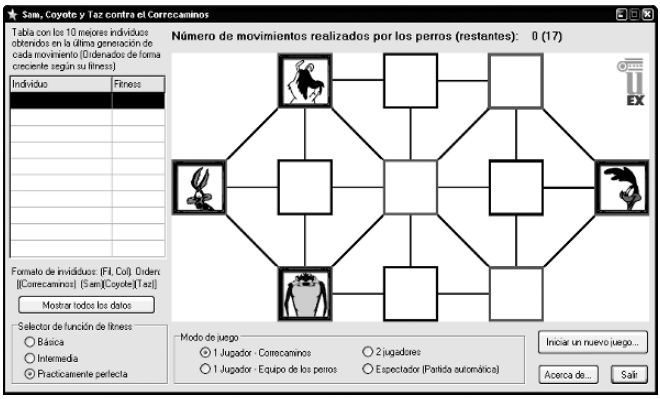

Figure 18-4. Main window of the game.

The main window of the game (see figure 18-4) has the following important parts:

- Just in the middle the board game is located. To move the characters in their appropriate turns, the user uses the mouse.
- In the top right corner there is a counter with the number of movements that the dogs have done until that moment to catch the Road Runner. If this counter reaches 17, the dogs lose the game.
- In the bottom right corner there are 3 buttons for the application management (Start a new game, quit the program and show the credits).
- In the bottom left corner, there is a control to select the fitness function which is going to be used during the game. The user can choose three different levels. Depending on the level of the fitness function, the individuals of each generation will have a better or worse behaviour.
- Down and in the centre of the window, a box to configure the game mode is placed. There are 4 different modes: the user can play with the dogs' team, or with the Road Runner team, but also it is possible that two users play a game between them, one controlling the dogs and another playing with the Road Runner (no genetic algorithm is working in this case). Finally, the program can be configured to play with no users. In this last case the application manages (by using genetic algorithms) the two teams and the user only sees how the game is performed and how it finishes.
- In the top left corner, there is a table with the 10 best individuals obtained in the last generation of the current movement, ordered from the lower to the upper value according to their calculated fitness. So, the individual placed in the first row of the table is the one which is selected to perform the actual movement. The individual is situated on the left column

(following a specific format that we will explain later) of the table and the value of the fitness function for that individual in that concrete movement is located on the right column.

- Just under the table described in the previous point, there is a button to open the file where all the genetic information about the game is saved. The format of the file that contains such information is explained in the *Results* subsection.

2.2 Genetic configuration of the game

The configuration of the game parameters is the first step to start a new game. These parameters are: *the fitness function* (there are three different functions, one for each level of difficulty in the game: easy, intermediate and advanced), *the game mode* (we have to decide which team is going to be controlled by the computer), and *the genetic parameters* of the game. These genetic parameters are fixed using the window shown in figure 18-5.

The configurable genetic parameters are the following:

- The number of individuals for each generation. This parameter can take values from 1 to 50. If the user does not change this parameter, it takes the value 16. We have checked the genetic algorithm and we know that 16 individuals in the population are enough to obtain good results. In case that the user selects only 1 individual per generation, no crossover is done (the individual will only change its genes from the mutation process). Therefore, it makes nonsense to select a population of only one individual, but the application allows it for didactic purposes (to compare different genetic configurations, even extreme values). The highest number of individuals is 50, because we have tested the game and we know that a higher number is useless for the genetic algorithm used.

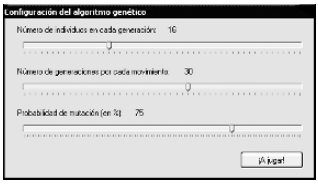

Figure 18-5. Window for the configuration of the genetic algorithm.

- The number of generations for each movement in the game. This parameter takes values from 1 to 50. As we explained with the previous parameter, it does not make a lot of sense to configure the number of generations with only one generation, but that can be useful to do some comparative studies. Besides, less than 30 generations are enough to produce the best movement in a game (but it depends on the number of individuals, which is configurable), so if the user does not touch this parameter, it takes the value 30.
- The mutation probability (in %). The mutation process consists in a random change in one of the genes of the genome of an individual. The default percentage is 75%, which means that the 75% of the new individuals in a generation suffer a mutation.

As we pointed out, both the genetic parameter ranges and the concrete default values have been obtained with the experience, making a lot of tests with lots of different values. We have specially taken care that the extreme values are wide enough to make comparative studies with even extreme configurations.

In the following sections we explain the representation of the genome, the correspondence between genotype and phenotype, how the initialization of the population is done, how we perform the individuals evaluation, which are the crossover and mutation operators, the selection and replacement strategies and the termination condition for the genetic algorithm.

2.3 Genome description

The design of the genome is very important in the development of a genetic algorithm. The genome contains the characteristic information which is inherited and improved from a generation to the next one. As we can see in figure 18-6, the genome for this game is made up with an array of 8 genes (1 byte per each cell), where the two first cells are the coordinates (row and column) of the Road Runner on the board. The following two cells are the row and column of Sam in the board, the two next ones are the coordinates (row and column) of Coyote and the two last are the row and column where Taz is situated in the board.

We have to point out that although the dimensions of the game board are 3x5, we do not consider the four corners (see figure 18-2). So, the rows of the board are 0, 1 and 2 (from the up to the down side of the board) and the columns are 0, 1, 2, 3 and 4 (from the left side to the right side). The positions (0,0), (0,4), (2,0) and (2,4) do not exist in the board.

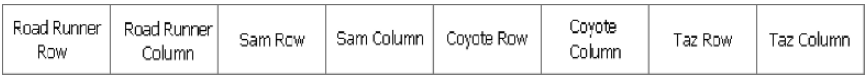

Figure 18-6. Genome format used in the game.

The relation between the phenotype and the genotype is quite easy. The genotype is the array which is described in figure 18-6, and the phenotype is the game board. The genotype which represents the initial state of the board (figure 18-2) is: [1,4,0,1,1,0,2,1], or if it is shown as a list of coordinates: [(1,4), (0,1), (1,0), (2,1)]. This means, according to figure 18-6, that the Road Runner is situated in the position (1,4) on the board, Sam in the coordinate (0,1), Coyote in the (1,0) position and Taz' devil in (2,1). When we started the work, we thought that a good genome could be the full game board, which means to consider an array with 11 genes (one byte per each cell in the board), but we have chosen a representation that allows to hold all the necessary information with less bytes, so we think it is a quite better structure for the final representation of the genotype.

2.4 Fitness function

This game has two different evaluation functions. One is for Sam, Coyote and Taz and another different one is for the Road Runner. It can not be the same, because the two teams have different objectives. An individual which is good for the dogs, because for instance, it almost traps the Road Runner, probably will be bad for the Road Runner; so the evaluation functions have to be different. We start studying the fitness function for the dogs.

One of the most difficult tasks in this work was to get that the dogs' evaluation function works perfectly. It is proved that if the dogs' team plays with no errors, they win the game, so the fitness function has to be perfectly fixed to obtain this result. However, the user probably is going to get boring if the dogs always win; so, we have developed three different versions for the evaluation function. But it is not only important the evaluation function, a right genetic configuration is also very important for the right behaviour of the characters. With too few generations per movement or too few individuals in the population, the artificial intelligence of a character will not be very high, although the evaluation function is very good. We can observe the pseudo-code for the full fitness function used for the dogs' team in figure 18-7.

```
int FitnessForDogs (TSIndividual *individual)
{
     fitness = 0;
     nMovRabbit = NumberMovementsRabbit (individual);

     //We give a lot of influence to the winner movement
     if (!nMovRabbit) fitness += 50000;

     fitness -= 1000 * RabbitOvertakesDogs (individual);
     fitness += DogGetsCentre (individual) ? 500 : 0;
     fitness -= RabbitCanGetCentre (individual) ? 3000 : 0;
     fitness -= 100 * RabbitNumMovements (individual);
     fitness -= RabbitRowEmpty (individual) ? 50 : 0;
     fitness += OneDogPerRow (individual) ? 20 : 0;
     fitness += RabbitAlmostTrapped (individual) ? 1000 : 0;
     fitness += Conditional3InARow (individual) ? 1000 : 0;
     fitness -= BadCase3InARow (individual) ? 2000 : 0;
     fitness += GoodStart (individual) ? 120 : 0;
     fitness -= EmptyColBetweenDogs&Rabbit (individual) ? 10000 : 0;
     fitness -= TwoDogsInARowInFinalCols (individual) ? 600 : 0;
     fitness -= LooserRhombus (individual) ? 10000 : 0;
     fitness -= StartALoop (individual) ? 10000 : 0;
     fitness += WinningCover (individual) ? 30000 : 0;
     return fitness;
}
```

Figure 18-7. Pseudo-code for the fitness function when the computer plays with dogs.

The value of the fitness is initialized to 0 at the beginning of the fitness function. The final value can be positive or negative, because, as we can see in figure 18-7, some functions add something to the final result and another subtract something from the final fitness. The number of movements that the Road Runner has is the first value which is calculated. If the result of this function is 0, the Road Runner is trapped, so, that individual makes the dogs win the game. For this reason, we give a lot of points (+50000) to this function (this individual has to be chosen if it appears). If this winning individual does not appear, the fitness function has 15 more little functions. The first of them is very valuable, because it is very important for the dogs that the Road Runner does not overtake them (because this would mean that he wins the game). So, for each dog overtaken by the Road Runner, 1000 points will be taken away from the fitness. Moreover, it is very important for the dogs to get the central square, because it is the cell which gives more freedom of movements. If the Road Runner gets this cell, he will very probably win the game. So, if an individual represents that one of the dogs is in that cell, that individual adds 500 points to its fitness. On the other hand, if an individual represents that the Road Runner is in the central cell, that individual subtracts 3000 points to its fitness value, because that individual makes the Road Runner wins the game (and for this reason is a very bad individual for the dog team).

The common sense and the experience acquired doing tests have been the principal rules that we have used to create all the expressions of the fitness function. It is true that some functions control little details of the game, but the fitness function shown in figure 18-7 supplies an almost perfect behaviour for the dogs team, so, it is important to take into account little details.

To sum up, the fitness for a dog individual is calculated with all the considerations that can be observed in figure 18-7. With the fitness obtained in the last generation of a movement, it is evaluated which move is performed when the dogs play in their turn. The higher value of the fitness, the better movement for the dogs, so, the individual with the highest value will be the selected one for the movement that the dogs' team will do in their turn.

On the other hand, the Road Runner fitness function is quite easier than the function for the dogs, because there are fewer parameters to consider in this case. It is easier to decide the movement for a single character than for three characters, and besides, in this game, the heuristic to escape is simpler than the heuristic to trap (because of the characteristics of the game). Figure 18-8 shows the evaluation function code for the Road Runner.

Similar to the dogs, when the Road Runner plays, its fitness function is calculated for each individual type "Road Runner". In this case, the criteria to calculate the fitness change, because the objective is that the Road Runner escapes from the dogs. As we can see in figure 18-8 the evaluation function

```
int FitnessForRR (TBoard *board, TSIndividual *individual)
{
     fitness = 0;
     nMovRabbit = NumberMovementsRabbit (individual);
     nDogsOverTaken = RabbitOvertakesDogs (individual);
     //We give a lot of influence to the winner movement
     if (nDogsOverTaken == 3) fitness += 6000;

     if (!nMovRabbit) fitness -= 1000;
     if (nMovRabbit == 1 && !RabbitProgressLevel (board, individual))
        fitness -= 2000;
     fitness += 100 * RabbitNumMovements (individual);
     fitness += EmptyColBetweenDogsRabbit (individual) ? 50 : 0;
     fitness += LooserRhombusForDogs (individual) ? 1000 : 0;
     fitness += RabbitGetsCentre (individual) ? 1000 : 0;
     fitness += RabbitProgressLevel (board, individual) ? 300 : 0;
     fitness += RabbitInLevellor2 (individual) ? 500 : 0;
     fitness += 500 * NumDogsSameLevelOrHigher (individual);
     return fitness;

}
```

Figure 18-8. Pseudo-code for the fitness function when the computer plays with the Road Runner.

gives a lot of points to the movement that makes the Road Runner win (+6000). It is penalized that the Road Runner does not have movements (-1000). It is good that he has a lot of possibilities of movement, or also that an empty column is between the dogs and the Road Runner, etc. To summarize, the individuals that represent good movements add more or less points depending on how good is the movement. A movement is good if the Road Runner has a possibility to escape from the dogs, although if they play with no mistakes, it is impossible for the Road Runner to win the game.

2.5 Description of the genetic operations

In the first generation, all bytes of all individuals in the population are randomly initialized (with correct values) and the fitness of each individual is calculated. New generations are created from the previous ones. From a generation to the next one, not all the individuals are replaced, because the elitism policy is applied between the individuals of a generation and the individuals of the following. It depends on the number of individuals configured, but approximately, the 25% of the individuals of a generation pass to the next one due to the elitism mechanism. The other 75% of the new population is created from the crossover of two random parents chosen from the old population. After the crossover, some mutation is done over the new offspring (depending on the configuration that the user made up for the mutation). The mutation is done so that not all descendants come from the cross between individuals of the previous population. The process consists in the random selection of a concrete gene inside the genome of an individual and the random change of this gene (taking the new value from a valid range). Once the processes of crossover and mutation are done, the fitness for the new generation of individuals is calculated and the cycle starts again until we arrive to the last generation. The termination condition is established by the user when he/she sets the number of generations for each movement (in figure 18-5 we can see the window to configure the genetic parameters). When the algorithm gets the last generation, the best individual, according to the fitness value, is selected to perform the most appropriate movement. This movement will be the best one that the team which is playing can do in that moment of the game.

2.6 Results

All the genetic information of the game is saved in a file of results. The structure of this file is described in figure 18-9.

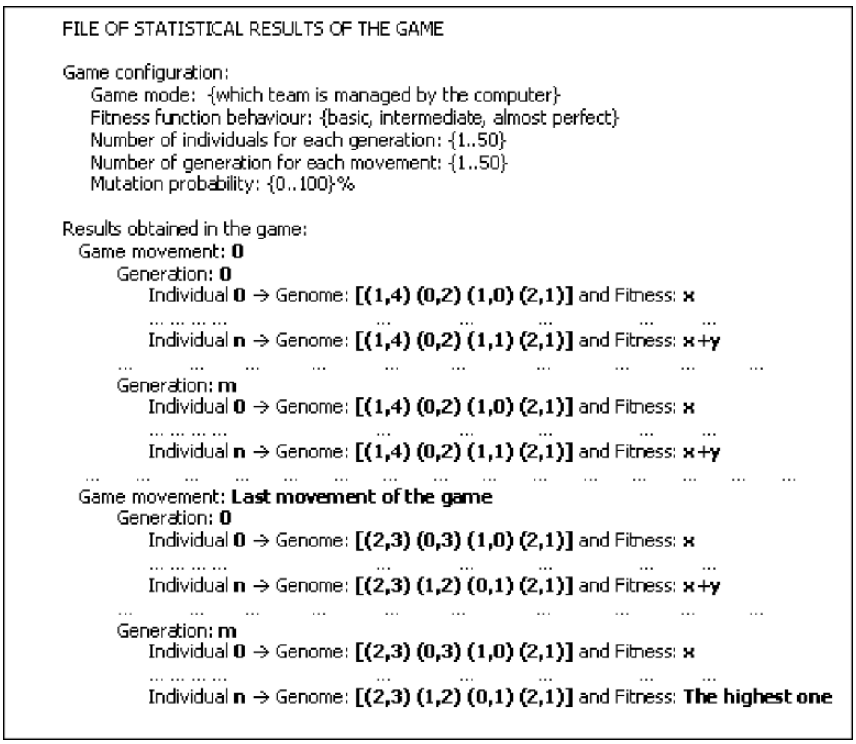

Figure 18-9. General structure of the results file.

The file of results, which structure is shown in figure 18-9, is very important, because it makes possible to study in depth the behaviour of the genetic algorithm used in the game (for each generation) and to compare different genetic configurations in a quantitative way. The file is divided in two parts. In the first part is detailed the configuration of the game (the configurable parameters are explained in the previous subsections and can be briefly explored at the beginning of the file -figure 18-9-). However, the most relevant information in the file appears in its second part (in the subsection "Results obtained in the game") because in this subsection, all the genetic data generated during the game are detailed. The file of results contains each single individual (with the specific genome) and the value of the fitness for that individual. Besides, they are ordered by the fitness value from the worse to the best individual. So, the last individual is the best in that generation. Moreover, the information is detailed for all generations of each movement in the game. So, the last individual of the last generation in a movement is the individual chosen to perform the movement, and the last individual of the

last generation in the last movement will be the individual with highest value in its fitness, because it is the individual that makes the final winning movement (if we consider that the team which is managed by the computer wins the game).

The information which is hold in the file of results is very useful, for example, to determinate the number of generations, or how many individuals are needed to find a valid solution for each level of the fitness function. It is also possible to study what happens with the individuals if too many generations are fixed (when the genetic algorithm finds the solution, all the individuals for all the generations from that moment, have the same fitness). On the other hand, if fewer generations than the necessaries are established, the characters will make bad movements, and the fitness will be low.

In conclusion, with the information contained in the file of results, we can study in depth how the behaviour of the individuals changes (and in what quantity) depending on the different configurations established (for the genetic algorithm), which is very important from an educational point of view.

3. THE TEN AND A HALF GAME

The ten and a half game is a card game based on a popular card game called "seven and a half" that is played with Spanish playing cards. Ten and a half is played with British playing cards. This game was selected in order to complete the genetic algorithms study that is initiated with the game explained in the previous section. In "the ten and a half" game we use different techniques to develop different genetic algorithms, so both games make this work more complete and richer in didactic experiences and contents.

3.1 Game description

The objective of the game for each player (computer or human) is to get 10.5 points or the nearest possible, without going further. At the end of each round, the player that reaches the nearest sum to 10.5, but does not overtake it, is the winner. Anyway, there are some special cases that may modify this rule and that we explain below.

The game is played with British cards, with the four card suits (hearts, clubs, diamonds and spades), 13 cards each. In each suit, there are cards numbered from 1 to 10, plus a Jack, a Queen and a King. Each numbered card scores the number it has. J, Q and K scores 0.5 points.

To get points, one of the players asks for one card after another to an automatic croupier pressing a button in the interface. In each turn the player can ask for as many cards as he/she wants (one is the minimum) or until he/she goes over 10.5 points. If this happens, he/she has to tell the other player that he/she went over 10.5. If he/she is below 10.5, the cards stay hidden on the table, waiting for the other player to play his/her turn. The turn to ask for cards is randomly assigned before each round. When a round is over, the following rules are evaluated to decide who the winner is. The order of the rules is relevant:

1. If only one player has a punctuation of 10.5, he/she is the winner.
2. If both players have a punctuation of 10.5, the winner is the player who took cards first.
3. If not, if both players have a punctuation below 10.5, the winner is the player with the greatest punctuation below 10.5. If they both have the same number of points, the winner is the player who took cards first.
4. If not (both players went over 10.5), the winner is the player who took cards first.

This game is a two player game. One of the players is human and the other is controlled by the computer by means of genetic algorithms. The genetic algorithm looks for a good playing strategy before the game starts. The game also has a didactic value because it includes the following features:

1. The ability to adjust the genetic algorithm to generate a good playing strategy for the ten and a half game.
2. The ability to log *and analyze* the genetic algorithm execution.
3. A counter of the rounds won by each player to evaluate the goodness of the strategy obtained with the genetic algorithm.

The main application window is shown in figure 18-10. The following elements are present:

- Information about the current game status, on the top of the window.
- At the bottom and in the middle of the window there is the rounds counter (which also counts how many rounds the human player has won and lost).
- The playing zone in the centre of the window. Cards are always in the playing zone. They appear while the players ask for them.
- User controls, below the playing zone. The application has buttons to: run the genetic algorithm, play (start a new round, ask for a card, and stop asking for cards), get some useful information (how to play, analyzed genetic algorithm execution results and the about dialog) and exit the application.

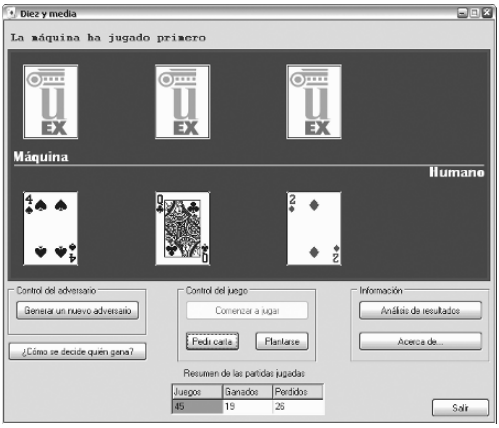

Figure 18-10. Application main window.

3.2 Genetic configuration of the game

As in "the rabbit and the dogs" game, before the game starts, it is compulsory to generate an opponent. The window shown in figure 18-11 has the necessary controls to do that.

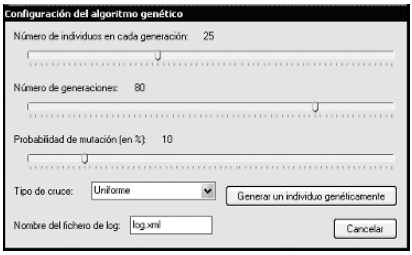

Figure 18-11. Genetic algorithm configuration window

The following parameters can be configured (fig. 18-11):

- The number of individuals per generation, between 2 and 100. Default value is 25.
- The number of generations, between 1 and 100. Default value is 80.
- The mutation probability (in %). Default value is 10%.

- The crossover. The user can choose uniform crossover, one point crossover or two point crossover.
- The name of the XML log file that stores the data generated from the genetic algorithm execution. This feature is an improvement in the log system of the previous game. Now the results are in a XML file, which is more standard than the text file used in "the rabbit and the dogs" game. However, both log files have the same purpose: analyze and help to understand and improve the genetic behaviour of the games.

3.3 Description of the individual

The genome of the individual has a length of 21 bits; each bit represents a boolean value. The individual can decide to ask for another card or to stop asking for cards during a game by querying its genome. Each bit in the genome relates to a possible accumulated score in that game. The first bit is required by the individual to decide if it wants another card when its score is 0, second when its score is 0.5, third when its score is 1 and so on. The 21st bit is queried when the score is 10 (since it makes nonsense to ask for another card when you have scored 10.5!). Figure 18-12 is a graphical representation of an individual's genome.

A good player has true boolean values in the first cells of the genome (when the score is low) and false values in the last cells (it refuses to take more cards when it has a considerable score, because if it goes over 10.5, he/she can lose the game easily).

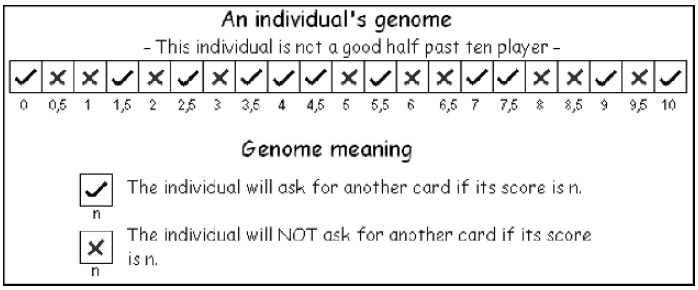

Figure 18-12. An individual's genome

3.4 Description of genetic operations

There is a simple statistical model to calculate how the perfect ten and a half player should behave. We could have calculated this individual and design a fitness function that somehow measures the "distance" between this perfect individual and a given one. But instead, we decided not to employ any external information to evaluate players. We use a tournament selection method. In each generation, each individual plays several games against every individual in that generation. Each individual has a counter of how many rounds it won and how many it lost (actually, a counter that counts victories minus defeats). This is a good measure of how good a player is in its own generation, so, we choose the parents for the next generation taking into account that counter. We also introduce elitism (parents survive from a generation to the next one) to ensure that if a good player arises in early generations it survives until it can not win its own offspring any more. The crossover can be chosen among random (bits are taken randomly from mother or father), in one, or two points crossover; so we can evaluate the results with each of them. Finally, there is also a chance for the mutation in each individual. A mutation consists in switching randomly the boolean value of a concrete gene (chosen randomly too).

3.5 Results

This game not only logs the results of a game, but also analyzes them. As we told in 3.2, when a new player is generated by the genetic algorithm, a XML log file is created. This file logs the algorithm configuration and tracks each generation by logging each individual, its genome, and its fitness function value. The log file can be analyzed with any XML parser, but the application has a simple built-in tool to do it. The dialog window which analyzes the results from the XML file generated in a game is shown in figure 18-13. Using this window, the user can load an XML file generated by the application to browse the data contained on it.

As it is shown in figure 18-13, on the left side there is a list with every generation which was generated during the execution. Clicking on a concrete generation, more detailed information is obtained. This information is shown on the right side of the window, and it consists in a list with every individual of the generation and the value of the fitness function for each. This list is also graphically displayed, with each individual represented by a bar. The more fitness the individual has, the higher its bar is. Clicking on an individual also it is shown its genome at the top of the window. Finally, at

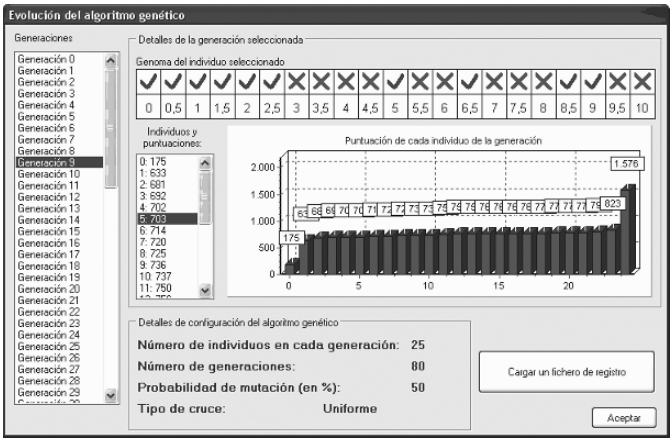

Figure 18-13. Results analysis window.

the bottom, the configuration parameters are situated. These parameters were the ones used in that concrete execution of the genetic algorithm (and they were selected using the window in figure 18-11).

4. CONCLUSIONS

The work expounded in this chapter describes both the main features of the games our students build and the educational experience we get with their development. It is possible that the inner characteristics of the chosen applications do not take all advantages that genetic algorithms offer, because genetic algorithms are used in problems where their solution is difficult to reach, and in our case, the developed games have quite easy solutions, and maybe other techniques of artificial intelligence would have been better to solve them. However, the development of this type of applications has, what we can consider, educational advantages. The implemented programs are quite simple, but students have to understand complex concepts to build them. So, if we propose an interesting, easy to understand and funny activity to a student, we will probably get better results than the obtained ones with a boring or hard activity. The reason is clear: motivated students (or motivated people in general) get involved in which they are studying (Cordova and Lepper, 1996; Lepper and Cordova, 1992), and they will probably obtain better results that the students that learn something because they have the obligation to do it. If a student, not only studies something, but also understands it, uses it in practice and besides, he/she does it in an entertaining way, he/she is motivated and learns much more.

So we can conclude that this kind of practices is a very interesting pedagogical experience that makes the study of genetic algorithms (in our case) going from theory assimilation to good understanding of how these algorithms work. Therefore, the implementation of games as a practical usage of artificial intelligence problems based on genetic algorithms motivates the students to go further when they build a first version of the program, because probably the behaviour of the automatic player will not be as good as the students want. Then, the students will analyze the results obtained with the program, they will try different configurations, they will change the algorithms used and, at the end, they will change the classical learning process into an amusing and entertaining sort of game that gives them much more satisfaction that any other theoretical lesson.

REFERENCES

Aznar, F.; Suau, P.; Compañ, P.; Rizo, R. Aprender Jugando: ¿Qué Opinan los Alumnos?. XII JENUI, Bilbao, Spain, pp. 199-206, July 2006. (in Spanish)

Cordova, D.I.; Lepper, M.R. Intrinsic Motivation and the Process of Learning: Beneficial Effects of Contextualization, Personalization, and Choice. Journal of Educational Psychology, vol. 88, no. 4, pp. 715-730, 1996.

Gallego, F.; Satorre, R.; Llorens, F. Computer Games Tell, Show, Involve,… and Teach. 8th Int. Symposium on Computers in Education, León, Spain, pp. 157-165, October 2006.

Garris, R.; Ahlers, R.; Driskell, J.E. Games, Motivation and Learning: A Research and Practice Model. Simulation & Gaming, vol. 33, no. 4, pp. 441-467, 2002.

Goldberg, D.E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, January 1989.

Granado-Criado, J.M.; Vega-Rodríguez, M.A.; Ballesteros-Rubio, J.; Sánchez-Pérez, J.M.; Gómez-Pulido, J.A. Teaching Reconfigurable Computing in the New EHEA by means of the Multicycle MIPS Machine Implementation. 6th International Workshop on Microelectronics Education, Stockholm, Sweden, pp. 104-107, June 2006.

Lepper, M.R.; Cordova, D.I. A Desire to be Taught: Instructional Consequences of Intrinsic Motivation. Motivation and Emotion, vol. 16, pp. 187-208, 1992.

Martens, R.L.; Gulikers, J.; Bastiaens, T. The Impact of Intrinsic Motivation on E-Learning in Authentic Computer Tasks. Journal of Computer Assisted Learning, vol. 20, no. 5, pp. 368-376, 2004.

Mitchell, M. An Introduction to Genetic Algorithms. The MIT Press, February 1998.

Overmars, M. Teaching Computer Science through Game Design. IEEE Computer, vol. 37, no. 4, pp. 81-83, 2004.

Paredes-Juárez, R.G.; Ramírez-Morales, A.A.; Saito-Vázquez N. A. http://www.cs.tcu.edu/people/professors/asanchez/cosc40503/RasGA/index.html, 2006.

Vega-Rodríguez, M.A.; Sánchez-Pérez, J.M.; Gómez-Pulido, J.A. An Educational Tool for Testing Caches on Symmetric Multiprocessors. Microprocessors and Microsystems, vol. 25, no. 4, pp. 187-194, June 2001.