Dealing with Noisy Fitness in the Design of a RTS Game Bot*

A.M. Mora, A. Fernández-Ares, J.J. Merelo, and P.García-Sánchez

Departamento de Arquitectura y Tecnología de Computadores.
Universidad de Granada (Spain)
{amorag,antares,jmerelo,pgarcia}@geneura.ugr.es

Abstract. This work describes an evolutionary algorithm (EA) for evolving the constants, weights and probabilities of a rule-based decision engine of a bot designed to play the Planet Wars game. The evaluation of the individuals is based on the result of some non-deterministic combats, whose outcome depends on random draws as well as the enemy action, and is thus noisy. This noisy fitness is addressed in the EA and then, its effects are deeply analysed in the experimental section. The conclusions shows that reducing randomness via repeated combats and re-evaluations reduces the effect of the noisy fitness, making then the EA an effective approach for solving the problem.

1 Introduction

Bots are autonomous agents that interact with a human user within a computer-based framework. In the games environment they run automated tasks for competing or cooperating with the human player in order to increase the challenge of the game, thus making their *intelligence* one of the fundamental parameters in the video game design. In this paper we will deal with real-time strategy (RTS) games, which are a sub-genre of strategy-based video games in which the contenders control a set of units and structures distributed in a playing area. A proper control of these units is essential for winning the game, after a *battle*. Command and Conquer™, Starcraft™, Warcraft™ and Age of Empires™ are some examples of these type of games.

RTS games often employ two levels of AI: the first one, makes decisions on the set of units (workers, soldiers, machines, vehicles or even buildings); the second level is devoted to every one of these small units. These two level of actions, which can be considered *strategical* and *tactical*, make them inherently difficult; but they are made even more so due to their real-time nature (usually addressed by constraining the time that can be used to make a decision) and also for the huge search space (plenty of possible behaviours) that is implicit in its action. Such difficulties are probably one of the reasons why Google chose this kind of games for their AI Challenge 2010. In this contest, real time is sliced in one

* This work has been supported in part by project P07-TIC-03044, awarded by the Andalusian Regional Government.

second *turns*, with players receiving the chance to play sequentially. However, *actions* happen at the *simulated* same time.

This paper describes an evolutionary approach for generating the decision engine of a bot that plays *Planet Wars*, the RTS game that was chosen for the commented competition. The decision engine was implemented in two steps: first, a set of parametrised rules that models the behaviour of the bot was defined by means of human players testing; the second step of the process applied a Genetic Algorithm (GA) for evolving these parameters offline (i.e., not during the match, but prior to the game battles).

The evaluation of the quality (fitness) of each set of rules in the population is made by playing the bot against predefined opponents, being a pseudo-stochastic or *noisy* function, since the results for the same individual evaluation may change from time to time, yielding good or bad values depending on the battle events and on the opponent's actions.

In the experiments, we will show that the set of rules evolve towards better bots, and finally an efficient player is returned by the GA. In addition, several experiments have been conducted to analyse the issue of the cited *noisy fitness* in this problem. The experiments show its presence, but also the good behaviour of the chosen fitness function to deal with it and yield good individuals even in these conditions.

2 State of the Art

Video games have become one of the biggest sectors in the entertainment industry; after the previous phase of searching for the graphical quality perfection, the players now request opponents exhibiting intelligent behaviour, or just human-like behaviours [1].

Most of the researches have been done on relatively simple games such as Super Mario [2], Pac-Man [3] or Car Racing Games [4], being many bots competitions involving them.

RTS games show an emergent component [5] as a consequence of the cited two level AI, since the units behave in many different (and sometimes unpredictable) ways. This feature can make a RTS game more entertaining for a player, and maybe more interesting for a researcher. There are many research problems with regard to the AI for RTSs, including planning in an uncertain world with incomplete information; learning; opponent modelling and spatial and temporal reasoning [6].

However, the reality in the industry is that in most of the RTS games, the bot is basically controlled by a fixed script that has been previously programmed (following a finite state machines or a decision tree, for instance). Once the user has learnt how such a game will react, the game becomes less interesting to play. In order to improve the users' gaming experience, some authors such as Falke et al. [7] proposed a learning classifier system that can be used to equip the computer with dynamically-changing strategies that respond to the user's strategies, thus greatly extending the games playability.

In addition, in many RTS games, traditional artificial intelligence techniques fail to play at a human level because of the vast search spaces that they entail [8]. In this sense, Ontano et at. [9] proposed to extract behavioural knowledge from expert demonstrations in form of individual cases. This knowledge could be reused via a case-based behaviour generator that proposed advanced behaviours to achieve specific goals.

Evolutionary algorithms have been widely used in this field, but they involve considerable computational cost and thus are not frequently used in on-line games. In fact, the most successful proposals for using EAs in games correspond to off-line applications [10], that is, the EA works (for instance, to improve the operational rules that guide the bot's actions) while the game is not being played, and the results or improvements can be used later during the game. Through offline evolutionary learning, the quality of bots' intelligence in commercial games can be improved, and this has been proven to be more effective than opponent-based scripts.

This way, in the present work, an offline GA is applied to a parametrised tactic (set of behaviour model rules) inside the Planet Wars game (a basic RTS), in order to build the decision engine of a bot for that game, which will be considered later in the online matches.

3 The Planet Wars Game

It is a simplified version of the game Galcon, aimed at performing bot's fights which was used as base for the Google AI Challenge 2010 (GAIC)[1].

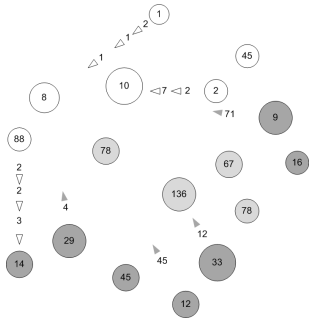

Fig. 1. Simulated screen shot of an early stage of a run in Planet Wars. White planets belong to the player (blue colour in the game), dark grey belong to the opponent (red in the game), and light grey planets belong to no player. The triangles are fleets, and the numbers (in planets and triangles) represent the starships. The planet size means growth rate of the amount of starships in it (the bigger, the higher).

A Planet Wars match takes place on a map (see Figure 1) that contains several planets (neutral or owned), each one of them with a number assigned to

[1] http://ai-contest.com

it that represents the quantity of starships that the planet is currently hosting. The objective of the game is to defeat all the starships in the opponent's planets. Although Planet Wars is a RTS game, this implementation has transformed it into a turn-based game, in which each player has a maximum number of turns to accomplish the objective. At the end of the match (after 200 actions, in Google's Challenge), the winner is the player owning more starships.

There are two strong constraints (set by the competition rules) which determine the possible methods to apply to design a bot: a simulated turn takes *just one second*, and the bot is *not allowed to store any kind of information* about its former actions, about the opponent's actions or about the state of the game (i.e., the game's map). Therefore, the goal in this paper is to design a function that, according to the state of the map in each simulated turn (input) returns a set of actions to perform in order to fight the enemy, conquer its resources, and, ultimately, win the game.

For more details, the reader is invited to revise the cited webs and our previous work [11].

4 Genetic Approach for the Planet Wars Game

The competition restrictions strongly limit the design and implementation possibilities for a bot, since many algorithms are based on a memory of solutions or on the assignment of payoffs to previous actions in order to improve future behaviour. Moreover most of them are quite expensive in running time. Due to these reasons, there was defined a set of rules which models the on-line (during the game) bot's AI. The rules have been formulated through exhaustive experimentation, and are strongly dependent on some parameters, which ultimately determine the behaviour of the bot.

Anyway, there is only one type of action: move starships from one planet to another; but the nature of this movement will be different depending on whether the target planet belongs to oneself or the enemy. As the action itself is very simple, the difficulty lies in choosing which planet creates a fleet to send forth, how many starships will be included in it and what will the target be.

Three type of bots are going to be tested in this paper, all of them previously introduced in our previous work [11][2].

The first one is ***GoogleBot***, the basic bot provided by Google for testing our own. It is quite simple, since it has been designed for working well independently of the map configuration, so it may be able to defeat bots that are optimised for a particular kind of map. It just choose a planet as a base (the one with most of its starships) and a target chosen by calculating the ratio between the growth rate and the number of ships for all enemy and neutral planets. It wastes the rest of time until the attack has finished.

The second bot is known as ***AresBot***, and it was defined as the first approach for solving the problem. It models a new hand-coded strategy better than the one

[2] The source code of all these bots can be found at: forja.rediris.es/svn/geneura/Google-Ai2010

scripted in the GoogleBot. So, several rules were created based on the experience of a human player. As a summary, this bot tries to firstly find a *base planet* depending on a score function, the rest of its planets are considered as *colonies*. Then, it determines which *target planet* to attack or to reinforce, if it already belongs to it in the next turns (since it can take some turns to get to that planet). The base planet is also reinforced with starships coming from colonies; this action is called *tithe*. Furthermore, colonies that are closer to the target than to the base also send fleets to attack the target instead of reinforcing the base. The internal flow of AresBot's behaviour with these states is shown in Figure 2. It can be seen in that figure a set of seven parameters (weights, probabilities

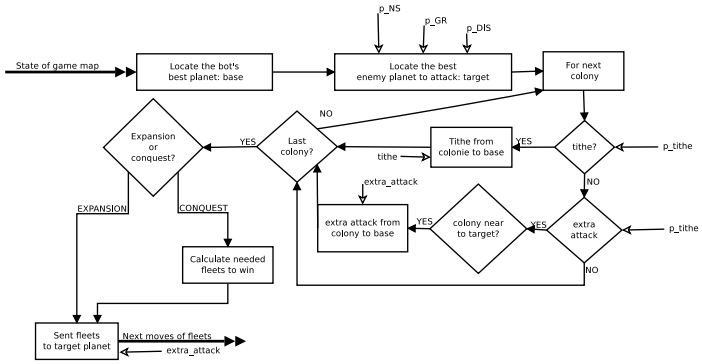

Fig. 2. Diagram of states governing the behaviour of AresBot and GeneBot. The parameters that will be evolved are highlighted.

and amounts to add or subtract) which has been included in the rules that model the bot's behaviour. These parameters have been adjusted by hand, and they totally determine the behaviour of the bot. Their values and meaning can be consulted in the previous work [11]. As stated in that paper, their values are applied in expressions used by the bot to take decisions. For instance, the function considered to select the *target planet*.

This bot already had a behaviour more complex than GoogleBot, and was able to beat it in 99 out of 100 maps; however, it needed lots of turns to do that; this means that faster bots or those that developed a strategy quite fast would be able to beat it quite easily. That is why we decided to perform a systematic exploration of the values for the parameters shown above, in order to find a bot that is able to compete successfully (to a certain point) in the google AI challenge.

The third bot is an evolutionary approach, called ***GeneBot***, and it performs an offline AresBot's parameter set optimisation (by means of a GA). The objective is to find the parameter values that maximise the efficiency of the bot's behaviour. The proposed GA uses a floating point array to codify the parameters, and follows a *generational* scheme with *elitism* (the best solution always survives). The genetic operators include a *BLX-*α crossover [12], very common in this kind of chromosome codification to maintain the diversity, and a *gene*

mutator which mutates the value of a random gene by adding or subtracting a random quantity in the $[0, 1]$ interval. The *selection mechanism* implements a *2-tournament*. Some other mechanisms were considered (such as roulette wheel), but eventually the best results were obtained for this one, which represents the lowest selective pressure. The elitism has been implemented by replacing a random individual in the next population with the global best at the moment. The worst is not replaced in order to preserve diversity in the population.

The evaluation of one individual is performed by setting the correspondent values in the chromosome as the parameters for GeneBot's behaviour, and placing the bot inside *five different maps* to fight against a GoogleBot. These maps were chosen for its significance and represent a wide range of situations: bases in the middle and planets close to them, few and spread planets, planets in the corners, bases in the corners, both planets and bases in the corners. The aim is to explore several possibilities in the optimisation process so, if the bot is able to beat GoogleBot in all of them, it would have a high probability of succeeding in the majority of 'real' battles. The bots then fight five matches (one in each map). The result of every match is non-deterministic, since it depends on the opponent's actions and the map configuration, conforming a *noisy fitness* function, so the main objective of using these different maps is dealing with it, i.e. we try to test the bot in several situations, searching for a good behaviour in all of them, but including the possibility of yielding bad results in any map (by chance). In addition, there is a reevaluation of all the individuals every generation, including those who remain from the previous one, i.e. the elite. These are mechanisms implemented in order to avoid in part the noisy nature of the fitness function, trying to obtain a real (or reliable) evaluation of every individual.

The performance of the bot is reflected in two values: *the number of turns* that the bot has needed to win in each arena, and the second is *the number of games that the bot has lost*. In every generation the bots are ranked considering this last value; in case of coincidence, then the number of turns value is also considered, so the best bot is the one that has won every single game or the one that needs less turns to win. Thus the *fitness* associated to an individual (or bot in this case) could be considered as the minimum aggregated number of turns needed for winning the five battles.

5 Experiments and Results

In order to test the GA proposed in previous section, several experiments and studies have been conducted. These are different from those performed in the previous work [11], being more complete in the first step (parameter optimisation), and analysing the pseudo-stochastic fitness function, and the value of the dealing mechanisms for avoiding it.

First of all, the (heuristically found) parameter values used in the algorithm can be seen in Table 1. 15 runs have been performed in the optimisation of the AresBot's behaviour parameter, in order to calculate average results with a certain statistical confidence. Due to the high computational cost of the evaluation

Num. Generations	*Num. Individuals*	*Crossover prob.*	α	*Mutation prob.*	*Replacement policy*
100	200	0.6	0.5	0.02	2-elitism

Table 1. Parameter setting considered in the Genetic Algorithm.

of one individual (around 40 seconds each battle), a single run of the GA takes around two days with this configuration. The previously commented evaluation is performed by playing in 5 representative maps, but besides, Google provides 100 example/test maps to check the bots, so they will be used to evaluate the value of the bots once they (their parameters) have been evolved. The following sections describe each one of the studies developed for demonstrating the value of the presented method and also the correct performance of the noisy fitness function.

5.1 Parameter optimisation

In the first experiment, the parameters which determine the bot's behaviour have been evolved (or improved) by means of a GA, obtaining the so-called GeneBot. The algorithm yields the evolved values shown in Table 2.

	$tithe_{perc}$	$tithe_{prob}$	ω_{NS-DIS}	ω_{GR}	$pool_{perc}$	$support_{perc}$	$support_{prob}$
AresBot	0.1	0.5	1	1	0.25	0.5	0.9
GeneBot (Best)	0.018	0.008	0.509	0.233	0.733	0.589	0.974
GeneBot (Average)	0.174 $\pm$0.168	0.097 $\pm$0.079	0.472 $\pm$0.218	0.364 $\pm$0.177	0.657 $\pm$0.179	0.524 $\pm$0.258	0.599 $\pm$0.178

Table 2. Initial behaviour parameter values of the original bot (AresBot), and the optimised values (evolved by a GA) for the best bot and the average obtained using the evolutionary algorithm (GeneBot).

Looking at Table 2 the evolution of the parameters can be seen. If we analyse the new values for the best bot of all the 15 executions, yielded in run number 8, it can be seen that the best results are obtained by strategies where colonies have a low probability of sending tithe, $tithe_{prob}$, to the base planet (only 0.008 or 0.09 in average value). In addition, those tithes send ($tithe_{perc}$) just a few of the hosted starships, which probably implies that colonies should be left on its own to defend themselves, instead of supplying the base planet. On the other hand, the probability for a planet to send starships to attack another planet, $support_{prob}$, is quite high (0.97 or 0.59 in average), and the proportion of units sent, $support_{perc}$, is also elevated, showing that it is more important to attack with all the available starships than wait for reinforcements. Related to this property is the fact that, when attacking a target planet, the base also sends ($pool_{perc}$) a large number of extra starships (73.3% or 65.7% in average of the hosted ships). Finally, to define the target planet to attack, the number of starships hosted in the planet, ω_{NS-DIS}, is much more important than the growth range ω_{GR}, but also considering the distance as an important value to take into account.

In order to analyse the value of these bots, a massive battle against AresBot has been conducted. The best bot in every run is confronted against it in 100 battles (one in each of the example maps provided by Google in the competition

pack). Table 3 shows the percentage of battles won by each bot. It can be seen that the best individual is the one of execution 8, i.e. the one considered as the best in the previous experiment (meaning a robust result). In addition, the improvement of the best bots with regard to AresBot can be noticed, since all of them win at least 82 out of 100 matches.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15
Winning Percentage	90	82	87	83	98	85	98	**99**	98	94	91	89	90	98	84

Table 3. Winning percentage of the best individuals against AresBot in 100 battles.

One fact to take into account in this work is that even as this solution looks like a simple GA (since it just evolves seven parameters) for a simple problem, it becomes more complicated due to the noisiness and complex fitness landscape; that is, small variations in parameter values may imply completely different behaviours, and thus, big changes in the battle outcome. This fitness nature is studied in the next section.

5.2 Noisy Fitness study

A good design of the fitness function is a key factor in any EA for getting success. It has to model the value of every individual. In a pseudo-stochastic environment (the victory or defeat depends on the opponent's actions) as is this, it is important to test the stability of the evaluation function, i.e. check if the fitness value is representative of the individual quality, or if it has been yielded by chance. In order to avoid this random factor a re-evaluation of the fittest individuals has been implemented, even if they survive for the next generation, testing continuously them in combat. In addition (and as previously commented), the fitness function performs 5 matches in 5 representative maps for calculating an aggregated number of turns, which ensures (in part) strongly penalising an individual if it gets a bad results.

The first study in this line is the evolution of fitness along the generations. Since the algorithm is a GA, it would be expected that the fitness is improved (on average) in every generation. It is shown in Figure 3, where it can be seen that as the evolution progresses (number of generations increases), the aggregate number of turns needed to win on five maps decreases (on average) on the three cases; however, since the result of every combat, and thus the fitness is pseudo-stochastic, it can increase from one generation to the next (it oscillates more than usual in GAs).

The second study tries to show the fitness tendency or stability, that is, if a bot is considered as a good one (low aggregated number of turns), it would be desirable that its associated fitness remains being good in almost all the battles, and the other way round if the bot is considered as a bad one (high aggregated number of turns). We are interested in knowing whether the fitness we are considering actually reflects the ability of the bot in beating other bots. It could be considered as a measure to determine if the algorithm is robust.

Figure 4 shows the fitness associated to two different GeneBots when fighting against the GoogleBot 100 times (battles) in the 5 representative maps. Both of

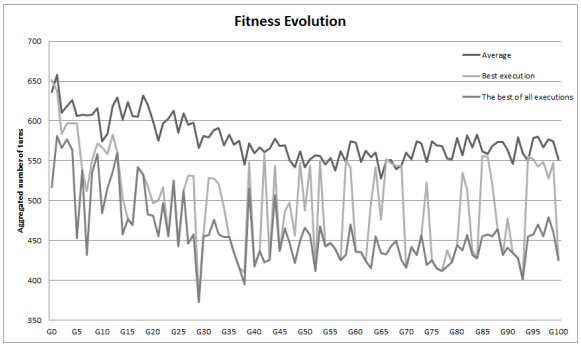

Fig. 3. The graphs show the complete execution of the best bot (best execution), the distribution of the best individuals in every run (best of all executions), and the average of the best bots in 15 runs.

them have been chosen randomly among all the bots in the 15 runs, selecting one with a good fitness value (578 turns), called *Promising Bot*, and another bot with a bad fitness value (2057 turns), called *Unpromising Bot*.

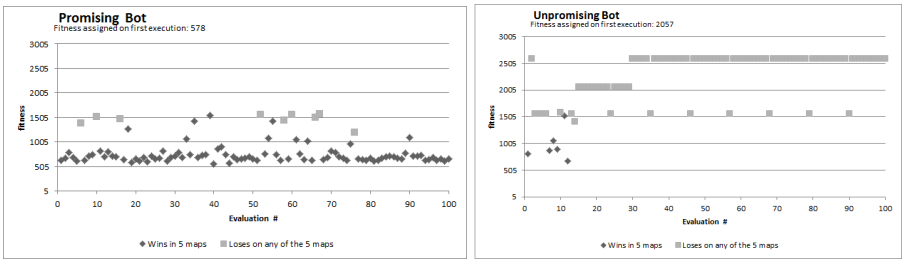

Fig. 4. Fitness tendency of two different and random individuals (bots) in 100 different battles (evaluations), everyone composed by 5 matches in the representative maps, against the GoogleBot.

As it can be seen, both bots maintain their level of fitness in almost every battle, winning most of them in the first case, and losing the majority in the second case. In addition, both of them win and lose battles in the expected frequency, appearing some outlier results due to the pseudo-stochastic nature of the fights.

5.3 GeneBots Fighting

Finally, a study concerning the behaviour of several GeneBots (the best in every execution) has been conducted to establish the validity of the fitness choice (better fitness means better bot). To do this, battles of 5 matches (in the 5 representative maps) have been performed. The winner in each battle is the bot who wins 3 out of 5 combats. Figure 5 shows the results, along with the fitness value for each bot.

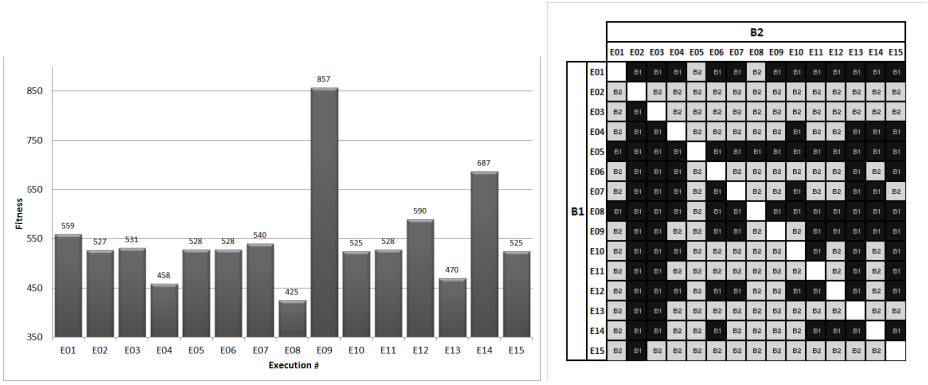

Fig. 5. Battles results between each pair of bots (the best in every execution). The winner in 3 out of 5 matches is marked (black for B1 victory and grey for B2 victory). The associated fitness to each bot (in its corresponding execution) is shown on the left graph.

The results demonstrate that individuals with lower fitness can hardly win to the fittest ones and the other way round, as it is desirable. However, it also proves the *noisy* nature of fitness, with a non-zero chance of the worst bot beating the best one.

6 Conclusions and Future Work

This paper shows how Genetic Algorithms (GAs) can be applied to the design of one autonomous player (bot) for playing Planet Wars game, which held the Google AI Challenge 2010. It have been proved that Genetic Algorithms (GeneBot) can improve the efficiency of a hand-coded bot (AresBot), winning more battles in a lower number of turns.

Besides, from looking at the parameters that have been evolved, we can draw some conclusions to improve overall strategy of hand-designed bots; results show that it is important to attack planets with almost all available starships, instead of keeping them for future attacks, or that the number of ships in a planet and its distance to it, are two criteria to decide the next target planet, much more important than the growing rate.

In addition, the presence of noisy fitness (the evaluation of one individual may strongly vary from one generation to the next) has been addressed by performing several battles in each evaluation, in addition to a re-evaluation of all the individuals in each generation. This subject has been studied in several experiments, concluding that the proposed algorithm yields results which have a good deal with it, being quite robust.

As future work, we intend apply some other techniques (such as Genetic Programming or Learning Classifier Systems) for defining the initial set of rules which limit the improving range of the bot by means of GAs. In the evolution-

ary algorithm front, several improvements might be attempted. For the time being, the bot is optimised against a single opponent; instead, several opponents might be tried, or even other individuals from the same population, in a co-evolutionary approach. Another option will be to change the bot from a single optimised strategy to a set of strategies and rules that can be chosen also using an evolutionary algorithm. Finally, a multi-objective EA will be able to explore the search space more efficiently, although in fact the most important factor is the overall number of turns needed to win.

References

1. Lidén, L.: Artificial stupidity: The art of intentional mistakes. In: AI Game Programming Wisdom 2, Charles River Media, INC. (2004) 41–48
2. Togelius, J., Karakovskiy, S., Koutnik, J., Schmidhuber, J.: Super mario evolution. In: Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09), Piscataway, NJ, USA, IEEE Press (2009)
3. Martín, E., Martínez, M., Recio, G., Saez, Y.: Pac-mant: Optimization based on ant colonies applied to developing an agent for ms. pac-man. In Yannakakis, G.N., Togelius, J., eds.: Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On. (2010) 458–464
4. Onieva, E., Pelta, D.A., Alonso, J., Milanés, V., Pérez, J.: A modular parametric architecture for the torcs racing engine. In: Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09), Piscataway, NJ, USA, IEEE Press (2009) 256–262
5. Sweetser, P.: Emergence in Games. Game development. Charles River Media, Boston, Massachussetts (2008)
6. Hong, J.H., Cho, S.B.: Evolving reactive NPCs for the real-time simulation game. In: Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05). (2005)
7. Falke-II, W., Ross, P.: Dynamic Strategies in a Real-Time Strategy Game. E. Cantu-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 1920-1921, Springer-Verlag Berlin Heidelberg (2003)
8. Aha, D.W., Molineaux, M., Ponsen, M.: Learning to win: Case-based plan selection in a real-time strategy game. In: in Proceedings of the Sixth International Conference on Case-Based Reasoning, Springer (2005) 5–20
9. Ontanon, S., Mishra, K., Sugandh, N., Ram, A.: Case-based planning and execution for real-time strategy games. In Weber, R., Richter, M., eds.: Case-Based Reasoning Research and Development. Volume 4626 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 164–178
10. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Improving opponent intelligence through offline evolutionary learning. International Journal of Intelligent Games & Simulation **2**(1) (2003) 20–27
11. Fernández-Ares, A., Mora, A.M., Merelo, J.J., García-Sánchez, P., Fernandes, C.: Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In: Evolutionary Computation, 2011. CEC '11. IEEE Congress on. (2011) accepted for publication
12. Herrera, F., Lozano, M., Sánchez, A.M.: A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. International Journal of Intelligent Systems **18** (2003) 309–338