Adaptive bots for real-time strategy games via map characterization

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

***Abstract*— This paper presents a proposal for a fast on-line map analysis for the RTS game Planet Wars in order to define specialized strategies for an autonomous bot. This analysis is used to tackle two constraints of the game, as featured in the Google AI Challenge 2010: the players cannot store any information from turn to turn, and there is a limited action time of just one second. They imply that the bot must analyze the game map quickly, to adapt its strategy during the game. Building in our previous work, which evolved bots for this game, in this paper we have evolved bots for different types of maps, and make the bot turn into one or the other in each turn depending on the geographical configuration of the game.**

Several experiments have been conducted to test the new approach, which outperforms our previous version, based on an off-line general training.

I. 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 tha affect all 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 the huge search space (plenty of possible behaviors) that is implicit in its action. Such difficulties are probably one of the reasons why Google chose this kind of game for their AI Challenge 2010 (in fact, a similar game, Ant Wars, was also chosen for the Fall 2011 challenge). In this contest, real time is sliced in one second *turns*, with players receiving the chance to play sequentially, although the resulting *actions* happen at the

simulated same time.

Department of Computer Architecture and Technology, University of Granada, Spain, {antares,pgarcia,amorag,jmerelo}@geneura.ugr.es

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.

In this work, 100 maps of the game have been studied and classified into nine different types, according to their *dispersion* and *distance* between bases. For each configuration, a bot has been trained using the evolutionary algorithm (EA) presented in [1], which evolves the parameters of a set of parametrized rules of the bot's engine. Then, an adaptive algorithm that includes the best parameters for each map configuration is compared with the best bot obtained in the previous work [2]. With this strategy we expect to achieve a certain degree of adaptability which is impossible to do online due to the constraints imposed on the game. Two criteria to analyze the map are proposed: distance among bases and planet dispersion. We have chosen this two factors because the number of planets is always the same, and the distance is one of the most important factors in our algorithm and during gameplay.

The paper is structured as follows: Section II addresses the problem by describing the game of Planet Wars. Then the literature is revised looking for related approaches to behavioral engine design in similar game-based problems in Section III. Section IV presents the proposed method for map analysis. After this, the expert bots design is introduced in Section V. The experiments and results comparing both, the new expert bot *Exp-GeneBot* and the previous bot *GeneBot*, are described and discussed in Section VI. Finally, the conclusions and future lines of work are presented in Section VII.

II. The Planet Wars Game

The Planet Wars game is a simplified version of the game Galcon, aimed at performing fights between bots, which was used as base for the Google AI Challenge 2010 (GAIC)[1].

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 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

[1]http://http://planetwars.aichallenge.org/

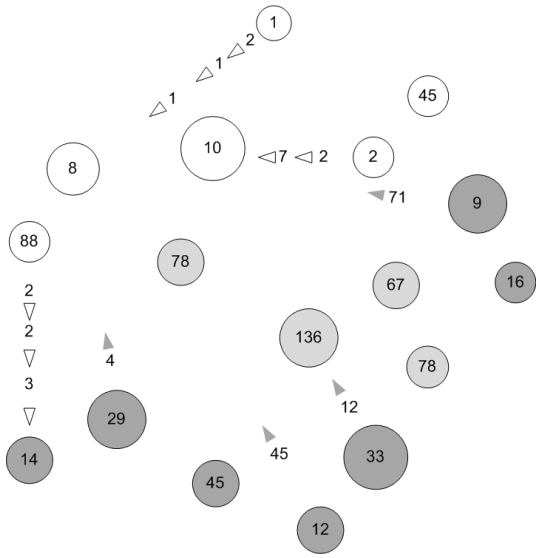

Fig. 1. Simulated screen shot of an early stage of a run in Planet Wars. White planets belong to the player (blue color in the game), dark grey belong to the opponent (red in the game), and light grey planets belong to no player. The triangles are fleets, and the numbers (in planets and triangles) represent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

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 [1].

III. 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 [3].

Most researchers have focused on relatively simple games such as Super Mario [4], Pac-Man [5] or Car Racing Games [6], since there are many bots competitions that involve them. Our paper is based in one of these competitions.

RTS games show an emergent component because several AIs are working at the same time. This feature can make a RTS game more entertaining for a player, and maybe more interesting for a researcher, and, in fact, there are many research issues related to the AI for RTSs, including planning in an uncertain world with incomplete information, learning, opponent modeling and spatial and temporal reasoning [7].

However, the reality in the industry is that in most 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. [8] proposed a learning classifier system that can be used to endow the computer with dynamically-changing strategies that respond to those displayed by the user, thus greatly extending the games playability. Strategy Game Description Language (SGDL) is used to model game strategies and scenarios [9]. They conclude that different fitness measurements depends of the scenario (RTS vs static games, for example), and there also exist differences in fitness according the used agent, so it is difficult to establish a standardized agent for other comparisons.

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 [10]. In this sense, Ontano et at. [11] 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 proposes advanced behaviors to achieve specific goals. In our work, similar ideas are applied, but not to imitate human player's behavior.

Other techniques, such Evolutionary Algorithms (EAs), 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 corresponds to off-line applications [12], 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 off-line 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. Our work also uses an EA for this purpose. In [13] co-evolutionary experiments in an abstract RTS game to evolve the strategies to apply in next turns are presented. In our work, we train a new bot in different scenarios fighting with previously trained ones. Evolutionary algorithms are not only used to adapt the bots behavior: in [14] a whole game is evolved (rules, scenario and level).

Nowadays, adaptive gaming have a great influence: reinforcement learning and evolutionary algorithms are used in [15] to dynamically adapt on-line the AI behavior to the player skills. Several adaptative controllers were tested against static controllers (such as neural networks or rule-based controllers), being adapted during the gameplay (without off-line training). Authors in the survey [16], create a taxonomy to explain the different levels of adaptation. *Targets* are the game components to adapt (narrative or AI, for example), and *methods* are the techniques to do the adaptation. Authors propose two levels of *methods* to adapt a game: on-line and off-line, but they claim that a combination of both levels can create better games, understanding "better" as the specific *purpose* of the game. Also, separating the problem into smaller subproblems or tasks (training camp) increases the agent learning, as demonstrated in [17], were a Ms Pac-Man agent performs better than using the standard

GP algorithm. This ideas are applied to our work for the Planet Wars game: first the bot is trained in the different maps (off-line), and the different sets of obtained parameters are grouped into an algorithm to adapt to the current state of the map (online). We compare the results obtained in this work to decide if off-line training and on-line adaptation performs better than a more generalistic, but not-adaptative, bot presented in [1], [2].

IV. Map Analysis

A characterization of the 100 maps provided by Google has been carried out according two different criteria: *distance* and *dispersion*. After many games played, we realized than the fleets are travelling across planets most of the time, so, distances among planets are one of the most important factors during the game.

The first criteria (distance) measures the distance between player bases (planets with highest number of ships), and can be grouped into three classes: far (more than 22 distance units), medium (between 16 and 22) and close (less than 16). Since the bases can change during the run, this value can also change in time. The other measurement is dispersion, which indicates the aggregation of the planets, and it can also be divided into three different classes: uniform, peripheral or centered.

Because the one second per turn restriction, a fast way to calculate the dispersion must be used. To do that, the map is divided in a 3x3 matrix and the number of planets in each cell is counted. Then, the average of the difference in number of planets of the peripheral cells with the planets in central cell is calculated. Figure 2 shows an example of the analysis in the Map 5 included in the development kit. In this case it is a map with periferal dispersion and medium distance. The dispersion is then classified as follows:

- Uniform: If dispersion is inside the -1 and 1 (the map is uniformly distributed) range.
- Peripheral: If dispersion is lower than -1, the map is decentralized (there is empty space in the galaxy and the planets are disperses in the periphery).
- Centered: If dispersion is greater than 1 the map is centralized (planets are grouped in the center of the galaxy).

Table I shows how the 100 maps provided with the GAIC development kit are divided into one of the 9 possibilities.

	Dispersion		
Distance	Peripheral	Uniform	Centered
closer	6	12	13
medium	5	26	16
far	4	6	12

TABLE I

Number of maps divided by type.

In each turn, our bot classifies the map as one of the 9 different combinations, to chose a fixed set of parameters depending the map and present situation.

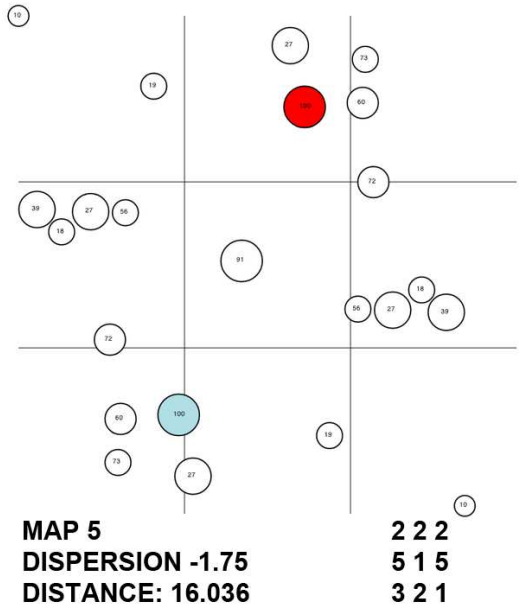

Fig. 2. Characterization of a map with medium distance and periferal dispersion.

V. Genetic Expert Bots

The initial bot presented in our previous work was based in a hand coded set of rules, whose behavior depends from several parameters. These parameters were evolved using an EA, to obtain a better bot. Figure 3 shows the internal flow of the bot's behavior. The set of parameters (weights, probabilities and amounts to add or subtract) has been included in the rules that model the bot's behavior. These parameters have been adjusted by hand, and they totally determine the behaviour of the bot. Their values and meaning are:

- $tithe_{perc}$: percentage of starships which the bot sends (regarding the number of starships in the planet).
- $tithe_{prob}$: probability that a colony sends a tithe to the base planet.
- ω_{NS-DIS}: weight of the number of starships hosted at the planet and the distance from the base planet to the target planet; it is used in the score function of target planet. It weights both the number of starships and the distance instead two different parameters since they would be multiplied and will act as just one as it is.
- ω_{GR}: weight of the planet growth rate in the target planet score function.
- $pool_{perc}$: proportion of extra starships that the bot sends from the base planet to the target planet.
- $support_{perc}$: percentage of extra starships that the bot sends from the colonies to the target planet.
- $support_{prob}$: probability of sending extra fleets from the colonies to the target planet.

Each parameter takes values in a different range, depending on its meaning, magnitude and significance in the game. These values are used in expressions used by the bot to take

decisions. For instance, the function that assign a score/cost to a *target planet* p is defined as (following a structure-based notation for p):

$$Score(p) = \frac{p.NumStarships \cdot \omega_{NS-DIS} \cdot Dist(base, p)}{1 + p.GrowthRate \cdot \omega_{GR}} \quad (1)$$

where ω_{NS-DIS} and, ω_{GR} are weights related to the number of starships, the growth rate and the distance to the target planet. $base$, as explained above, is the planet with the maximum number of starships, and p is the planet to evaluate. The divisor is added 1 to avoid a zero division.

More information about the parameters is explained in [1]

To train the bot in each map a Genetic Algorithm (GA) [18] has been used. A standard GA's procedure goes as follows: First, a population of chromosomes is randomly generated. All the chromosomes in the population are then evaluated according to the fitness function. A pool of parents (or mating pool) is selected by a method that guarantees that fitter individuals have more chances of being in the pool - tournament selection, fitness proportionate selection. Then a new population is generated by recombining the genes in the parents' population. This is usually done with a *crossover operator* (1-point crossover, uniform crossover, or BLX-α (for real-coded chromosomes), amongst many proposals that can be found in Evolutionary Computation literature) that recombines the genes of two parents and generates two offspring according to a crossover probability p_c that is typically set to values between 0.6 and 1.0 (if the parents are not recombined, they are copied to the offspring population).

After the offspring population is complete, the new chromosomes are mutated before being evaluated by the fitness function. *Mutation* operates at gene level, randomly changing the allele with a very low probability p_m (for instance, p_m is usually set to $1/l$ in many binary GAs, being l the chromosome length).

Once the evaluation of the newly generated population has been performed, the algorithm starts the *replacement of the old population*. There are several techniques for replacement, that is, for combining the offspring population with the old population in order to create the new population, but in our case an $e - elitism$ strategy is used, i.e., the best e chromosomes from the old population are copied without mutation to the new population. The remaining individuals are selected according to any method.

This process goes on until a stop criterion is met. Then, the best individual in the population is retrieved as a possible solution to the problem.

A GA was used in [1] to obtain a good trained bot: Genebot. However, after the training, this set of rules and parameter are fixed. In this work, 9 kinds of bots are trained (one for each map type), and 9 different set of parameters are obtained. Then, the map analysis explained in previous section is performed to select the parameter set for every turn.

The *evaluation* of one individual is performed by setting the correspondent values in the chromosome as the parameters for behavior, and placing the bot inside *five different maps* to fight against the best bot obtained in previous works (*GeneBot*). In every generation, the bots are ranked considering the number of game that the bot has lost (being better the lower this number is); in case of coincidence, then the number of turns to win in each arena is also considered (minimizing this value).

Source code of the presented algorithms is available in `https://forja.rediris.es/svn/genebot`, under a GNU/GPL license.

VI. Experiments and Results

Two kind of experiments have been performed: training the rules used in GeneBot per each type of map to obtain the Exp-GeneBot sets of parameters, and then, confront both types.

A. Expert-Bot Parameter optimization

As previously did in previous works, the EA (heuristically found) parameter values used in the training algorithm can be seen in Table II. These values were set according to what is usual in the literature and tuned up by systematic experimentation.

One run per type of map has been performed in the optimization of the Exp-GeneBot behavior parameters. Due to the high computational cost of the evaluation 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 the bot versus the *GeneBot* in 4 different map of the sets of Table I. Obtained sets of parameters can be seen in Table III. Parameters of the GeneBot used in next section are also shown.

Because the distance is not fixed during the algorithm, the maps are selected taking into account the distance among bases in the first turn. This decision has been taken because the first distance is the most usual during the run.

Looking at Table III the evolution of the parameters can be seen. If we analyze the values for each map, no evident correlation can be seen in all parameters. Grouping by *dispersion*, higher ω_{GR} are obtained when the planets are distributed in an uniform way. This parameter represent the weight of the planet growth rate in the target planet score function (explained in [1]), so it make sense that when planets are equally distributed, the planet to attack must be the one with higher growing rate (to obtain more fleets in future turns after conquered). However, this parameters are lower in the case the distance among bases is far, in part due to the way the bot is implemented. In the case that planets are more centered, reduced space among allied planets allows more support to the base (attacking at the same time). That is the reason why $support_{perc}$ is higher in this type of maps. Note that the $pool_{perc}$ value is higher in the GeneBot, mainly because it is a good behavior to send the most of fleets to attack in general.

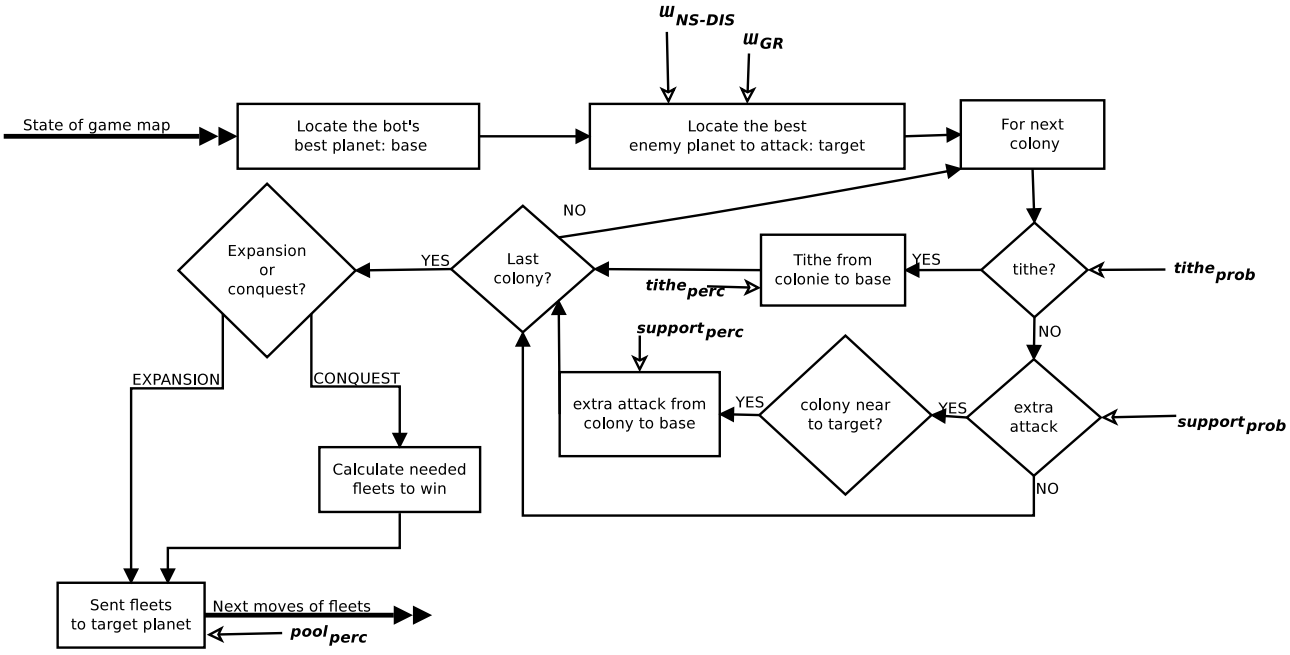

Fig. 3. Diagram of states governing the behavior of GeneBot, with the parameters that are evolved highlighted. These parameters are fixed in GeneBot, and adaptive in Exp-GeneBot.

Parameter	Value
GA type	Generational scheme with 4-elitism
Crossover (BLX-α crossover with α = 0.5)	0.6
Mutation of a random parameter in the $[0, 1]$ interval	0.02
Selection	2-tournament
Generations	100
Population size	200

TABLE II

PARAMETER SETTING CONSIDERED IN THE GENETIC ALGORITHM.

Distance/Dispersion	$tithe_{perc}$	$tithe_{prob}$	ω_{NS-DIS}	ω_{GR}	$pool_{perc}$	$support_{perc}$	$support_{prob}$
Closer/Peripheral	0.028	0.187	0.733	0.570	0.238	0.660	0.506
Closer/Uniform	0.251	0.063	0.0003	0.691	0.442	0.715	0.157
Closer/Centered	0.810	0.366	0.349	0.210	0.472	0.908	0.225
Medium/Peripheral	0.234	0.063	0.191	0.087	0.601	0.392	0.758
Medium/Uniform	0.045	0.215	0.286	0.794	0.512	0.388	0.294
Medium/Centered	0.479	0.082	0.182	0.498	0.198	0.906	0.254
Far/Peripheral	0.737	0.005	0.586	0.188	0.376	0.267	0.353
Far/Uniform	0.161	0.148	0.249	0.223	0.240	0.279	0.847
Far/Centered	0.022	0.947	0.170	0.356	0.587	0.963	0.741
GeneBot	0.018	0.008	0.509	0.233	0.733	0.589	0.974

TABLE III

OBTAINED VALUES OF THE BEST INDIVIDUALS FOR EACH OF THE MAP TYPES, AND THE BEST GENEBOT.

B. Expert vs GeneBot

In order to analyze the value of Exp-GeneBot, a massive battle against GeneBot has been conducted. Both bot have been confronted in 100000 battles (100 battles per map). Table IV shows the percentage of battles won by Exp-GeneBot. Due to the adaptative behavior of the algorithm, the type of map based in the *distance* criteria is obtained from the distance among bases in the last turn. Many battles ends without enemy base, so in this case this value is not taken into account (“No base” in Table). It can be seen that

this bot wins in almost all the types of maps, but it is slightly worse in centered maps. This could be based in the fact that distances among planets are too small to obtain advantage of the different sets of values, being GeneBot more optimized due to its general training and higher $pool_{perc}$ value.

Although the results show an improvement in Exp-GeneBot over GeneBot, several issues should be addressed. For example, an over-training could exist: maybe the bot has been optimized for the training maps and it is difficult to win GeneBot in non-trained maps. Also, due to the noisy fitness of the problem [2] and the amount of time to perform an experiment, the obtained set of parameters (the best bot for that for a map type) could be not significative.

VII. Conclusions and Future Work

This paper shows how map training technique can improve the performance of a autonomous player (bot) in Planet Wars game. Nine different sets of parameters to model a bot have been obtained after train the bot in 9 different types of maps using a Genetic Algorithm. Then this sets are grouped into a new bot that analyses the map in every turn to select the specific set of parameters for that turn (due to restrictions in the game in time and memory). This new bot wins more time than the best bot obtained in our previous work [1], where no map analysis exist.

Besides, from looking at the parameters that have been evolved, we can draw some conclusions to improve overall strategy of hand-designed bots taking into account the map. Results show that when the distances among planets are uniform, is better to send ships to the planet with higher growth ratio, and sending high number of supporting ships to that target planet.

It has been demonstrated in the experimental section that spent a small amount of time in evaluate the map confers some leverage to win more battles, which in turn, increase its performance. This indicates that an evolutionary algorithm holds a lot of promise in optimizing any kind of behavior, even a parametrized behavior such as the one programmed in GeneBot; at the same time, it also shows that when the search space is constrained by restricting it to a single strategy, no big improvements should be expected.

As future work, we intend to 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. More thresholds and criteria for the map analysis should be studied, for example, taking into account the size of the planets. In the evolutionary algorithm front, several improvements might be attempted. For the time being, the bot is optimized against a single opponent; instead, several opponents might be tried, or even other individuals from the same population, in a co-evolutionary approach. Finally, a multi-objective EA will be able to explore the search space more efficiently.

Acknowledgements

This paper has been funded in part by the P08-TIC-03903 project awarded by the Andalusian Regional Government, FPU Grant AP2009-2942 and TIN2011-28627-C04-02.

References

[1] A. Fernández-Ares, A. M. Mora, J. J. Merelo, P. García-Sánchez, and C. Fernandes, "Optimizing player behavior in a real-time strategy game using evolutionary algorithms," in *Evolutionary Computation, 2011. CEC '11. IEEE Congress on*, June 2011, pp. 2017–2024.

[2] A. M. Mora, A. Fernández-Ares, J. J. M. Guervós, and P. García-Sánchez, "Dealing with noisy fitness in the design of a rts game bot," in *EvoApplications*, ser. Lecture Notes in Computer Science, vol. 7248. Springer, 2012, pp. 234–244.

[3] L. Lidén, "Artificial stupidity: The art of intentional mistakes," in *AI Game Programming Wisdom 2*. Charles River Media, INC., 2004, pp. 41–48.

[4] J. Togelius, S. Karakovskiy, J. Koutnik, and J. Schmidhuber, "Super mario evolution," in *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09)*. Piscataway, NJ, USA: IEEE Press, 2009.

[5] E. Martín, M. Martínez, G. Recio, and Y. Saez, "Pac-mant: Optimization based on ant colonies applied to developing an agent for ms. pac-man," in *Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On*, G. N. Yannakakis and J. Togelius, Eds., August 2010, pp. 458–464.

[6] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, and J. Pérez, "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, pp. 256–262.

[7] J.-H. Hong and S.-B. Cho, "Evolving reactive NPCs for the real-time simulation game," in *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05)*, 4-6 April 2005.

[8] W. Falke-II and P. Ross, "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.

[9] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Modelling and evaluation of complex scenarios with the strategy game description language," in *CIG*, S.-B. Cho, S. M. Lucas, and P. Hingston, Eds. IEEE, 2011, pp. 174–181.

[10] D. W. Aha, M. Molineaux, and M. Ponsen, "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, pp. 5–20.

[11] S. Ontanon, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, R. Weber and M. Richter, Eds. Springer Berlin / Heidelberg, 2007, vol. 4626, pp. 164–178.

[12] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Improving opponent intelligence through offline evolutionary learning," *International Journal of Intelligent Games & Simulation*, vol. 2, no. 1, pp. 20–27, February 2003.

[13] D. Keaveney and C. O'Riordan, "Evolving coordination for real-time strategy games," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 3, no. 2, pp. 155–167, 2011.

[14] M. Cook, S. Colton, and J. Gow, "Initial results from co-operative co-evolution for automated platformer design," in *EvoApplications*, ser. Lecture Notes in Computer Science, vol. 7248. Springer, 2012, pp. 194–203.

[15] C. H. Tan, K. C. Tan, and A. Tay, "Dynamic game difficulty scaling using adaptive behavior-based ai," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 3, no. 4, pp. 289–301, 2011.

[16] R. Lopes and R. Bidarra, "Adaptivity challenges in games and simulations: A survey," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 3, no. 2, pp. 85–99, 2011.

[17] A. M. Alhejali and S. M. Lucas, "Using a training camp with genetic programming to evolve ms pac-man agents," in *CIG*, S.-B. Cho, S. M. Lucas, and P. Hingston, Eds. IEEE, 2011, pp. 118–125.

[18] A. Eiben and J. Smith, "What is an evolutionary algorithm?" in *Introduction to Evolutionary Computing*, G. Rozenberg, Ed. Addison Wesley, 2005, pp. 15–35.

[19] *Applications of Evolutionary Computation - EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, and EvoSTOC, Málaga, Spain, April 11-13, 2012, Proceedings*, ser. Lecture Notes in Computer Science, vol. 7248. Springer, 2012.

Map type (distance at final turn)	Num. of Battles	Perc. of victories
Closer/Periferal	438	76.94%
Closer/Uniform	1597	61.99%
Closer/Centered	1891	47.65%
Medium/Periferal	321	75.39%
Medium/Uniform	660	61.36%
Medium/Centered	361	44.04%
Far/Periferal	11	90.91%
Far/Uniform	60	81.67%
Far/Centered	32	43.75%
No base/Periferal	726	76.89%
No base/Uniform	2022	66.27%
No base/Centered	1870	65.28%

TABLE IV

VICTORIES OF EXP-GENEBOT VS. GENEBOT.

[20] S.-B. Cho, S. M. Lucas, and P. Hingston, Eds., *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011, Seoul, South Korea, August 31 - September 3, 2011*. IEEE, 2011.