GENERATING EMERGENT TEAM STRATEGIES IN FOOTBALL SIMULATION VIDEOGAMES VIA GENETIC ALGORITHMS

Antonio J. Fernández, Carlos Cotta and Rafael Campaña Ceballos
ETSI Informática,
Departmento de Lenguajes y Ciencias de la Computación,
University of Málaga,
Málaga, Spain
E-mails: {afdez,ccottap}@lcc.uma.es

KEYWORDS
AI, Evolutionary Algorithm, Simulation, Robot Football.

ABSTRACT

This paper defends the use of evolutionary algorithms to generate (and evolve) strategies that manage the behavior of a team in simulated football videogames. The chosen framework to develop the experiments is Robocup, an internatonal project that promotes the use of Artificial Intelligence in socially significant areas. One of these areas is related to computer games in the form of a simulated soccer league that allows two teams of 11 simulated robotic autonomous players to play football without human intervention. This paper proposes to generate emergent behaviors for the teams, via an evolutionary training process. The proposal is an alternative to implementing specific AI controllers for both players and teams in football videogames.

INTRODUCTION

The main aim of videogames (VG) is to provide entertainment to the player(s). In the past, research on commercial VGs was mainly focused on having more realistic games by improving graphics and sound (i.e., having higher resolution textures, more frames-per-second, ...etc). However, in recent years, hardware components have experienced exponential growth and players, with higher processing power computers, demand higher quality opponents exhibiting intelligent behavior.

In many simulated sports video games, the opposing team / the opponent (i.e., the enemy) is basically controlled by a fixed script. This is previously programmed and often comprises hundreds of rules, in the form *if the game is in state S then execute action A*, to control the behavior of the components (e.g.. members or players) of the team under specific conditions of the framework (i.e., a specific state of the game). This is quite a problem from both the developer and player point of view. For the former, it is a problem because these scripts are becoming more and more complex and thus it is not easy to program all the possible situations that could potentially happen. In fact, most of the games contain 'holes' in the sense that the game stagnates or behaves incorrectly under very specific conditions. As a consequence, the reality of the simulation is drastically reduced and so too the interest of the player. This problem relies on the category of 'artificial stupidity' (Lidén 2004). Also, for players, these scripts that model the opponent behavior are pre-programmed schemas whose behavior can become predictable for the experienced player, again causing a decrease in player interest.

To solve these problems, existing sports games employ some kind of artificial intelligence (AI) technique with the aim of making the opponents more intelligent, thereby making the game more attractive and increasing player satisfaction. However, even in these cases, the reality with respect to current sports videogames is that game AI is either not really AI and often consists of very specialized scripts (with the same problems as those already mentioned), or else game AI basically mimics a human player behavior. Nevertheless, even in this latter case, a problem remains: the AI controlled strategy rarely evolves according to the behavior of the player during the game. Again the reality is that most videogames are divided into levels, and the opponents are pre-programmed according to these. In the most complex levels the player faces high-quality opponents who behave like humans. Once the player is able to beat all the opponents in each level, they lose interest. In this context, the generation of opponents whose behavior evolves in accordance with the player's increasing abilities would be an appealing feature and would make the game more attractive. For instance, an amateur player expects an amateur opponent (not necessarily pre-programmed) whereas a very experienced player demands high-quality opponents. Moreover, the addition of emergent behavior in a football simulation game can make it more entertaining and less predictable in the sense that emergent behavior is not explictly programmed but simply happens (Holland 2000;Sweetser 2007).

This paper represents a step in this direction and deals with football simulation videogames. It proposes the use of genetic algorithms (GAs), to generate, in a dynamic way, opponents that depend on both the user skills and the game level. The main contributions of the paper are the natural encoding of the team strategies, that make our proposal very simple to manage, and the definition of a fitness function based on two heterogeneous components to guide the processes of learning and improvement of the team strategies inside the genetic algorithm. We report our experience using

Genetic Algorithms (GAs) in the context of Robocup, an international robot soccer simulation framework in which researchers can develop new ideas in the area of AI, and their developments can be evaluated via a competition mechanism in which any AI proposal is tested against another one. It should be observed that our experience can be extrapolated to commercial football simulation videogames.

RELATED WORK

AI can play an important role in the success or failure of a game and some major AI techniques have already been used in existing videogames (Johnson and Wiles 2001; Millington 2006). Traditionally, game developers have preferred standard AI techniques such as Artificial Life, Neural Networks, Finite State Machines, Fuzzy Logic, Learning and Expert Systems, among others (Bourg and Seemann 2004; Mikkulainen et al. 2006).

Evolutionary algorithms (EAs) (we use this term in a broad sense to refer to any kind of evolutionary procedure, including genetic algorithms and genetic programming) offer interesting opportunities for creating intelligence in strategy or role-playing games and, on the Internet, it is possible to find a number of articles related to the use of evolutionary techniques in VGs. For instance, (Sweetser 2004) shows how EAs can be used in games for solving pathfinding problems; also (Buckland 2002) focused on bot navigation (i.e., exploration and obstacle avoidance) and proposed the use of evolutionary techniques to evolve control sequences for game agents. However, in general, most of the work published on the use of EAs in games is aimed at showing the important role EAs play in Game Theory and, particularly, their use in solving decision-taking (mainly board) games. For example, (Kim and Cho 2007) presented several methods to incorporate domain knowledge into evolutionary board game frameworks and chose the board games checkers and Othello to experimentally validate the techniques. Also, it is worth mentioning the works of Fogel, which explored the possibility of learning how to play checkers without requiring the addition of human expertise via co-evolutionary techniques (Fogel 2000; Chellapilla and Fogel 2001). Other decision-taking games that have been handled via evolutionary methods are for example poker (Barone and While 1999), Backgammon (Pollack and Blair 1998) and Chinese Chess (Ong et al. 2007).

Evolutionary techniques involve considerable computational cost and thus are rarely used in on-line games. One exception however, published in (Fernández and Jiménez 2004), describes how to implement a genetic algorithm used on-line in an action game (i.e. a first/third person shooter). In fact, the most successful proposals for using EAs in games correspond to off-line applications, that is to say, the EA works on the user's computer (e.g., to improve the operational rules that guide the opponent's actions) while the game is not being played and the results (e.g., improvements) can be used later online (i.e., during the playing of the game). Through offline evolutionary learning, the quality of opponent intelligence in commercial games can be improved, and this has been proven to be more effective than opponent-based scripts (Spronck, Sprinkhuizen-Kuyper, and Postma 2003). Also, genetic algorithms have been used to evolve combat strategies for agents or opponents in between games (i.e., offline learning) as was done in the classical *Unreal Tournament* (Dalgaard and Holm 2002). Some realistic VGs that have used genetic algorithms are *return Fire II*, *The Creatures Series*, *Sigma and Cloak*, *Dagger and DNA*, and *Quake III*. For more detailed information about the use of EA in games the reader is referred to (Fogel, Blair, and Miikkulainen 2005; Lucas and Kendall 2006).

Regarding sports simulation VGs, readers can find some papers describing the evolutionary experiences mainly in simulated robot soccer frameworks. For instance, (Luke 1998) reported his experience of applying genetic programming (GP) to evolve team strategies. The teams were represented by trees and the GP system used low-level atomic functions designed for the soccer environment to build the strategies. Also, (Agah and Yanie 1997) used a genetic algorithm to evolve a team of 5 robots in a simulated soccer setting. Here, the agents (i.e., the robots) were built based on the approach of the Tropism-Based control architecture. (Nakashima et al. 2005) proposed to encode the set of action rules for soccer agents into integer strings. Although this is similar to what we have done, the approach is radically different as Nakashima et al. divided the soccer field into 48 subareas, and the action of the agent is specified for each subarea. Also, the replacement policy in the evolutionary process was quite different and they used a standard evolution strategy-type generation replacement schema i.e., the (μ+λ)-strategy (Eiben and Smith 2003).

GENETIC ALGORITHMS

A genetic algorithm is a population-based optimization algorithm that uses techniques inspired by evolutionary biology such as selection, inheritance, mutation, and recombination. Genetic algorithms manipulate a population of candidate solutions (also known as individuals), traditionally represented by binary strings that evolve towards better solutions. A typical genetic algorithm schema is shown in Figure 1:

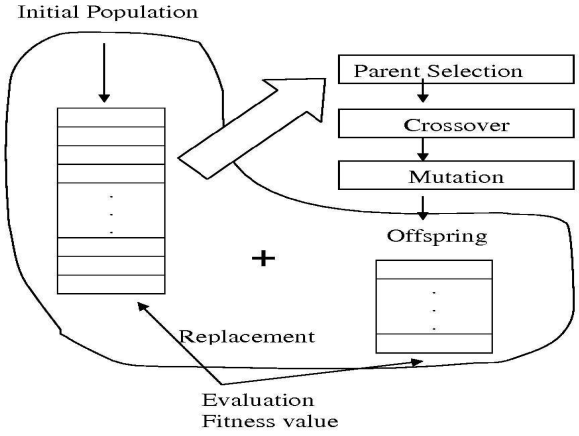

Figure 1: Standard Schema of a Genetic Algorithm

The basic process is as follows: initially a population of individuals is (often randomly) generated and the fitness of each member of this population is evaluated. Then the algorithm is executed until a termination condition is satisfied (e.g., a number of generations –iterations – or evaluations is reached, a solution is found, a desirable fitness value is obtained, etc). In each generation, some individuals (i.e., parents) from the current population are selected stochastically (this selection is usually based on their fitness values) and recombined to produce an offspring. The newly generated individuals can be modified via a mutation process; then the new individuals are evaluated. Finally, the population for the next generation is constructed from the individuals belonging to the current population and the new ones, now referred to as the offspring. This new population is then used as the current population in the next iteration of the algorithm (Eiben and Smith 2003).

ROBOCUP: THE SIMULATION LEAGUE

Robocup is an international framework in which researchers can develop new ideas in the area of AI and their developments can be evaluated via a competition mechanism in which any AI proposal is tested against another one. The basis of the research is robot soccer. Among the five leagues in RoboCup soccer, the simulation league is the most active research domain in terms of the development of computational intelligence techniques. This league is based on a simulator called "soccer server", which is a software system that simulates a football match. The server is a real time system that provides support for the competition between multiple virtual players in a multi-agent framework. It eases the communication with the client program, manages the entire physical environment (i.e., communications, objects, and hardware issues), and allows the visualization of the match in an X-window system. Figure 2 shows an example of a simulated match. The advantage of using the soccer server is that developers can focus on conceptual tasks such cooperation and learning. We will not go into more details as it is preferible to concentrate on the artificial evolution of the team strategies. The reader is referred to (Noda and Stone 2003) for more information on the server.

Figure 2: Soccer Server: An Example of a Simulated Match

The rules of a simulated match - very similar to those of the FIFA – (Fédération Internationale de Football Association) - are imposed (e.g., kick-off, goal, off side, half-time, time-up, players must not block the movement of other players, etc.). A Robocup agent (i.e., an autonomous player/bot)) receives, in real-time, information from the environment via three sensors: *aural sensor* (to detect the messages from the referee, trainers and the rest of the players); *visual sensor* (to detect information into a limited region, called the *agent visual range*; the information is about the current state of the match in the form of distance to and direction of the close objects i.e., players, balls, etc, all within a specific area of vision); and *corporal sensor* (to detect the physical state of the agent i.e., energy, velocity and direction).

EVOLUTION OF TEAM STRATEGIES

Our aim is to generate controllers to govern the behavior of an entire team (i.e., a set of 11 autonomours players/bots). A description of the technical issues (e.g., management of communication with soccer server or implementation of basic agent actions e.g., shot, pass, run, turn, etc., among others) of our proposal is beyond the scope of this paper. We concentrate thus on the process of developing controller behaviour.

The first step consists of generating (and evolving) a set of rules that will control the reactions of the agents at each instant of the game. These reactions depend on the current state of play. As already mentioned, each agent is continuously informed about the ongoing state of the game through the communication with the soccer server (via the three sensors mentioned above). Depending on the situation, the agent executes a particular action that may modify the state of the agent itself. The global team strategy is then the result of the sum of the specific behavior of all the agents.

However, the definition of specific strategies for each agent is complex, costly, requires a profound knowledge of the required behavior of each agent, and results in predictable behavior. To avoid these problems, all the agents (except the goalkeeper, who is manually implemented) will be managed by the same evolved controller. This means that we have to produce just one controller (thereby making the evolution process very much cheaper) to devise a global team strategy. Note, this does not mean that all the agents execute the same action because this depends on the individual situation of the agent in the match.

In the following we provide details of the genetic algorithm used to evolve team strategies.

Chromosome Representation (encoding): each individual in the population represents a team strategy, that is to say, a common set of actions that each agent has to carry out under specific conditions of the environment. These conditions depend on the information that the agent receives in its specific visual range (this differs from one agent to another). The information comes in the form of parameters that can take values from a range of values. Then, an individual in the population is represented as a vector v of k cells where k is the number of different situations in which the agent can be, and $v[i]$ contains the action to be taken in a specific

situation. In other words, if there are m parameters, and the parameter p_i (for $0 \leq i \leq m-1$) can have k_i possible values (numbered for 0 to k_{i-1}), then the cell:

$$v[e_{m-1}+e_{m-2}*k_{m-1}+e_{m-3}*(k_{m-1}*k_{m-2})+e_{m-4}*(k_{m-1}*k_{m-2}*k_{m-3})+\ldots]$$

contains the action to be executed when the parameters p_0, $p_1,\ldots,$ p_{m-1} take the values e_0, $e_1,\ldots,$ e_{m-1} respectively. Managing a large number of parameters (such as those provided by the server) is complex. Thus to reduce complexity, in our experiments we have considered the following manageable set of parameters:

- Advantage state (A_s): This paremeter can take two values that evaluate a possible advantage situation. 0: if the agent is supported by more team mates than rival ones. 1: otherwise.
- Ball kick (B_k): can the agent kick the ball? Two values. 0: the agent cannot; 1: the agent can.
- Agent position in the field (A_p): three values. 0:closer to its own goal area; 1: closer to rival goal area, 3: not defined.
- Ball possession (B_p): which agent is closer to the ball ? Four values. 0: the agent, 1: a team mate, 2: a rival agent, 3: not known.

The encoding of an individual in the population consists of a vector of 48 genes i.e., all the possible conditions that can happen from the different combinations of these parameter values. An additional parameter was also considered:

- Position of Ball (P_b). Three values depending on the proximity of the ball to the goal areas. 0: closer to agent goal area, 1: closer to the rival goal area, 2: not defined.

Considering the addition of this parameter the representation is a vector of length 144. We note that some combinations of values in the representations may make no sense (e.g., The combination $A_p = 0$ and $P_b = 1$ makes no sense with $B_k = 1$ because the agent would never be able to kick the ball). The addition of this kind of knowledge can lead to a simplified representation but in our experiments we did not limit any combination (in any case, this kind of optimization can be done at a later stage).

Each cell of the vector encoding a candidate solution will contain an action. In our experiments, 9 actions were considered:

- Go back: each agent has two positions by default: the *starting position* that corresponds to a fixed position in the field in which the agent is placed at the beginning of the match as well as after scoring a goal, and the *required position* that corresponds to a strategic position in which the agent can be placed during the game (e.g., forwards and defenders should be placed close to the rival area or team area respectively).
- Look for the ball: turn the agent body to align it with the ball if this is visible, otherwise turn 15º (an arbitrary value) to the right to modify the environmental conditions received in the visual area of the agent.
- Intercept ball: If ball is visible then the agent accelerates in that direction.
- Look around. Turn 15º right.
- Kick. Shoot in direction to the rival goal area.
- Pass to closest team mate.
- Pass to farthest team mate.
- Kick out: try to put the ball as far as possible from the team goal area.
- Drive ball: conduct the ball in the direction of the rival goal area.

This means that the search space (i.e., the number of different strategies that can be generated from this representation) is $9^{144} = 3^{288}$ if we consider the second representation, and $9^{48} = 3^{96}$ if we consider the 48 length representation. Figure 3 displays an example of a possible encoding of length 48. The optimal solution (if it exists) would be that strategy which always select the best action to be executed for the agents under all possible environmental conditions. In fact, these vast search spaces make this problem impracticable for many exact methods and ideal for genetic algorithms.

	Ball possesion (B_p)			
$A_sB_kA_p$	agent	Team mate	Rival agent	Not known
000	1	6	9	3
001	8	8	3	7
002	5	1	2	4
010	4	2	5	6
011	6	9	3	5
012	1	1	2	7
100	7	4	9	2
101	1	2	3	5
102	6	5	4	8
110	3	6	1	7
111	7	2	3	1
112	2	9	8	5

Example: Execute action 7 (pass away) if the player is not supported by team mates, can kick the ball, is positioned close to its lodge, and is the player placed closest to the ball in the field.

Actions:
1. Go back default position
2. Look for the ball
3. Intercept ball
4. Look around
5. Kick
6. Pass to friend
7. Pass away
8. Kick out
9. Drive ball

Parameters:
As: Advantage state
Bk: ¿ can the player kick the ball ?
Ap: ¿ Where is the player placed in the field?
Bp: ¿ Who is closer to the ball ?

Figure 3: Example of Encoding for an Arbitrary Individual

Fitness function evaluates the adequacy of a team strategy. Let *pop* be the population considered in the GA. Then, evaluating the fitness of an individual *pop[i]* (for $1 \leq i \leq$ population size) requires the simulation (in the soccer server) of the match between the opponent strategy (e.g., the one followed by a human in a previous game) and the strategy encoded in *pop[i]*. The fitness function depends on the statistical data collected at the end of the simulation (e.g., who won, goals scored by both teams, etc). The higher the number of statistical data to be collected, the higher the computational cost will be. A priori, it seems a good policy would be to consider a limited number of data. Five data were used in our experiments during the simulation of a match between *pop[i]* and the opponent.

1. Ball closeness (c_i): distance average from team players to the ball.
2. Ball possession (p_i): average time that the ball is in the rival field.
3. Ball in area (a_i): average time that the ball is in the rival area.
4. Scored goals (sg_i).
5. Received goals (rg_i).

The fitness function to evaluate any individual *pop[i]* in the population is then defined as follows::

$$fitness(pop[i]) = f_1(c_i, p_i, a_i) + f_2(sg_i, rg_i)$$

where

$$f_1(c,p,a) = w_1 \cdot c + w_2 \cdot p + w_3 \cdot a \qquad (1)$$

and

$$f_2(sg,rg) = \begin{cases} 0, & \text{if } sg = 0 \\ w_4 \cdot ((sg - rg) + 1), & \text{if } ((sg \geq rg) \\ (w_4 - 1)/(rg - sg), & \text{otherwise} \end{cases} \qquad (2)$$

The higher the fitness value of a strategy is, the better the strategy. Observe that the fitness function is based on two very different components. The first, defined in (1), has the aim of teaching to the strategy *pop[i]* how to play according to the basic rules of football (note that initially the population is randomly initialized and thus most individuals will not even know how to play football). Weights were assigned as w_1=0.01, w_2=0.1, and w_3=1; the reason for assigning these values was to promote the evolution towards strategies in which the ball is in play more time in the rival field (if possible near the goal area) than in the team field (it seems reasonable to think that this policy will result in less "received" goals). The second component of the fitness function, defined in (2), should help the strategy *pop[i]* to evolve towards better solutions (i.e., those able to beat the opponent). The weight w_4 is assigned to 100 so that in the case of a victory or a draw, the function returns a number that is multiple of 100, thus giving priority to the higher difference of goals; otherwise (in the case of defeat), priority is given to a minor difference of goals.

The fitness function is non-deterministic (i.e., different simulations of the same match can produce different results) as the simulation is affected by random factors that the soccer server adds to provide a greater sensation of realism. This fact, which can be viewed as another reason for not using complete solving techniques, is really an added incentive to use genetic algorithms as it is well-known that GAs incorporate a certain degree of randomness that makes them suitable for handling this problem.

Evolutionary operators: Our GA is a steady state algorithm that uses single point crossover, binary tournament for the parent selection, and elitism for the replacement policy. Mutation is done in the gene – i.e., action – level by changing an action to any other action.

EXPERIMENTAL SECTION

Extensive tests, varying the probability of mutation P_M (i.e., 0.1, 0.01, 0.001, 0.0001), the number of generations (i.e., 150,300), offspring length (i.e., 1,2,3,4), and individual size (i.e., 48 and 144) were carried out. The population length and the crossover probability P_X were set to 30 and 1.0, respectively in all the tests (i.e., instances) and the population was always initialized randomly. Ten runs per test instance were executed. In addition, two type of tests were conducted: one (mode 1) in which the GA was executed with the aim of finding one strategy (i.e., a winning strategy) to beat a manually implemented opponent; and another type (mode 2) in which each winning strategy obtained was incorporated into the objective function (i.e., the objective was not only to beat an opponent but to beat all the opponents in the objective function). The experiments demonstrated the validity of our proposal in the first mode. Due to the large number of experiments done, only some examples of evolutionary advances are shown in the following.

For instance, Figure 4 displays the average fitness (from ten runs) resulting from one of the test instances executed. It is interesting to note that initially the team strategies encoded in the population as individuals play really very badly (in fact, they do not know how to play), but they evolve quickly within a few generations. This behaviour of the evolutionary process is common in all the tests.

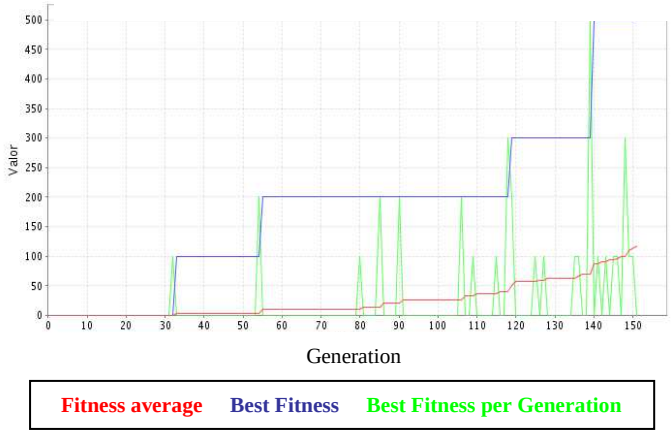

Figure 4: Pop size: 30, Offspring size: 1, P_M = 0.001. Average Results for Fitness Value/Generations

Figure 5 shows for the same instance, the evolution of the data values used to define the fitness function; one can observe for instance that the "received" goals decrease whereas the scored goals increase. Also, note the pressure to place the ball close to the rival area.

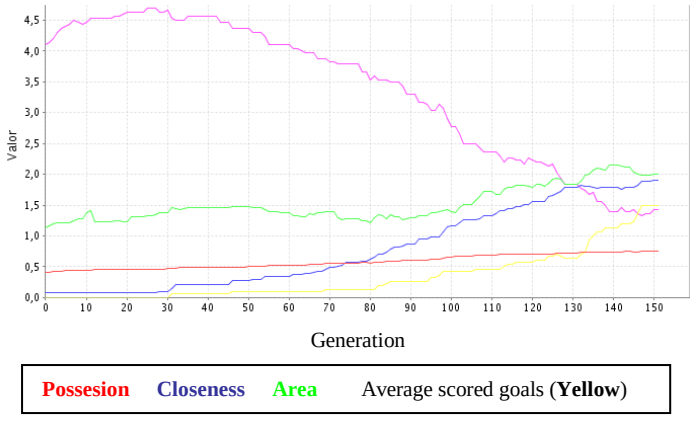

Figure 5. Evolution of Values Defining the Fitness Function

Tests in mode 2 were also developed with poor results: the algorithm could be executed with two opponents in the objective function but not with three. The reason for this bad performance is the cost of evaluating the fitness values since for each individual, three simulations had to be carried out at

considerable computational cost (note that the strategy encoded in the individual has to be evaluated against each of the three teams included in the objective function). Note however that this mode rarely has applications in simulated football games in which there is just one opponent (i.e., the human player). Mode 2 is interesting if one wants to produce strategies to be tested in a competitive environment such as for instance Robocup, but this was not the original motivation for this work.

CONCLUSIONS AND FURTHER RESEARCH

In this paper we have shown that genetic algorithms are a simple mechanism to obtain emergent behavior strategies to control teams in football simulation videogames (such as for example *FIFA, Pro-evolution or Football Manager* series). The genetic algorithm described in this paper has the particularity that is guided by a fitness function defined with two very heterogeneous components: one that guides the basic learning of the football principles, and the other that strives to find winning strategies. These two components, although radically different, are complementary however. Experiments were conducted to validate the feasibility of our approach with promising results. The evolutionary learning described in this paper could be used in existing football simulation games taking as opponent the player's game strategy, which can be deduced by collecting statistical data during the game. In this sense, the evolutionary algorithm would produce player's skills-based self-adaptive opponents. This, together with the randomness associated with GAs, would lead the evolutionary process towards the generation of opponents with non-predictable behavior.

The main drawback of our technique is the computational cost associated to the simulation of matches that are necessary to execute in order to evaluate the fitness values of the evolved teams. This however can be minimized in our working framework as the soccer server provides facilities for parallel execution and therefore several matches can be running on different machines at the same time, thus reducing the computational cost. Nevertheless, this is an issue for further research.

ACKNOWLEDGMENTS

This work has been partially supported by projects TIN2007-67134 and TIN2008-05941 (from Spanish Ministry of Innovation and Science) and P06-TIC2250 (from Andalusia Regional Government).

REFERENCES

Agah, A. and Yanie, K. 1997. "*Robots Playing to Win: Evolutionary Soccer Strategies*". In Proceedings of the 1997 IEEE lnternational Conference on Robotics and Automation, 632-637.

Barone, L. and While, L. 1999. "*An Adaptive Learning Model for Simplified Poker Using Evolutionary Algorithm*". In Proceedings of the Congress on Evolutionary Computation, IEEE Press, 153-160.

Bourg, D.M. and Seemann, G. 2004. *AI for Game Developers*. O'Reilly.

Buckland, M. 2002. *AI Techniques for Game Programming*. Premier Press.

Chellapilla, K. and Fogel, D.B. 2001. "Evolving an expert checkers playing program without using human expertise". In *IEEE Trans. Evolutionary Computation* 5, No. 4, 422-428.

Dalgaard, J.and Holm, J. 2002. "Genetic Programming applied to a real time game domain". Master Thesis, Aalborg University - Institute of Computer Science, Denmark.

Eiben, A.I., and Smith, J.E. 2003. *Introduction to Evolutionary Computing*. Springer.

Fernández, A.J. and Jiménez, J. 2004. "Action Games: Evolutive Experiences". In *Computational Intelligence: Theory and Applications*, Bernd Reusch (ed.), Springer, 487-501.

Fogel, D.B. 2000. "Evolving a checkers player without relying on human experience". In *Intelligence* 11, No. 2, 20-27 (2000)

Fogel, D B; Blair, A; and Miikkulainen, R (eds). 2005. "Special Issue: Evolutionary Computation and Games". In. *IEEE Trans. Evolutionary Computation* 9, No. 6,

Holland, J.H. 2000. *Emergence: from Chaos to Order*. Oxford University Press.

Johnson, D. and Wiles, J. 2001. "Computer Games with Intelligence". In *Australian Journal of Intelligent Information Processing Systems* 7, 61-68.

Kim, K-J., and Cho, S-B. 2007. "Evolutionary Algorithms for Board Game Players with Domain Knowledge". In *Advanced Intelligent Paradigms in Computer Gam*es, Baba, N, Jain L.C. and Handa H. (eds), Springer, 71-89.

Lidén, L. 2004, "Artificial Stupidity: The Art of Making Intentional Mistakes", In *AI Game Programming Wisdom 2*, S. Rabin, ed., Charles River Media, Inc., pp. 41-48.

Lucas, S.M., and Kendall, G. 2006. "Evolutionary Computation and Games". In *IEEE Computational Intelligence Magazine* 1, No.1, 10-18.

Luke, S. 1998. "*Evolving Soccerbots: a Retrospective*". In Proceedings of 12th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI). Invited paper.

Miikkulainen, R; Bryant, B.D.; Cornelius, R.; Karpov I.V.; Stanley, K.O.; and Yong, C.H. 2006. "Computational Intelligence in Games". In *Computational Intelligence: Principles and Practice*, Yen, G. Y. and Fogel, D. B. (editors), IEEE Computational Intelligence Society, 155-191.

Millington, I: 2006. *Artificial Intelligence for Games*. Morgan Kaufmann.

Nakashima, N; Takatani, M; Udo, M; Ishibuchi, H; and Nii, M. 2005. "*Performance Evaluation of an Evolutionary Method for RoboCup Soccer Strategies*". In *Proceedings of RoboCup 2005*, LNCS 4020, Springer, 616-623.

Noda, I: and Stone P. 2003. "The RoboCup Soccer Server and CMUnited Clients: Implemented Infrastructure for MAS Research**".** *Autonomous Agents and Multi-Agent Systems* 7 , No.1-2 (July-September), 101-120.

Ong, C.S.; Quek, H.Y.; Tan, K.C.; and Tay, A. 2007. "*Discovering Chinese Chess Strategies through Coevolutionary Approaches*". In Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games, 360-367.

Pollack, J.B. and Blair, A.D.1998. "Co-Evolution in the Successful Learning of Backgammon Strategy". In *Machine Learning* 32, No. 1, 22-240.

Spronck, P.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2003. "Improving Opponent Intelligence through Offline Evolutionary Learning". In *International Journal of Intelligent Games&Simulation* 2, No. 1, 20-27.

Sweetser, P. 2004. "How to Build Evolutionary Algorithms for Games". In *AI Game Programming Wisdom 2*, S. Rabin, ed., Charles River Media, Inc., pp. 627-638.

Sweetser, P. 2007. *Emergence in Games*. Charles River Media.