AN EVOLUTIONARY ALGORITHM APPROACH FOR A REAL TIME STRATEGY GAME

Rodrigo de Freitas Pereira Claudio Fabiano Motta Toledo
Marcio Kassouf Crocomo Eduardo do Valle Simões

University of São Paulo, SSC, ICMC, São Carlos, SP, Brazil

Figure 1: The user interface of the BOS War game

Abstract

The present paper reports the preliminary results of the application of an evolutionary algorithm developed to adjust artificial intelligence scripts in a real time strategy game. The evolutionary algorithm (EA) is embedded in the game engine of the Bos Wars, which is a real time strategy game coded in C++ and LUA language. The proposed algorithm is an adaptation of a similar approach introduced in [Crocomo 2008], which was applied to the battle system of a Role Playing Game. A tailor-made representation of the individual chromosomes is proposed, as well as a set of new genetic operators. The computational results indicate a superior performance of the scripts produced by the presented evolutionary algorithm when playing against the set of the standard scripts available in Bos Wars [2010].

Keywords: Artificial Intelligence, Evolutionary Algorithm, Real Time Strategy Games

Authors' contact:
rodrigofp@grad.icmc.usp.br, claudio@icmc.usp.br, marciokc@gmail.com, simoes@icmc.usp.br.

1. Introduction

The game industry innovates constantly to satisfy its customers. It is already possible to play using sensors that detect movement and a lot of effort has been spent to allow massive multiplayer matches. There have been also advances in the degree of realism with the development of sophisticated audio and graphics engines that lead players closer to the game environment [Bittencourt and Osório 2006].

Another factor about the realism in games that has received special attention in the last years is the artificial intelligence (AI) [Millington 2007]. And, more specifically, intelligent scripts responsible to control the decision-making process for non-player characters (NPCs). A well planned AI script should be able to provide gaming experiences against NPCs that are more similar to playing against other human players. In that case, real time adaptation of the NPCs behavior to player strategies can increase the level of entertainment [Ponsen et al. 2007].

In [Sweetser 2002], several AI techniques applied to programming game scripts are reported, such as fuzzy logic, flocking, decision tree, finite state machines, artificial neural network and evolutionary algorithms. The finite state machines (FSM) have been most frequently used to implement AI in games, since they are relatively easy to code and understand, and usually they are able to hit the objective proposed in the game [Sweetser 2002]. However, FSM can make the game strategy predictable leading the players to lose interest early.

The use of evolutionary algorithms (EAs) to provide AI scripts, in the other hand, is advantageous according to Lucas and Kendal [2006] due to its natural adaptability that allows them to generate different and unpredictable strategies. For this reason, the motivation for using EAs as the game AI goes beyond making the computer win or lose. Since EAs give the computer the ability to create strategies that sometimes outperform the player current abilities, it forces the player to create new strategies, improving the overall ability to entertain of the game.

The skills of the human player can also be improved by evolutionary algorithms as reported by Smith et al. [2010]. The authors argue that the use of evolutionary algorithms as AI by NPC improved the human player abilities more than playing against other humans.

In this context, the present paper proposes a novel EA that can be used to program scripts to control the AI for a Real Time Strategy (RTS) game called Bos Wars. This is a game that demands resource management and action planning to attack other opponents. A screenshot of the Bos Wars game is presented in Figure 1.

The developed EA is an adaptation of the evolutionary algorithm presented in Crocomo [2008] that was applied to optimize NPC behavior in a Role-Playing Game (RPG). However, this algorithm was modified to incorporate a tailor-made representation of individuals, as well as specific initialization, mutation and crossover operators.

The paper is organized as follows. The related works are presented in Section 2 and the RTS game Bos Wars is described in Section 3. The proposed EA is detailed in Section 4 and the results found are reported in Section 5. The conclusions follow in Section 6.

2. Related Works

Appolinario and Pereira [2007] developed a game where a tank must reach a target avoiding randomly inserted obstacles. The authors applied an EA to execute actions for this tank. An individual is coded as a sequence of actions which are defined by several possibilities of tank rotation. This is different from the proposal of this paper, since our objective in the Bos Wars game is to generate strategies for controlling multiple individuals, considering that basic individual behaviors, such as path finding, is solved by the game engine.

Spronck et al [2006] proposed a method called Dynamic Scripting (DS) to be used in online learning for games. The authors say that EAs lack the necessary qualities needed by an algorithm to be applied in the online learning of commercial games.

Ponsen et al. [2007] proposed an EA as a learning routine in a RTS game called Wargus[1]. Each individual is represented by 20 building states and a set of actions is determined from each construction type defined by each state. Although an EA is used by the proposed AI, it is responsible for off-line learning in the game, meaning that the evolutionary process occurs before the game begins. The Dynamic Scripting algorithm is used for online learning, i.e., during the match.

The off-line learning consists in the acquisition of knowledge by the NPCs without contact with a player, it can be done before the game release and it can test many variations of AI whereas it is impossible by a human in a short amount of time [Posen et al. 2007].

The on-line learning makes possible the adaptation of behaviors used by NPCs during the gameplay, according to the actions performed by a player [Spronck et al. 2004].

An EA is also applied by Smith et al. [2010] on a RTS game. The aim of this EA is to define simple tactics spatially oriented to control the overall strategy of game NPCs. The authors verified that players were able to develop creative strategies that were effective against these tactics when playing against the evolutionary algorithm. They argue that the learning process was more effective in this way than playing against human opponents.

The behavior of NPCs was also improved by Jang et al. [2009] using an evolutionary algorithm in a RTS game. The algorithm is used to make decisions about what actions should be executed. The authors developed a game called Conqueror, where the computational tests showed a superior performance of the proposed method.

Crocomo [2008] proposed an EA as the AI for a Role Playing Game (RPG) based on the Baldur's Gate game. The EA individuals (or chromosomes) encode rules that have to be executed by the NPC through the matches. These actions are selected from a data base of rules previously defined. The implemented EA presented good results in the game online learning, despite the statements presented by Spronck et al [2006].

In this paper, we use an EA based on the one developed in [Crocomo 2008], where the EA was divided in three steps: generate initial population, evaluate individuals and generate the next population. In the former EA, each individual has a chromosome that stores game rules. These rules are executed during the battles by the NPCs and they are chosen from a set of rules previously defined. The rules consist of basic actions such as attack, move, drink potion or cast spell. The initial population was formed by 4 individuals generated by a tournament, where the best individual was always kept in the next population. The other 3 individuals of the next population were obtained executing a uniform crossover between the best individual and each one of the other individuals.

The EA approach in the present paper is responsible for the game online learning. In doing so, we aim to expand the results from [Crocomo 2008], showing that EAs can also be successfully applied to the online learning of RTS games. This is also a different approach than the EA developed in [Ponsen et al. 2007], which does not deal with online learning.

Another goal of this project is to create a new AI strategy for the Bos Wars Game, which is an open source RTS game. In doing so, we intend to contribute to other academic researches, which will be able to test other learning strategies against the one reported on this paper.

To sumarize the main contributions of this paper, we may highlight: i) the generalization of the EA proposed in [Crocomo 2008] to a different type of game (from

[1] Wargus. Available in :< https://launchpad.net/wargus>.[Accessed 16 December 2011]

RPG to RTS) and ii) the creation of an AI for the Bos Wars game, which is an open source game that can be used as a test platform for new AI techniques developed from academic research.

3. The Bos Wars Game

The game we chose is called Bos Wars [2010]. It is a RTS game set in a futuristically environment, where players must not only fight against enemy teams, but also develop their economy. Thus, it is necessary to find a balance among keeping the economy growing and at the same time produce strong armies to fight off enemy invasions. The game allows playing against human as well as against computer scripts. It is an open source game that runs under Windows and Linux.

The game starts with one or more opponent teams, where each team starts with some structures like a town center, a power plant and some engineers as illustrated in Figure 2.

The aim of the player is to completely destroy all the enemy units and structures. The resources must be collected as well as armies need to be built to attack and repel the enemies.

There are two types of resources available in Bos Wars: magma and energy. These resources are used to the creation of structures and armies. To collect magma, it is necessary to build magma pumps on hot spots spread on the map or to use engineers to collect it from rocks. There are three ways to get energy: building a power plant, a nuclear power plant, or collecting it from trees and morels with engineers.

Figure 2: The game start.

Bos Wars offers a wide variety of buildings and units to create. The buildings include vaults, power plants, aircraft factories, turrets for defense among others, totalizing 14 available structures. The unities include assault tanks and aircrafts in a total of 15 available types.

The game comes with five pre-made scripts which represent strategies to be executed by NPCs as their AI. There are three offensive scripts, one defensive, and one that is more balanced, having a balance between offensive and defensive actions. All these scripts are coded in two parts: a set of instructions that are executed just once and another set that are executed repeated times in a loop until the game is over. This looping is used to keep the NPCs executing different actions during the gameplay.

The game engine was developed in C++ and the AI scripts such as definitions of units, buildings, images, sounds and data capture functions were coded in the LUA language [LUA 2012]. Thus, the EA code was embedded into the game engine. The EA, which was also developed in C++, must communicate with both environments to make possible the loading of AI scripts into the LUA environment. Figure 3 shows a scheme of this communication mechanism.

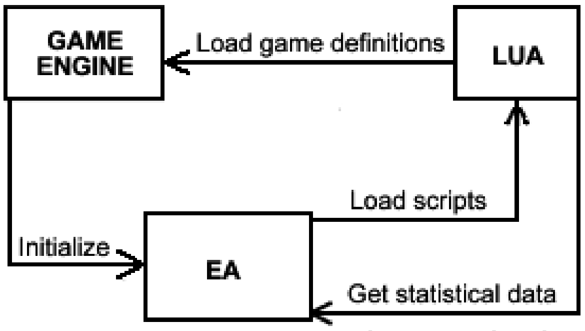


Figure 3: Communication among the Genetic Algorithm, C++ and LUA

4. The Proposed Evolutionary Approach

The proposed EA is an adaptation of the method developed by Crocomo [2008], originally coded for a Role-Playing Game (RPG) simulator based on the Baldur's Gate game. The first part of the adaptation lies on the representation of the EA individuals. The individual is constituted of genes, which represent a sequence of atomic actions. These actions are divided in two groups: actions to build constructions and actions to create armies. Figure 4 illustrates the two possible gene representations.

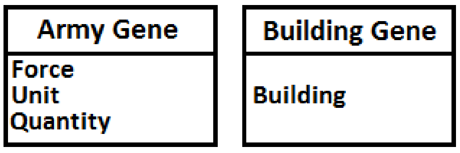


Figure 4: Representation of Genes

The "Building Gene" just contains the information about what kind of structure will be built. The "Army Gene" contains information concerning what kind of unit will be created, its quantity and its force. The force is an identification integer value between 0 and 9. This number will be used to identify the army during the decode phase of the gene by the EA. Figure 5 illustrates a possible individual representation.

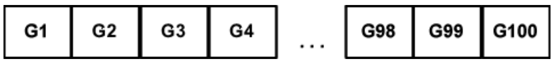
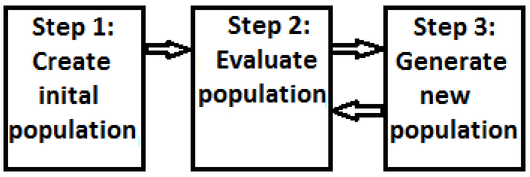


Figure 5: Representation of an individual containing 100 genes

For instance, the gene G1 of the individual in Figure 5 can encode the action "Build a vehicle factory" and G100 can mean "Create 3 helicopters unities.

The genes in each individual are decoded into an instruction of the game script. Thus, all the information encoded in each individual can be decoded generating a script that will be executed as an AI control strategy by the NPCs. The EA executes basically the three steps presented in Figure 6.


Step 1: Create inital population
Step 2: Evaluate population
Step 3: Generate new population
Figure 6: Evolutionary Algorithm Steps

In the first step, the initial population is generated by a tournament. A total of 8 individuals are randomly generated and the champion of this tournament is inserted in the initial population. A total of 4 tournaments are executed with a total of 32 individual randomly generated and evaluated. Thus, the EA will have an initial population of 4 individuals (the 4 winning strategies), as previously proposed in Crocomo [2008]. Figure 7 has an example of a tournament.

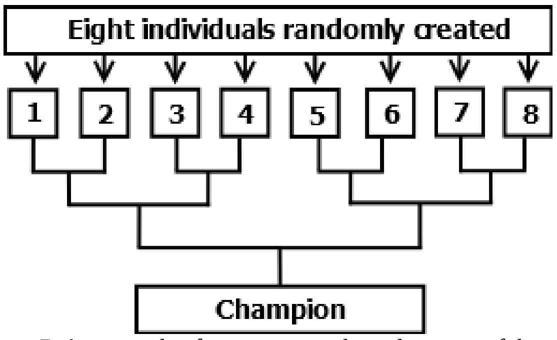


Figure 7: An example of a tournament that selects one of the four initial individuals

The individuals are evaluated playing a match against each other. At this moment, the information encoded by each individual is decoded as a script that is executed by a team of non-player characters in the game.

At the end, this individual receives a score that will be used as its fitness value by the EA. The applied fitness function is defined by expression (1).

$$F(t) = \begin{cases} 1 - t/(2 * T_{max}) & \text{if NPC won the match} \\ t/(2 * T_{max}) & \text{if NPC lost the match} \end{cases} \quad (1)$$

where:

- *F(t)*: fitness function
- *t*: time spent in the match
- T_{max}: time limit for a match.

The fitness function *F(t)* assigns scores to the winners or losers regarding the game time. A higher fitness value is associated to a winner strategy which took less time to win than a different winner strategy. On the other hand, a loser strategy which survived longer have a higher fitness value than a strategy that is quickly defeated.

The evolutionary process continues after the initial population has been created. At this step, the same evaluation process can be conducted even if the individual plays a match against the game scripts or against human players. In this case, if a match has reached the time limit (*Tmax*), the individual evaluated is considered a loser and victory is assigned to the game script or human player.

The selection, crossover and mutation operators are executed during the evolution process. The elitism strategy is applied in selection, where the individual with the highest fitness value is selected to mate with all the others and is maintained in the population for the next generation. The crossover recombines the best individual with all the other 3 individuals in the population. The new created offspring always replaces the 3 worst individuals and with the best individual that is maintained in the population, the population size is kept in 4 individuals. In the mating process, the uniform crossover is executed, which means that each gene of the new individual has a 50% chance to come from the best individual, otherwise, the gene is inherited from the other parent. Figure 8 illustrates this crossover.

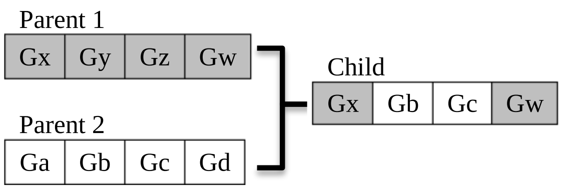


Figure 8: Example of a uniform crossover

The mutation operator is then applied to all new individuals, except the best one which is not modified. Each gene of the new individual has a small probability to be changed, given by the mutation rate. If a gene is mutated, it has one or more of its parameters modified as illustrated in Figure 9.

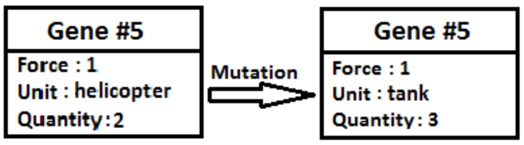


Figure 9: Example of a mutation

The loop between steps 2 and 3 takes place until a stopping criteria is achieved, that can be the number of matches or the execution time.

5. Computational Results

The computational results reported in this section evaluate the EA configurations aiming to find the best one to be used to work the AI for Bos War. The EA is set with a population of 4 individuals and a mutation rate of 10%. Each computational test is composed of 200 matches. In each match, during the evolutionary process, each individual plays against the default script of the Bos War. This default script is represented by the more balanced one, containing offensive and defensive actions.

To evaluate the EA stability, each test is repeated 10 times and all presented results (graphics and tables) are evaluated taking into account the average performance of the propose EA. The first assessment is on the size of the individual, which means evaluating the impact that the amount of actions encoded in the individual has in the EA adaptability. Figures 10, 11 and 12 compare the average values of the fitness function. The presented charts use the moving average to identify the trends regarding the fitness obtained by the used strategies. Each represented point is calculated by the average of the last 10 fitness values. The expression (1) in section 3 was also used to evaluate he performance of the default script.

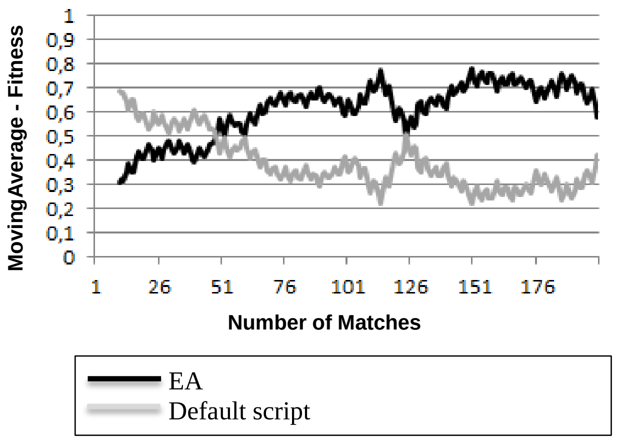


Figure 10 – Adaptability of the EA using individuals encoded with a total of 50 actions.

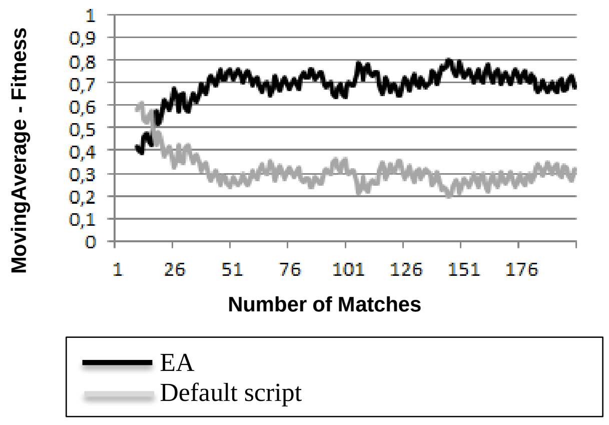


Figure 11 – Adaptability of the EA using individuals encoded with a total of 100 actions.

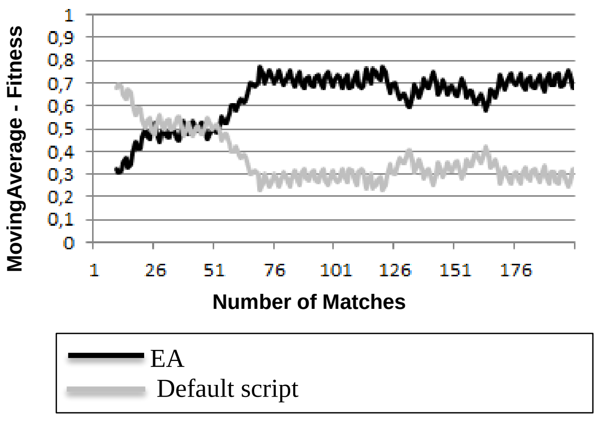


Figure 12 – Adaptability of the EA using individuals encoded with a total of 150 actions.

Figure 11 reveals that an EA containing individuals with 100 actions encoded present the best performance. The moving average values found by EA outperforms the default script before 26 matches take place. The same happens after 51 matches for individuals with size 50 and 150 (Figures 10 and 12).

The moving average of the EA is in the interval [0.7; 0.8] after 51 matches in Figure 10 and after 76 matches in Figure 12. These values are more unstable in Figure 10 where they are ranging in the interval [0.5; 0.8] after 51 matches.

Another evaluation measure used in Crocomo [2008] and Spronck et al [2006] is the average equilibrium point (AEP). It provides the number of the first match from which the method reached at least a minimum number of consecutive wins. Thus, if a method is able to reach an AEP earlier this can also indicates a better performance.

Table I compares the AEP values and the percentage of wins obtained by the EA. The values of 5 and 10 were considered as minimum number of consecutive runs to determine the AEP. The percentage of victory is calculated over 200 matches.

Table I – Percentage of wins and AEP values for EA individuals with sizes 50, 100 and 150.

Size	% Wins	AEP – 5	AEP - 10
50	58,85	54	73
100	64,5	35	71
150	62,5	64	62

The EA with individual length of 100 is, once more, the one with the better results. This configuration reached the best percentage of wins as well as the earliest AEP for 5 consecutive wins. The EA with individual length of 150 found competitive results with the best value of AEP for 10 consecutive wins. To summarize these experiments, the results reported so far indicate that EA with 100 actions encoded in the individual has the better performance.

The next test evaluated different values for mutation rate with individual length of 100 genes. For this purpose, the EA was executed with mutation rates of 5%, 10% and 20%. The same number of 200 matches and 10 repetitions were applied. Figures 13, and 14 show the moving average results using these mutation rates.

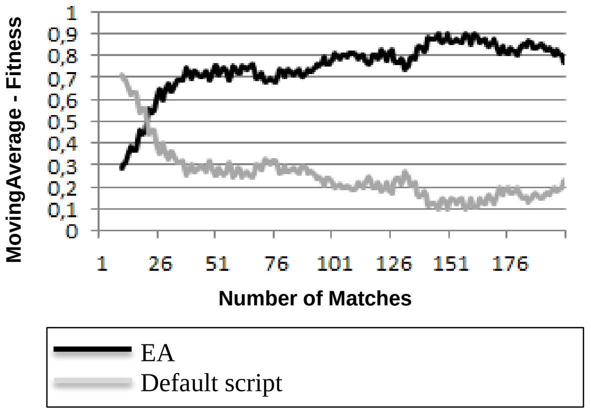


Figure 13 – EA with mutation rate of 5%

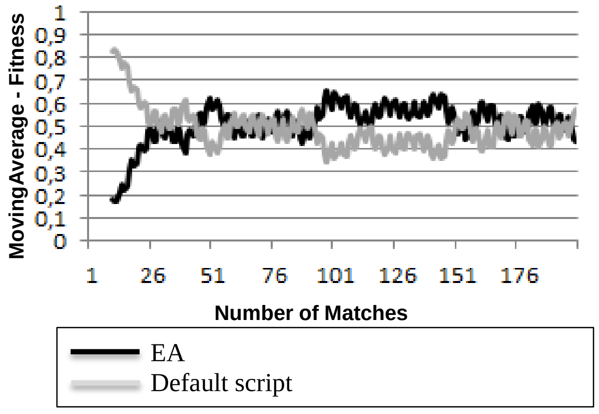


Figure 14 – EA with mutation rate of 20%

The Figure 13 and the previous Figure 10 show that EA with mutation rate of 5% and 10%, respectively, outperforms the default script before 26 matches. However, the mutation rate of 5% leads the method to reach better moving average values, in this case, on the interval [0.8;0.9]. The worst performance of the EA was obtained using the highest mutation rate (Figure 14).

Table II compares the mutation rates in terms of AEP and percentage of wins.

Table II – Percentage of wins and AEP values for EA individuals with mutation rate 5%, 10% and 20%.

Rate (%)	% Wins	AEP - 5	AEP – 10
5	72	44	68
10	64,5	35	71
20	48	51	110

While the EA with 10% mutation rate presented the better AEP with 5 consecutive wins, the EA with 5% mutation rate presented better values for the percentage of wins and improved the AEP value for 10 consecutive wins. Thus, it is possible to conclude that the best found configuration works with individuals composed by 100 actions and a mutation rate of 5%.

Next, the EA with these parameters is evaluated against the other four default scripts of the game. There are three offensive scripts called Tank Rush, Rush and Blitz. The Tank Rush emphasizes the use of several types of tanks to attack whereas Rush does the same using several types of assault units. The Blitz attacks mixing tanks, assault units and air craft units. The fourth script is called Spacious and is a defensive strategy where it is emphasized the construction of gun-turrets.

Figures 15-18 show the moving average results of the EA against each one of these scripts.

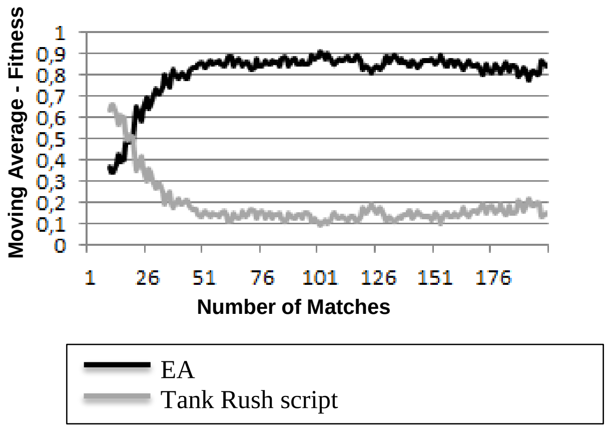


Figure 15 – EA against Tank Rush

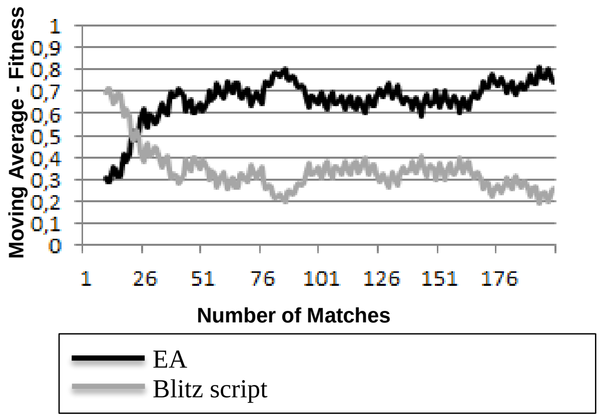


Figure 16 – EA against Blitz

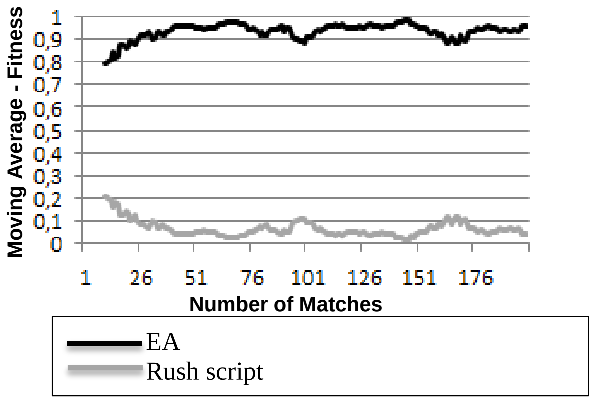


Figure 17 – EA against Rush

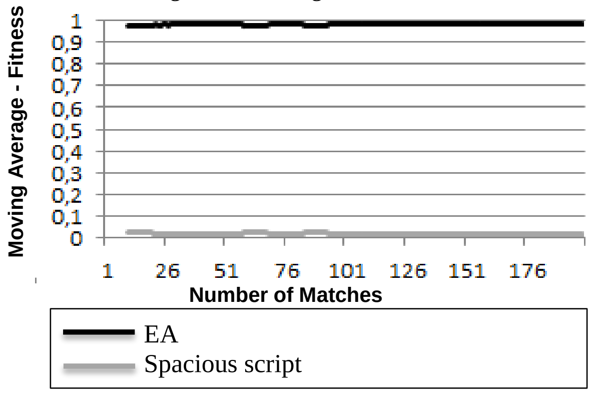


Figure 18 – EA against Spacious

The EA reaches better moving average values before 26 matches against Tank Rush and Blitz scripts. The method is better since the beginning than Rush and Spacious scripts, where the majority of average fitness values are in the interval [0.9; 1.0]. This result means that even the initial population of the EA, built with the champions of the 4 tournaments, was able to beat the Rush and the Spacious scripts.

Table III presents the percentage of wins and AEP values taking into account all scripts.

Table III – Percentage of wins and AEP values for EA playing against the five scripts of the game.

Script	% Wins	AEP - 5	AEP - 10
Default	72	44	68
Tank Rush	81,1	26	42
Rush	95	10	13
Spacious	99,9	1	1
Blitz	65,3	44	71

The proposed method wins easily when playing against the defensive strategy called Spacious. In this case, the EA won almost 100% of the matches since the first match of the game. Our method had more trouble to win against the Blitz strategy, since it alternates different type of units. However, it reached a satisfactory rate of wins (65,3%) with AEP values found after 44 and 71 matches. A similar behavior of EA occurred when playing against the default script that also alternates between defensive and offensive actions.

6. Conclusion

The presented paper reported the preliminary results found when applying an EA to produce the artificial intelligence control scripts in a real time strategy game called Bos Wars. The proposed EA was an adaptation of a similar EA approach applied previously by some of the authors in a Role Playing Game. However, a tailor-made representation of individual as well as new genetic operators were proposed and evaluated in this paper using the Bos Wars engine.

The proposed EA was first evaluated playing against one of the default scripts of the Bos Wars game. The results indicated a better performance when each individual encodes 100 actions for the resulting script. Also this representation of individuals performs better with a small mutation rate. In the next phase, the configuration found to the EA was validated playing against the other four standard scripts available in the game. The proposed method outperformed all the other game scripts.

It is possible to conclude that the EA had more difficulty to win against default scripts that mixed defensive and offensive actions, as well as the Blitz script that alternated several types of attack units. On the other hand, the method outperformed those offensive scripts that prioritize a specific type of action. Also the method had no problems to win against the standard defensive script.

As future work, we will be evaluating modifications in the individual representation, as well as different selection criteria for crossover. Also, an adaptation of the described EA approach to the Wargus game is under development

Acknowledgements

This research received financial support from Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), project grant 2012/00995-0.

References

APPOLINARIO, B. V., PEREIRA, T.L., 2007.*Navegação autônoma em jogos eletrônicos utilizando algoritmos genéticos*. Exacta, São Paulo, v.5, n.1, p.79-92.

BITTENCOURT, J. AND OSÓRIO, F., 2006. *Motores de jogos para criação de jogos digitais – gráficos, áudio, interface, rede, inteligência artificial e física*. In: Anais da V ERI-MG SBC, v. 1, 1–36.

BOS WARS ©2004-2010. Available in: http://www.boswars.org/ [Accessed May 05 2012].

CROCOMO, M. K., 2008.Um Algoritmo Evolutivo para Aprendizado On-line em jogos Eletrônicos. Proceedings of SBGames 2008: Computing Track. Available in: http://www.sbgames.org/papers/sbgames08/computing/full/ct22_08.pdf [Accessed October 02 2011].

JANG, S., YOON, J., CHO, S., 2009.*Optimal Strategy Selection of Non-Player character on Real Time Strategy Game using a Speciated Evolutionary Algorithm*. IEEE Conference on Computational Intelligence and Games (CIG'09) p. 75-79.

LUA 2012. Available in : http://www.lua.org/. [Accessed May 05 2012]

LUCAS, S.M. AND KENDALL, G. 2006, *Evolutionary Computation and Games*. IEEE Computational Intelligence Magazine., February, p.10-18.

MILLINGTON,I. G, 2006, Artificial Intelligence for Games. The Morgan Kaufmann Series in Interactive 3D Technology, Morgan Kaufamann.

PONSEN, M., SPRONCK, P., MUÑOZ-AVILA, H. AHA, D., 2007*Knowledge Acquisition for Adaptive Game AI*. Science of Computer Programming, v.4, n.1, p. 59-75.

SMITH, G., AVERY, P., HOUMANFAR, R., LOUIS, S., 2010.*Using Co-evolved RTS Opponents to Teach Spatial Tactics*. IEEE Conference on Computational Intelligence and Games (CIG'10) p. 146-153.

SPRONCK, P., PONSEN, M., SPRINKHUIZEN-KUYPER, I. ANDPOSTMA E., 2006. *Adaptive Game AI with Dynamic Scripting. Machine Learning*, Vol. 63, No. 3, pp. 217-248. (Springer DOI: 10.1007/s10994-006-6205-6)

SPRONCK, P., SPRINKHUIZEN-KUYPER, I., POSTMA, E., 2004. *Online Adaptation of Game Opponent AI with Dynamic Scripting*. International Journal of Intelligent Games and Simulation, Vol.3, No.1, ISSN: 1477-2043, pp. 45-53.

SWEETSER, P., 2002. *Current AI in games: a review*. Technical Report, School of ITEE, University of Queensland.