AN EVOLUTIONARY ALGORITHM APPLIED TO EVOLVE SCRIPT STRATEGIES FOR A RTS GAME

Claudio F.M. Toledo, Rodrigo de Freitar Pereira, Márcio da Silva Arantes,
Márcio K. Crocomo, Eduardo do Valle Simões
University of São Paulo, Institute of Mathematics and Computer Science
São Carlos, Brazil.
claudio@icmc.usp.br, rodrigofp@grad.icmc.usp.br, marcio@icmc.usp.br,
marciokc@gmail.com, simoes@icmc.usp.br

***Abstract*— The present paper proposes an Evolutionary Algorithm (EA) as Artificial Intelligence (AI) for a Real Time Strategy (RTS) game. The engine of the game Bos Wars is used as a battle system, where the EA is able to create and evolve game strategies represented by scripts coded in the LUA language. To accomplish this goal, the EA communicates with the game engine sending scripts, playing matches and capturing statistical data to evaluate its individuals. The preliminary computational results indicate a superior performance of the EA that beats Bos Wars' standard scripts.**

Keywords*— Evolutionary algorithms, Artificial intelligence, Game, Real time strategy games.

I. INTRODUCTION

The present paper applies an Evolutionary Algorithm (EA) to generate and evolve strategies for a Real Time Strategy (RTS) game called Bos Wars [1].

The Bos Wars game is about future warfare, where the goal is to destroy all the enemies. Furthermore, this game demands resource management and action planning to attack other opponents. The artificial intelligence of the game uses a sorted list of actions (script) executed by non-player characters (NPC).

There are five scripts available with different strategies, where the player can choose to play against any one of them. The proposed EA will be able to act as an alternative AI, generating and evolving scripts that control the NPC actions during the games. Thus, the player will play against different scripts at each match.

Real time adaptation of the NPCs behavior can increase the level of entertainment [2] and the natural adaptability of the EAs can allow finding different and unpredictable strategies [3].

However, the application of EAs as the game AI is not limited to make a static NPC strategy. The adaptability of EAs can lead the computer to outperform human players, forcing them to improve their game ability, making the entertainment experience better.

The authors in [4] argue that playing against adapting AI scripts, produced by the application of EAs, can improve human player abilities more than if the games are played against other humans.

In the last years, several articles report the use of EAs in computer games [5], [6], [7], [8], [9], presenting good results. As an example, in [9], EAs are used to construct highly competent players for the Reverse game. Furthermore, in [7], the authors say that the use of EAs allowed the construction of FreeCell solvers that outperform the best FreeCell solvers up to date.

EAs have also been successfully applied in RTS games [2], [4], [10], [11] and [12]. Wargus is a RTS game where an EA is applied by Ponsen et al [2]. The method acts as a learning routine where individuals are defined by states representing different possible buildings. The evolutionary process produces competent scripts before the game begins (off-line learning) and a dynamic scripting algorithm is applied during the game play (on-line learning).

The authors in [10] also use an EA to produce AI in the RTS game called Conqueror. The method makes decisions about actions that should be executed by NPCs. The presented results report a better performance of the EA.

A navy-style RTS game based on capture the flag is also developed in [4] where another EA is applied. The method defines simple tactics spatially oriented that are able to control the overall strategy of NPCs. The authors report that it was more challenging to human players to play against the EA strategies than it playing against other human players.

Another example of the application of EAs to computer games is the framework created in [11] to simulate battles amongst AI bots for the RTS game StarCraft. The proposed EA evolves the strategies executed by these bots.

A hybrid approach combining a Genetic Algorithm (GA) and a Neural Network is presented in [12]. The hybrid method creates agents for the RTS game Wargus. The created agents are capable of displaying complex and adaptive behavior.

The goal of the proposed paper is to expand the work presented in [13], which shows that it is possible to use EAs to dynamically construct and evolve script strategies for the RTS game Bos Wars.

The present paper introduces another representation of individuals as well as tailor-made crossover and mutation operators to deal with the proposed representation. The preliminary results found against standard scripts of the Bos Wars are reported.

The paper is organized as follows. The Bos Wars game is described in Section 2 and the EA is introduced in Section 3. The computational results are reported in Section 4 and conclusions follow in Section 5.

II. BOS WARS

Bos Wars is an open source RTS game containing a C++ coded engine with AI scripts coded in the LUA language [14]. The game environment is set in the future, where several battles amongst different nations can simultaneously happen. The players have to plan attacks as well as manage resources. It is possible to play against human players or against default scripts.

Each player controls a nation in a given territory on the map. The control allows selecting one or more units of their nation. These units must perform tasks such as move around the map, collect resources, or attack units of the opposing team. Thus, the player must keep collecting the resources and plan the construction of armies to attack and repel the enemies.

The resources available in Bos Wars are magma and energy that are obtained to allow creating structures and armies. Magma pumps are built on hot spots in the map to collect magma.

However, engineer teams can also be used to collect magma from rocks. Energy is obtained building a power plant, a nuclear power plant or using engineers to collect it from trees.

A total of 14 available structures can be built including vaults, power plants, aircraft factories, turrets for defense, among others. A total of 15 units are available, such as assault tanks and aircrafts.

The main objective of this game is to destroy all enemy units and structures. Figure 1 gives an overview of a battle field.

All actions executed by NPCs are previously defined in scripts that work as the AI of the game. There are three offensive scripts, one defensive, and one that is more balanced between offensive and defensive actions. The player can choose to play against one of these five scripts.

Each script has two sets of instructions. The first set is composed of instructions to be executed just once, in the beginning of the game. The second set is a loop of commands responsible to control NPCs in a second phase. The actions in this loop are executed repeatedly until the game is finished.

Figure 1: Overview of the Bos Wars game battle field.

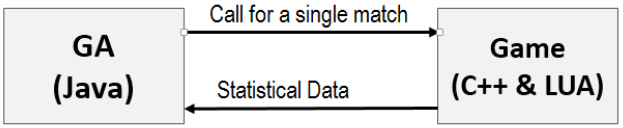

Figure 2: Communication between the EA and the game engine

The proposed EA was coded in JAVA language and is integrated with the Bos War structure as shown by Figure 2.

The EA creates and evolves several scripts, represented as individuals, which are evaluated by playing a single match. After the match, statistical data about the match are obtained and the individual of the EA is evaluated.

One notable feature of Bos Wars is the absence of noise on matches played among scripts, i.e. Bos Wars does not generate a new random seed for each match. This means that, if two or more scripts are always selected to play against each other, the result is always the same. Thus, the EA can find a winner script that will always be better against the script it is playing.

III. THE PROPOSED EVOLUTIONARY ALGORITHM

The EA proposed in this paper is responsible to create and evolve scripts that work as the AI that control construction and NPC behavior for the Bos Wars game.

Each individual will represent a script or a game strategy, where each gene is a possible action. The genes are classified as army genes or building genes that encode information about actions to create armies or actions to construct buildings, respectively.

Figure 3 presents the parameters of these two types of genes.

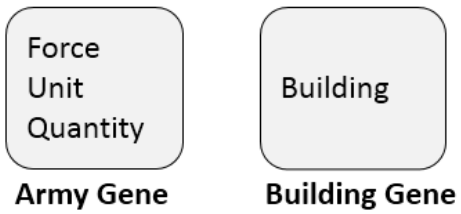

Figure 3: Representation of genes

In the army gene, it is defined what unit must be created as well as its quantity and force. The parameter force is an integer value from 0 to 9. This value is used by the EA to identify the type of army during the decode process of the gene.

The decode process is responsible to transform an individual into a game script. The building gene has the type of structures that will be built, i.e., magma pumps, vaults, power plants and aircraft factories, among others.

As explained, each script of Bos Wars is compounded by two set of instructions (actions): one set that is executed just once and another set that is executed repeatedly. The representation of the proposed individual takes this into account.

Each set of instructions is encoded separately and they can be of different sizes. Figure 4 shows a possible representation of an individual.

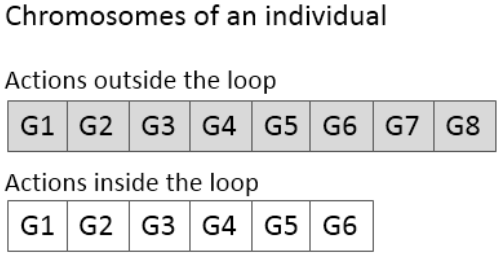

Figure 4: Representation of an individual

There are two chromosomes for each individual. The first chromosome has genes with information about actions that will be in the beginning of the script, outside of the loop. The second chromosome has the actions that will be decoded inside the loop of the game script.

The gene G1 of the upper chromosome can encode the action “Build a vehicle factory” in Figure 4, for instance, while gene G2 represents the action “Create 3 helicopters units”.

The information encoded by the proposed individual can be completely decoded into a game script. This script is executed by a NPC during a match. Thus, all of the genes from the two chromosomes of an individual are decoded into a game script in LUA language.

The pseudo code in Figure 5 summarizes the decoding process.

```
1.  begin Decoding(chromo)
2.    for i=1  to (chromo.size)
3.        if chromo[i].type = ARMY then
4.            decodeArmyGene(chromo[i])
5.        else
6.            decodeBuildingGene(chromo[i])
7.        end if
8.    end for
9.  end
```

Figure 5: Pseudo code for the decoding process

During the decode process, each chromosome is traversed from the first to the last gene, and each gene is decoded to one or more lines of a game script. The quantity of lines decoded depends on of the type of the gene and its position on the chromosome.

The so called Building Genes are decoded into one line of a script as illustrated on Figure 6 for the genes G1 and G2. In this case, it is only enough to define the type of structures that will be built.

If the gene encodes an Army, it can be decoded into one or three lines of a game script. There are some game features that need to be taking into account. For example, if a gene in position N has the same Force number than gene N+1, it is decoded into one line as shown by gene G3 in Figure 6.

If a gene N is the last gene on the chromosome, or it has a different Force number than the gene (N+1), or if the gene (N+1) is a Building Gene, gene N is decoded into three lines of a game script, as shown by the gene G4 in Figure 6. In this case, some specific actions related with the Army need to be performed. Details about these actions can be found in [1].

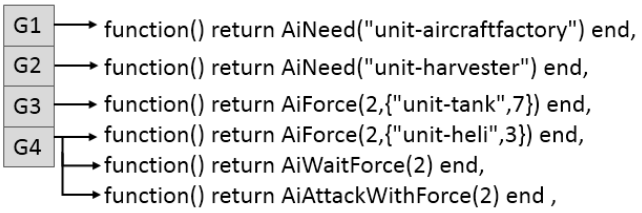

Figure 6: Example of a decoding

At the end of each match, the fitness value is determined by expression (1), where Fitness(t) is the fitness function, Tgame is the time the match took and Tmax is the time limit set for a match.

The Fitness(t) calculates a fitness value taking into account if a script won or lost the match. A higher value is assigned to winner scripts that spent a shorter time to win than other winner scripts. On the other hand, a loser script that played longer has greater fitness than scripts that quickly lost.

If the time limit (Tmax) is reached, there is no winner in the match. Thus, the individual being evaluated is assumed to be a loser and the enemy is named the winner. The idea is to consider a tie match a defeat for the EA when playing against other scripts.

$$Fitness(t)=\begin{cases} 1-\left(\dfrac{Tgame}{2*T\max}\right), & \text{Script won a match} \\ \dfrac{Tgame}{2*T\max}, & \text{Script lost a match.} \end{cases} \quad (1)$$

The pseudo code of the proposed EA is presented in Figure 7.

```
1.   begin EA(population)
2.     enemy = chooseEnemy()
3.     playMatch(pop,enemy)
4.     calculateFitness( child)
5.     while (stopping criteria not reached)
6.       for i=1 to (populationSize-1)
7.         parents ← pop.selectParents()
8.         child  ← crossover( parents )
9.         mutation( child, mRate )
10.        playMatch(child,enemy)
11.        calculateFitness( child)
12.       pop.insertion( child, counterMatch)
13.      end for
14.     pop.update()
15.    end while
16.  end
```

Figure 7: Pseudo code for the EA.

To improve the initial scripts, the method evolves a population of scripts previously selected by an off-line learning phase that is responsible to provide a better initial population to the on-line evolutionary process. Initially, two populations with the same length are randomly created.

Next, these two populations evolve through eight generations with their individuals playing against each other. If a tie match happens in this case, both individuals receive the same fitness value (0.5).

The evolutionary process executed by each population follows independently steps described in Figure 7. In this case, for each match, an individual of the other population will be selected as an enemy script to play against and evaluate an individual of the evolving population.

At the end of this initial (off-line) evolutionary process, the individuals with the best fitness value from both populations are selected to compose the initial population of the on-line evolutionary process. Thus, a learning phase can be executed before the game actually begins (off-line), in the attempt to select efficient scripts to be played against during the game.

The evolutionary process that performs matches against other players (enemy) happens in fact after the initial population has been created and improved by off-line evolution.

In this second phase, a new enemy is chosen (in line 2 of Figure 7). This enemy could be a pre-defined game script (such as any one of the five already included in Bos Wars), a script generated by the EA (as in the learning phase), or a human player.

Every individual of the initial population plays one match against that enemy (line 3), then its fitness value can be calculated (line 4). This value will be used latter by the selection method (line 7).

For the new generation, a total of population size-1 new individuals are created (line 6). A roulette wheel is applied as selection operator to determine two parents (line 7). Thus, the parents with higher fitness values have more chance to be selected.

However, an elitism strategy is also applied in this step. The individual with the highest fitness value is always selected as the first individual for both selection methods at the first time, i.e. when i=1 in line 6.

The crossover operator (line 8) is applied over the two selected parents chosen in the selection phase. Two crossovers methods were evaluated to see which would work best with the representation of individual proposed in this paper: uniform and one-point crossover. In both operators, chromosomes that encode actions outside the loop as well as chromosomes that encode actions inside the loop can only be recombined with chromosomes of the same type

In the uniform crossover, each gene of the produced individual has 50% of chance to be inherited from one of the parents.

If the chromosome of one parent is larger than the same chromosome in the other parent, the procedure follows the larger chromosome until its end, with 50% of chance of each gene to be copied to the resulting chromosome.

In the one point crossover, a random cut point is defined for each chromosome. This random value is determined taking into account the chromosome with the smallest length.

The genes before the cut point are inherited from one parent (randomly defined) and the genes after this point come from the other parent.

Figure 8 shows an example of the application of the uniform crossover.

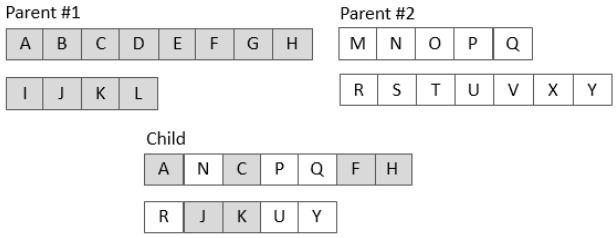

Figure 8: Example of uniform crossover operator.

In the mutation phase, each offspring has a probability, given by the mutation rate, to have one of its genes changed (line 9). If an individual is to be mutated, its two chromosomes can be modified. A total of four types of mutation operators were proposed:

- Swap: two genes are randomly selected to exchange positions with each other.
- Change: the parameters in the gene are modified.
- Removal: the gene is removed from the chromosomes.
- Addition: a new gene is created and inserted in a random position in the chromosome.

The number of mutations to be performed over the individual is randomly selected, so more than one type of mutation can be executed at the same time. The mutation is exemplified in Figure 9, where swap, insertion and remove mutations are applied in Child.

The new individuals are always inserted in the intermediate population (line12) that has a total of 2*population size – 1. After all new individuals have been inserted; the population is resized (line 14) to population size, where the worse individuals are discarded.

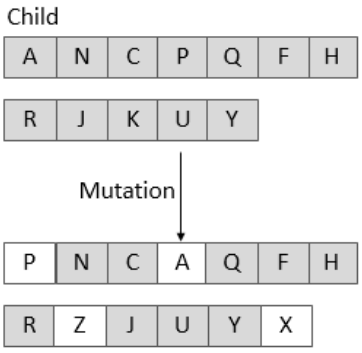

Figure 9: Example of mutation operator.

IV. Computational Results

The EA was set with a population of seven individuals and 0.5 of mutation rate. The individuals were initialized with different lengths of chromosomes. Chromosome size for actions outside the loop can range from 7 to 120 genes, while the chromosome size for actions inside the loop ranges from 7 to 25 genes. All these values were obtained based on some empirical tests previously conducted.

Each computational test reported in this section was repeated 10 times to evaluate the performance and stability of the EA. Therefore, the results reported takes into account the average performance of the method.

The first studied assessment is on the type of crossovers applied over the proposed representation of individuals. In these tests, two EAs are set to compete against each other, playing 50 matches where they control their NPC teams on Bos Wars.

One EA is set to execute uniform crossover and the other EA executes one-point crossover. The other values of EA parameters are seven of population size and 0.5 of mutation rate. The roulette wheel is applied as selection.

In this experiment, there is no off-line learning phase and both methods starts with the same initial population randomly generated. Figure 10 shows the moving average, where each point depicted in the chart is the average of the last 5 values. These values are the average fitness obtained in each match taking into account the 10 executions of the EA.

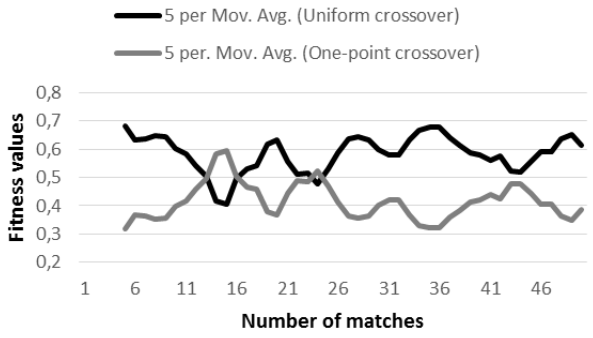

Figure 10: Uniform crossover vs. One-point crossover

It is shown in Figure 10 that uniform crossover outperformed one point crossover, when applied in the proposed representation of individuals.

The EA with uniform crossover always outperforms one-point crossover after 26 matches, and the most part of its moving average values are in the interval [0.6;0.7]. During the 50 matches, repeated 10 times for each execution, the uniform

crossover wins 57% of the matches; one-point wins 38% and 5% are tied matches.

In the next experiments, the EA was executed with 7 individuals, uniform crossover, 0.5 of mutation rate and roulette wheel selection.

The method plays now against the three most difficult standard scripts of Bos Wars. Previous work [13] showed that the strategies named as Default, Tank Rush and Blitz were harder to beat by EAs than the other two included in the game.

First, the EA was evaluated against each script separately, where the stopping criterion adopted is the first victory reached by the EA. The aim of this strategy was to evaluate how long (how many matches) it takes to the EA to generate a script able to outperform a game script.

As stated before, there is not noise in the game; therefore, this winner script evolved by the EA will always beat the game script. During each execution, the EA plays 50 matches against a game script. A total of 10 executions were performed.

Table I presents the performance of the EA, showing the minimum, maximum and average number of matches necessary to beat each one of the scripts.

Table I. Number of matches to find a winner individual fighting against each one of the game script

	Script		
	Default	**Tank Rush**	**Blitz**
Min.	1	1	1
Max.	7	27	36
Avg.	3	7	8

The results indicate that the EA needs to play a small number of matches to evolve a script able to outperform those available in Bos Wars.

The minimum number of matches necessary to outperform all three game scripts was only one match. This means that the learning phase of the method was able to produce an initial population containing individuals that already beat the scripts provided in the game.

The proposed method had no problems to beat the Default script where only 7 matches (maximum) in average were spent to generate a better script.

On the other hand, the Blitz script was harder to beat, and it took 36 matches in the worst case for the EA to win. On average, a total of 10 matches were enough for the EA to defeat the game scripts.

The next experiment aimed to simulate a human player playing against the EA scripts. If human players lose a match they usually change the strategy to counter the AI of the game. Thus, a routine was developed to simulate a player that always changes the strategy every time the EA wins.

In this routine, one of the three scripts (Default, Tank Rush and Blitz) is randomly chosen to play against the EA. If the EA loses a match, the selected game script remains playing while the EA continues generating new individuals to compete. If the EA wins the match, the current individual will be kept to play another match and a new enemy script is randomly chosen.

The goal of this experiment was to find a robust individual, which can beat the three scripts of the Bos Wars game. The stopping criterion of each test was to play 200 matches or to find an individual able to win against the three scripts.

The random choice of game scripts does not select the same script. For instance, if the EA individual wins Blitz, the next possible choice is against Default or Tank Rush. If Tank Rush is selected and the EA individual wins again, the next option to be selected is only Default. If this individual wins the Default script, than the EA execution is finished. Otherwise, a new individual is generated by the EA to play against Default and the steps explained before are repeated.

Moreover, the fitness function for this experiment was modified so that the value returned is the average of the fitness values of the three matches played, against each one of the tree scripts.

Suppose that one new individual wins two matches, but it loses the third one. In this case, its fitness value is determined as the average of the fitness value, obtained using equation 1 (in section 3), in the three matches played by this individual. Thus, this average will include the fitness value related with the two victories and one defeat.

Figure 11 shows the number of matches necessary to find this robust individual, i.e. the EA script that wins the tree game scripts sequentially.

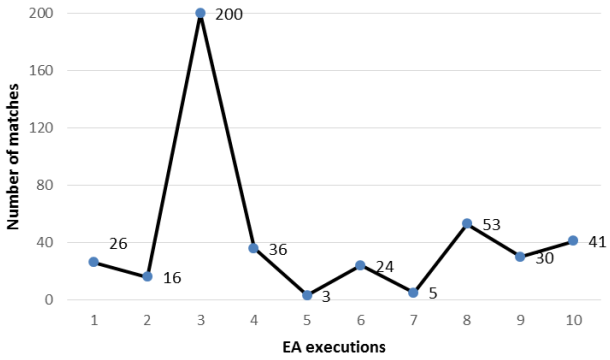

Figure 10: Number of matches to find the robust individual.

The EA found this robust individual in 9 out of 10 executions, where less than 60 matches were necessary. Table II shows the number of matches required for the EA to beat one, two and three scripts in sequence.

Table II. Number of matches to find a winner individual fighting against three scripts randomly selected

	Min.	Max.	Avg.
1 script	1	24	7
2 scripts	2	171	30
3 scripts	3	Not Found	37

The minimum number of matches in Table 2 reveals that a robust individual was already evolved in the initial population in some executions.

In this case, the initial population evolved during the learning phase already generated one individual that was able to win the three game scripts sequentially.

In the worst case, the method spent 24 matches to beat one script and it took longer to win sequentially two scripts. This situation is the third execution shown on Figure 10, where the sequence of three victories was not reached.

However, on average, the method took 7 matches to the first win, 30 matches to sequentially beat two scripts and 37 to find the winner script. The average values did not take into account the third execution results.

V. CONCLUSION

The paper presented preliminary results applying an EA to produce (evolve) scripts that act as artificial intelligence, controlling NPCs in a real time strategy game called Bos Wars. The proposed EA introduces a representation of individual with tailor-made crossover and mutation operators.

The performance of the proposed EA was first evaluated when uniform or one-point crossovers were separately applied over the proposed representation of individuals.

The results showed that uniform crossover outperformed one-point crossover. The representation of individuals encodes information about actions to be executed during the matches.

The uniform crossover was more able to exchange this information, producing more diversity and generating better individuals than one-point crossover. Thus, this method was set as crossover in following experiments.

Next, the EA was validated against the hardest standard scripts available in the Bos Wars game. First, the performance of the EA was evaluated against each game script.

The method was able to evolve a script that outperformed all the other three game scripts, spending a reduced number of matches on average.

In several experiments, the off-line learning phase of the proposed approach itself was able to generate a very capable script, able to defeat the Bos Wars ones.

This means that the method can provide AI control scripts with superior performance than those available in the game with very few generations or number of matches.

The last experiment tried to simulate the behavior of a human player, changing the strategy every time the EA wins. The main idea was to verify the EA ability to find a robust individual able to win the three scripts.

Although the method did not find this robust individual in one of the tests, in the other nine executions, a very robust individual that was able to beat all other three scripts was found in less than 60 matches.

On average, the method spent around 37 matches to win the three scripts. These results indicate the potential adaptability of the EA to strategy changes. The method seems to learn quickly from its defeats, evolving the scripts even when the opponent strategy changes.

As future work, the EA will be evaluated against human players. An adaptation of the proposed EA to the Wargus game is also under development.

VI. ACKNOLEDGEMENTS

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

VII. REFERENCES

[1] BOS WARS ©2004-2010. DOI= http://www.boswars.org/.

[2] 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.

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

[4] 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.

[5] 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.

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

[7] ELYASAF, A.; HAUPTMAN, A.; SIPPER, M. 2012; , Evolutionary Design of FreeCell Solvers, *Computational Intelligence and AI in Games, IEEE Transactions on , vol.4, no.4, (Dec. 2012), 270-281*. DOI= 10.1109/TCIAIG.2012.2210423

[8] BRANDSTETTER, M. and AHMADI, S. 2012. Reactive Control of Ms. Pac Man using Information Retrieval based on Genetic

Programming. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2012*. Granada, Spain, September 11 - 14, 2012). IEEE, USA, 250-256.

[9] BENBASSAT, A. AND MOSHE, S. 2012. Evolving Both Search and Strategy for Reversi Players using Genetic Programming. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2012*. Granada, Spain, September 11 - 14, 2012). IEEE, USA, 250-256.

[10] 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.

[11] OTHMAN, N., DECRAENE, J., CAI, W., LOW, M.Y.H., GOUAILLARD, A. 2012. Simulation-based Optimization of StarCraft Tactical AI through Evolutionary Computation. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2012.* Granada, Spain, September 11 - 14, 2012). IEEE, USA, 394-401.

[12] TRAISH, J. AND TULIP, J.. 2012. Towards Adaptive Online RTS AI with NEAT. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2012*. Granada, Spain, September 11 - 14, 2012). IEEE, USA, 430-437.

[13] PEREIRA, R. F.; TOLEDO, C. F. M.; CROCOMO, M. K.; SIMÕES, E. V. An Evolutionary Algorithm Approach for A Real Time Strategy Game. In: *Proceedings of SBGames*, Brasília, Brazil, November 2-4, 2012. 56-63

[14] .LUA 2012. DOI= http://www.lua.org/.