Coevolutionary Procedural Generation of Battle Formations in Massively Multiplayer Online Strategy Games

André Siqueira Ruela
Graduate Program in Electrical Engineering
Federal University of Minas Gerais
Av. Antônio Carlos 6627, Belo Horizonte, MG, Brazil

Frederico Gadelha Guimarães
Department of Electrical Engineering
Federal University of Minas Gerais
Av. Antônio Carlos 6627, Belo Horizonte, MG, Brazil

Abstract

This paper presents a coevolutionary genetic algorithm for the development of battle formations in Massively Multiplayer Online Real-Time Strategy games. We consider the context of the game Call of Roma, where the battles are turn based and two sides fight each other, each side involving one or more players. The coevolutionary genetic algorithm takes as input a predetermined battle formation, presented by the defense side, and returns a battle formation adapted for the Attack side. The algorithm aims to maximize the battle performance, which is a subjective concept that varies from player to player. The individual encodes the various characteristics of the fighting heroes. We tested the algorithm over a cloud-computing platform, considering test cases modeled by several active players. The results illustrate that the proposed algorithm is able to find a victorious solution for the Attack team, even when it is under unfavorable conditions.

Keywords: Massively Multiplayer Online Real-time Strategy, Coevolutionary Algorithm, Procedural Content Generation, Game Design

Author's Contact:

andre.siqueira.ruela@gmail.com, fredericoguimaraes@ufmg.br

1 Introduction

Massively Multiplayer Online Real-Time Strategy (MMORTS) games have attracted the attention of millions of users in many countries around the world. MMORTS is a kind of game that involves real-time strategy (RTS) with a massive number of simultaneous players on the Internet. As indicated in a study available online [Yee 2005] these players spend 21 hours a week on average in online games. Such amount of time spent in virtual worlds can lead to different socializing experiences [Canossa et al. 2013]. This possibility of interacting with such a high number of players is one of the main factors that might explain the success of these kind of games. For the companies that maintain MMO games, a massive number of active players is an important source of income. To give a perspective, only in the United States, the market of free-to-play online games category totalled U$2,893 million in sales in 2013, up from U$1,991 million in 2012 (+45%) [SuperData Research 2014].

In MMORTS, players can either fight each other in PvP (Player versus Player) battles or engage artificial players, known as NPC (Non-Player Character), in PvE (Player versus Environment) battles; they can also take different missions which are integrated to the story of the game. Since such NPCs have predefined, static battle strategies, more experienced players find no challenge in engaging in PvEs. After some time, the game becomes tedious for these experienced players, who can easily defeat NPCs. Because of this, the evasion of players in the server increases. With few active players remaining in the server, the possibility of interactions reduces, affecting the income of the company that hosts the game.

One idea to overcome this problem is create attractive balanced content for PvE battles. If NPCs present more dynamic, efficient and unanticipated battle formations in PvE, players will have to put more effort in the preparation for the battles, and more experienced players can still find stimulating battles to fight. Such characteristics are important to maintain their interest in the game and to reduce the evasion of players in the server.

Another common source of players' complaint is the handicraft design of unbalanced game content. Sometimes the game designers fail to produce new content for the game and generate bad and unbalanced content that affects the game play. The use of computational intelligence algorithms to help in the game design can potentially produce a better content with lower efforts. This has been one of the central ideas of the Procedural Content Generation (PCG) research [Smith 2014; Shaker et al. 2014]. In recent years, PCG has been receiving attention from both academic researchers and industrial developers, for its potential to turn computers into creators of massive, varied worlds for players to explore, at a fraction of the cost of human authorship [Tozour and Champandard 2012].

It was pointed by [Togelius et al. 2011a; Togelius et al. 2011b] that PCG differs from the *mainstream* AI game research. Togelius defined that the development of agent behavior, or the intelligence behind a robot mind, which is very common in RTS research, is not PCG. In other words, the agent behavior is not the content in the scope of the PCG. By this way, this work consists in a *search-based* PCG, featuring cooperative coevolution, where the content is a team of the evolved battle formations. In this case, there is no agent behavior or robot AI controlling the actions of the NPCs, but just the generation of content that influences the battle performance. This content will be better explained in section 3.

In this paper, we employ coevolutionary algorithms to implement our ideas. In [Togelius et al. 2014], the authors argue that artificial intelligence and machine learning techniques have the potential to increase the longevity of electronic games and reduce its production costs. Heuristic techniques such as evolutionary techniques have been recognized as a promising approach to generate strategies for many different games [Benbassat and Sipper 2012; Stanley et al. 2005; Othman et al. 2012]. The complexity of handcrafting strategies for artificial players makes the evolutionary process very appealing, since useful strategies can be generated and discovered automatically. Strategies generated by evolutionary algorithms can be competitive to human strategies, see [Avery and Michalewicz 2008; Miles et al. 2004]. In this way, high level NPCs can be generated by a coevolutionary process as an additional tool to reduce the evasion of users from the server, giving them motivation to keep playing and investing in their accounts.

In our previous work [Ruela and Guimarães 2012] a Genetic Algorithm (GA) was proposed for finding a battle formation for a single hero. That work was taken as a starting point, but it is very outdated, due to the changes in the game and in the way we handle the problem. Since then, many improvements have been achieved, like the storage of players' modeled heroes, for being used as a standard benchmark and the multithreaded implementation allowing a better use of computational resources from cloud computing web services. Additionally, this work proposed a new evaluation approach based on five measurements, considering different perspectives of what is a good battle performance. This evaluation can be extended to other games, since it is based only on the number of soldiers in both input and output. Finally, this work also evolves an entire team, while the previous one handled just a single hero.

The next section introduces the context of this work, detailing the battle system of the bench-marked game. Section 3 presents our proposed approach. It is detailed how to encode the solutions and the ideas behind the evaluation process. Section 4 shows the base steps to extend this work to other games. Section 5 shows the results of our method and discusses about the effect of the design of the evaluation function. Finally, section 6 presents the conclusions and points towards some possible interesting future works.

2 Context and Problem Definition

In this work we consider the context of the game *Call of Roma* [1], formerly known as *Caesary*. Call of Roma is based on the history of the Roman Empire, the post republic phase of the ancient roman civilization. Players can build an empire, exploit resources, organize troops and fight enemies. The game was developed by Heroic Era and is similar to *Evony*[2], *Maegica*[3] and *Senatry*[4]. Although considering a specific context, the model proposed in this paper can be extended to several MMORTS games, since it shares common features of the battle system of these games, which are based on basic principles of role-playing games. It is important to understand that MMORTS games differs from classic RTS games in various aspects, and the use of well-known benchmark and frameworks, like [Senior 2010], does not apply here.

Units are divided into frontal and rear units. Frontal units engage in close combat, while rear units fight at distance. Among the different unit attributes, in this paper we consider only those that affect the performance of the soldiers during the battle, since the effects of the other attributes are beyond the scope of this research. The attributes considered are: *Offense* (Off), the capacity of attacking opponents; *Defense* (Def), the capacity of defending from the enemies' attacks; *Damage* (Dmg), indicating the damage of an attack performed by the unit; *Life* or *Hit Points* (HP), related to the capacity of absorbing the damage inflicted by enemies. The damage of the units is uniformly distributed within a given interval, introducing some degree of randomness to the battles.

The 3 frontal units are: *Hastatus*, consisting in one soldier with light armor, low lethality, performing close range combat with a spear; *Equites*, consisting in a mounted soldier, capable of flanking and ambushing rear units; *Principes*, the heavy roman infantry, with high production cost, but high armor and lethality. The 3 rear units are: *Sagittarius*, consisting in an archer with light armor, a bow and arrows; *Ballistae*, consisting in medium war machines that throws darts on the enemies; *Onagers*, heavy catapults that throws lethal fireballs over the opponent. When the frontal units are killed, the rear units become incapable of fighting at distance and they engage in close combat, with a penalized damage (50% reduction). The Table 1 shows the base attributes of the evaluated units in this work. Moreover, all units have special skills, for example the Sagittarius' *Dispersion*, that spread arrows over all divisions of the opponent hero with reduced damage, but for convenience and lack of space, all units' details are not described in this paper.

Table 1: *Units*

Units	Attributes				
	Dmg	Off	HP	Def	Sol
Hastatus	3 - 6	8	60	8	1
Equites	20 - 25	16	250	16	3
Principes	40 - 60	19	400	16	6
Sagittarius	3 - 5	9	45	5	1
Ballistae	12 - 15	15	100	12	4
Onager	50 - 50	32	600	13	8

Units are sent to a battle under the leadership of a *Hero*. Heroes have 3 traits that influence their performance in a battle: *Sway*, *Bravery*, and *Parry*. Sway is related to the leadership *Faculty* of the hero. The higher this faculty the higher the number of soldiers that can be allocated to this hero. The column Sol in Table 1 refers to the number of soldiers occupied by a single unit. Bravery affects the offensive (Off) performance of the units, increasing their damage (Dmg) in combat. Parry affects the defensive (Def) performance of the units, reducing casualties in combat. Units have fixed values for their attributes but the values for the heroes' traits can be chosen by the user according to his/her strategies.

[1]Call of Roma: `http://www.callofroma.com/index.do`
[2]Evony: `http://www.evony.com/`
[3]Maegica: `http://maegica.browsergamez.com/`
[4]Senatry: `http://www.senatrywar.com/`

Users can assign integer values to each attribute under the constraint that the highest value cannot surpass the sum of the other two. Each hero has a total amount of points to be distributed among the attributes. The leadership faculty of a hero can be assigned by the formula:

$$faculty = 30000 + 500 \times Sway$$

Thus, a single hero can lead hundreds of thousands of soldiers. Usually, this number of soldiers (or faculty value) lies between 460,000 and 670,000. In this work, we assume that each hero has 969 Unassigned Trait Points (UTPs). In the real game, these point are received every time a hero reach a new level, and the number of points received depends on the hero's characteristic. Additionally, there is a random number of base starting points. To avoid this bias, in this work all heroes are equal, with the same base number of points, same potential and the same 969 UTPs.

Heroes can be equipped with five different types of basic equipment, which are boots, shield, helmet, armor, weapon; and other five types of accessories, which are necklace, pendant, belt, ring and a horse. The five types of accessories have levels that vary from Lv1 to Lv9 and can be improved by players. Each equipment gives a special bonus to all soldiers led by the hero. In this paper, we consider seven of the most important basic sets of equipment in Call of Roma: Saturn, Fearlessness, Hard-Core, Mars, Bright, Green and White; and the four available accessory sets: Loyal, Holly, Glory and Green. The Table 2 shows the sum of the bonus values of these equipment sets. For convenience, only the values of the ninth level of the accessories are shown.

Equipment can also give extra special skills to an unit, for example the *Rain of Arrows* (RoA). The RoA ability allows the heroes' Sagittarii to cast Dispersion over the entire battlefield, raising drastically the power of those long ranged units. Players can build any combination of equipment, but only one per type, i.e., they can have any mixing of sets, but not two boots or two swords, for example. It looks simple, but considering the nine levels of each accessory set, there are more than 10^{12} ($7^5 \times 36^5$) possible different combinations. It is easy to see that there is a clear dominance relationship between the accessories sets, so that it can be said that the Green set is better than the Glory set. This behavior is not so clear for the basic equipment sets. The White set has more damage than the Saturn set, but the Saturn set has more life. There are fewer exceptions, like White and Bright that can dominate few others.

Table 2: *Sets of equipment considered in the work: basic and accessory types*

Set	Attributes				Skill
	Dmg	Off	HP	Def	
Basic Sets					
Saturn	25	45	325	11	RoA
Fearlessness	43	78	210	58	Testudo
Hard-Core	60	44	150	74	—
Mars	25	75	225	60	RoA
Bright	50	105	255	70	Roa, Testudo
Green	67	55	247	38	—
White	65	75	235	94	—
Accessory Sets					
Loyal	0	30	100	20	—
Holly	0	39	115	39	—
Glory	0	49	135	48	—
Green	0	61	161	65	—

The battle system in Call of Roma tries to simulate the way battles were fought in the time of Roman civilization. In the battles, only two sides face each other, the *Attack* and *Defense*. Each side take turns to attack the opponent. The units killed in battle are removed from the hero's divisions at the end of each turn. There can be at most 3 heroes in action simultaneously, assuming the positions left flank, center and right flank. The first hero sent to the battle takes the left flank, the second hero, in turn, assumes the center and, finally, the third hero takes the right flank. If there is a fourth hero, or even more, these are arranged in a replacement queue outside the battlefield, in the order they were sent. Queued heroes only enter into combat by replacing a defeated hero in the next turn, maintaining the order of placement. Figure 1 illustrates the battle formation and frontal and rear divisions.

Figure 1: *The battle system in Call of Roma*

Figure 2: *A sample hero*

For the allocation of soldiers, there are 6 different divisions placed side by side, 3 frontal divisions (left d_1, center d_2 and right d_3) and 3 rear divisions (left d_4, center d_5 and right d_6). Players are free to organize their divisions into formations. Figure 2 illustrates a sample hero. The dark-red text resumes this section and introduces the codification of the evolutionary algorithm, which will be explained in the next section. As can be seen in Figure 2, the frontal divisions are filled with Hastatus, while only one rear division, d_6, is filled with Sagittarius. The divisions d_4 and d_5 are empty.

Other factors influence the performance of soldiers in combat. For instance the level of research in the academy of the city where the heroes originated from. The effect of these factors were implicitly taken into account in this work, however they are not included as variables of the solution. Therefore, in the conception of a battle formation, a player must consider which equipment are more suitable for the units, how the UTPs are going to be distributed among the attributes of the hero, and the disposition of the units in frontal and rear divisions.

The goal of this paper is to propose a computational intelligence algorithm able to generate a winning and efficient battle formation, considering all the relevant factors. This is a complex combinatorial problem, which would require a heuristic approach. Among heuristic methods, we have selected GAs [Goldberg 1989], which are methods inspired by the adaptation principle of the Darwinian Natural Selection Theory [Darwin and Huxley 2003] to evolve candidate solutions to the problem.

3 Proposed Approach

GAs belong to a family of methods inspired by nature, based on the principles of evolution by natural selection. They are very general and high level heuristics, applicable to a wide range of real-world problems. Individuals in the population of the GA represent candidate solutions that compete for reproduction and survival. In biology, [Ricklefs 2008] defines *Coevolution* as a series of reciprocal responses between populations. In general, two or more species that have a close ecological relationship evolve together, such that each species adapts to the evolution of the other. This section describes our proposed approach, which consists in a Cooperative Coevolutionary Algorithm.

There are N parallel GAs, where N is the number of heroes in the enemy team. Each GA has its own independent population, mostly called here as subpopulation, and its own evolutionary cycle. The parallel GAs run asynchronously until they achieve the cooperation interval, when they share information and exchange individuals between each other.

3.1 Codification

In the problem considered in this paper, an individual codes for a particular battle formation (a single hero) in the game Call of Roma. In Figure 2 the red text indicates each element present in the codification. Given the factors that influence the battle, the proposed codification employs three "chromosomes", E, A and D. To ensure the fully comprehension of the proposed encoding, Table 3 shows the genotype-phenotype mapping adopted in this work. The union of those three chromosomes, as EAD, forms the complete individual, which codes a single hero. The three chromosomes can be described as follows:

(i) Chromosome E codes for the 10 types of equipment used by the hero. Each gene in this chromosome represents an integer index to a corresponding equipment. In Table 3, such genes are described from e_1 to e_{10};

(ii) Chromosome A codes for the distribution of the UTPs among the attributes of the hero. Each gene in this chromosome is an integer value directly representing the value of a corresponding hero trait. Once the hero trait is an integer value, there is no need for genotype-phenotype mapping function for the chromosome A. In Table 3, such genes are described from a_1 to a_3;

(iii) Chromosome D codes for the organization of soldiers in the 6 divisions. The values available in the D chromosome were normalized to an integer percentage scale, since the faculty of a hero varies with its Sway. Therefore, the sum of the values of all elements present in the D chromosome should be equivalent to 100 points. In Table 3, such genes are described from d_1 to d_6;

The algorithm receives as input a team of N battle formations for the Defense side. Individuals of the GA represent formations for the Attack side. To evolve those formations, an N-population coevolutionary GA was used. The number of subpopulations (N) is equal to the number of enemy heroes present in the input team. Each subpopulation evolves battle formations for particular placements in the battlefield. For example, if there are 3 heroes in the defense side, the algorithm instantiates 3 subpopulations. Heroes that are placed at the left flank of the battlefield form the first subpopulation, heroes placed at the center form the second subpopulation, and heroes at the right flank, following the battle system idea explained in the previous section, form the third subpopulation. Thus, a complete solution for this problem is obtained by grouping one individual from each subpopulation, in their respective order, forming a team of heroes for the Attack side.

3.2 Evaluation

The evaluation of an individual is done by using the core of the Armageddon Battle Simulator (ABS)[5], developed by the first author based on information available at *Quest Unlocked*[6] and the official Call of Roma website[1]. It is important to highlight that the ABS was used in this work just as a tool to simulate battles of Call of Roma, and it is not essential to the proposed approach, so it can be easily replaced. Although most of the information needed to implement a simulator of Call of Roma are available online, some essential equations for the battle system are not provided. More specifically, it is not clear how the final damage is calculated. Even with the analysis of several battle replays, such hidden equations were not obtained with a good precision.

[5]Armageddon Battle Simulator: `http://armageddonbattlesimulator.blogspot.com`

[6]Quest Unlocked: `http://guides.questunlocked.com/caesary`

Table 3: *A hero encoding*

E - Equipment										A - Traits			D - Divisions					
Boots	Shield	Helmet	Armour	Weapon	Necklace	Pendant	Belt	Ring	Horse	Sway	Bravery	Parry	Frontal Left	Frontal Center	Frontal Right	Rear Left	Rear Center	Rear Right
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	a_1	a_2	a_3	d_1	d_2	d_3	d_4	d_5	d_6

To overcome this problem, an Artificial Neural Network (ANN) was used to emulate the hidden equations. The neural network was developed with the support of the *Encog*[7] framework [Heaton 2008] and it was trained over more than one thousand battle replays, obtained directly from the game. There is no secret behind this particular ANN. It consists in a simple feedforward multilayer perceptrons, trained by the resilient back-propagation algorithm, in a supervised training [Igel and Hüsken 2003]. The ANN receives as input the Offense of the attacking unit, the Bravery of the attacking hero, the Defense of the defending unit and the Parry of the defending hero. Given these four input values, the ANN returns the damage inflicted by the attacking unit. Hundreds of active players have tested the Armageddon Battle Simulator with a high approval rate. Once the details about the development of the battle simulator or its neural network are not in the scope of this paper, interested readers may access [5] for further information.

Designing the evaluation function is not such a trivial task. The task of the coevolutionary algorithm is to improve the *battle performance* over the generations, finding a *better* team, if it is possible. But the ideas over battle performance varies from player to player. The algorithm attempts to maximize the fitness value of the individual. The fitness value represents the in-battle performance of the whole team (N heroes).

To illustrate the different perspectives of a good battle performance, let's suppose a hypothetical situation. Imagine a battle of 10 attacking heroes against 3 defending heroes. The defense is outnumbered and is defeated, but it killed 9 of the attacking heroes. Once the victory is given to the last standing side, the attacking team is victorious, but it lost 3 times more than the opponents did. There is no agreement among the players about what is in fact a good battle performance, and hence it is a subjective concept. Some players prefer to conquer the victory, no matter the casualties. So, in this case, if the Attack won, they are better. Others are more interested in killing the maximum number of opponent's soldiers, winning or not. In this case, the Defense are the real killer, and so they are better. The proposed fitness evaluation tries to gather in a single function all the different views of what is a good performance. Thus, the capacity of the individual to adapt to the environment is directly related to its performance in the battle simulation against a given test case.

In our approach, the complete solution is a team of N heroes, and the fitness of the whole team is the sum of the fitness of each hero. The fitness value of a single hero is the sum of five different individual *scores*. The individual scores are calculated for each particular hero in such team. In other words, each hero has five different factors that compounds its *internal* fitness. The sum of the internal fitness of every hero present in the evaluated team, results in the *external* fitness. More details about this internal and external fitness will be described in subsection 3.3. Following, we introduce the five scores used to evaluate a single hero.

Victory Point: The first score is the Victory Point, which is 1 if victory is achieved or 0 otherwise. The idea behind this score is to push the algorithm towards the victory and also avoid the optimization over defeated individuals. By creating this new level on the fitness landscape, defeated individuals will have less chance for survival and reproduction, in comparison with the victorious ones.

[7]Encog Framework: `http://www.heatonresearch.com/encog`

Offensiveness: The second score is called offensiveness and it measures the lethality of the soldiers. In the simulation, the number of soldiers killed by each side is counted. The offensiveness of a given hero is the total number of soldiers killed, S_{killed}, by this hero, divided by the initial number of soldiers led by him, which is his faculty as described in section 2 (so it is $S_{killed}/faculty$). By this way, the offensiveness may vary from 0 to values greater than 1. In a $N \times N$ battle, scores greater than 2 are a good sign of victory and it is possible to score values over N.

Defensiveness: The third score is called defensiveness and it is just the ratio of remaining soldiers after the battle, varying from 0, which means that the hero was killed, to 1, meaning that the hero returned untouched. There is only one way to have a defensiveness score higher than 0, which is winning. So, by trying to maximize defensiveness, the evolutionary algorithm is also trying to win.

Usage: It is possible to win using less heroes than N, for $N \geq 4$. Let us suppose $N = 5$, so it is a 5×5 battle. There are initially 3 heroes in the battlefield and 2 queued. It is possible to the first 3 heroes to defeat the 5 opponents, so the queued heroes will not even join the battlefield. This is the concept of the fourth measurement, called usage score. Each unused hero gives and extra $1/N$ score to the N heroes, including himself, i.e., 1 point to the team divided equally between all members. For a particular hero, it does not matter if it is being used or not. In the previous example, there are 5 total heroes, 3 used and 2 unused heroes, so it is $2/5$ for each one.

Before introducing the fifth score, it is important to reinforce the idea that the concept of a good battle performance depends on the point of view. There are situations that it seems to be impossible to find a victorious solution. In this case, the victory, defensiveness and usage scores are exactly zero. The algorithm then attempts to maximize the offensiveness by reducing the number of led soldiers, i.e., reducing the faculty, which means reducing the hero's Sway. And reducing the Sway implies increasing the Bravery and/or Parry, in such a way that the minimum Sway implies the maximum of both Bravery and Parry, as described in section 2. And so, both maximized Bravery and Parry mean the best possible offensive and defensive bonus granted by the hero's traits, but granted to the as lower as possible number of soldiers. It looks like the algorithm is evolving heroes that are avoiding the battle and this can be seen as a bad behavior, but it is not at all. Remember the 10 versus 3 example and the different perspectives of good performance.

Participation: Nevertheless, to appease those who see the defeat always as a bad thing, the fifth score, called participation score, can replace the offensiveness score. This last score is the ratio of killed soldiers in all the opposing team, which is the total number of soldiers killed by this hero, divided by the average number of initial soldiers in all enemies. By this way, the participation score may vary from 0 to N. An offensiveness score of 2 means that each soldier killed, on average, two other soldiers, while a participation score of 2 means that the considered hero killed, on average, two other enemies' heroes. Despite the idea of replacement, in this work, we consider both measures and so the fitness value of a single hero is the sum of all five scores.

However, there are N heroes being evolved and the team fitness (external) value consists in the sum of the individual fitness (internal) of all N team members. The external fitness is what determines the survival chances of the candidate solutions.

3.3 Cooperation

In a cooperative coevolutionary algorithm, the individuals from the subpopulations are taken together to produce solutions for the global problem. In our case, solutions that encode battle formations are gathered to form the Attack team. This cooperative time is called cooperation interval. Each subpopulation evolves its own individuals in an asynchronous, parallel, and independent way, until the cooperation interval is achieved. The time of the cooperation interval is determined by the cooperation gap parameter, $\gamma = 5$. This means that in every elapsed γ generations, a subpopulation thread stops its execution and waits to perform the cooperation with the others. When the cooperation interval is finished, all threads run independently for more γ generations.

Inside the subpopulation, each particular individual is evaluated by placing it in its respective battle position among the other ones. Each individual is teamed up with the best solution found so far from the other subpopulations. If an improvement is achieved in the team fitness, the evaluated individual is stored as the representative of its corresponding subpopulation. The external fitness assessment and the update of the best team are done in every cooperation interval. If there are more than one subpopulation presenting a new improvement, the best representative, considering the team fitness, is tested first. Then, the second is tested and so on. If a real external fitness improvement occurs, the corresponding representative is updated and the best attack side team is stored. In the next generation, the best-updated representative hero of a subpopulation is the corresponding hero present in the best-updated attack side team.

This best first cooperation strategy is not a good approach for the first generations and may delay the process of coevolution. Once the first individuals are randomly generated and the fitness assessment is performed considering the best heroes found in the previous cooperation intervals, the representatives of the first generations consist in individuals that evolved in combat while forming teams with random ones. So the information about what is in fact a good battle formation is vague at this point. To speed-up the coevolution, a number n_α of other cooperations are performed by randomly picking individuals from the subpopulation to form the attacking team. It is considered an α factor, $0 \leq \alpha \leq 1$, as input, which is the probability of picking the best hero from each subpopulation. So, α acts as a *greedy factor*, such that when $\alpha = 0$ all heroes of the attack team are randomly chosen from their current subpopulations, when $\alpha = 1$ all heroes of the attack team correspond to the current best individual in each subpopulation, and intermediate values of α promote a balance between random and greedy pickings. Intermediate values of α would be very interesting because a greedy picking may lead the algorithm to a premature convergence, while a random picking leads to a very unstable and inefficient coevolution. The value of n_α is defined as:

$$n_\alpha = \texttt{Max}\,(1, 50 - (\frac{gen}{2}))\,,$$

where gen is the current generation. So The value of n_α reduces over the generations from 50 to 1. The α value starts in 0 and for every random picking, it is increased by $1/n_\alpha$. The idea behind these formulas is simple. The presence of randomly selected individuals tends to be more effective in earlier generations, while in the later stages of the algorithm, this approach tends to yield no gain. Thus, in earlier generations, more α-based cooperations are performed.

3.4 Genetic Operators

The initial subpopulations contain $\mu = 100$ individuals, generated randomly according to a uniform distribution. Individuals are selected for reproduction by means of a binary tournament. In a binary tournament, two candidate individuals are randomly selected from the population. Then, their fitness are compared and the individual with the best fitness is selected for mating (or reproduction). The tournament continues until a total of 100 (μ) parents are selected for matting.

Selected individuals undergo crossover with recombination rate $\rho_r = 0.9$. Recombination takes two parents and produces two offspring. Remember that each individual has three chromosomes, E, A and D. Each chromosome is divided by one randomly selected cutting point, and the genetic information are permuted only between chromosomes of the same type. There is no recombination between chromosomes of different types, like E with D. In the event of generating an invalid solution by crossover, a simple repair operator fixes the individual, keeping the rules and the proper encoding described in sections 2 and 3.1.

After recombination, the offspring is subject to the mutation operators. The idea of the mutation operators is to help the algorithm to avoid local minima and premature convergence. We employ different mutation operators, suitable for each chromosome. The following mutation rates correspond to the probability of each gene, in the respective chromosome, being affected. It may seem that the low mutation rate has no effect on the evolutionary process, but this thought is wrong. This rate should be lower enough to promote small perturbations over the individuals, and not a random search.

The chromosome E suffers mutation with rate $\rho_m^E = 0.01$. Mutation operator ME replaces one element in the chromosome by another random equipment. Chromosome A has mutation rate $\rho_m^A = 0.02$ and a mutation operator is applied randomly. Mutation MA1 performs a simple swap between two genes and Mutation MA2 adds a random perturbation to the value of one of the genes subtracting the same value from another gene in A such that the limit of 969 of total points and the limit of 484 points per trait, are not violated. The chromosome D has mutation rate $\rho_m^D = 0.04$ with three mutation operators chosen randomly. Mutation MD1 is a simple swap, like MA1; Mutation MD2 selects two divisions $d_i \neq d_j$ and randomly allocates some points from d_i to d_j keeping the sum equal to 100; Mutation MD3 removes all points from a randomly chosen division and add it to the biggest division, concentrating soldiers into one division. This operator was modeled based on the author's intuition and expertise on this particular game. In other words, the goal of this operator is to reduce the losses caused by the Dispersion from opponent Sagittarii. In fact, a reduction on the number of divisions implies a reduction on the number of Dispersion's targets, and this might lead to a better performance. It can be argued that MD3 is not a "mutation" operator and it is more likely to operate as a "refinement" operator. But refinement operators are likely to accept changes only if an improvement is devised, which is not the case of MD3. We decided to avoid such discussion. Also, earlier experimental results on our previous work showed that the MD3 operator brought a significant improvement to the algorithm.

The offspring replace the current population, characterizing a generational GA. The algorithm stops after 200 generations or when there is no improvement in elapsed 50 generations. For more details about evolutionary algorithms, in general, see [Jansen 2013].

4 Generalization

It can be argued that this approach is very specific or *ad hoc*. In fact, it considers the battle system of just one game, called Call of Roma. But the principles presented here are common to a large number of games. It consists in a war game, fought by a large number of *units*, under the leadership of a small number of *heroes*. Both units and heroes have properties or *attributes* that are elementary to the most RTS games and also Role Playing Game (RPG) styles, such as hit-points, armor or defense, damage, attack, etc. In this paper it was considered the heroes' traits, but these characteristics are not specific. Usually, it must be decided how to *build* this attributes according to a strategy or playing style. Sometimes, in more complex games, this building process is not trivial.

In Call of Roma, there are also a large number of collectable *equipment*. The capacity of collecting items from the environment and using it to improve the player performance in battle is another well-known characteristic from strategy and RPG based games. In general, there are a lot of items to collect, and a very large number of possibilities of how to combine, use or equip these items, according to the build or the strategy. In our case, even restricting the number of equipment to seven basic sets, there are more than 7^5 different combinations. Obviously, it is not a trivial decision, and again it is not an ad hoc consideration.

Finally, in most of strategy games, the fights are performed between two or more sides and these sides follow some implicit or explicit *spatial tactic*. One may fast rush toward the first enemy, other may try to kill the weakest first. This spatial tactic is called here as battle formation. In other words, the player also has to decide how to organize his/her troops to achieve his/her goals. Also, there are a lot of ways of how to organize the units, or the heroes, and this is not trivial. And finally, spatial tactics are not ad hoc.

It is not hard to understand that a successful build must combine a good set of decisions that are not easy to conceive. Here, these decisions are summarized into these three abstract dimensions: attributes (traits), equipment and tactics (formation). But, of course, there may be many other dimensions not considered here. To transfer this decision making to an algorithmic approach, such attributes must be represented (or encoded) in some way. In our case, we try to find these decisions and generate content by coevolutionary means. To generalize the proposed approach to other games the representation must be adapted to fit into a different game content. Of course, there will be different equipments, units, attributes and formations, requiring that the genotype-phenotype mapping function must be adjusted.

Call of Roma has some constraints related to the number of UTPs and restrictions on the distribution of these points. These constraints were taken into consideration by the genetic operators, to avoid the generation of invalid content. It may be expected that different games have different constraints. Related to the adjustment of the representation, pointed above, the adjustment of these constraints must also be considered. And so, the implementation of proper operators or repair functions may be required.

Also, the generated content of another game must be evaluated according to some function. This paper uses five different scores as a measure of good battle performance. These scores represents *desirable* properties and not *necessarily* properties. And it may be thought that these properties are exclusive to the Call of Roma context, but they are not. They take into consideration just one element: the number of soldiers. It is measured the killing ratio, the surviving ratio, the capability of winning, using less resources (i.e. soldiers), etc. It is all about the number of soldiers on the input, and the number of soldiers on the output. This consideration is one of the most common considerations of the RTS game researchers [Synnaeve and Bessiere 2012; Ballinger and Louis 2013]. The evaluation function must also be adjusted to fit into another game context.

In this paper, we perform a *simulation-based* evaluation [Togelius et al. 2011b; Shaker et al. 2014]. This means that the generated content, i.e. the attacking team, is tested by a simulator and the fitness is assessed based on the values obtained on this simulation. This simulator is specific to the game Call of Roma. Obviously, considering a simulation-based evaluation, to evolve content to other games, based on this proposed approach, new simulators must be implemented to fit on another battle system. The exchange of the simulator engine does not change our cooperation techniques or the measurement of the five scores. The battle simulator is ad hoc, but our approach is not.

5 Results and Discussion

In order to test the proposed coevolutionary algorithm, it was executed over different sizes of N, i.e., it was tested against enemies of different number of heroes. This number varies from 1 to 9, so it performs tests of up to 9×9 heroes. The benchmark for testing the algorithm used in this paper was developed by the users of the Armageddon Battle Simulator [5]. The users simulate battles to get information about the performance of their builds against a specified opponent, to test new ideas or to find something that overcomes their enemies. Every simulation is stored on the server in agreement with the user. The simulator's server application is running on the Google App Engine[8] platform. There are currently 37 thousand simulations stored. Most of them are just irrelevant registers and should be obviously discarded. Of course it is not possible to run the coevolutionary algorithm over all these test cases, but a set of 10 interesting simulations for each size considered (from 1 to 9) was carefully taken as input. So the algorithm is tested against a total of 90 test cases that express what active players and users of the simulator have in their mind.

Every user modeled hero was carefully inspected to ensure that they perfectly follow the contents presented in section 2. However, it is very important to highlight that to add a new level of challenge to the proposed approach, both Green and White basic sets are forbidden to the coevolutionary algorithm and also the Green accessory set. This means that the solutions found by the algorithm will never contain any equipment from these mentioned sets. Looking only to the accessories table (see Table 2), it is easy to see that the evolutionary team is under a disadvantage situation. Additionally, the hero's trait points of the users can exceed in up to 10 points every trait designed by the algorithm, due to the initial base points differences. Finally, the randomness provided by the unit's minimum and maximum damage differences were removed, in such a way that the coevolutionary heroes always inflict the minimum damage while the user's heroes always inflict the maximum damage.

The algorithm is implemented in Java 1.7 and designed to run on the Amazon Elastic Compute Cloud[9] (EC2). The EC2 is an Amazon's cloud computing platform that allows users to rent virtual computers on which to run their own computer applications. The idea here is to show that the algorithm is able to run as a web service that can be explored by the same companies that develop and host MMORTS games. Once the algorithm was implemented under a multithreaded design, it requires the configuration of just one parameter, the number of concurrent processes, to adjust its execution to the corresponding cloud computing machine. It was used a 64-bit image of the Amazon Linux AMI (Amazon Machine Image) 2014.03 over the M1.xlarge instance type. This instance is equipped with 8 computing units, 4 virtual CPUs and 15GiB of memory. For convenience, read the following link[10] for further information about the Amazon Web Services and the EC2's M1 instance type. It also would be interesting to run tests on different types of hardware to check the parallel performance and the scalability of the proposed approach, but this kind of research escapes a little bit of current goals and the scope of this paper. Therefore, these tasks will be left for future research.

The algorithm was tested 3 times for each test case. Therefore, that is 30 executions for each battle size and 270 overall executions. The Table 4 presents the average scores of that 30 executions and its respective standard deviation (small number at the right of the $\pm$ symbol), for each $N \times N$ size.

It is not hard to see that the greater is the size N, the higher is the expected fitness. Considering the performance of the whole team, the participation score varies from 0 to N and, when the victory is achieved, this score is necessarily equal to N. In addition, if victory is achieved, each hero receives a $+1$ from the victory score, so the team receives additional N score. In addition, the usage bonus is available only for $N \geq 4$.

In the other hand, the greater the value of N, the harder the problem due to its increasing complexity. The effects of the high complexity level are visible for the scores of the 9×9 solutions. The worst values in victory rate are in this size. In addition, the offensiveness is lower than the scores of 7×7 and both defensiveness and usage scores are both poor.

Achieving the victory is a good way to increase the fitness value. The victory score was designed to add a new level to the fitness landscape. The idea is exactly to force the algorithm to "forget" defeated solutions, preventing the "avoiding battle behavior" described in section 3.2, while improving the victorious ones, by maximizing the defensiveness and the usage scores. Note that the defensiveness score will only be greater than 0 if a hero survives, i.e., if there are remaining troops after the battle, which implies victory. The same is truth for the usage score.

[8] Google Cloud: `https://cloud.google.com/products/app-engine`

[9] Amazon Web Services: `https://aws.amazon.com`

[10] `http://aws.amazon.com/ec2/instance-types/`

Table 4: *Average Scores*

Size	Victory Rate	Fitness	Participation	Offensiveness	Defensiveness	Usage
1x1	100%	$5.143_{\pm 1.554}$	$1_{\pm 0}$	$2.611_{\pm 1.667}$	$0.532_{\pm 0.194}$	$0_{\pm 0}$
2x2	90%	$7.874_{\pm 2.367}$	$1.918_{\pm 0.249}$	$3.102_{\pm 1.902}$	$1.053_{\pm 0.477}$	$0_{\pm 0}$
3x3	100%	$14.759_{\pm 5.28}$	$3_{\pm 0}$	$6.983_{\pm 5.639}$	$1.777_{\pm 0.633}$	$0_{\pm 0}$
4x4	100%	$20.795_{\pm 2.886}$	$4_{\pm 0}$	$10.871_{\pm 3.522}$	$1.491_{\pm 0.726}$	$0.867_{\pm 1.008}$
5x5	96.667%	$22.5_{\pm 4.606}$	$4.983_{\pm 0.093}$	$10.587_{\pm 3.289}$	$1.564_{\pm 0.955}$	$1.067_{\pm 1.639}$
6x6	96.667%	$28.104_{\pm 4.46}$	$5.971_{\pm 0.161}$	$13.762_{\pm 4.12}$	$1.805_{\pm 1.101}$	$1.533_{\pm 2.08}$
7x7	90%	$29.716_{\pm 5.821}$	$6.793_{\pm 0.657}$	$14.193_{\pm 4.455}$	$1.664_{\pm 1.315}$	$1.533_{\pm 2.08}$
8x8	76.667%	$30.134_{\pm 7.391}$	$7.446_{\pm 1.167}$	$15.455_{\pm 4.774}$	$0.966_{\pm 0.802}$	$0.267_{\pm 0.868}$
9x9	60%	$28.658_{\pm 8.912}$	$8.066_{\pm 1.376}$	$13.925_{\pm 2.869}$	$0.967_{\pm 1.136}$	$0.6_{\pm 1.589}$

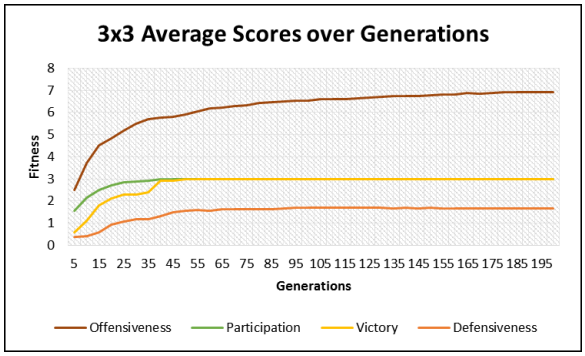


Figure 3: *The evolution of the five scores for the* 3×3 *instances*

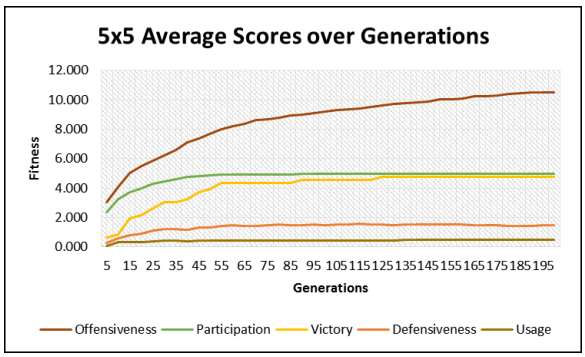


Figure 4: *The evolution of the five scores for the* 5×5 *instances*

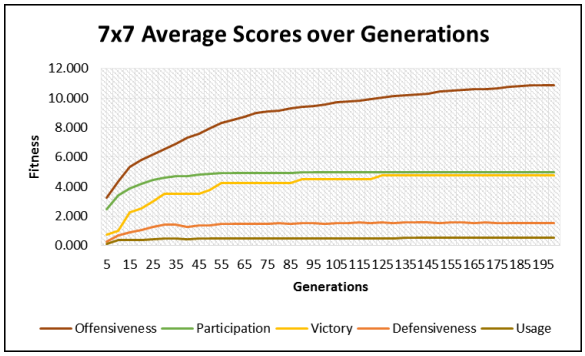


Figure 5: *The evolution of the five scores for the* 7×7 *instances*

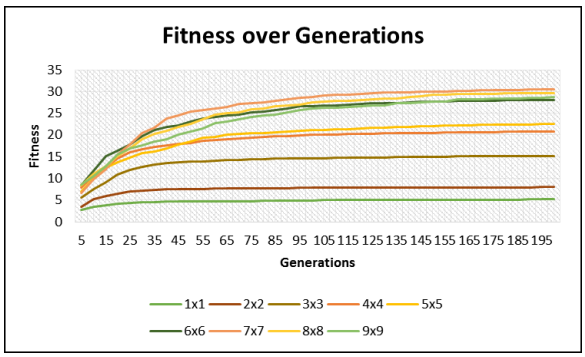


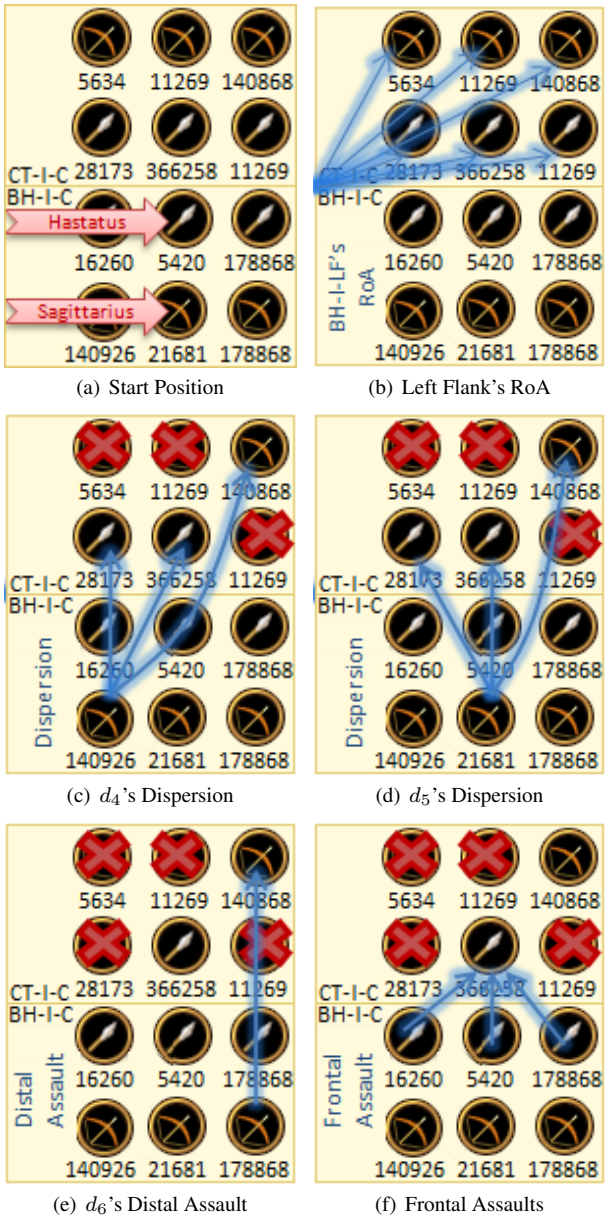
Figure 6: *Average fitness over generations*

To illustrate the differences between the scores or the fitness measurements, the figures 3, 4 and 5 show the average values of the five scores over the generations for three different sizes. It is easy to see what was just being described before. The scores of the participation, offensiveness and victory are high scores in comparison with the other ones, following the value of N. Maybe a new adjustment over the defensiveness and usage scores, or even some kind of normalization of these measurements can be tested to check if a better performance is obtained. Despite the initial idea of replacing the offensiveness by the participation score, it was not tested yet. The correct adjustments of the fitness function is directly related to the algorithm's capability of finding better solutions. But, just to reinforce, the offensiveness measurement cannot be taken as a bad measurement. For a large number of players, this ratio of killed soldiers (instead of killed heroes) is all that matters.

Also discussing about offensiveness, it is possible to see that, on average, for all battle sizes, the offensiveness value is higher than N. In other words, on average, there is always a good battle performance being achieved, in some sense. In fact, if offensiveness is higher than N, it can be said that the evolved units killed more than one of the opponent's units, on average. And again, some players consider this ratio as the most important measure. So, it can be argued that, even losing on 40% of executions for the 9×9, the algorithm still able to find good battle performances. In other words, the algorithm ensures an economical compensation, considering the cost of the losses, even in a defeat situation.

There are additional parameters that must be considered, which are the stopping criteria variables. Once the complexity increases, it may be necessary to take more generations to converge into a proper winning solution. Moreover, the stopping criteria are the same for all executions. The Figure 6 shows the average fitness of the best attacking team over the generations, for each size. As can be seen, the fitness improvements are higher at the first generations and as the generation is approaching the end, the fitness curves are smoothed. At this scale, it is not clear but the fitness values keep growing slowly until the last generation. Most of the best solutions were found at the last cooperation interval. This means that it is possible to keep evolving and improving the solutions until a later moment.

Figure 7 illustrates a sample first round scenario of an evolved hero, called BH-I-C , which consists in one of the most common tactics applied by the coevolutionary heroes. Figure 7(a) shows the initial state, before the battle begins. Figure 7(b) shows the effect of the Rain of Arrows skill, provided by left-flank's Bright Helmet, that helps center hero to kill its opponent hero, called CT-I-C , eliminating the enemy's d_3, d_4 and d_5 before center hero starts the fight. Figures 7(c) and 7(d) shows the first actions of the evolved hero, applying Dispersion over all remaining units, that kills the opponent's d_1 division. At this moment, applying Dispersion is no more advantageous for him, so its d_6 division performs a Distal Assault, as shown in figure 7(e). Finally, in figure 7(f), there is only the main frontal division remaining, which is the single target of all frontal units.



(a) Start Position (b) Left Flank's RoA

(c) d_4's Dispersion (d) d_5's Dispersion

(e) d_6's Distal Assault (f) Frontal Assaults

Figure 7: *BH-I-C vs. TC-I-C: Round 1 illustration.*

On the other side, all enemy's Sagittarii cannot eliminate hero d_2 division, because this hero has a higher Def value and a high Parry. Therefore, there is no alternative to the enemy's main frontal division, but attacking the weak hero's d_2 division, thus wasting a good offense. Notice that it would be more advantageous for the enemy if their main division attack fell on a stronger division of our evolved hero. At the end of the first round, while four divisions of the enemy are dead and the two remaining divisions are seriously damaged, only one weak division of our hero was eliminated and the main divisions were little affected. This good performance can be observed in most of the winning evolved heroes.

It can be argued that nine heroes are not enough to handle the reality of PVP battles in MMORTS, where hundreds of players can join a single battle. It is true, but remember that the goal here is not designing teams for human players or for PvP battles, but to procedurally generate content for PvE battles. In Call of Roma the most common PvE size is 3×3 and there is no environmental character with a number of heroes higher than 6. In fact, in PvE battles, the quality of the heroes usually is more important than the quantity. And this is truth for most MMORTS games, like Senatry, for example. In the most important NPCs there is a size limit, restricting the number of heroes of the human players, usually three or six. For the considered context, the performance of the algorithm is satisfactory.

Finally, it is common to the developers to design overpowered and unfair NPCs to add a new level of challenge for the most experienced players. For example, adding a number of troops that exceeds the heroes' faculty or giving some attributes boost that is impossible to players in game. This practice works, of course, because it is hard to beat the overpowered NPC. But once the player is aware of that difficulty is due to an unfair factor, he feels disappointed with the game. Nobody likes to play with a cheater. The use of the approach presented on this paper can solve this problem. It has been proved that it is possible to procedurally generate victorious solutions, based on builds designed by the players. This victory is achieved by the evolved heroes under unfavorable conditions, as described before, proving that it is possible to be competitive with human players with balanced and fair game content. Due to the game complexity, there is no overall winner, i.e. every strong hero has a weak point and is likely to be defeated in some way.

It is important to highlight that all the input, output and any other files used in this work are being shared in this online repository[11], so that further researchers interested in this work can use the same benchmark to compare and share their results. Once the files provided by the coevolutionary algorithm are compatible with the ABS, players may also be interested in new ideas for developing their builds. But, for future researchers, it is important to keep in mind that the game is always changing and so the ABS is changing too, according to the game. Thus, the forthcoming research might get different results from the simulator and direct comparison may not be proper. Also, it was argued in section 4 that the ABS is specific to the Call of Roma battle system, but it can be easily replaced by any other simulation tool, tailored to other researchers purposes.

In general, the coevolutionary algorithm has a good evolution, avoiding premature convergence, but it takes several minutes in only one complete execution (e.g. 25min for a 9×9 run). That occurs because each battle simulation must take into consideration the actions of all units of all divisions of heroes in the test case, for each subpopulation. Timing performance was not detailed here because it may vary a lot, depending on several other configurations. On the other hand, if the time is short and the results should be returned as soon as possible, winning solutions may be returned as soon as they are found. Looking at the victory score curve, in figures 3, 4 and 5, it can be seen that victory is achieved in most executions at an earlier or middle stage. As well as the algorithm's scalability and other parallel characteristics, this study is left for future research. Although, interested readers may check the output files for detailed information.

6 Conclusion and Future work

This paper presented a coevolutionary approach for the automatic discovery of battle formations for a MMORTS game. Currently, the NPC developed for these games present predefined and static strategies. This work follows the assumption that the procedural generation of content for the NPCs, that are adapting to human strategies, can contribute to making these games more attractive and challenging to human players. To illustrate the proposed approach, we considered the game Call of Roma, for which we developed a simple representation for the candidate solutions and suitable genetic operators. This paper takes the previous work [Ruela and Guimarães 2012] as a starting point, but the current proposal is largely different and expanded from how it was at the referred beginning. There are several changes, specially the new evaluation function and the cooperation method.

The coevolutionary GA is able to find a victorious and efficient solution for overcoming formations developed by players of Call of Roma. In all cases, the algorithm presents a convincing victory rate, but it shows its limitations for instances of higher number of heroes. Moreover, the GA was tested against different scenarios, that consist in teams formed by heroes modeled by active players and users of the simulator. Despite the disadvantages imposed by the battle simulator implemented, the GA was able to keep a positive balance, finding honorable victories.

[11] https://www.dropbox.com/s/fr2fed7yvmxieyh/ArmageddonEvolution.rar

Cooperative coevolution of several subpopulations is not such trivial task. Sometimes the best solution 'a' for the subpopulation A might not be the best solution 'a' for the subpopulation B, and in addition both A and B might not agree with the rest of the team about what would be better for all. So an α based method was used to choose between random or greedy cooperation. The fitness function was carefully designed to attend our goals and to cover different perspectives of a good battle performance, pushing the algorithm forward to victory, in most of cases. Our simulation based evaluation was able to assess an individual fitness for the candidate solutions and an external fitness for the whole team, based only on the number of soldiers.

As reported in section 3.2, there are different concepts of what would be a good battle performance. In future work, new visions of this subjective measurement may be studied, e.g., the cost behind each equipment, which is not considered in this work. The insertion of the building cost would result in complete different solutions and may lead the algorithm to a Multi-Objective approach. More cooperation methods can be explored to find better ways of producing a good Attack team.

The population of the GA is evaluated using only one test case per time, which results in a very specific battle strategy. The application of multiple test cases, such as the approach adopted in CIGAR [Louis and Miles 2005], would allow the generation of more general and robust strategies, able to beat not only one but also a set of test cases. Each subpopulation runs in a single thread but no research has been done about the algorithm's parallelism or the scalability. The number of concurrent process was controlled by a single parameter to correspond to the considered hardware. Therefore, the parallelism can be studied at a deeper level, trying to reduce the execution time.

Finally, the authors intend to use the coevolutionary algorithm in a parallel project for developing new balanced resources, such as new units, equipment and skills, for example, by the means of search-based Procedural Content Generation [Shaker et al. 2014], in an attempt to convince the game developers that there are new promising alternative ways for game design.

Acknowledgment

The authors would like to thank to: State of Minas Gerais Research Foundation - FAPEMIG; Coordination for the Improvement of Higher Level Personnel - CAPES; National Council of Scientific and Technological Development - CNPq (Grants 30506/2010-2, 312276/2013-3).

References

AVERY, P., AND MICHALEWICZ, Z. 2008. Adapting to human game play. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*.

BALLINGER, C., AND LOUIS, S. 2013. Robustness of coevolved strategies in a real-time strategy game. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, 1379–1386.

BENBASSAT, A., AND SIPPER, M. 2012. Evolving both search and strategy for reversi players using genetic programming. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 47–54.

CANOSSA, A., MARTINEZ, J., AND TOGELIUS, J. 2013. Give me a reason to dig: Minecraft and psychology of motivation. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 1–8.

DARWIN, C., AND HUXLEY, J. 2003. *The Origin Of Species: 150th Anniversary Edition*. Signet Classics.

GOLDBERG, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

HEATON, J. 2008. *Introduction to neural networks with Java*. Heaton Research, Inc.

IGEL, C., AND HÜSKEN, M. 2003. Empirical evaluation of the improved rprop learning algorithm. *Neurocomputing 50(C)*, 105–123.

JANSEN, T. 2013. *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Natural Computing Series. Springer Berlin Heidelberg.

LOUIS, S. J., AND MILES, C. 2005. Combining case-based memory with genetic algorithm search for competent game ai. In *ICCBR Workshops*, 193–205.

MILES, C., LOUIS, S., COLE, N., AND MCDONNELL, J. 2004. Learning to play like a human: case injected genetic algorithms for strategic computer gaming. In *Congress on Evolutionary Computation*, vol. 2, 1441–1448.

OTHMAN, N., DECRAENE, J., CAI, W., HU, N., LOW, M., AND GOUAILLARD, A. 2012. Simulation-based optimization of starcraft tactical ai through evolutionary computation. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 394–401.

RICKLEFS, R. E. 2008. *The economy of nature*. Macmillan.

RUELA, A. S., AND GUIMARÃES, F. G. 2012. Evolving battle formations in massively multiplayer online strategy games. In *SBC - Proc. of the Brazilian Symposium on Games and Digital Entertainment - SBGames 2012*, 49 – 55.

SENIOR, T., 2010. Computer program finds devastating starcraft 2 build orders, November.

SHAKER, N., TOGELIUS, J., AND NELSON, M. J. 2014. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

SMITH, G. 2014. Understanding procedural content generation: A design-centric analysis of the role of pcg in games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, CHI '14, 917–926.

STANLEY, K. O., BRYANT, B. D., AND MIIKKULAINEN, R. 2005. Evolving neural network agents in the NERO video game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 182–189.

SUPERDATA RESEARCH, I., 2014. US digital games market update: December 2013, January.

SYNNAEVE, G., AND BESSIERE, P. 2012. Special tactics: A bayesian approach to tactical decision-making. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 409–416.

TOGELIUS, J., KASTBJERG, E., SCHEDL, D., AND YANNAKAKIS, G. N. 2011. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, ACM, New York, NY, USA, PCGames '11, 3:1–3:6.

TOGELIUS, J., YANNAKAKIS, G. N., STANLEY, K. O., AND BROWNE, C. 2011. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on 3*, 3, 172–186.

TOGELIUS, J., SHAKER, N., AND NELSON, M. J. 2014. Introduction. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer.

TOZOUR, P., AND CHAMPANDARD, A. J., 2012. Making designers obsolete? evolution in game design. Open Interview.

YEE, N., 2005. MMORPG hours vs. TV hours, January.