Competitive Algorithms for Co-evolving both Game Content and AI. A Case Study: Planet Wars

Mariela Nogueira-Collazo, Carlos Cotta Porras, and Antonio J. Fernández-Leiva

Abstract

The classical approach of Competitive Coevolution (CC) applied in games tries to exploit an arms race between coevolving populations that belong to the same species (or at least to the same biotic niche), namely strategies, rules, tracks for racing, or any other. This paper proposes the co-evolution of entities belonging to different realms (namely biotic and abiotic) via a competitive approach. More precisely, we aim to coevolutionarily optimize both virtual players and game content. From a general perspective, our proposal can be viewed as a method of procedural content generation combined with a technique for generating game Artificial Intelligence (AI). This approach can not only help game designers in game creation but also generate content personalized to both specific players' profiles and game designer's objectives (e.g., create content that favors novice players over skillful players). As a case study we use Planet Wars, the Real Time Strategy (RTS) game associated with the 2010 Google AI Challenge contest, and demonstrate (via an empirical study) the validity of our approach.

Index Terms

Coevolution, RTS game, virtual player, Automatic Content Generation.

I. Introduction

Game designers recognize game Artificial Intelligence (AI) as one of the primary ways to turn good games into best-sellers. The application of AI, and Computational Intelligence (CI) (as the AI representative of the nature-inspired computational techniques for learning and optimization) in games is a research field that poses significant challenges for the game development community [1], [2], [3].

From the set of AI techniques, *Coevolution* promotes the interaction between different species and that has already been applied in the game area. Several models have been shown to be successful in different applications but coevolution also has intrinsic drawbacks that are difficult to solve. Many approaches analyze the dynamics of the coevolutionary process to identify both its weaknesses and strengths. This produces more solid techniques for

M. Nogueira, C. Cotta and A.J. Fernández are with the Department of Languages and Computer Sciences, University of Malaga, Andalucia Tech, Malaga, Spain. E-mail: {mnogueira,ccottap,afdez}@lcc.uma.es.

coevolution support. One of the first approaches in this aspect was that of Rosin and Belew [4]. They proposed the use of a *Hall-of-Fame* (HoF) based mechanism as an archive method. The HoF acts as a long-term memory mechanism in Competitive Coevolutionary (CC) algorithms to manage the historical set of champions, during the individuals' evaluation. In this line of work we have already employed the basic archive methods for generating virtual players (i.e., a traditional game AI perspective) in a capture-the-flag game, with the aim of coping with the aforementioned problems in coevolutionary methods. In particular, in [5] we analyzed, in the context of the RTS game *RobotWars*[1], how the diversity and the growth of the HoF can influence the quality of the solutions obtained by HoF-based CC algorithms. Later in [6], we considered another RTS game, namely *Planet Wars* (the Google AI challenge in 2010), and added novel strength indicators which were independent from the fitness function, to avoid the appearance of cycling. We also incorporated into our prime CC algorithm, an additional archive (termed *Hall-of-Celebrities*, HoC) that contained a team of experienced virtual players that were used to evaluate how strong a candidate was. This combined use of two halls (HoF and HoC) with the (possibly combined) employment of diversity and quality metrics helped the optimization to obtain competitive bots that self-adapted to beat their (co)evolved enemies. The results obtained in our previous work shown that diversity management is a key point in guaranteeing the progress of the arms race.

The current state of game development requires revising the term of game AI which should be "enhanced with non-traditional research and development areas beyond NPC control [7] as AI/CI is applied to most aspects of game development and design" [8]. Procedural Content Generation (PCG) in particular, applies AI techniques to automatically generate game content. This content refers to all aspects of the game that affect gameplay other than NPCs, such as maps, levels, dialogues, characters, rule-sets and weapons [9]. PCG represents one of the most interesting issues in the CI-in-games community, as game content is an important factor to keep "players engaged in the gaming world" [10]. However, it is well-known that player satisfaction is not easy to measure as it depends on many variables such as personality, age, culture, skills, preferences, and player gender. In fact, developing a general solution to satisfy all players is extremely complicated, (possibly even a utopian task). Although it is known that games which are too easy or too hard for the players will not be successful, design decisions relative to the hardness of a game (or a level in a game) are difficult to make. Players have different skills ranging from novice to those with professional skills. So, how to balance a game? In this context, game adaptability to the player is an important issue to consider.

With this issue in mind, this paper represents a step forward with regard to our previous work and also considers the generation of game content (i.e., different from bot's behavior) to compose a multi-species CC algorithm that promotes the self-generation of both NPC's AI and game content. CC represents a perfect scenario in which this (bi-)evolution can be defined. Our proposal can assist both programmers and designers. It not only allows the automatic generation of game AI (i.e., game strategies to govern the behavior of the NPCs), but it can also help designers to create content that dynamically adapts to reach specific objectives. Imagine, for example, a map/level

[1]://wp.me/p2cObl-60{}

that adapts to favor novice players to face more experienced players so that all of them enjoy the match (i.e., the match is neither too easy for the skillful player nor too hard for the in experienced players). From this design perspective, our approach can be viewed as a model for the dynamic adaptation of the game according to specific players' profile via PCG.

Our work here represents another attempt at interspecific coevolution (but, as usual, bridging some gaps with reality), whereby we manage (two) species which are intrinsically different (they are different-in-nature domains). One of them groups bot strategies and the other represents game content (that might be in the form of maps or levels, for example). So, the relationship of competitiveness between them is slightly beyond that of the typical scenario of two competing species, such as a *predator - prey* or a *parasite - host*, in which the performance of each individual is closely related to the performance of the competitor [11]. Therefore, how to establish the competition between the content game and bots is a key point in our proposal and makes it new in the CI-in-game field. In addition, to the best of our knowledge, this is the first time that a CC algorithm is used to handle the co-evolution of game content as well as game AI. There is a single, obligatory condition for the viability of this approach, and it is competition, because the existence of a competitive relationship between the coevolved species is strictly necessary.

It is important to emphasise that our coevolutionary system is applicable to any genre of games which promote competition between players, and for this we first describe a generic schema. We provide a proof-of-concept to show its suitability by instantiating it to a specific RTS game. Finally, we proved, via an experimental validation, that it can generate not only content that adapts to specific objectives but also game AI that performs at the level of the best bot described in the scientific literature for this RTS game.

II. Background

Games have already been used as environments over which different coevolutionary models have been tested. The range of these games is wide, from simple games (e.g., board games) to others that can generate an extensive search space and that involve intrinsically complex domains (e.g., RTS games). The concept of coevolution involves individuals which interact with each other and that might belong (or not) to the same species. From a general categorization of the coevolutionary models studied in the scientific literature one can observe a trend towards the use of models that coevolve populations belonging to the same species. This means that all the individuals have the same genetic structure or codification, and from this idea, two approaches can be identified.

The first approach uses a unique population and the evaluation process is carried out by having individuals face each other according to a selection mechanism. A direct consequence is that here reproductive relationships emerge as a natural process. The second case employs different populations, to encourage the arms race. There is a tendency to create these populations as lineages, distinguished from each other by modifying some aspects of the coevolutionary cycle (e.g., the individuals' generation process, fitness function, genetic operators, etc.). Another variant is just to generate several populations over the same features and conditions. It is interesting to see how, in both of them, even though they are dealing with a unique species, it is possible to produce "interspecific coevolution" (according to the classifications defined in [12]) whenever the competition occurs in populations that do not have reproductive or parental relationships. Along these lines, [13] worked with competitive fitness functions in the Tic Tac Toe game, whereas [14] evolved strategies in a pursuit-evasion game. Also, [15] analyzed the evolution of both the morphology and behaviors of artificial creatures through competition in a predator-prey environment. [16] proposed a method for finding game strategies with a basic coevolutionary algorithm for zero-sum and non-zero-sum games. More complex scenarios such as those that emerge in strategy games have also been considered as, for instance in [17] which coevolved AI opponents to train human players in the context of a capture-the-flag game. [18] presented a spatial decision making system that used a basic implementation of co-evolution. Another interesting perspective was presented in [19], which employed the game of Tempo as the test suite, and evaluated the results of inserting an expert strategy into two populations.

All the aforementioned approaches use a co-evolutionary model based on a single species which is, as far as we know, the most-used CC variant in the literature. However, there are computational coevolutionary models based on multispecies interaction, and they produce a prime and proper "interspecific coevolution". This type of coevolution is less common in games although one can find a pair of proposals following this schema as for instance, that presented in [20] in the context of Pac-Man versus Ghost Teams Competition. The authors coevolved two different populations, one for the Pac-Man strategies and the other for the Ghost team. In this approach the two populations were derived from different species, although both of them shared the same nature, in this case both populations evolve "virtual players" (i.e., the classical game AI application). Another interesting approach was described in [21], which compared the performance between nine-population co-evolution and single-population co-evolution. The goal was to develop controllers for a simple car racing game; each population represented a controller (i.e., again, same nature for the populations).

Note that this section does not mention other forms of coevolution which have also been applied in games because they are not directly related to competition (e.g., [22] the focus is cooperation and not on competition).

III. General model

This section presents our general (and primary) model for a Multi Content Competitive Coevolutionary Algorithm (MuCCCo) in Section III-A. Subsequently three variants of this algorithm are also described.

A. *The general schema for MuCCCo*

Our proposal employs two populations to automatically generate both game strategies and game content[2] at the same time. These two domains are completely different in nature and have conflicting goals. Due to this, the population inter-relationship is not standard and will be described below.

In a general competitive co-evolutionary system, game AI evolves with the goal of finding 'good' virtual players that defeat the champions of its adversarial populations. Our proposal maintains this approach but the game scenario

[2]In the rest of the paper we will use the terms 'game strategy', 'bot' and 'game AI' (resp., 'game content', 'maps' and 'levels') interchangeably.

is influenced by the champions of the game content population. The process of generating content is usually driven by constraints or effectiveness criteria which are associated with a specific goal like, improving content diversity [23], or satisfying aesthetical measures or human preferences with respect to the balance of difficulty [24], just to name a few. The key point is that the two domains to evolve are not intrinsically competitive (by their very nature) and the evolution of game content cannot be driven by a direct confrontation against the champions of the bot population. Even in the case of a multi-player game, the generation of game content requires a specific goal to determine what a (game content) champion is when it is used as the game scenario in a game that pits the champions of the different bot populations against each other. In this context, it makes sense to fix a game AI that provides the goals (surely imposed by the game designer) to direct the evolution of maps. Consider this bot (termed as the *fixed rival bot* (FRB)), for instance, as a model of a specific player profile so that bots –of the other game AI populations– evolve with the goal of beating players with this profile, whereas game content evolves driven by objectives related with this FRB. The game content, may for instance, adapt to favor (resp. disfavor) the FRB if this represents a model of novice (resp. an skilfull) player. This basically means that the game scenario adapts to satisfy a design decision i.e., in this case, preventing a game from being too hard (resp. easy) for the novice (resp. experienced player). Of course, other objectives for the game content evolution with respect to the FRB(s) can be included.

Note that any game which uses content to affect gameplay other than NPCs and promotes competition between players can apply this model, and the *fixed rival bot* is the seed for encouraging the specialization process. As shown in our study case, the designer might use one or many of them, depending on the objectives required. Note also, that a specific player profile can be extracted using standard or advanced player modeling techniques (beyond the goal of this paper).

Our proposal can be adjusted to multi-player games and multiple FRBs (see Figure 1); for a n-player game, we have $n-1$ game AI populations that evolve according to some specific objectives (again, associated with design decisions) related with all the FRBs. We can also have m populations, one for each kind of content (that should be different in nature from the other contents), that set the game scenario for the battles between the $n-1$ players and the FRBs. The form of these fights can vary depending on the design objectives.

Algorithm 1 shows the schema of MuCCCo adapted (for simplicity) to a 2-player game. This schema manages two populations, one that encodes game content, and the other for game strategies. Each population uses its own HoF as a long-term memory mechanism to keep the winning individuals found in each coevolutionary step, and each member of the HoF is used in the evaluation process (although this schema of evaluation can vary). The algorithm has two parameters: α that indicates (in percentage) the portion of the HoF to be removed so as to maintain only those worthwhile champions that might contribute to the solution according to some specific metric (see details below), and λ the frequency of executing this updating of the HoF.

One or several fixed rival bots can be considered in the evaluation step. In the general explanation we consider one FRB although we employ several of them in the experiments. So, applying this model to our domain, in the n_{th} coevolutionary step, each candidate in the bot population is faced with a fixed rival bot FRB in $n-1$ matches

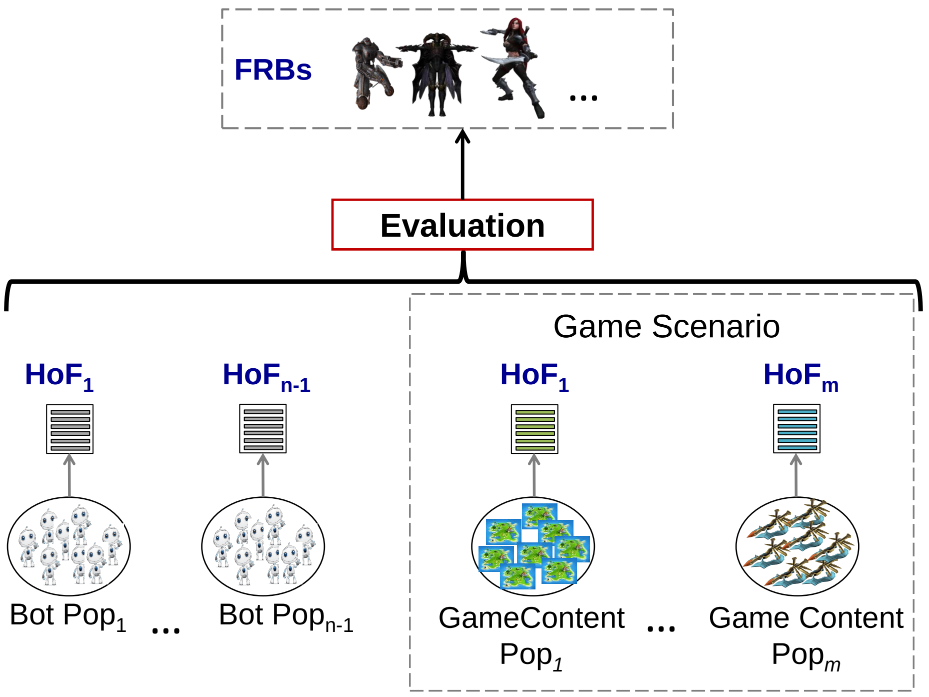
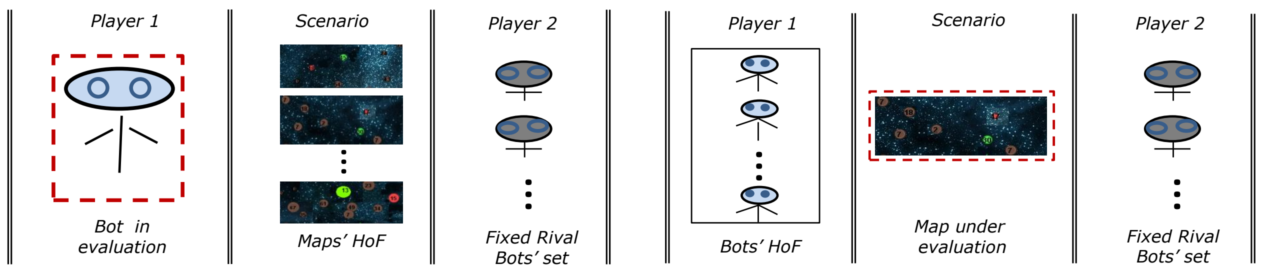


Fig. 1. Multi-content generation coevolutionary schema for a n-player game.

executed over each map in $\{map_1, map_2, ...map_{n-1}\}$, where map_i is the game content champion found in the i-th ($1 \leqslant i \leqslant n-1$) coevolutionary step. Additionally, each candidate in the game content population is also evaluated in the n_{th} coevolutionary iteration. This is done by organizing a set of battles, in the scenario imposed by the candidate, between the fixed rival bot FRB and each member of the bot's HoF, that is to say, $\{bot_1, bot_2, ...bot_{n-1}\}$. Figure 2 shows how the battle scenario is configured to evaluate each population in a two-player game, and where 'player 1' and 'player 2' denote each of the players considered in a confrontation. To sum up, the bot fitness function evaluates the capacity that an individual has to defeat the fixed opponent(s) in a finite set of maps, whereas the map fitness function evaluates the nature of a map that favors the fixed opponent(s) when it(they) is(are) competing against a set of evolved bots in different matches.

Observe that the initial objective is to find a winning individual for bots (i.e. the population marked as active – line 1 – and which will firstly be evolved). During the execution, the population marked as active (resp. non-active) is the one that will be (resp. will not be) evolved in a coevolutionary cycle, and this role will be interchanged between the two populations (line 20) until the end of the process. The HoF of the non-active population (i.e., the maps) is initially loaded with some (randomly or manually generated) individuals (line 2, method PRELOADMAPS) whereas the bot's HoF is initially empty (line 2). The HoF of the population being evolved is updated (i.e., only robust champions are kept) according to specific criteria and all λ coevolutionary steps (lines 4-6). The active population is randomly initialized (line 7) and the strength of its initial candidates is evaluated in line 8. Then a standard



(a) Bot evaluation (b) Map evaluation

Fig. 2. Configuration of the battle scenario for both populations, note that the object under evaluation is highlighted with dashed lines.

evolutionary process tries to find a victorious individual in the active population (lines 11-24). If one individual is found to be victorious (line 17), it is added to the HoF of the active population (line 19) and the process is initiated again but with the population roles reversed (line 20). This means the population not being evolved will take the active role in the following co-evolutionary cycle and vice versa; otherwise (i.e. no winning individual is found) the search process is restarted. If, after a number of coevolutionary steps, no winning individual is found in the active population (i.e., bots or maps) then the search is considered to have stagnated and the coevolution ends (see the while condition in line 3). At the end of the process we obtain as a result, two sets of winning solutions associated accordingly with each of the populations.

B. *Variants of MuCCCo*

This section describes three variants of our basic MuCCCo algorithm that have been adjusted from those proposed in [5] and are denoted as: MuCCCo-Diversity, MuCCCo-Quality and MuCCCo-U. The variants differ from each other in the way in which they periodically update the archives employed on the basis of quality and diversity metrics. The objective is to establish a correct policy for updating the HoF (by removing some of its members) whilst maintaining in this set, only those champions which can be considered robust.

The number of individuals to be removed from the memories (i.e., strategies' HoF and game content' HoF) in each update is a percentage value (α) (see line 5 in Algorithm 1), set by the programmer. This indicates the portion of the HoF to be removed; in other words, the HoF (with cardinality $\#HoF$) is ordered according to the metric value (i.e., diversity, quality or a combined criteria) in a descending order and the last $\lceil \frac{\#HoF \times \alpha}{100} \rceil$ individuals in this ordered sequence are removed unless the HoF is empty after executing this update. The update is executed every λ coevolution.

1) MuCCCo-Diversity: This variant updates the archive by eliminating those solutions identified as "less diverse". In a previous approach we proved that the control of diversity helps to avoid redundancy in the champion's memory as well as combat over-specialization, and for this reason we decided to employ it here.

2) MuCCCo-Quality: In this variant, the criteria for eliminating solutions of the HoFs is a measure of quality (which, as for diversity factor, has to be specifically defined for the game under consideration).

Algorithm 1: MuCCCo(α, λ)

```
nCoev ← 0; activePop ← Bots; nonActivePop ← Maps; $\phi$ ← thresholdvalue;
$HoF_{Bots} \leftarrow \emptyset$; $HoF_{Maps}$ ← PRELOADMAPS(); frb ← FIXEDRIVALBOT();
while $nCoev < MaxFailCoevolutions \wedge \neg timeout$ do
    if $nCoev \bmod \lambda = 0$ then //HoF updating every $\lambda$ Coevolutions
        PURGE$_\alpha$($HoF_{activePop}$); // HoF updating
    end if
    $pop_{activePop}$ ←RANDOMSOLUTIONS(); // Active pop randomly initialized
    $pop_{activePop}$ ← EVALUATEPOP$_{frb}$($HoF_{nonActivePop}$);// Evaluate candidates in Active population against rival
        HoF and FRB
    $i \leftarrow 0$;
    $foundWinner \leftarrow false$;
    while $(i < MaxGen) \wedge \neg foundWinner$ do
        parents ←SELECT ($pop_{activePop}$);
        children ← RECOMBINE ($parents, p_X$);
        children ← MUTATE ($childs, p_M$);
        $pop_{activePop}$ ← REPLACE(childs);
        $pop_{activePop}$ ← EVALUATEPOP$_{frb}$($HoF_{nonActivePop}$);
        if (FITNESS$_{activePop}$(BEST($pop_{activePop}$)) $\geq \phi$) then //winner found!
            $foundWinner \leftarrow true$;
            $HoF_{activePop} \leftarrow HoF_{activePop} \cup \{$BEST$(pop_{activePop})\}$
            temp ← activePop; activePop ← nonActivePop; nonActivePop ← temp;// interchange populations'
                activity roles
        else
            $i \leftarrow i + 1$;
        end if
    end while
    if $(foundWinner = true)$ then
        $nCoev \leftarrow 0$; // start new search
    else
        $nCoev \leftarrow nCoev + 1$; // continue search
    end if
end while
```

3) MuCCCo-U: This variant continues the idea of optimizing the champions memory, but now we propose a multiobjective approach where each solution has both a diversity value and a quality value, as previously described, associated with it. Then, a percentage value (α) from the set of dominated solutions according to the multiobjective values is removed.

Note that, in general, the adaptation of our proposal to a specific game can be done by just providing game-specific definitions for the evaluation function and the diversity measure. We agree that this might be not simple to do but, in general, the adaptation does not require additional efforts.

IV. Study Case on a RTS game: Planet Wars

To demonstrate the applicability of our proposal, this section describes an instance of it in a specific RTS game. Section IV-A introduces the game, then Section IV-B centers on codification issues. Finally, Section IV-C provides specific details to adapt our algorithms to the game Planet Wars.

A. *Planet Wars*

Real-Time Strategy (RTS) games are one of the most exciting genre of games or research because they provide a range of challenging problems for AI design.These include planning in an uncertain world with incomplete information, learning, opponent modeling, and spatial and temporal reasoning [25]. Here, we consider the Planet Wars game, which is an easy-to-understand instance of an RTS game. Planet Wars is a two player game that was proposed by the Google AI Challenge international contest 2010[3] the objective of which was to optimize the Artificial Intelligence (AI) of the game.

The game takes place on a map which contains several planets, each of which has a specific number of ships on it. The planets may belong to the player, the opponent, or just be neutral. Each planet owned by a player (not those that are neutral) will increase their forces there according to the "growth rate" of the planet.

At the start of each turn, the player receives the current status of the game (i.e., information about the planets and fleets) and can only take one type of action: send fleets of ships to another planet. After sending fleets, each planet owned by a player increases the number of ships remaining proportional to the planet growth rate. The fleets that were sent in a previous order take a certain number of turns to reach their destination, according to the distance between the planet of origin and the destination. Upon arrival, if both the planet and the ships belong to the same player, then the number of ships increases by adding together the current number of ships on the planet and the number of newly arrived ships. Otherwise, if the arrival planet is neutral, then it has a fixed number of ships (NS) and the player must send at least $NS + 1$ ships to own the planet (i.e. reach the neutral planet in order to conquer it); and if the player sends fleets to an enemy planet (i.e. the player attacks an enemy planet) a fight is initiated to own the planet.

Although the players issue their orders on a turn-by-turn basis, they carry out these orders at the same time, so we can treat this game as a real-time one. The player with the most ships at the end of the game wins. The game may also end earlier if one of the players loses all his/her ships or if one player exceeds the time limit without completing his/her orders and therefore forfeits the game. If both players have the same number of ships when the game ends it is considered to be a draw.

Planet Wars has been used as a test scenario in other research on AI applied to video games. The first issue was a proposal by a team from the University of Granada, they designed a Genetic Algorithm to generate and train bots which were apt for participating in the Google AI competition. The result was the "GeneBot" [26], [27], which was a virtual player that defeated baseline bots in most playing environments and obtained a top-20% ranking position in the contest. Other examples are described in [24] and [9] where the authors presented two procedurally balanced map generators for this game. As can be seen, in this game the generation of maps and bots has been addressed earlier but independently from each other, without any relation between them (i.e. maps and bots). Our purpose here, is to experiment with the unification of these two objectives. The idea is to contribute to the (PCG) with a mechanism which allows the simultaneous creation of multiple games' resources, and which exploits the

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

possibilities offered by CC for co-evolving multiple species.

B. Representation issues

Our coevolutionary algorithm tries to achieve two main conflicting goals. The first is to find optimal winning strategies to govern a bot (i.e., the virtual player) in the game Planet Wars. This is done optimizing the rules that guide its decision making mechanism, so that, in each turn the bot should select the best action according to the current state in the game. The second objective is to automatically generate game maps to avoid the victory of the evolved bots. In other words, this second goal leads to maps which enhance the performance of the opponent at the expense of the evolved bots.

Two different populations (i.e., one for bots and the other for maps) are thus managed, and their specific representations are now detailed. For bots (i.e., strategies), we take into account that the state, at a specific instance of the game is determined by the advantage (with respect to its opponent) of the bot in terms of ships and growth rate. Both advantage metrics have three possible values ('high', 'null' or 'low') which indicate the level of advantage in each case. Lets say that, for player p (resp. opponent o), GR_p (resp. GR_o) is the total growth rate, according to the growth rate of the planets owned by player p, and let $\Delta GR_{po} = GR_p - GR_o$; then if $\Delta GR_{po} > 0$, we say that player p has a 'high' advantage over opponent o in terms of the growth rate; 'null' in the case of a draw (i.e., $\Delta GR_{po} = 0$), and 'low' if $\Delta GR_{po} < 0$. The calculation of the advantage in terms of ships is similar but considers NS_p instead of GR_p (resp. NS_o instead of GR_o) as the total number of ships owned by player p (resp. opponent o) at the current instant of the game; in other words, $\Delta NS_{po} = NS_p - NS_o$. However, we considered different thresholds for the advantage values so that if $\Delta NS_{po} > 10$, then player p has a 'high' advantage, 'null' if $0 \leqslant \Delta NS_{po} \leqslant 10$, and 'low' otherwise (note that we consider that a difference of 10 ships is not significant enough to distinguish between the two players; we settled on this value through our playing experience).

So, a virtual player strategy is coded as a bidimensional matrix where the first dimension symbolizes the player's advantage over his/her opponent in terms of the number of ships (i.e. the ΔNS_{po}), and the second dimension represents the advantage in terms of the total growth rate (i.e. the ΔGR_{op}). As we explained above, each axis has three possible values ('high', 'null' or 'low'). Each cell in the matrix acts as a gene and stores one of the following possible actions:

1) attack the strongest enemy planet (AS) (i.e., the enemy planet that owns the greatest number of ships),
2) attack the weakest enemy planet (AW),
3) attack the closest enemy planet (AC),
4) conquer the strongest neutral planet (CS) (again in terms of number of ships),
5) conquer the weakest neutral planet (CW),
6) conquer the closest neutral planet (CC),
7) follow the enemy (FE): it sends fleets to one of the destination planets to which the enemy is now sending his/her own fleets, chosen by first planet on the list of the enemy fleets for which we have sufficient number of ships.

In this way, the whole matrix represents a strategy that controls, deterministically, the behavior of a bot during the game by executing the action associated with a specific instance of the game. For a virtual player there are 9 possible different states (i.e. 3×3, all the possible value combinations considering the two dimensions of the matrix). Basically, in a specific turn of the game the player will execute the action stored in the state in which he perceives that he is. Note that the search space is $7^9 = 40353607 \in [2^{25}, 2^{26}]$, which cannot be exhaustively assessed due to the cost of the evaluation that requires a game simulation and thus metaheuristic techniques are used.

Our maps are coded differently. A map is a list of n planets where $n \in [15, 30]$ and each planet is represented by the following information: two float values that indicate, on the map, the x and y coordinates corresponding to the center of the planet, where $x, y \in [0.0, 15.0]$; the owner attribute which identifies the player who dominates the planet (i.e., 0 if the planet is a neutral and thus has not owner; 1 if player 1 is the owner of the planet, and 2 if player 2 owns the planet); the total number of ships ($NS \in \mathbb{N}$) that the planet hosts (when the map is created $NS \in [1, 100]$); and an integer value $GR \in [1, 5]$ that denotes the specific growth rate of the planet.

C. Specific issues

A game strategy is considered ‘winning’ if it is able to beat the *fixed rival bot*(s) when they are faced on all the maps belonging to the map’s HoF. On the other hand, a map is considered a victorious individual if it meets two constraints: the first being that the *fixed rival bot* has to defeat (or tie with) each member in the bots’ HoF in a match played, considering this map (i.e. the evaluated map) as the confrontation scenario. The second requires that the *fixed rival bot* does not exceed a limit of planets conquered. This second constraint is imposed so as to avoid the generation of corrupt (i.e., degenerated) game content, that is to say, content that provides no possibility at all for the game strategies to beat the *fixed rival bot*.

More specifically, bots are evaluated by a direct confrontation against the FRB on all the maps in the corresponding HoF. So, given a specific strategy $s \in Pop_{Bots}$, its fitness (F) is computed as follows:

$$\begin{aligned} \mathrm{F}_{Bots}(s) = \frac{1}{k} \sum_{m \in \mathrm{HoF}_{Maps}} \big(& r_m^{s,frb} + (C_1 - nTurn_{s,frb}(m)) \\ & + C_2 \cdot Q_{s,frb}(m) \big) \end{aligned} \tag{1}$$

- $k = \#HoF_{Maps} \in \mathbb{N}$ is the cardinality of the maps’ HoF;
- function $r_m^{a,b} \in \mathbb{R}$ returns ϕ points if strategy a beats strategy b on the map m, $\frac{\phi}{2}$ if the confrontation ends in a draw, and 0 if FRB wins over strategy s;
  So, for instance, $r_m^{s,frb}$ would return ϕ points if strategy s beats the FRB on the map m;
- function $nTurn_{a,b}(m) \in \mathbb{N}$ returns the number of turns spent on a game that, played on the map m, ends in a victory of a or in a tie, and 0 otherwise;
- and $Q_{a,b}(m) \in \mathbb{R}$ is the percentage of conquered planets (with respect to the total number of planets existing on the map m) that strategy a owns at the end of the confrontation between a and b on map m.

- C_1 and C_2 are two constants used for scaling/weighting. We consider $C_1 = 500$ and $C_2 = 10$, based on our game experience and with the goal of providing significance to both number of turns and conquered planets in the fitness evaluation.

In turn, map evaluation uses a different fitness function defined in Equation 2 where $m \in Pop_{Maps}$ is a map to be evaluated, function $r_m^{a,b} \in \mathbb{R}$ is defined as shown above, and $l = \#HoF_{Bots} \in \mathbb{N}$ is the cardinality of the bots' HoF. Also, in the new fitness equation we employ a modification of the function $Q_{a,b}(m)$ used in Equation 1. The reason is to penalize the *corrupt maps*, because in preliminary experiments, based on the function $Q_{a,b}(m)$ our algorithm generated (as champions) maps where the majority of the planets had a beneficial position with respect to the planet of origin associated with the *fixed rival bot*. Note that the number of turns is not considered in these experiments, initially it was included in the function but the first experiments showed that finding individuals which gave the victory to the FRB without crushing players' adversaries was very difficult for the maps' population. We therefore try to reduce the selection pressure during the search process centering only on the goal of achieving victorious maps.

$$\mathrm{F}_{Maps}(m) = \frac{\sum\limits_{s \in \mathrm{HoF}_{Bots}} \left(r_m^{frb,s} + C_2 \cdot Q'_{frb,s}(m) \right)}{l} \tag{2}$$

where

$$Q'_{a,b}(m) = \begin{cases} 0, & Q_{b,a}(m) < \rho \\ Q_{a,b}(m) + Plus_{b,a}(m), & Q_{b,a}(m) \geqslant \rho \end{cases}$$

and

$$Plus_{b,a}(m) = Q_{b,a}(m) - \rho$$

The maps' fitness function values positively those maps in which the bot champions are not radically defeated by the FRB. This basically means that the bot champion owns, at least, $\rho\%$ of the planets in the map at the end of the fight. This constraint prevents the convergence of the algorithm onto corrupt maps. Note also that a bonus is given to those maps in which the loser bot (resp. the winner FRB) conquers more than $\rho\%$. Using this bonus we try to intensify the search for maps which lead to games with a fairer share of resources between players. We tested different values for $\rho \in [0\%, 50\%]$ but found that when ρ was close to 0% all planets in the maps tended to be aligned, clearly close to the initial position of the *fixed rival bot* and far from the initial position of the opponent game strategy; o its turn, when ρ was close to 50% we experimented problems in finding map champions (probably because maps were not able, by themselves, to find the equilibrium of the game). Based on our game experience and preliminary experiments, we finally set $\rho = 20\%$ to prevent totally corrupt maps from being generated.

1) MuCCCo-Diversity: As for the strategies' HoF, the diversity provided by each of its members is calculated via the genotypic distance as follows: we manage the memory of champions as a matrix in which each row represents a solution and each column a gene (i.e., an action in the strategy). Note that there are 9 columns (see Section IV-B). Then, we compute the entropy value for a specific column j as follows:

$$H_j = -\sum_{i=1}^{\#HoF_{Bots}} (p_{ij} \log p_{ij}) \tag{3}$$

where p_{ij} is the probability of action i in column j, and $\#HoF_{Bots} \in \mathbb{N}$ is the length of the bots' memory. Finally the entropy of the whole set is defined by:

$$H = \sum_{j=1}^{9} H_j \tag{4}$$

The higher the value of H the greater the diversity of the set. To determine the diversity's contribution to a specific solution, we calculate the value of entropy with this solution inside the set. Then we find the corresponding value with this solution outside the set, and finally, the difference between these two values represents the contribution of diversity.

This notion of genotypic distance is similar to Novelty Search [28], as both aim to maintain the diversity in a set of solutions. However, our proposal is different because diversity is not an objective in our search process but rather it is used as a policy to keep the memories (i.e., HoFs) to a manageable size.

Regarding maps, the calculation is done by the analysis of the average distance between the planets of the map with respect to all the other maps in the memory as follows:

$$D(m_j) = \frac{\sum_{i=1}^{k}(S_{ij})}{k} \tag{5}$$

where $k \in \mathbb{N}$ is the cardinality of maps' HoF which the individual m_j is a member, and $\forall i \in HoF\{m_{i=0}, m_{i=1}, ..., m_{i=k}\}$; $i \neq j$ we calculate:

$$S_{ij} = \frac{(eS_{ij}) + (eS_{ji})}{2} \tag{6}$$

where eS_{ij} is the average of the Euclidean distance of each planet contained in the $i-th$ map (i.e. m_i in HoF of maps population) to each planet located in $j-th$ map (i.e m_j member in HoF); and eS_{ji} represents the same distance concept but this time starting from the $j-th$ planet as the origin for calculating the Euclidean value.

2) MuCCCo-Quality: The measure of quality for Planet Wars was defined based on our game experience, in the case of bots, as the number of defeats that a member in the champions memory suffered in the previous coevolutionary step. For maps, the quality was defined as the number of defeats that each *fixed rival bot* suffered when he had to face enemies on this map also in the previous coevolutionary step. In what follows, and abusing the language, we will say that a *map suffers a defeat against a game strategy* b (i.e., a bot) if the *fixed rival bot* is defeated by b in the map.

3) MuCCCo-U: This variant is based on a modification of the fitness functions defined in (1) and (2). This change is inspired by the Competitive Fitness Sharing (CFS) method [4]. The main idea is that a victory against opponent b can be considered more relevant if b beats a significant number of opponents. So, given a domain $d \in \{Bots, Maps\}$, a penalization value N for each individual i (for $1 \leqslant i \leqslant n$) in the population Pop_d is then calculated as follows:

$$N_i = 1 - \frac{1}{k}\sum_{j=1}^{k}\frac{v_{ij}}{V(j)} \tag{7}$$

where $v_{ij} = 1$ if the i-th individual of the population *defeats*[4] the j-th champion in the rival HoF (whose cardinality is k) and 0 otherwise; and

$$V(j) = \sum_{i=1}^{\#Pop_d} v_{ij} \tag{8}$$

is the number of individuals in the population which defeat the j-th opponent of the rival HoF. As a consequence, $N_i \approx 0$ if the i-th candidate defeats all opponents of the rival HoF and the i-th solution itself is one of the few candidates to do so; $N_i = 1$ if doesn't defeat any opponent; and $0 < N_i < 1$ depending on how many times it wins and how common it is to beat certain opponents. The fitness of a candidate i is then computed as follows:

$$F_i = P_i - \omega N_i \tag{9}$$

where P_i is the result obtained in the battles by applying (1) or (2) depending on whether the domain to be considered is $Bots$ or $Maps$, and $\omega \in \mathbb{N}$ is a coefficient that scales N_i in order to make it meaningful with respect to the value P.

V. Experiments and Analysis

This section describes the experimental analysis conducted on the Planet Wars game. We consider two instances for each algorithm (MuCCCo-Div, MuCCCo-Qua, and MuCCCo-U) that vary according to the value of $\alpha \in \{10\%, 50\%\}$. The notation *MuCCCo-Variant-*α (where $Variant$ $\in${*Div*, *Qua*, *U*} is used to denote each of the instances. In all cases, we set $\lambda = 3$ and performs 10 runs per algorithm instance using a steady-state genetic algorithm (GA) with the aim of finding winning solutions (maps or bots) with respect to the rival HoF. This employs binary tournament for selection, uniform crossover, bit-flip mutation and elitist replacement. At the beginning of the coevolutionary cycle, *map 10* (from the collection of maps designed for the original Google AI Challenge Competition 2010) is set as the *initial map opponent* (i.e. the map added initially to the maps' HoF). We also chose the *ProspectorBot*, from the set of bots that were originally provided as *example bots* in the competition because it is a fairly offensive strategy that can help to produce bots with fighting spirit. Next, we detail the configuration of the experiments, and analyze the results obtained.

[4]Remember that, in the case that $d = Maps$, we consider that a map 'defeats' a specific bot b if the FRB is defeated by the bot b when both the FBR and b are faced on the map. In addition, we will use the term 'rival HoF of maps' (resp. bots) as synonym of the HoF_{Bots} (resp. the HoF_{Maps}).

A. Configuration of the Experiments

All experiments were executed using the Planet Wars engine (version 1.2) [5]. The parameters used in the experiments are as follows: $maxFailCoevolutions$ (set to 10) represents the limit of continuous coevolutions that one of the domains (i.e., bots or maps) can consume without finding a champion solution. The timeout condition (line 3 in Algorithm 1) is associated with $maxCoevolutions$ (set to 100), which indicates the maximum number of total coevolutions that the algorithm execution can employ. $maxEvaluations$ (3000) is the limit of evaluations i.e., $timeout = (nCoev > maxCoevolutions)$ or $(numEvaluations > maxEvaluations)$, where $numEvaluations$ represents the number of evaluations consumed by that execution of the algorithm. Other parameters are: Mutation Probability ($p_M = 0.01$), Crossover probability ($p_X = 1.0$), ϕ is the threshold value to consider a solution as winning ($\phi Bots = 1600$, $\phi Maps = 1250$), $maxGen = 50$, $popSize = 30$ and $\lambda = 3$.

We have compared the results obtained by the different executions in the instances by applying the Kruskal-Wallis test [29] (a non-parametric statistical test which allows a comparison of different distributions that are independent from each other) with a significance of 95%. Moreover, in those cases in which this test detected significant differences in the distributions we additionally performed multiple tests using the Dunn$-$Sidak method [30] for determining which pairs are significantly different and which are not.

B. Analysis of the Results

Figures 3 and 4 show the fitness of the best champion found in each independent execution of the algorithm instances for the populations of maps and bots, respectively.

The Kruskal-Wallis test confirms that the differences between $Qua - 10$ and the 'U' versions (i.e., $U - 10$ and $U - 50$) are statistically significant (with $p-$value$= 9.0524e - 007$) and, according to the bot population results (Figure 3), the algorithms working on 'quality' values obtain the best results. These are followed by those based on the 'diversity' metrics, while the combined versions (i.e., the 'U' versions) show a less-competitive behavior. This last assertion might be made because in 'U' versions it is more difficult to obtain a high fitness value due to the penalty that is applied to the score obtained by the individuals according to Eq.(9). Although this penalty affects the numerical value of fitness it should not affect the robustness of the solution; it should even, in theory, improve the focus in the search process.

With respect to the maps' fitness displayed in Figure 4, the behavior of the algorithms is very similar to the bots' results. Once again, a clear distinction between the three families of algorithms is seen, the Multicompare test showed that the "MuCCCo-Div" results are in the middle of the mean ranks and there are statistically significant differences between $Qua - 10$ and the 'U' versions (with $p-$value$= 4.5878e - 006$). In order to compare the algorithms from another point of view, Figure 5 presents the best bot fitness, the best map fitness and the number of evaluations consumed in the execution (in axes X, Y, and Z respectively) for each independent execution. Here each point (x, y, z) corresponds to one of the 10 executions performed by each algorithm instance. For clarity, the

[5]http://planetwars.aichallenge.org/starter_packages.php

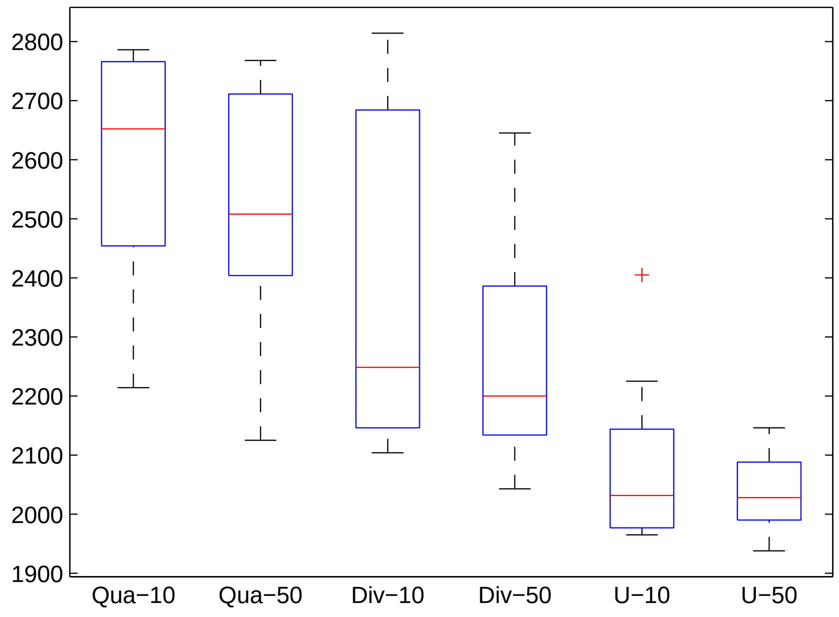


Fig. 3. Best bot fitness distributions: $Qua-10$ obtained the best results and showed statistically significant differences with respect to $U-10$ and $U-50$.

algorithms are grouped in families, where each family comprises 20 points corresponding to the 10 executions of the algorithm for both $\alpha = 10\%$ and $\alpha = 50\%$. In addition, we have considered the families of algorithms because the instances of our prime algorithm distinguished in terms of their results as we have shown previously, and because no significant differences between instances of the same family were found. Considering a multi-objective approach, note that there are only two non-dominated solutions in the Pareto front, one belonging to MuCCCo-Qua and the other to MuCCCo-Div. By analyzing them in depth, we note that in both cases the dominance is due to a low value in the Z axis (i.e., number of evaluations consumed to obtain the solutions) which corresponds to very short coevolutionary cycles. In such a case, this result may be interpreted as a sign of an early stagnation of the search process. However, considering that the fitness values achieved in these short cycles correspond to the “best values” we could also think that there is a rapid convergence towards robust solutions. It may also be the case that in large and short cycles the algorithms find similar fitness values which could be a sign that the search process is affected by a loss of requirements and so it only finds mediocre solutions. To complement this analysis, in the following paragraphs the behavior of another ‘robustness’ indicator for bot individuals is used to help us reach more accurate conclusions. Table I presents the results of the robustness test for bot individuals. Here, the 10 bot champions obtained in the independent executions of the algorithm instances, were pitted against each other in an *All vs. All* tournament, and the results of each family of algorithms (i.e., 20 results for each family) were grouped together.

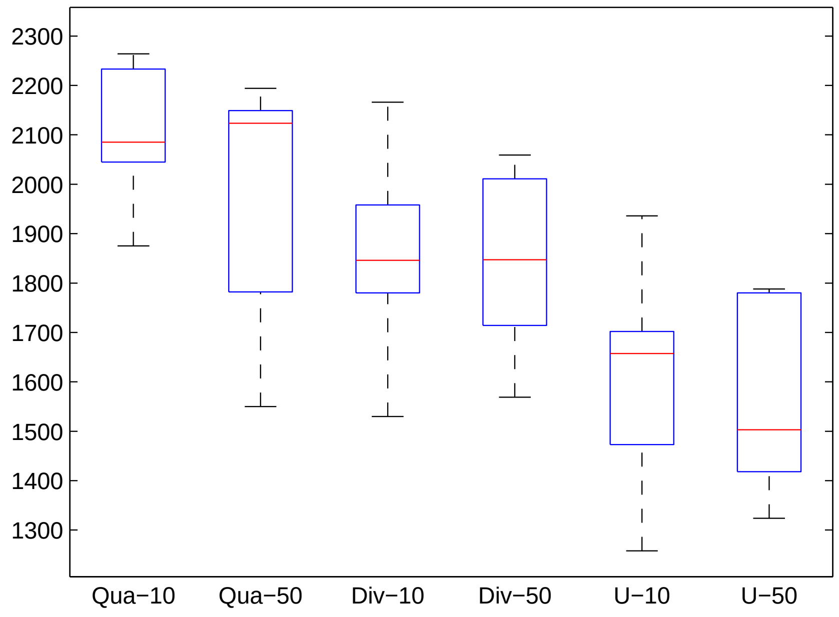


Fig. 4. Best map fitness distributions: $Qua-10$ obtained the best results and showed statistically significant differences respect to $U-10$ and $U-50$.

In other words, each individual competes against the other 20 champions of the opponent algorithm family, and, in each confrontation, two battles (each of them centered on a distinct scenario) are executed. The game scenarios were two maps, randomly chosen from the original set given in the Planet Wars starter package. Columns in Table I show, respectively, the algorithm families that competed against each other, the number of victories that were obtained by each of them and the percentage that each value represents with respect to the total number of battles. It also shows the number of draws with the corresponding percentage. We note that 800 battles were performed in total in each algorithm confrontation (i.e., 20 individuals $\times$ 20 opponents $\times$ 2 maps).

Note also that we have differentiated the battles according to the order of the players (i.e., first player as Player1 (P1) and second player as Player2 (P2)) which is necessary because when we execute the Planet Wars' game engine sometimes the player's position on the map can result in an advantage or disadvantage in the match. Table I shows a slight variation (between 1 and 2 per cent) with respect to the number of victories and defeats when we pit the same bots against each other on the same maps and only permute their positions, and so we conclude that the three families behave similarly, in both cases (i.e., for two player positions) MuCCCo-Diversity seems to obtain the most difficult set to overcome and MuCCCo-U is the most vulnerable in the competitions.

Next we repeat the previous test, using the same bots that have been evaluated before but this time the battles take place on two maps that have been generated by the algorithms implemented, and which have been chosen at

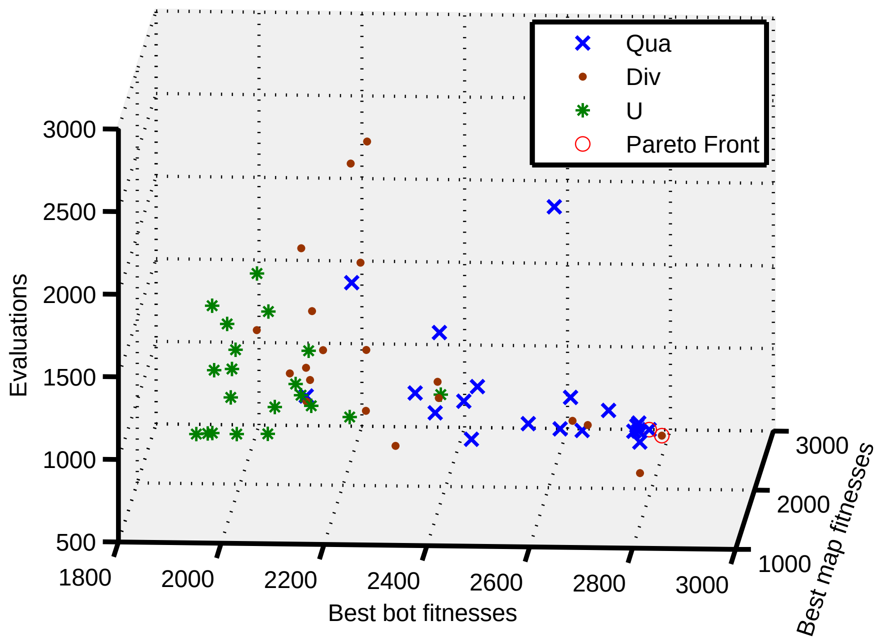


Fig. 5. A Pareto Front representation.

TABLE I
RESULTS OF ALL (VS) ALL TOURNAMENT ON MAP30 AND MAP70 (FROM THE PLANET WARS MAPS' SET).

Player1	Player2	Victories-P1	Victories-P2	Draws
Qua	Div	203 (25%)	211 (26%)	386 (49%)
Qua	U	287 (36%)	231 (29%)	282 (35%)
Div	Qua	224 (28%)	217 (27%)	359 (45%)
Div	U	284 (36%)	193 (24%)	323 (40%)
U	Qua	218 (27%)	286 (36%)	296 (37%)
U	Div	199 (25%)	299 (37%)	302 (38%)
Qua	Qua	220 (27.5%)	196 (24.5%)	384 (48%)
Div	Div	192 (24%)	216 (27%)	392 (49%)
U	U	272 (34%)	240 (30%)	72 (36%)

TABLE II

RESULTS OF ALL (VS) ALL TOURNAMENT ON TWO MAPS CHOSEN AT RANDOM FROM THE CHAMPIONS' SETS GENERATED BY OUR ALGORITHMS.

Player1	Player2	Victories-P1	Victories-P2	Draws
Qua	Div	306 (38%)	466 (58%)	28 (4%)
Qua	U	307 (38%)	458 (57%)	35 (5%)
Div	Qua	274 (34%)	516 (65%)	10 (1%)
Div	U	324 (41%)	427 (53%)	49 (6%)
U	Qua	231 (29%)	544 (68%)	25 (3%)
U	Div	265 (33%)	477 (60%)	58 (7%)
Qua	Qua	152 (19%)	648 (81%)	0 (0%)
Div	Div	96 (12%)	704 (88%)	0 (0%)
U	U	120 (15%)	672 (84%)	8 (1%)

random from the set of map individuals classified as winners. So, looking at Table II we observe a very similar behavior in the outcomes of individuals who are positioned as Player 2. Note that in all cases the maps clearly favor the second player, and this time there is a notable variation in the results when the positions of the bots in the battles are interchanged, the numbers of victories/defeats for the same bots' family always vary between 20 and 30 per cent (see the differences between the cells with the same color). Note how, unlike the previous table, the number of wins (on the same family of bots) increases considerably when the positions of the bots in the battles are interchanged. It is noteworthy that our algorithm coevolution in maps aims to find solutions to ensure the victory of the FRB in the HoF individual opponents (as mentioned in the previous sections), and it is precisely this FRB that always occupies the position of Player 2 in the battles that are executed during the evaluation process. Therefore, this interesting behavior suggests how the learning process that emerges in the mapping population converges towards the creation of more favorable structures for Player 2. Next some tests were applied to our maps so as to analyze them in more detail.

First, we show the results from a geometrical evaluation of the maps, for this we use the same maps that are analyzed in Figure 4 (which were the best individuals obtained by each algorithm in the experiments described above) but this time instead of having six algorithms we have just 3 because they are grouped by families (i.e "Div", "Qua", "U") and another set is added that contains 20 maps (which were chosen at random) from the official maps' set of Planet Wars. Some of the measurements used for this testing are based on the maps aesthetic analysis presented in [31]. These are indicators related to the spatial distribution of the planets and their features, such as number of ships and the growth rate. The first test explored the differences between the maps sets with respect to the average distance from the planets to the home planet of Player 2 in each map, this distance is calculated by the Euclidean distance formula. Figure 6 shows that in our maps the average distance is significantly lower than the Google maps set. The values distributions of our maps behave similarly, no one set stands out for finding especially radical solutions, but the test shows that there are significant differences between our maps and the Google maps

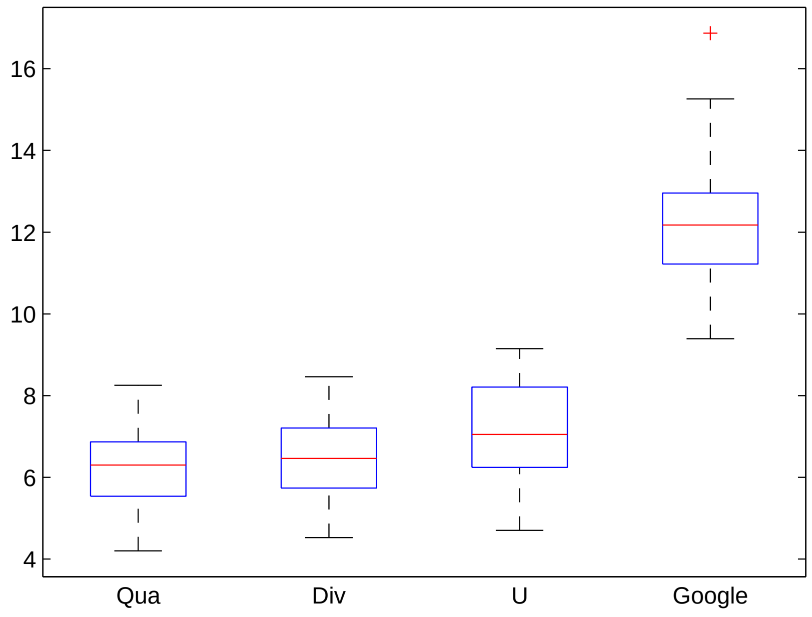


Fig. 6. Statistical analysis of the average distance from the planets to Player 2's home planet for each map in the maps' sets.

set. A second test was applied to the maps. It focused on the "number of ships" and "growth rate" and checked whether there was a relationship between the values assigned to these features when the maps were created by our algorithm. Let s_i and w_i, respectively, be the number of ships and growth rate of the i_{th} planet on a map (N is the number of planets in this map), then we specify the average and standard deviation of this feature (μ_s and σ_s, respectively) and the Pearson's correlation between the number of ships and the growth rate on ρ as follows:

$$\mu_s = \sum_{i=1}^{N} s_i \tag{10}$$

$$\sigma_s = \sqrt{\frac{\sum_{i=1}^{N}(s_i - \mu_s)^2}{N}} \tag{11}$$

$$\rho = \frac{\sum_{i=1}^{N} s_i w_i - N\mu_s\mu_w}{N\sigma_s\sigma_w} \tag{12}$$

According to Figure 7 in all distributions the relationships between the analyzed variables is positive, for the cases of the "Div" and "U" sets they display less dispersion in the values but it is not a significant difference with respect to the other sets. To sum up, it was found that there is a relationship which may be classified as "high" because in all the cases the median value is larger than 0.5, this is an expected result because the correlation between the size and the number of ships in a planet is reasonable.

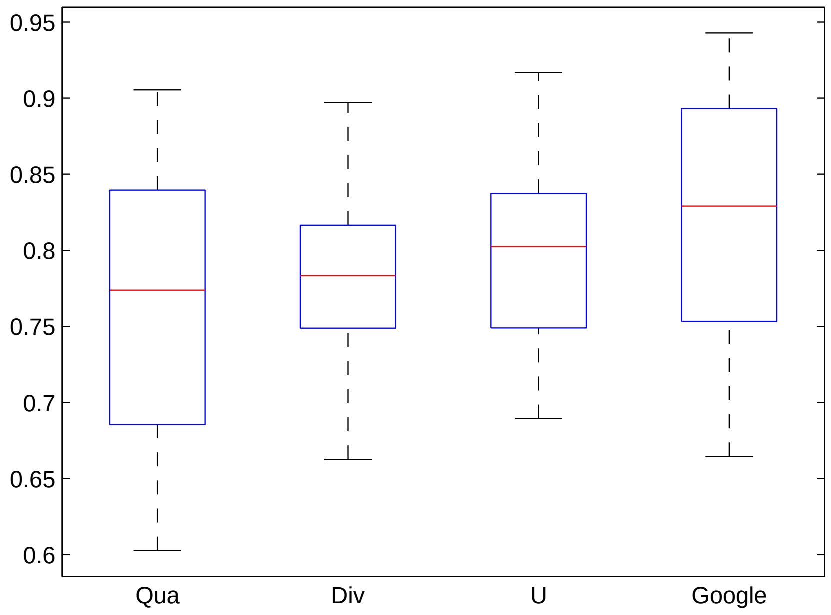


Fig. 7. Statistical analysis of the Pearson Correlation between “number of ships” and “growth rate”.

Lastly we show the analysis of the diversity in the maps sets using the map diversity metric defined in Equation 5 which computes the average distance between the planets of a map with respect to all the other maps in the set. So we calculate the diversity metric D_i for each map in the set according to Equation 5, then the average of the diversity values is computed for each set an it is denoted as μ_d, and the standard deviation σ_d of these diversity values is obtained, as follows:

$$\sigma_d = \sqrt{\frac{\sum_{i=1}^{N}(D_i - \mu_d)^2}{N}} \tag{13}$$

Figure 8 shows a clear distinction between the Google maps set and our maps, the first has less dispersion in its values distribution and it produces significant differences with regards to the maps set of our algorithms which are more disperse. This result may be an indicator that our algorithms are able to maintain an adequate diversity level in the HoF, in fact, note how the distribution of the “Div” family has the greatest dispersion because this variant of algorithms (i.e. “Div10” and “Div50”) pay particular attention to the diversity control in the HoF during the coevolutionary process. So far, we have shown those results which have been useful in our research, in the next section they will be summarized, trying to establish relationships and reach conclusions.

Next, we conducted three additional experiments, one to show the capacity of the system to evolve competitive bots, another to assess the adequacy of our method to cope with a set of fixed rival bots (for instance to specialize content/bots with respect to several adversaries or profiles of players), and the third other to demonstrate the

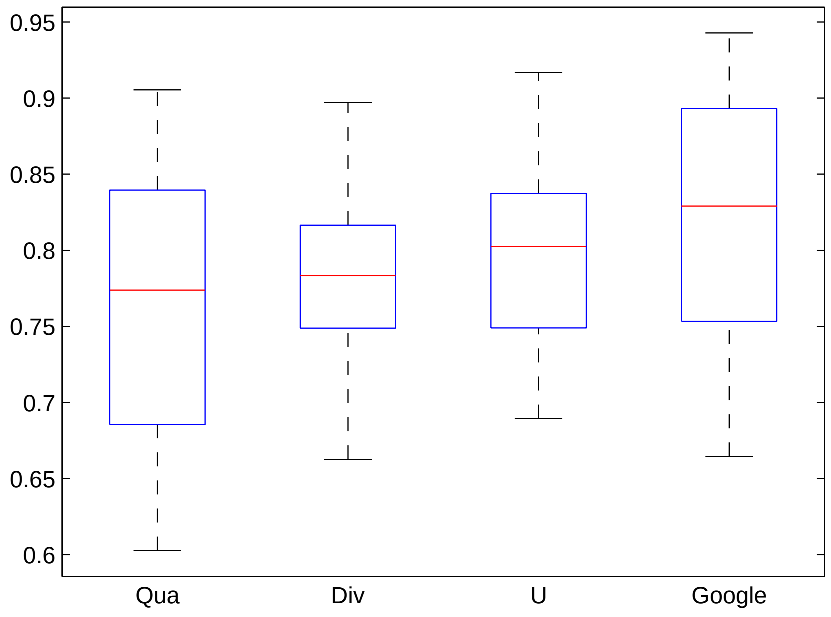


Fig. 8. Statistical analysis of the diversity in the maps sets.

adaptation of our maps to favor an FRB when this is faced with one of the best experts known in the game. In the first case, we fixed the FRB to be a highly specialized bot, namely, the GeneBot [26], [27]. This bot is mentioned in Subsection IV-A and represents the best method described in the scientific literature for playing Planet Wars. The encoding of GeneBot consists of a set of numeric values, and each bot to evolve, tries to optimize them. GeneBot had been subjected to a computationally expensive optimization process so that the task to improve it seemed to be hard. In the second experiment, we considered three FRBs: GeneBot, ProspectorBot, and RandomBot (i.e., other bot from the Google bot's set for PlanetWars). All the variants of 'MuCCCo' were considered, and each experiment was run 10 times. Table III (Rows 1-18) shows the results from the All vs All tournament between 15 victorious individuals (taken from the execution of the different algorithm variants), and where we used two balanced maps (obtained from the set of Google maps) that were not involved in the search process. The prefix 'Many' (resp. 'Fixed') indicates the cardinality of FRB's set is 3 (resp. 1). Note also that the original version of GeneBot is also included in the tournaments. In all cases, the individuals obtained from 'Fixed' instances are more robust in the battles and show the best performances against their adversaries. The reason might be that a single FRB leads an efficient specialization of the solutions during the search process whereas the use of many FRB generates less specialized solutions (at the expense of obtaining a wider profile). Observe that GeneBot is not an invincible adversary and that even our bots evolved via our 'Fixed' versions, clearly beat it (particularly, the version obtained by 'Fixed U'). This is a promising result, especially if we consider that our bots were not optimized in the two

TABLE III

RESULTS OF ALL (VS) ALL TOURNAMENT (IN GOOGLE MAPS, EXCEPT LAST ROW * IN TWO EVOLVED MAPS). 'FIXED' MEANS THE BOTS WERE EVOLVED VIA ONE FRB (GENEBOT). 'MANY' MEANS THE BOTS WERE EVOLVED VIA THREE FRBS (GENEBOT, PROSPECTORBOT, RANDOMBOT).

Player 1	Player2	Victories-P1	Victories-P2	Draws
ManyDiv	FixedDiv	129	321	0
ManyDiv	FixedQua	139	311	0
ManyDiv	FixedU	83	367	0
ManyQua	ManyDiv	256	194	0
ManyQua	ManyU	210	240	0
ManyQua	FixedDiv	170	280	0
ManyQua	FixedQua	146	304	0
ManyQua	FixedU	102	348	0
ManyU	ManyDiv	276	174	0
ManyU	FixedDiv	182	268	0
ManyU	FixedQua	169	281	0
ManyU	FixedU	111	339	0
ManyDiv	GeneBot	5	25	0
ManyQua	GeneBot	9	21	0
ManyU	GeneBot	15	15	0
FixedDiv	GeneBot	15	15	0
FixedQua	GeneBot	19	11	0
FixedU	GeneBot	25	5	0
ZerlingRush	GeneBot	29	1	0
ZerlingRush	FixedU	30	0	0
ZerlingRush*	FixedU	0	30	0

confrontation maps.

In the third experiment, we considered the bot *ZerlingRush* (developed by *GreenTea*) that was ranked 8th in the Google AI Contest 2010 (with over 4600 submissions) and was publicly available as a jar file. This robot beat all our bots (including the original GeneBot) in the set of balanced maps provided by Google. Rows 19 and 20 in Table III show its manifest superiority wrt. GeneBot and our 15 evolved solutions obtained from $FixedU$. Then, we fixed *ZerlingRush* as the FRB in $FixedU$, and let Genebot co-evolve to beat it at the same time as the game maps evolved to disfavor it. These 15 new evolved GeneBot beat *ZerlingRush* in our generated (non-corrupted) maps as shown in the last row of the Table (marked with *). This is a proof that the maps adapt to satisfy the objectives we have established.

VI. CONCLUSION

This paper has described an evolutionary model that allows the co-evolution of game AI and game content via competition. The intrinsic differences between these two domains prevent direct confrontation between their candidates so that the model has a set of fixed opponents that play the role of adversaries in the evolution of bots, and allies in the optimization of game content. The overall schema can be used to automate the generation of game AI as well as to dynamically adapt the game for specific objectives via procedural content generation.

We have considered *Planet Wars* as the testbed, and have shown that our proposal can not only generate maps that favor (resp., disfavor) allies (resp., adversaries) but also game AI that performs very efficiently (on a par with the best game AI reported in the scientific literature for playing the game).

The number of generations needed to adapt the game AI and content can vary significantly according to the underlying game and to the expected quality of the results. The efficiency of the application of our algorithm depends thus on many factors and, as in any other evolutionary algorithm, one can tune the basic parameters (such as population size, number of co-evolutionary iterations, number of FRBs, and the sizes of the HoFs) to significantly reduce the running time at the expense of solution quality.

In future work, we will focus on testing other multi species co-evolutionary approaches to explore the parallel generation of different contents for games, trying to reduce the vulnerabilities of this model. We will pay particular attention to the use of other populations with the aim of generating more complex game components such as game rules.

ACKNOWLEDGMENT

This work is partially funded by Junta de Andalucía (project P10-TIC-6083 DNEMESIS), Ministerio Español de Economía y Competitividad (project TIN2014-56494-C4-1-P, UMA-EPHEMECH), and Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

REFERENCES

[1] R. Lara-Cabrera, C. Cotta, and A. Fernández-Leiva, “A review of computational intelligence in rts games,” in *IEEE Symposium on Foundations of Computational Intelligence*. Singapore: IEEE, 2013, pp. 114–121.

[2] R. Lara-Cabrera, C. Cotta, and A. Fernández-Leiva, “An analysis of the structure and evolution of the scientific collaboration network of computer intelligence in games,” *Physica A: Statistical Mechanics and its Applications*, vol. 395, no. 0, pp. 523 – 536, 2014.

[3] S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, “Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191),” *Dagstuhl Reports*, vol. 2, no. 5, pp. 43–70, 2012.

[4] C. Rosin and R. Belew, “New methods for competitive coevolution,” *Evolutionary Computation*, vol. 5, no. 1, pp. 1–29, 1997.

[5] M. Nogueira, C.Cotta, and A. J. Fernández-Leiva, “An analysis of hall-of-fame strategies in competitive coevolutionary algorithms for self-learning in rts games,” in *Learning and Intelligent Optimization - 7th International Conference*, ser. LNCS. Catania, Italy: Springer, 2013, pp. 174–188.

[6] M. N. Collazo, C. Cotta, and A. J. Fernández Leiva, “Virtual player design using self-learning via competitive coevolutionary algorithms,” *Natural Computing*, vol. 13, no. 2, pp. 131–144, 2014.

[7] G. Yannakakis, “Game ai revisited,” in *Computing Frontiers Conference*. Caligari, Italy: ACM, 2012, pp. 285–292.

[8] S. M. Lucas, “Computational Intelligence and AI in Games: A New IEEE Transactions,” *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 1, no. 1, pp. 1–3, 2009.

[9] R. Lara-Cabrera, C. Cotta, and A. J. Fernández-Leiva, “On balance and dynamism in procedural content generation with self-adaptive evolutionary algorithms,” *Natural Computing*, vol. 13, no. 2, pp. 157–168, 2014.

[10] M. Hendrikx, S. Meijer, J. V. D. Velden, and A. Iosup, “Procedural content generation for games: A survey,” *TOMCCAP*, vol. 9, no. 1, p. 1, 2013.

[11] D. Floreano, S. Nolfi, and F. Mondada, “Competitive co-evolutionary robotics: From theory to practice,” in *Fifth International Conference on Simulation of Adaptive Behavior on From Animals to Animats 5*, Zurich, Switzerland, 1998, pp. 515–524.

[12] R. Dawkins and J. R. Krebs, “Arms races between and within species,” *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 205, no. 1161, pp. 489–511, 1979.

[13] P. J. Angeline and J. B. Pollack, “Competitive Environments Evolve Better Solutions for Complex Tasks,” in *5th International Conference on Genetic Algorithms (ICGA93)*, S. Forrest, Ed. Urbana-Champaign, IL, USA: Morgan Kaufmann, 1993, pp. 264–270.

[14] C. Reynolds, “Competition, coevolution and the game of tag,” in *Proceedings of Artificial Life IV*, Brooks and P. Maes, Eds. Cambridge, Massachusetts: MIT Press, 1994, pp. 59–69.

[15] K. Sims, “Evolving 3D Morphology and Behavior by Competition,” *Artificial Life*, vol. 1, no. 4, pp. 353–372, 1994.

[16] D.Ashlock, W. Ashlock, S. Samothrakis, S. M. Lucas, and C. Lee, “From competition to cooperation: Co-evolution in a rewards continuum,” in *IEEE Conference on Computational Intelligence and Games*. Granada, Spain: IEEE, 2012, pp. 33–40.

[17] G. Smith, P. Avery, R. Houmanfar, and S. J. Louis, “Using co-evolved RTS opponents to teach spatial tactics,” in *IEEE Conference on Computational Intelligence and Games*, G. N. Yannakakis *et al.*, Eds., Copenhagen, Denmark, 2010, pp. 146–153.

[18] C. Miles and S. J. Louis, “Co-evolving real-time strategy game playing influence map trees with genetic algorithms,” in *International Congress on Evolutionary Computation*. Vancouver: IEEE, 2006.

[19] P. M. Avery and Z. Michalewicz, “Static experts and dynamic enemies in coevolutionary games,” in *IEEE Congress on Evolutionary Computation*. Singapore: IEEE, 2007, pp. 4035–4042.

[20] A. B. Cardona, J. Togelius, and M. Nelson, “Competitive coevolution in ms. pac-man,” in *IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 1403–1410.

[21] J. Togelius, P. Burrow, and S. M. Lucas, “Multi-population competitive co-evolution of car racing controllers,” in *IEEE Congress on Evolutionary Computation*. Singapore: IEEE, 2007, pp. 4043–4050.

[22] M. Cook, S. Colton, and J. Gow, “Initial Results from Co-operative Co-evolution for Automated Platformer Design,” in *Applications of Evolutionary Computation(EvoGAMES)*, ser. LNCS, C. D. Chio *et al.*, Eds., vol. 7248. Málaga,Spain: Springer, 2012, pp. 194–203.

[23] A. Liapis, G. Yannakakis, and J. Togelius, “Enhancements to constrained novelty search: two-population novelty search for generating game content,” in *Genetic and Evolutionary Computation Conference*, C. Blum and E. Alba, Eds. Amsterdam, The Netherlands: ACM, 2013, pp. 343–350.

[24] R. Lara-Cabrera, C.Cotta, and A. Fernández-Leiva, “A procedural balanced map generator with self-adaptive complexity for the real-time strategy game planet wars,” in *Applications of Evolutionary Computation - 16th European Conference, EvoApplications 2013*, ser. LNCS, vol. 7835. Vienna, Austria: Springer, 2013, pp. 274–283.

[25] M. Buro, “Call for AI research in RTS games,” in *4th AAAI Workshop on Challenges in Game AI*, D. Fu and J. Orkin, Eds., San Jose, California, 2004, pp. 139–141.

[26] A. Fernández-Ares, A. M. Mora, J. J. M. Guervós, P. García-Sánchez, and C. Fernándes, “Optimizing player behavior in a real-time strategy game using evolutionary algorithms,” in *IEEE Congress on Evolutionary Computation*. New Orleans, LA, USA,: IEEE, 2011, pp. 2017–2024.

[27] A. M.Mora, A. Fernández-Ares, J. J. Merelo, P. García-Sánchez, and C. M. Fernándes, “Effect of Noisy Fitness in Real-Time Strategy Games Player Behaviour Optimisation Using Evolutionary Algorithms,” *J. Comput. Sci. Technol.*, vol. 27, no. 5, pp. 1007–1023, 2012.

[28] J. Lehman and K. O. Stanley, “Abandoning objectives: Evolution through the search for novelty alone,” *Evol. Comput.*, vol. 19, no. 2, pp. 189–223, 2011.

[29] W. Kruskal and W. Wallis, “Use of ranks in one-criterion variance analysis,” *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[30] R. Sokal and J. Rohlf, *Biometry: the principles and practice of statistics in biological reseach*. W.H. Freeman and Company, 1995.

[31] R. Lara-Cabrera, C. Cotta, and A. J. Fernández-Leiva, “Evolving aesthetic maps for a real time strategy game,” in *First Spanish Symposium on Entertainment Computing*, Madrid, Spain, 2013.

CAPTIONS OF THE FIGURES

Figure nogue1.pdf

Caption: Multi-content generation coevolutionary schema for a n-player game.

Main caption for SubFigure nogue2a and SubFigure nogue2b

Caption: Configuration of the battle scenario for both populations, note that the object under evaluation is highlighted with dashed lines.

SubFigure nogue2a.pdf

Sub-caption: Bot evaluation.

SubFigure nogue2b.pdf

Sub-caption: Map evaluation.

Figure nogue3.pdf

Caption: Best bot fitness distributions: $Qua - 10$ obtained the best results and showed statistically significant differences with respect to $U - 10$ and $U - 50$.

Figure nogue4.pdf

Caption: Best map fitness distributions: $Qua - 10$ obtained the best results and showed statistically significant differences respect to $U - 10$ and $U - 50$.

Figure nogue5.pdf

Caption: A Pareto Front representation.

Figure nogue6.pdf

Caption: Statistical analysis of the average distance from the planets to Player 2's home planet for each map in the maps' sets.

Figure nogue7.pdf

Caption: Statistical analysis of the Pearson Correlation between "number of ships" and "growth rate".

Figure nogue8.pdf

Caption: Statistical analysis of the diversity in the maps sets.

CAPTIONS OF THE TABLES

Table nogue.t1.pdf

Caption: Results of All (vs) All tournament on map30 and map70 (from the Planet Wars maps' set).

Table nogue.t2.pdf

Caption: Results of All (vs) All tournament on two maps chosen at random from the champions' sets generated by our algorithms.

Table nogue.t3.pdf

Caption: Results of All (vs) All tournament (in Google maps, except last row * in two evolved maps). 'Fixed' means the bots were evolved via one FRB (GeneBot). 'Many' means the bots were evolved via three FRBs (GeneBot, ProspectorBot, RandomBot).

BIOGRAPHIES

Mariela Nogueira-Collazo obtained the Bachelors degree in Computer Science by the University of Computer Science of The Havana, Cuba; and the Masters degree in Software Engineering and Artificial Intelligence from the University of Málaga (UMA), Spain. Currently she is pursuing the Ph.D. in UMA and her research cover the application of artificial intelligence in videogames context.

Carlos Cotta Porras obtained his MSc and PhD in Computer Science from the University of Málaga (UMA), Spain in 1994 and 1998 respectively. He holds a tenured Professorship in the Department of *Lenguajes y Ciencias de la Computación* of this University since 2001. His main research areas involve metaheuristic optimization -in particular hybrid and memetic approaches- with a focus on both algorithmic and applied aspects (particularly combinatorial optimization) as well as complex systems.

Antonio J. Fernández-Leiva received, in 2002, the PhD degree in Computer Science from the University of Málaga (UMA), where he is currently associate professor in the *Lenguajes y Ciencias de la Computación* department. In the past, he worked in private companies as computer engineer. His main areas of research involve both the application of metaheuristics techniques to combinatorial optimization and the employment of Computational Intelligence in Games. He also leads a Master on Design and Programming of Videogames at UMA.