Using Self-Adaptive Evolutionary Algorithms to Evolve Dynamism-Oriented Maps for a Real Time Strategy Game

Raúl Lara-Cabrera, Carlos Cotta and Antonio J. Fernández-Leiva

Department "Lenguajes y Ciencias de la Computación", ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 Málaga – Spain
{raul,ccottap,afdez}@lcc.uma.es

Abstract. This work presents a procedural content generation system that uses an evolutionary algorithm in order to generate interesting maps for a real-time strategy game, called *Planet Wars*. Interestingness is here captured by the dynamism of games (i.e., the extent to which they are action-packed). We consider two different approaches to measure the dynamism of the games resulting from these generated maps, one based on fluctuations in the resources controlled by either player and another one based on their confrontations. Both approaches rely on conducting several games on the map under scrutiny using top artificial intelligence (AI) bots for the game. Statistic gathered during these games are then transferred to a fuzzy system that determines the map's level of dynamism. We use an evolutionary algorithm featuring self-adaptation of mutation parameters and variable-length chromosomes (which means maps of different sizes) to produce increasingly dynamic maps.

1 Introduction

Videogames, with a total consumer spent of 24.75 billion US dollars in 2011 [1], is a very important pillar of the entertainment industry. Until the last decade, the graphical quality of a game determined its quality but, since then, the attractiveness of video-games has fallen on additional features, such as music, interesting stories and the player immersion into the game. It is difficult to measure how much fun a game is since it depends on each player; however it is related to the player satisfaction: the higher the satisfaction, the higher the fun.

This high satisfaction can be achieved via the automated adaptation of the game in response to the player's needs [7] using computational intelligence (CI) techniques. Traditionally, CI has been applied to generate strategies that define the behaviour of the non-player characters (NPC), but it can be also applied to many other aspects of game development such as computational narratives, player modelling, learning in games, intelligent camera control, and procedure content generation (PCG), among other – see [6].

PCG involves algorithms and techniques devoted to create game content automatically, providing several advantages to game developers, such as reduced memory consumption, the possibility of create endless video-games (i.e. the game changes every time a new game is started) and a reduction in the expense of creating the game content. This work focuses in PCG in the context of the real-time strategy (RTS) game *Planet Wars* by means of evolutionary algorithms (EAs).

Planet Wars is a real-time strategy game based on *Galcon* and used in the *Google AI Challenge 2010*. The objective is to conquer all the planets on the map or eliminate every opponent. Every game takes place on a map on which several planets are scattered. These planets are able to host ships and they can be controlled by any player or remain neutral if no player conquer them. Moreover, planets have different sizes, a property that defines their growth rate (i.e., the number of new ships created every time step, as long as the planet belongs to some player). Players send fleets of ships from controlled planets to other ones. If the player owns the target planet the number of fleet's ships is added to the number of ships on that planet, otherwise a battle takes place in the target planet: ships of both sides destroy each other so the player with the highest number of ships owns the planet (with a number of ships determined by the difference between the initial number of ships). The distance between the planets affects the required time for a fleet to arrive to her destination, which is fixed during the flight (i.e., it is not possible to redirect a fleet while it is flying).

PCG for *Planet Wars* involves in this case generating the maps on which the game takes place. The particular structure of these maps can lead to games exhibiting specific features. In previous work [3,4] we focused on achieving balanced games, i.e., games in which none of the players strongly dominates her opponent. Such balanced games can be of little interest though, due to the lack of action. For this reason, we turn our attention to the evolution of maps resulting in interesting, action-packed games. We use the label dynamism to refer to this property of games. Next section is devoted to analyse the evolution of dynamism-oriented games.

2 Evolution of Maps with Dynamism

To study the evolution of *Planet Wars* maps leading to dynamic games, let us firstly analyse how to capture dynamism within an objective function. Subsequently, we focus on an evolutionary approach optimizing this objective function.

2.1 Capturing Dynamism

In order to evaluate the dynamism of the generated maps we had to specify which are the characteristics that define a dynamic game. To do so, we consider two groups of indicators. The first group reflects dynamism from a resource-based perspective (i.e., we try to relate dynamism with the variation in the amount of resources owned by either player); the second group focuses on confrontations between the players (i.e., dynamism is tried to be captured by the extent to which the players repeatedly clash). More precisely, the indicators for a game i are the following:

– Resource-based:
  - *Game length* T_i: this is the ratio of the maximum number of turns allowed $\tau_{\max}$ that have been played in the current game: $T_i = \tau_i/\tau_{\max}$.
  - *Conquering rate* K_i: this is the ratio of planets which are not neutral at the end of the game.
  - *Reconquering rate* Z_i: Let ζ_{ij} be the number of planets that were owned by a player in turn $j-1$ and conquered by the other player in turn j. Then $Z_i = \frac{1}{\tau_i}\sum_{j=1}^{\tau_i} \zeta_{ij}/n_p$, where n_p is the total number of planets.
  - *Peak difference*: this is a family of variables measuring the maximal amplitude of the variation in any of the resources accounted for, namely planets (π), growth capacity (γ), and ships (ξ). Let $\phi_{ij}^{(a)}$ be the amount of resource ϕ owned by player a in the j-th turn of the i-th game, we record the two points in which the relative difference is best for one player and the other one and sum both quantities, i.e., $\Delta_i^{\phi} = \max_{1\leqslant j\leqslant\tau_i}\{(\phi_{ij}^{(1)} - \phi_{ij}^{(2)})/(\phi_{ij}^{(1)} + \phi_{ij}^{(2)})\} - \min_{1\leqslant j\leqslant\tau_i}\{(\phi_{ij}^{(1)} - \phi_{ij}^{(2)})/(\phi_{ij}^{(1)} + \phi_{ij}^{(2)})\}$

– Confrontation-based:
  - *Battle rate:* B_i this is the ratio of planets under attack throughout the game. Let β_{ij} be the number of planets that were under attack during the j-th turn, then $B_i = \frac{1}{\tau_i}\sum_{j=1}^{\tau_i} \beta_{ij}/n_p$.
  - *Destroyed ships* S_i: this is the ratio of the generated ships that have been destroyed throughout the game. Let χ_i be the number of destroyed ships and ψ_i the number of created ships, then $S_i = \chi_i/\psi_i$.

Since we use a tournament system whereby a number of bots are paired and compete on the map under evaluation, the consider the average value of the above indicators across the N_g total games. We drop the sub-index to denote this average quantity. Subsequently, we have defined a set of fuzzy rules to express dynamism as a function of these indicators. The fuzzy rule base is depicted in Fig. 1. In general the underlying fuzzy sets (LO and HI) are defined so as to hit a maximum at the corresponding end of the value range of the variable under consideration, and decrease linearly towards the other end. The exceptions are Z and B whose usual values are far from the theoretical maximal value 1.0. In this case, we saturate the HI value to 1 when they reach the value 0.1 and 0.35 respectively (these values were empirically determined). For the output variable *dyn* a middle triangular fuzzy set MED is defined, hitting a maximum at 0.5 and linearly decreasing towards both ends. In this case, both HI and LO reach their minimum at this 0.5 value.

2.2 Evolutionary Approach

We have used a self-adaptive evolutionary approach in order to optimize the dynamism of the generated maps. These maps have been encoded in mixed real-integer vectors which define the characteristics of the planets (i.e. position, size and number of ships) in a way that each gene represents a planet. The mutation operator depends on the parameter's type: Gaussian mutation for the real-valued parameters and a method that generates suitable integer mutations [5,8] for the

Resource-based:

1. **if** K **is** HI **and** Z **is** HI **then** *dyn* **is** HI
2. **if** Δ^{π} **is** HI **and** Δ^{γ} **is** HI **and** Δ^{ξ} **is** HI **then** *dyn* **is** HI
3. **if** Δ^{π} **is** HI **and** (Δ^{γ} **is** LO **or** Δ^{ξ} **is** LO) **then** *dyn* **is** MED
4. **if** Δ^{γ} **is** HI **and** (Δ^{π} **is** LO **or** Δ^{ξ} **is** LO) **then** *dyn* **is** MED
5. **if** Δ^{ξ} **is** HI **and** (Δ^{γ} **is** LO **or** Δ^{π} **is** LO) **then** *dyn* **is** MED
6. **if** Δ^{π} **is** LO **and** Δ^{γ} **is** LO **and** Δ^{ξ} **is** LO **then** *dyn* **is** LO
7. **if** K **is** LO **or** Z **is** LO **or** T **is very** LO **then** *dyn* **is** LO

Confrontation-based:

1. **if** B **is** HI **and** S **is** HI **then** *dyn* **is** HI
2. **if** (B **is** HI **and** S **is** LO) **or** (B **is** LO **and** S **is** HI) **then** *dyn* **is** MED
3. **if** B **is** LO **and** S **is** LO **then** *dyn* **is** LO

Fig. 1. Fuzzy rule bases for dynamism.

rest of the parameters. The parameters of these operators have been included into the solutions, thus providing the means for self-adapting them. More precisely, in the case of real-valued parameters $\langle x_1, ..., x_n\rangle$ mutation is done by having $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$ and $x'_i = x_i + \sigma_i \cdot N_i(0,1)$ where σ_i is the mutation parameter for x_i, $\tau' = 1/\sqrt{2n}$, and $\tau = 1/\sqrt{2\sqrt{n}}$. Likewise, integer-valued parameters $\langle z_1, ..., z_m\rangle$ are mutated by having $\varsigma'_i = \max(1, \varsigma_i \cdot \exp(\tau \cdot N(0,1) + \tau' \cdot N(0,1)))$, $\psi_i = 1 - (\varsigma'_i/m)(1 + \sqrt{1 + (\varsigma'_i/m)^2})^{-1}$ and $z'_i = z_i + \lfloor \ln(1 - U(0,1))/\ln(1 - \psi_i)\rfloor - \lfloor \ln(1 - U(0,1))/\ln(1 - \psi_i)\rfloor$ where ς_i is the mutation parameter for z_i, $\tau = 1/\sqrt{2m}$ and $\tau' = 1/\sqrt{2\sqrt{m}}$.

As for recombination, we have used a "cut and splice" operator that recombines two individuals by swapping cut pieces with different sizes that provides new maps which contain a different number of planets in relation to their parents, hence adding again additional self-adapting capabilities to the algorithm.

3 Experimental Results

We have used the DEAP library [2] to implements the EA described previously. The algorithm has employed a population size of 100 individuals and a $(\mu + \lambda)$ generational scheme, with $\mu = 10$, $\lambda = 100$. The bots used to evaluate the individuals were obtained from the Google AI Challenge competition- These bots (*Manwe*[1], *Flagscapper's bot*[2] and *fglider's bot*[3]) ranked in the top 100 (there were over 4600 participants) and have their source code available. The duration of the games was limited to $\tau_{\max} = 400$ turns. Regarding the evaluation of fuzzy rules, we have used the min t-norm, the max t-conorm, and the centre of mass as defuzzification method.

We have run two sets of experiments focusing on the behaviour of the algorithm when optimizing dynamism using either a confrontation-based (CB) or a

[1] `https://github.com/Manwe56/Manwe56-ai-contest-planet-wars`
[2] `http://flagcapper.com/?c1`
[3] `http://planetwars.aichallenge.org/profile.php?user_id=8490`

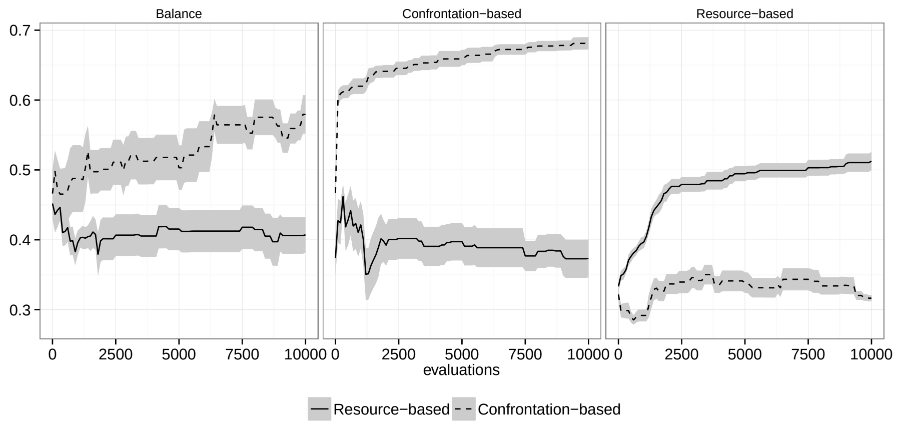

Fig. 2. Evolution of the different objective functions. In the middle and rightmost graph we depict the evolution of dynamism measured in both ways when one of them is subject to optimization (the one indicated in the sub-graph title). The leftmost graph indicates the evolution of balance when either objective function is being optimized. Each line represents the average of 10 runs of the evolutionary algorithm and the shaded area indicates the standard error of the mean.

resource-based (RB) approach. The results are shown in Fig. 2. Let us firstly focus on the middle and rightmost sub-figures which provide an indication on the correlation of both objective functions when one of them is being optimized. It is clear from these plots that both measures are fairly orthogonal, in the sense that when one of them is optimized the other one follows a rather flat trajectory. Thus they can be seen as truly complementary views on game dynamism. Notice also that CB fitness seems to converge faster, likely indicating that confrontations can be induced more easily than wide resource fluctuations by adjusting the map. Another interesting fact is shown in the leftmost sub-figure. Therein we show how balanced are the games when either objective function is being optimized. The measure of balanced was defined elsewhere aiming to analyse its trade-off with a RB version of dynamism, and essentially amounts to measure how the three resources (planets, growth capacity and ships) remain balanced (i.e., their absolute difference is small) for the two players throughout the game. While balance seems to follow a flat trajectory when RB dynamism is being optimized, there is an increasing trend in the case of optimizing CB dynamism. We believe this can be due to the fact that continuous battles prevent the accumulation of resources by either party and push towards their mutual cancellation.

Fig. 3 shows the progress of the variables used to measure the dynamism of the maps during evolution. Unsurprisingly, variables used in the function under optimization in either case exhibit an increasing trend in general. It is more interesting to note some cross-relationships. Firstly, the conquering rate K grows higher in the case of CB fitness than in RB fitness, despite the fact it

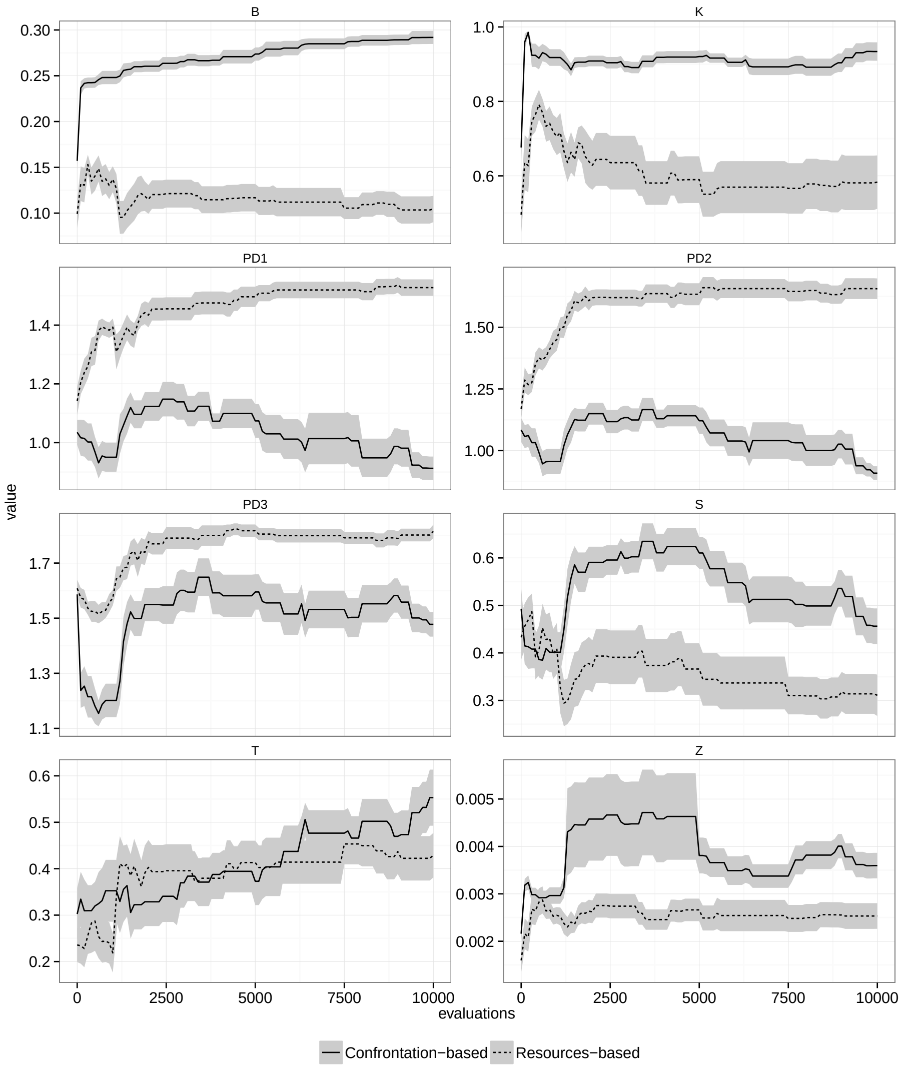

Fig. 3. Evolution of the different variables involved in the rules of the fuzzy sets.

is only explicitly included in the latter. This is side effect of the optimization of the battle rate B: in order to conquer a planet for the first time it must be placed under attack; hence an increase in the number of conquered plants implies another increase in the number of battles, a fact exploited by the evolutionary algorithm. Likewise the reconquering rate is also higher in CB fitness since a high number of battles can eventually lead to numerous planets changing hands (note also that CB fitness heavily revolves around B whereas RB fitness involves

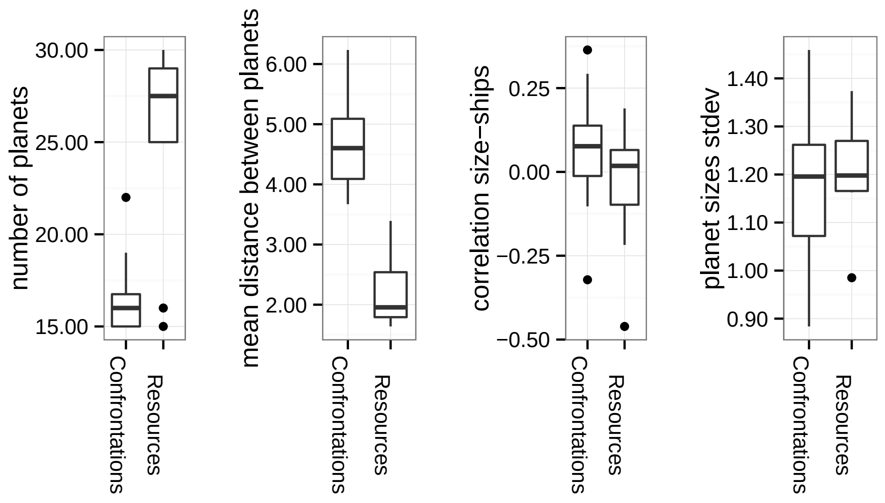

Fig. 4. Several characteristics of the best generated map for every run and objective function: number of planets in the map, average distance between these planets, correlation between planets' sizes and their initial number of ships and standard deviation of the planets' sizes.

a higher number of variables among which different trade-offs can be attained). Both objective functions tend to produce longer games (i.e., higher T); while this is explicitly stated in the RB function, it emerges implicitly in CB fitness since longer games increase the number of battles that take place. It also increases the number of ships ever produced which again forces an increase in the number of battles in order to keep a high ratio of ships destroyed.

Let us finally inspect the characteristics of the evolved maps (Fig. 4). As it can be seen, the maps obtained from optimizing both functions are similar in terms of having a similar correlation between the size of planets and the initial number of ships placed on them, and in terms of the variability of planet sizes. They do however differ in the number of planets and the mean distance among them. The lower number of planets in CB fitness can be explained by the fact that having a low number of planets reduce the expansion possibilities for players, thus forcing them to focus on the same planets more often and hence leading to a higher number of battles. Having a higher distance among planets has the effect of increasing the time-lag between the moment decisions are taken (i.e., ships are dispatched to a target) and the moment ships arrive to their destination. We hypothesize that this larger time-lag introduces a factor of instability by making it more difficult to hold positions and increasing the time of reaction upon attacks, thus promoting more battles to regain control of planets.

4 Conclusion and Future Work

This paper presents a PCG method that is capable of generating maps for the RTS game *Planet Wars*. These maps should fulfil some desirable requirements in terms of dynamism in order to obtain interesting and attractive games. This dynamism has been tackled from two different approaches: a RB approach that looks for a high level of dynamism in players' resources, and a CB approach, that focuses on battles and lost ships. Both approaches have been shown to be orthogonal, thus suggesting their joint optimization (either in a single- or a multi-objective scenario) as a potential line of future research. It is interesting to note the higher correlation of CB fitness with balance. We plan to analyse further this connection by introducing subjective evaluation of the generated maps in a future work, so as to analyse the attractiveness of games for human players.

Acknowledgements This work is partially supported by Spanish MICINN under project ANYSELF (TIN2011-28627-C04-01), by Junta de Andalucía under project P10-TIC-6083 (DNEMESIS) and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

References

1. Entertainment Software Association: Essential facts about the computer and video game industry (2012), `http://www.theesa.com/facts/pdfs/esa_ef_2012.pdf`
2. Fortin, F.A., Rainville, F.M.D., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research 13, 2171–2175 (jul 2012)
3. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: Procedural map generation for a RTS game. In: Leiva, A.F., et al. (eds.) 13th International GAME-ON Conference on Intelligent Games and Simulation, pp. 53–58. Eurosis, Malaga (Spain) (2012)
4. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: A procedural balanced map generator with self-adaptive complexity for the real-time strategy game planet wars. In: Esparcia-Alcázar, A., et al. (eds.) Applications of Evolutionary Computation, pp. 274–283. Springer-Verlag, Berlin Heidelberg (2013)
5. Li, R.: Mixed-integer evolution strategies for parameter optimization and their applications to medical image analysis. Ph.D. thesis, Leiden University (2009)
6. Lucas, S.M., Mateas, M., Preuss, M., Spronck, P., Togelius, J.: Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). Dagstuhl Reports 2(5), 43–70 (2012)
7. Nogueira, M., Cotta, C., Fernández-Leiva, A.J.: On modeling, evaluating and increasing players' satisfaction quantitatively: Steps towards a taxonomy. In: Chio, C.D., et al. (eds.) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 7248, pp. 245–254. Springer-Verlag, Málaga, Spain (2012)
8. Rudolph, G.: An evolutionary algorithm for integer programming. In: Davidor, Y., Schwefel, H.P., Männer, R. (eds.) Parallel Problem Solving from Nature III, Lecture Notes in Computer Science, vol. 866, pp. 139–148. Springer-Verlag, Jerusalem, Israel (1994)