Evolving Aesthetic Maps for a Real Time Strategy Game

Raúl Lara-Cabrera, Carlos Cotta, 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 paper presents a procedural content generator method that have been able to generate aesthetic maps for a real-time strategy game. The maps has been characterized based on several of their properties in order to define a similarity function between scenarios. This function has guided a multi-objective evolution strategy during the process of generating and evolving scenarios that are similar to other aesthetic maps while being different to a set of non-aesthetic scenarios. The solutions have been checked using a support-vector machine classifier and a self-organizing map obtaining successful results (generated maps have been classified as aesthetic maps).

1 Introduction

The video-game industry has become one of the most important component in the entertainment business, with a total consumer spent of 24.75 billion US dollars in 2011 [5]. The quality and appealing of video-games used to rely on their graphical quality until the last decade, but now, their attractiveness has fallen on additional features such as the music, the player immersion into the game and interesting story-lines. It is hard to evaluate how much amusing a game is because this evaluation depends on each player, nevertheless there is a relationship between player satisfaction and fun.

Procedural Content Generation (PCG) [12] includes algorithms and techniques dedicated to produce 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. These benefits are well known by the industry as demonstrated by the use of

PCG techniques during the development of commercial games such as *Borderlands*, *Borderlands 2*, *Minecraft* and *Spore*.

Search-based procedural content generation [22] (SBPCG) techniques apply a generate and test scheme, that is, the content is firstly generated and then evaluated according to some criteria. This evaluation sets the quality level of the generated content automatically. Then, new content is created based on previous evaluations. This process, which should be automated, is repeated until the content reach a certain quality level, hence making evolutionary algorithms a perfect tool for this process. In this work we have used a SBPCG technique to generate aesthetic maps for a real-time strategy (RTS) game. This method is based on a multi-objective evolution strategy that generate maps which are similar to other aesthetic maps and different from other non-aesthetic maps at the same time.

2 Background

Real-time strategy games offer a large variety of fundamental AI research problems [1] such as adversarial real-time planning, decision making under uncertainty, opponent modelling, spatial and temporal reasoning and resource management. This genre of game has been widely used as a test-bed for AI techniques [14]. *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 all the opponents. Games take 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 [15, 16] we focused on achieving balanced (i.e., games in which none of the players strongly dominates her opponent) and dynamic (i.e., action-packed) games. PCG techniques are usually employed to generate maps, as exhibited by the large number of papers on this topic [12]. For example, Mahlmann et al. [18] presented a search-based map generator for an simplified version of the RTS game *Dune*, which is based on the transformation of low resolution matrices into higher resolution maps by means of cellular automata. Frade et al. introduced the use of genetic pro-

gramming to evolve maps for videogames (namely terrain programming), using either subjective human-based feedback [9, 10] or automated quality measures such as accessibility [8] or edge-length [11]. Togelius et al. [21] designed a PCG system capable of generating tracks for a simple racing game from a parameter vector using a deterministic genotype-to-phenotype mapping.

Maps are not the only content that is generated with these methods. For example, Font et al. [6] presented initial research regarding a system capable of generating novel card games. Collins [2] made an introduction to procedural music in video games, examining different approaches to procedural composition and the control logics that have been used in the past. The authors of [19] have created a prototype of a tool that automatically produce design pattern specications for missions and quest for the game Neverwinter Nights.

3 Materials and methods

As stated before, this work focuses on the design of aesthetic maps for the RTS game *Planet Wars*. This section describes the mechanisms that helped us to achieve our goal: firstly, there is a description about how the maps have been represented and characterized in order to get better aesthetics; next, there is a detailed explanation of the evolutionary algorithm used to generate the maps.

3.1 Representation and characterization

Game's maps are sets with a certain number of planets n_p located on a 2D plane. These planets are defined by their position on the plane (coordinates (x_i, y_i)), their size s_i and a number of ships w_i. The size s_i defines the rate at which a planet will produce new ships every turn (as long as the planet is controlled by any player) while the remaining parameter, w_i, indicates the number of ships that are defending that planet. Hence, we can denote a map as a list $[\vec{\rho}_1, \vec{\rho}_2, \cdots, \vec{\rho}_{n_p}]$, where each $\vec{\rho}_i$ is a tuple $\langle x_i, y_i, s_i, w_i \rangle$. A playable map needs to specify the initial home planets of the players, which have been fixed as the first two planets $\vec{\rho}_1$ and $\vec{\rho}_2$ because of simplicity. The number of planets n_p is not fixed and should range between 15 and 30 as specified by the rules of the *Google AI Challenge 2010*. This variable number of planets makes part of the self-adaptive evolutionary approach described later on.

In order to evaluate the generated maps' aesthetics , we have defined several measurements that characterize them. These are indicators related to the spatial distribution of the planets and their features, such as size and number of ships:

- Planet's geometric distribution: Let $\vec{p}_i = (x_i, y_i)$ be the coordinates of the i-th planet and N the total number of planets, so we defined the average distance between planets μ_d and the standard deviation of these distances

σ_d as follows:

$$\mu_d = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \|\vec{p}_i - \vec{p}_j\| \tag{1}$$

$$\sigma_d = \sqrt{\frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\|\vec{p}_i - \vec{p}_j\| - \mu_d)^2} \tag{2}$$

- Planet's features: Let s_i and w_i be the size (i.e. growth rate) and number of ships, respectively, of the i-th planet, then we specified the average and standard deviation of these sizes (μ_s and σ_s respectively) and the Pearson's correlation between the planet's size and the number of ships on it ρ as follows:

$$\mu_s = \frac{1}{N} \sum_{i=1}^{N} s_i \tag{3}$$

$$\sigma_s = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - \mu_s)^2} \tag{4}$$

$$\rho = \frac{\sum_{i=1}^{N} s_i w_i - N \mu_s \mu_w}{N \sigma_s \sigma_w} \tag{5}$$

  where μ_w and σ_w are the average and standard deviation of ships, respectively.

These measures has been applied to compare the likelihood between maps in the following way: each map is characterized by a tuple $\langle \mu_d, \sigma_d, \mu_s, \sigma_s, \rho \rangle$, then the euclidean distance between these tuples defined the similarity among the planets they represented. Additionally, we specified two sets of maps, one of them containing 10 maps with good aesthetics and the other one including 10 non-aesthetic maps. These sets made up a baseline to compare with in a way that the goal of generating aesthetic maps turned into an optimization problem about minimizing the distance between generated and aesthetics maps while maximizing their distance to non-aesthetic maps. The latter was necessary to insert diversity into the set of generated maps in order to avoid the generation of maps that were very similar to the aesthetic ones.

3.2 Evolutionary map generation

This procedural map generator used a multi-objective self-adaptive $(\mu + \lambda)$ evolution strategy (with $\mu = 10$ and $\lambda = 100$) whose objectives were to reduce the distance between the generated maps and those considered aesthetics and to increase the distance to non-aesthetic maps, in order to obtain procedurally generated aesthetic maps. Mixed real-integer vectors represented the solutions

(i.e., planets): planet's coordinates (x_i, y_i) are real-valued numbers but sizes s_i and initial number of ships w_i are positive integers.

Due to this situation we have considered a hybrid mutation operator that performed different methods for parameters of either type: for real-valued parameters, it used a Gaussian mutation; as for integer variables, it considered a method that generates suitable integer mutations [17, 20] – see also [16]. The latter is similar to the mutation of real values but it uses the difference of two geometrically distributed random variables to produce the perturbation instead of the normal distributed random variables used by the former. In either case, the parameters that ruled the mutation were also a part of the solutions, thus providing the means for self-adapting them. More precisely, regarding real-valued parameters $\langle r_1, ..., r_n \rangle$ they are extended with n step sizes, one for each parameter, resulting in $\langle r_1, ..., r_n, \sigma_1, ..., \sigma_n \rangle$. The mutation method is specified as follows:

$$\sigma_i' = \sigma_i \cdot e^{\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)} \quad (6)$$

$$r_i' = r_i + \sigma_i \cdot N_i(0,1) \quad (7)$$

where $\tau' \propto 1/\sqrt{2n}$, and $\tau \propto 1/\sqrt{2\sqrt{n}}$. A boundary rule is applied to step-sizes to forbid standard deviations very close to zero: $\sigma_i' < \epsilon_0 \Rightarrow \sigma_i' = \epsilon_0$ (in this algorithm, σ_0 comprises a 1% of the parameter's range). In the case of integer-valued parameters $\langle z_1, ..., z_m \rangle$ they are extended in a similar way as are real-valued parameters, resulting in $\langle z_1, ..., z_m, \varsigma_1, ..., \varsigma_m \rangle$. The following equations define the mutation mechanism:

$$\varsigma_i' = \max(1, \varsigma_i \cdot e^{\tau \cdot N(0,1) + \tau' \cdot N(0,1)}) \quad (8)$$

$$\psi_i = 1 - (\varsigma_i'/m) \left(1 + \sqrt{1 + \left(\frac{\varsigma_i'}{m} \right)^2} \right)^{-1} \quad (9)$$

$$z_i' = z_i + \left\lfloor \frac{ln(1 - U(0,1))}{ln(1 - \psi_i)} \right\rfloor - \left\lfloor \frac{ln(1 - U(0,1))}{ln(1 - \psi_i)} \right\rfloor \quad (10)$$

where $\tau = 1/\sqrt{2m}$ and $\tau' = 1/\sqrt{2\sqrt{m}}$.

We considered a "cut and splice" recombination operator that recombines two individuals by swapping cut pieces with different sizes. This operator selects one cut point for each individual and then exchanges these pieces, getting two new individuals with a different number of planets in relation to their parents. This endows the algorithm with further self-adaptation capacities, hence affecting the complexity of the maps, i.e., the number of planets in the solutions.

As described in section 3.1, we characterized every map as a vector of five elements so the euclidean distance between these vectors measures the likelihood between them, hence the fitness function used to evaluate the individuals is, precisely, the median euclidean distance from the individual to every map from the set of aesthetics (minimization objective) and non-aesthetics (maximization objective) maps.

Considering that we had a multi-objective optimization problem, we decided to use the selection method of the Non-dominated Sorting Genetic Algorithm II (*NSGA-II*) [4].

4 Experimental Results

We have used the DEAP library [7] to implement the aforementioned algorithm. We have run 20 executions of the algorithm during 100 generations each. We have also computed the cumulative non-dominated set of solutions from every execution – see figure 1. As we can see, there is a linear relationship between both distances in the middle range of the front. This hints at the density of the search space and the feasibility of linearly trading increasing distance to good maps by increasing distance to bad maps.

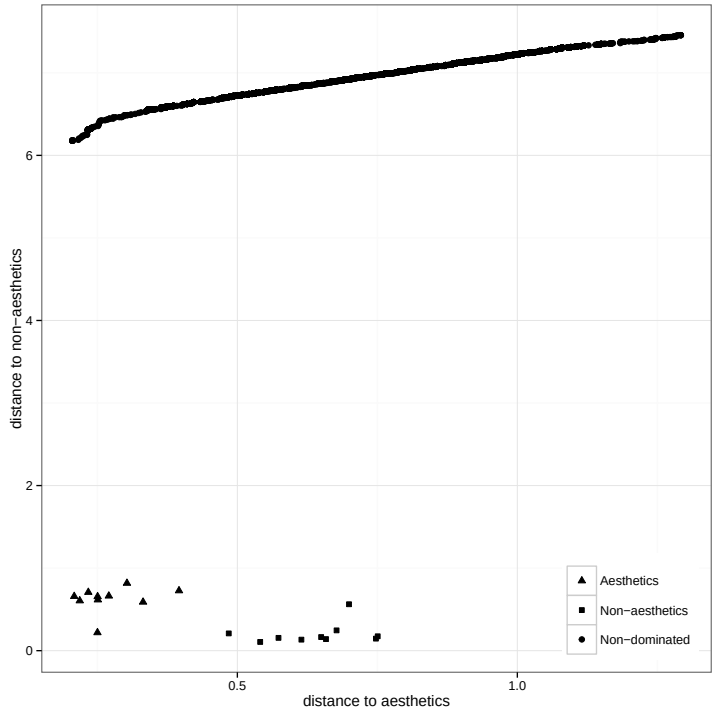

Figure 1: Cumulative set of non-dominated generated solutions (circle) and maps from aesthetic (triangle) and non-aesthetic (square) baseline sets.

Figure 2 shows how are distributed the values of the different variables that make up the characterization vector of a map. Note that there are some variables that are similar in both aesthetics and non-aesthetic maps, such as σ_d and σ_s. However, this variable is higher in the case of the non-dominated maps, which

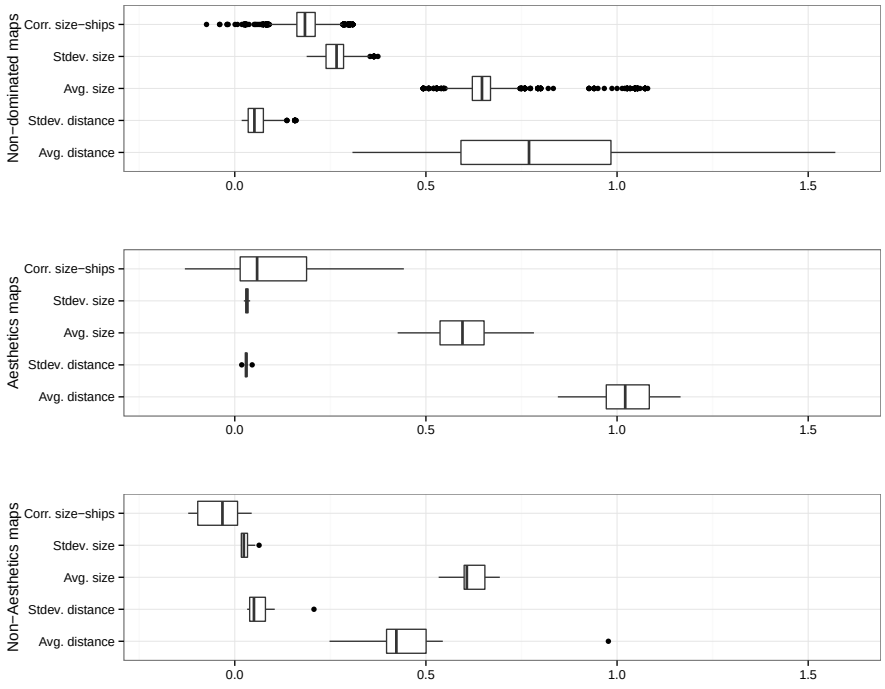

Figure 2: Characterization variables for both non-dominated and baseline maps

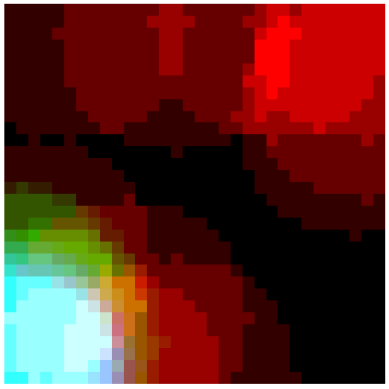

Figure 3: Map's distribution over the SOM. Red for non-aesthetic, green for aesthetic and blue for non-dominated.

should explain the high distance between many solutions in the front and the baseline maps, as seen in figure 1. Another interesting observation is the highly distributed values of μ_d in the non-dominated maps, which probably means that this variable has an uncertain effect over the fitness and hence the search space for this variable is wider with respect to other variables.

In order to check the validity of the generated maps, we built a support vector machine [3] (SVM) to classify maps (namely, characterization vectors) as aesthetic and non-aesthetic. Support vector machines are supervised learning models that recognize patterns and analyze data. They are able to perform linear and non-linear classification. The SVM was trained using the same sets of maps that the evolutionary algorithm has used to calculate the fitness. This SVM classified as aesthetic every map out of the 4289 non-dominated maps, which led us to think that the algorithm is capable of generating aesthetic maps.

In addition to the aforementioned classifier, we created a self-organizing map (SOM) [13] with 32×32 process units over a non-toroidal rectangular layout, using the same maps as the training set. Self-organizing maps are artificial neural networks that are trained using unsupervised learning to generate a discretized representation of the input space. As we can see in figure 3, this SOM established a separation between non-aesthetic (red zones, upper-right) and aesthetic maps (green zones, lower-left). Moreover, generated maps (blue zones) shared the same region as aesthetic maps, hence they should be considered aesthetic as well.

5 Conclusions

We have performed an initial approach towards the procedural aesthetic map generation for the RTS game *Planet Wars*. We have defined a method of map characterization based on several of its maps' geometric and morphologic properties in order to evaluate how aesthetic a map is. We have used two sets of maps (aesthetics and non-aesthetics) as a baseline to compare with, and an evolution strategy whose objectives are minimize and maximize the distance of the generated maps to the aesthetics and non-aesthetics maps of the baseline. The solutions have been tested with a SVM and a SOM. The SVM has classified each solution as aesthetic while the SOM was able to make a separation between aesthetic and non-aesthetic maps (the generated maps shared the same region as the aesthetic maps).

We have used a geometric characterization of the maps (namely planets' coordinates), which means that this characterization is affected by rotation, translation and scaling, thus suggesting the use of other kind of measurements, such as topological variables, as a potential line of future research.

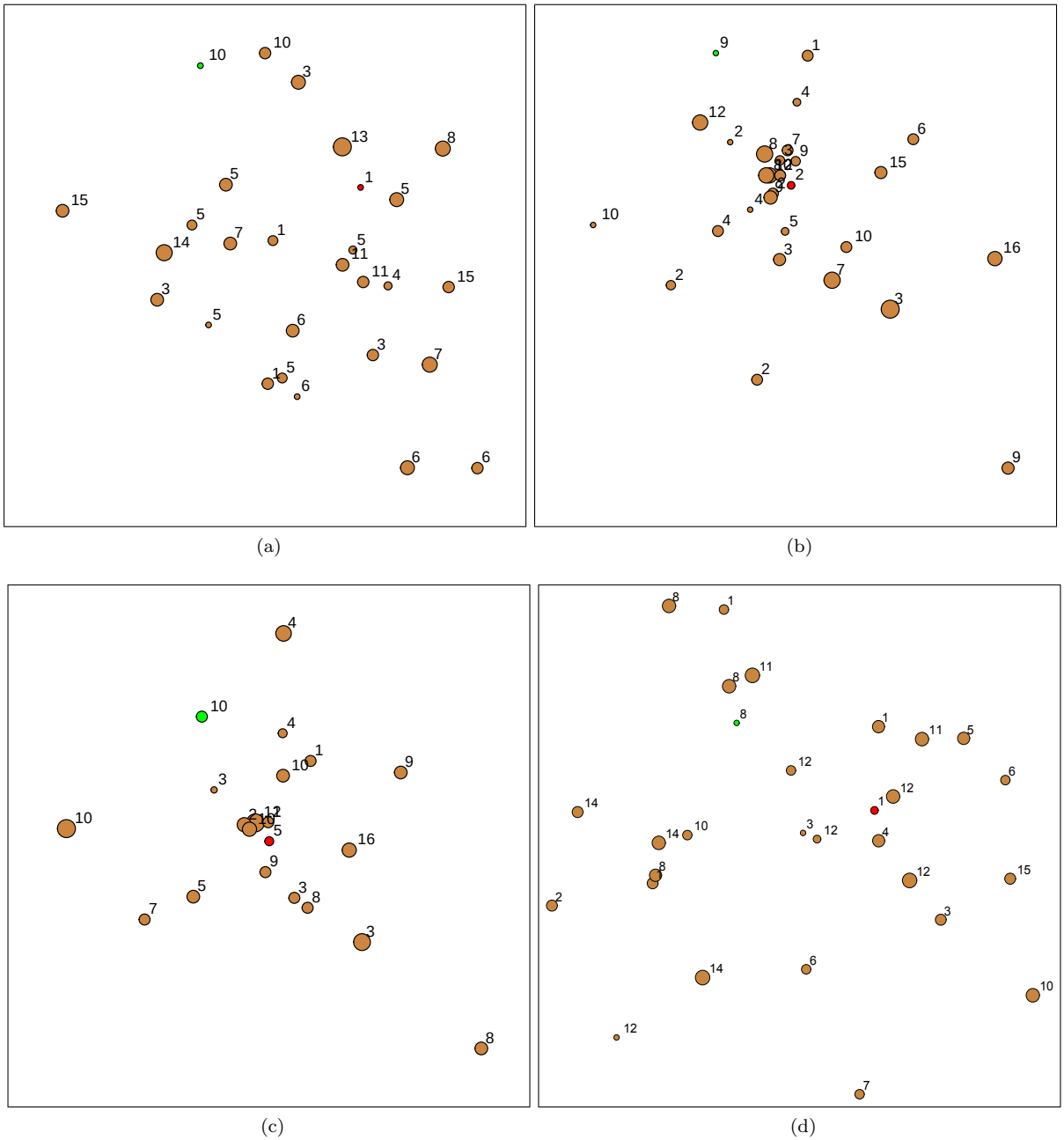

Figure 4: Examples of generated maps

Acknowledgments

This work is partially supported by Spanish MICINN under project ANYSELF[1] (TIN2011-28627-C04-01), Junta de Andalucía under project DNEMESIS[2] (P10-TIC-6083) and Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

References

[1] Buro, M.: RTS games and real-time AI research. In: Behavior Representation in Modeling and Simulation Conference. vol. 1. Curran Associates, Inc. (2004)

[2] Collins, K.: An introduction to procedural music in video games. Contemporary Music Review 28(1), 5–15 (2009)

[3] Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995)

[4] Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Schoenauer, M., et al. (eds.) Parallel Problem Solving from Nature VI. Lecture Notes in Computer Science, vol. 1917, pp. 849–858. Springer-Verlag, Berlin Heidelberg (2000)

[5] Entertainment Software Association: Essential facts about the computer and video game industry (2012), http://www.theesa.com/facts/pdfs/esa_ef_2012.pdf

[6] Font, J., Mahlmann, T., Manrique, D., Togelius, J.: A card game description language. In: Esparcia-Alczar, A. (ed.) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 7835, pp. 254–263. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-37192-9_26

[7] 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)

[8] Frade, M., de Vega, F., Cotta, C.: Evolution of artificial terrains for video games based on accessibility. In: Di Chio, C., et al. (eds.) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 6024, pp. 90–99. Springer-Verlag, Berlin Heidelberg (2010)

[9] Frade, M., de Vega, F.F., Cotta, C.: Modelling video games' landscapes by means of genetic terrain programming - a new approach for improving users'

[1] http://anyself.wordpress.com/
[2] http://dnemesis.lcc.uma.es/wordpress/

experience. In: Giacobini, M., et al. (eds.) Applications of Evolutionary Computing. Lecture Notes in Computer Science, vol. 4974, pp. 485–490. Springer-Verlag, Berlin Heidelberg (2008)

[10] Frade, M., de Vega, F.F., Cotta, C.: Breeding terrains with genetic terrain programming: The evolution of terrain generators. International Journal of Computer Games Technology 2009 (2009)

[11] Frade, M., de Vega, F.F., Cotta, C.: Evolution of artificial terrains for video games based on obstacles edge length. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2010)

[12] Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: A survey. ACM Trans. Multimedia Comput. Commun. Appl. 9(1), 1:1–1:22 (Feb 2013), http://doi.acm.org/10.1145/2422956.2422957

[13] Kohonen, T.: The self-organizing map. Proceedings of the IEEE 78(9), 1464–1480 (1990)

[14] Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: A Review of Computational Intelligence in RTS Games. In: Ojeda, M., Cotta, C., Franco, L. (eds.) 2013 IEEE Symposium on Foundations of Computational Intelligence. pp. 114–121

[15] 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)

[16] 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)

[17] Li, R.: Mixed-integer evolution strategies for parameter optimization and their applications to medical image analysis. Ph.D. thesis, Leiden University (2009)

[18] Mahlmann, T., Togelius, J., Yannakakis, G.N.: Spicing up map generation. In: Chio, C.D., et al. (eds.) Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 7248, pp. 224–233. Springer-Verlag, Málaga, Spain (2012)

[19] Onuczko, C., Szafron, D., Schaeffer, J., Cutumisu, M., Siegel, J., Waugh, K., Schumacher, A.: Automatic story generation for computer role-playing games. In: AIIDE. pp. 147–148 (2006)

[20] 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)

[21] Togelius, J., De Nardi, R., Lucas, S.: Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on. pp. 252–259 (2007)

[22] Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games 3(3), 172–186 (2011)