Modelling Video Games' Landscapes by Means of Genetic Terrain Programming - A New Approach for Improving Users' Experience

Miguel Frade[1], F. Fernandez de Vega[2], and Carlos Cotta[3]

[1] School of Technology and Management, Polytechnic Institute of Leiria, Portugal
mfrade@estg.ipleiria.pt
[2] Centro Universitario de Merida, Universidad de Extremadura, Spain
fcofdez@unex.es
[3] ETSI Informática, Campus de Teatinos, Universidad de Málaga, Spain
ccottap@lcc.uma.es

Abstract. Terrain generation algorithms have an important role in video games: they can provide a realistic scenario for the game experience, or can help keep the user interested in playing by providing new landscapes each time he plays. Nowadays there are a wide range of techniques for terrain generation, but all of them are focused on providing realistic terrains, often neglecting other aspects (e.g., aesthetic appeal or presence of desired features). This paper proposes a new technique, Genetic Terrain Programming, based on evolutionary design with GP to allow game designers to evolve terrains accordingly to their aesthetic feelings or desired features. The developed application produces Terrains Programs (TPs) that will always generate different terrains, but consistently with the same features (e.g. valleys, lakes). This characteristic will allow game designers to endow games with programs capable of generating different coherent landscapes each time users run the game.

Key words: terrain generation, video games, evolutionary art, genetic programming

1 Introduction

Video games constitute a crucial area of the entertainment industry, with impressive financial investments to make them more appealing and interesting. Game players seek continually for more enjoyable games as they spent 3.7 days per week playing an average of 2.01 hours per day [1]. Entertainment industry wants to maintain or increase this user's interest. In order to achieve richer human-machine interaction video games must be dynamic, they need to present game players with novel plots, intelligent artificial opponents, different goals and even scenario changes. Artificial terrain generation algorithms have an important role in the video games' dynamics. They help keep the user interested in playing by providing new landscapes each time he plays.

Nowadays there are a wide range of techniques for terrain generation, which are presented in Sect. 2.1, but all of them have key constrains. More elaborated methods depend highly upon designer's skills, time and effort to obtain acceptable results, and can not be used to automatically generate terrains in real time. The simpler methods allow only a narrow variety of terrain types and usually require a second algorithm (e.g. erosion algorithms) to provide a more natural looking result. Another key limitation of most techniques is the lack of control over localised terrain features.

A common feature with current techniques is that they all try to generate realistic terrains. Although this is an important aspect, imagine the possibility of a game designer to evolve their own terrain accordingly to his aesthetic feelings. This can lead to the generation of more exotic terrains at the cost of realism, but might also give players an awe reaction and increase their interest in playing. This paper presents a new technique, that we designated as Genetic Terrain Programming, that allows the generation of terrains based on aesthetic evolutionary design with Genetic Programming (GP).

Sect. 2 presents some background on the current terrain generation techniques, their main constrains and an overview of evolutionary systems, applied to terrain generation, and on similar domains. The details of the Genetic Terrain Programming technique and the developed application are described in Sect. 3. Some results and considerations are presented in Sect. 4. Finally, Sect. 5 presents the conclusions and possible future directions.

2 Background

A digital ground surface topography, or terrain, can be represented in many ways, but the most common representation is the height map. A height map is a scalar function of two variables, such that every coordinate pair (x, y) corresponds to an elevation value h, as shown in Eq. (1).

$$h = f(x, y) \quad . \tag{1}$$

A height map is normally implemented as a two-dimensional, rectangular grid of height values, and is equivalent to a grayscale image. Height maps have the limitation that they cannot represent structures where there are multiple heights for the same (x, y) coordinates (such as caves), but are sufficient for most uses and can be highly optimised for rendering and object collision detection [2].

2.1 Terrain Generation Techniques

Current terrain generation techniques can be divided in three main categories: measuring, modeling and procedural.

In the measuring techniques elevation data is derived from real-world measurements to produce Digital Elevation Models (DEM), commonly built using remote sensing techniques such as satellite imagery and land surveys [3]. This

is the most common basis for digitally-produced relief maps. Measuring has the advantage of producing highly realistic terrains with very little human effort, but at the expense of designer control. If the designer has specific goals for the the terrain's design and features (e.g. mountains, valleys, lakes) this approach may be very time-consuming, as the designer might have to search extensively to find real-world data that meets his specific criteria.

Modeling is by far the most flexible technique for terrain generation. A human artist models or sculpts the terrain morphology manually using a 3D modeling program (e.g. Maya[4], 3D Studio[4], or Blender[5]), or a specialised terrain editor program (e.g. the editors that ship with video games like Unreal Tournament[6] or SimCity 4[7]). The way the terrain is built is different depending on the features provided by the chosen editor, but the general principle is the same. With this approach the designer has unlimited control over the terrain design and features, but this might be also a disadvantage. By delegating most or all of the detail up to the designer, these technique imposes high requirements on the designer in terms of time and effort. Also the realism of the resulting terrain is fully dependent on the designer's skills.

Procedural techniques are those in which the terrains are generated programmatically. These category can be divided into physical, fractal and spectral synthesis techniques. Physically-based techniques simulate the real phenomena of terrain evolution trough effects of physical processes such as erosion by wind [4], water [5], thermal [6], or plate tectonics. These techniques generate highly realistic terrains, but require an in-depth knowledge of the physical laws to implement and use them effectively.

Another procedural approach is the spectral synthesis. This technique is based on the observation that fractional Brownian motion (fBm) noise has a well defined power spectrum. So random frequency components can be easily calculated and then the inverse Fast Fourier Transform (FFT) can be computed to convert the frequency components into altitudes [6]. This technique does not allow the designer much control on the outcome of terrains features.

Self-similarity is the key concept behind any fractal technique. An object is said to be self-similar when magnified subsets of the object look like the whole and to each other [7]. Terrain falls into this category. The jagged edge of a broken rock has the same irregularities as a ridgeline on a distant horizon. This allows the use of fractals to generate terrain which still looks like terrain, regardless of the scale in which it is displayed [8]. Every time these algorithms are executed they generate a different terrain due to the incorporated randomness. This class of algorithms is the favourite one by game's designers, mainly due to their speed and simplicity of implementation. There are several tools available that are predominantly based on fractal algorithms, such as Terragen[8] (which

[4] http://www.autodesk.com/fo-products
[5] http://www.blender.org
[6] http://www.mobygames.com/game/unreal-tournament-2004
[7] http://simcity.ea.com/about/simcity4/overview.php
[8] http://www.planetside.co.uk/terragen

is a fractal/modeling tool) and GenSurf[9] (a mapping tool for Quake 3 Arena video game). However, generated terrains by this techniques are easily recognised because of the self-similarity characteristic of fractal algorithms. Although these algorithms present some parameters that can be tweaked to control, e.g the roughness, the designer does not have control on the resulting terrain features.

2.2 Evolutionary Design

Teong Ong et al. proposed an evolutionary design optimisation technique to generate terrains [9]. They applied genetic algorithms to transform height maps in order to conform them to the required features. Their approach breaks down the terrain generation process into two stages: the terrain silhouette generation phase, and the terrain height map generation phase. The input to the first phase is a rough, 2D map laying out the geography of the desired terrain that can be randomly generated or specified by the designer. This map is processed by the first phase to remove any unnaturally straight edges and then fed to the second phase, along with a database of pre-selected height map samples representative of the different terrain types. The second phase searches for an optimal arrangement of elevation data from the database that approximates the map generated in the first phase. Since the height map generation algorithm is inherently random, the terrains generated from two separate runs of the algorithm will not be the same, even if they use the same map. While this has the benefit of allowing an infinite number of variations to be created. To control this, the seed for the random number generator can be kept the same across separate runs of the algorithm, allowing the same terrain to be regenerated as many times as desired.

Like the techniques described in Sect. 2.1, the solution proposed in [9] is focused only on generating realistic terrains. It is based on a database of real terrain samples and cannot generate terrains accordingly to the designer's aesthetic appeal, or terrains that look "out of this world". On the opposite side there are artists. They have used evolutionary art systems, for many years, to generate aesthetically pleasing forms rather than realistic ones.

Evolutionary art systems are similar in many ways, but they differ on their phenotype representations [10]. Because of the equivalence between height maps and grayscale images, it is possible to apply the same principle of evolutionary art to terrain generation. However, the phenotype will be a terrain surface instead of an image.

GP has been the most fruitful evolutionary algorithm applied to evolve images interactively. Karl Sims used GP to create and evolve computer graphics by mathematical equations. The equations are used to calculate each pixel [11], or create graphic movies by adding a time variable to the dynamic differential equations [12]. He created several graphic art pieces including *Panspermia* and *Primordial Dance* and also allowed visitors interact with his interactive art system at art shows and exhibitions. His *Galapagos*[10] is an L-system based In-

[9] http://tarot.telefragged.com/gensurf
[10] http://www.genarts.com/galapagos

teractive Evolutionary Computation (IEC) system that allows visitors to create their own graphic art through their interaction.

Tatsuo Unemi developed *SBART* (Simulated Breeding ART) [13,14], an IEC graphics system open to public. *SBART* uses GP to create mathematical equations for calculating each pixel value and its (x, y) coordinates. As GP nodes *SBART* assigns the four arithmetic fundamental operators ($+$, $-$, $\times$ and $\div$), *power*, *sqrt*, *sin*, *cos*, *log*, *exp*, *min* and *max*. The terminal nodes are constants and variables. Three values at each pixel are calculated using one generated mathematical equation by assuming that the constants are 3D vectors consisting of three real numbers and the variables are 3D tuples consisting of $(x, y, 0)$. The three calculated values are regarded as members of a vector (hue, lightness and saturation) and are transformed to RGB values for each pixel. These three values are normalised to values in $[-1, 1]$ using a saw-like function. It allows the creation of movies by replacing $(x, y, 0)$ with (x, y, t), where t is a time variable. The *SBART*'s functions were expanded to create a collage [15].

In *NEvAr* (Neuro Evolutionary Art) [16], of Peneusal Machado et al., the function set is composed mainly of simple functions such as arithmetic, trigonometric and logic operations. The terminal set is composed of a set of variables x, y and random constants. The phenotype (image) is generated by evaluating the genotype for each (x, y) pair belonging to the image. In order to produce colour images, *NEvAr* resorts to a special kind of terminal that returns a different value depending on the colour channel – Red, Green or Blue – that is being processed. This tool focus on the reuse of useful individuals, which are stored in an image database and led to the development of automatic seeding procedures.

Despite the interesting results achieved by the above evolutionary art systems, to the best of our knowledge, none of them has been applied this technique to evolve terrain landscaps. We believe the use of evolutionary art systems with GP will allow the creation of both aesthetic and real terrains (without requiring a database of real terrain data). Additionally some control over localised terrain features will be possible trough the use of several TPs to compose the full landscape, this is the main drawback of the procedural techniques. It will also require less effort and time than modeling techniques to create complex terrains and the result is not solely dependent on the designer's skills.

3 Genetic Terrain Programming

Our main goals are to create TPs capable of generating different terrains but consistently with the same features and obtain TPs capable of generating realistic or aesthetic terrains. We propose the use of aesthetic evolutionary design with GP to achieve them. This approach consists of a guided evolution of terrains (by means of Interactive Evolution) accordingly to a specific desired terrain feature or aesthetic appeal. We have coined the term Genetic Terrain Programming to denote this new technique. Although the concept is similar to the one used in evolutionary art systems, it has never been applied to terrain generation.

Our application, Generator of Terrain Programs (*GenTP*) has been developed with GPLAB [17], an open source GP toolbox for Matlab[11]. The initial population is created randomly, without restrictions on trees depth size and a fixed population size of 12. The number of generations is decided by the designer, who can stop the application at any time. The designer can select one or two individuals to create the next population and the genetic operators used depend upon the number of selected individuals. If one individual is selected only the mutation operator will be used. In case the designer chooses to select two individuals both the standard crossover and mutation operators [18] will be applied. Like in others IEC systems, the fitness function relies exclusively on designers' decision, either based on his aesthetic appeal or on desired features.

$$F = \{plus(a,b); minus(a,b); myMultiply(a,b); myDivide(a,b); myLog(a); myMod(a,b); mySin(a); myCos(a); myTan(a); myAtan(a); FFT(a); myPower(a); mySqrt(a); Smooth(a); Incline(a)\} \ . \quad (2)$$

$$T = \{squares; rectangles; circles; triangles; planes; myRandom; fftGen\} \ . \quad (3)$$

Each GP individual is a tree composed by mathematical functions shown in Eq. (2) and height maps as terminals, shown in Eq. (3). Some terminals depend upon a Random Ephemeral Constant (REC) to define some characteristics, such as inclinations of *planes*, spectrum values of *fftGen* and sizes of the geometric figures terminals. All these terminals depend upon a random number generator, which means that consecutive calls of one TP will always generate different terrains. This is a desired characteristic because we want to be able create different terrains with each TP, but that will share the same features. An example of a GP individual is given in Fig. 1.

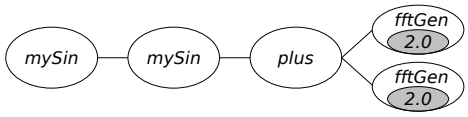

Fig. 1. Example of a GP tree individual with two RECs (in grey ellipses)

While in [14,15] the mathematical equations are used to calculate both the pixel value and its coordinates, in *GenTP* only the height will be calculated. The (x, y) coordinates will be dictated by the matrix position occupied by the height value.

The *GenTP* interface (see Fig. 2) presents a grid with 12 individuals as 3D surfaces to allow the designer to select one or two of them to create the next generation. This interface also permits to see the TP of the selected individuals and record them as a formula and the generated terrain as a VRML 2.0 file.

[11] Matlab is a product of Mathworks (http://www.mathworks.com/)

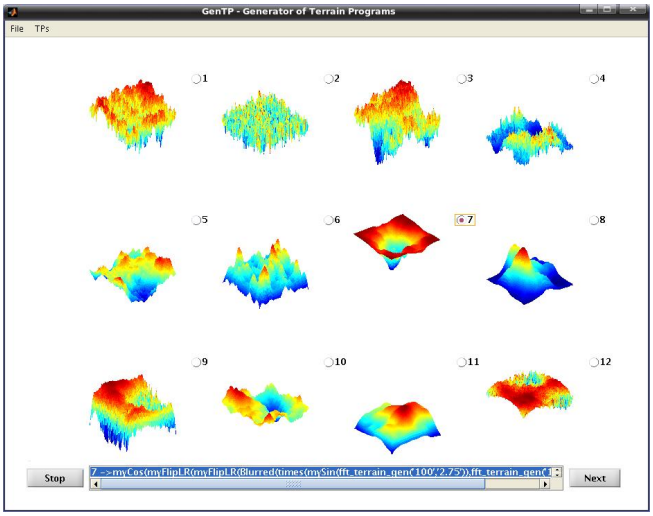

Fig. 2. *GenTP* graphical user interface

4 Experimental Results

Two kind of experiments were conducted, the first one consisted on obtaining aesthetic appealing terrains (regardless of their realism) and the second one to achieve a realistic terrain with a specific feature in mind.

On the first kind of experiments we were able to get aesthetic appealing terrains after about 30 to 70 generations. On those experiments we were able to obtain very different kinds of terrains types. Must of them are difficult to describe due to their exotic look (see Fig. 3).

$$H = \quad myLog(Incline(mySin(mySqrt(Smooth(\textit{fftGen}(1.25)))))) \ . \qquad (4)$$

For example, the TP represented in Eq. (4) creates terrains with a bank of knolls with two ridges that give them an alien look (see Fig. 3). In this TP the REC in *fftGen* allows us to control the number of knolls, e.g. if we change the REC from 1.25 to 2.5 we can decrease their number. Fig. 3 has examples of terrains generated from three different TPs. Each column has pictures of terrains generated by three consecutive executions of the same TP. In this set of pictures is visible that each TP is capable of generate different terrains, but with the same features.

On the second kind of experiments we tried to obtain TPs to generate terrains with a specific features, such as mountains, cliffs or corals. In this case the number of necessary generations varies widely until we are able to get acceptable results. These number is highly dependent on the initial population and could vary between 10 to more than 100 generations. When running the experiments, if after a number of generations an interesting result is not obtained, we have preferred to cancel the experiment and begin again, avoiding this way a long

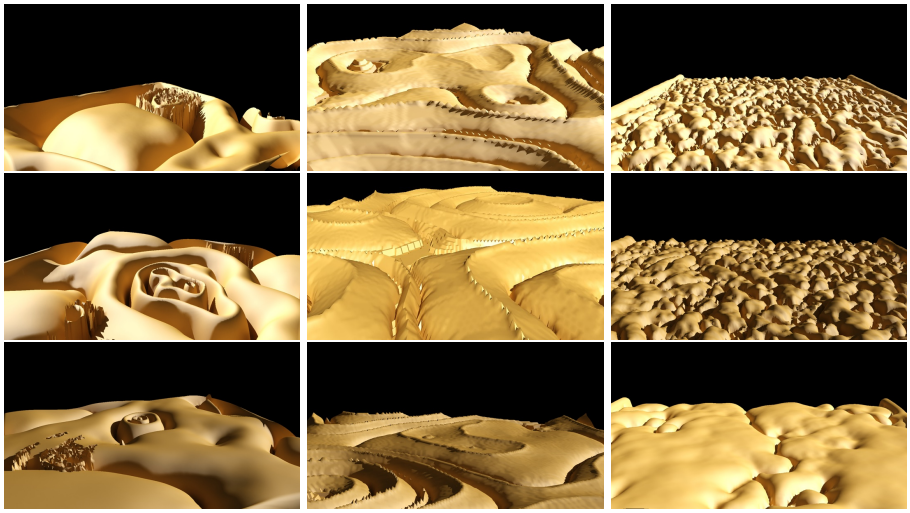

Fig. 3. Exotic terrains generated by three different TPs (rendered with 3DS Max). The pictures of the third column were generated by Eq. (4), the two top terrains with $REC = 1.25$ and on the bottom one with $REC = 2.5$.

run. We also verified that, for realistic landscapes, the range of terrains types were narrower than in the first experiment. Eq. (5) has an example of a TP that was evolved having in mind to achieve a coral looking terrain. In the set of pictures on Fig. 4 it is visible that the terrains generated by each TP are allways different, but still present the same features.

$$\begin{aligned} H = \quad & myLog(minus(fftGen(2.75), myLog(minus(\\ & Smooth(fftGen(1.50)), fftGen(2.50))))) \ . \end{aligned} \tag{5}$$

The evolution is influenced by the number of selected TPs, if just one TP is selected - only mutation operator applied - the next generation will present few variations of the selected individual and the TP will evolve slowly. On the other hand, if the designer opts to select two individuals, the next generation will present more diversity and the evolved TPs can change their look more dramatically. Some robustness tests, on a few TPs, showed that the functions *myLog*, *myPower*, *myTan* and *myAtan* are the ones that have more influence in the terrain look, followed by the *Smooth*. Changes on the REC also influence the terrain look, but that change is not always noticeable.

5 Conclusions and Future Work

On this paper we present the Genetic Terrain Programming technique for terrain generation. The idea behind this new approach is to use interactive evolution with GP to generate TPs. To employ this technique a first implementation of

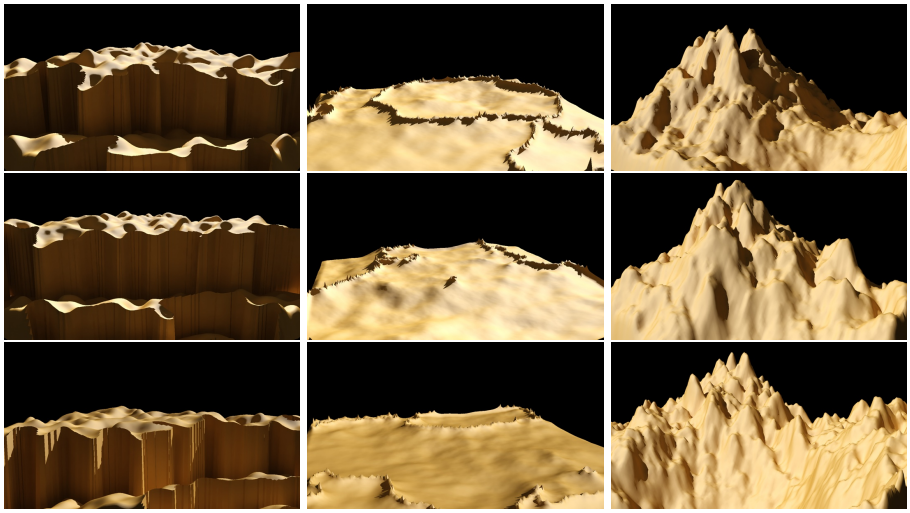

Fig. 4. TPs evolved with specific features in mind (rendered with 3DS Max). From left to right column: cliffs, corals (Eq. (5)) and mountains.

GenTP has been carried out with GPLAB and Matlab. Through a series of experiments we have shown that multiple execution of the same TP will allways generate differrent terrains, because of the randomness present on its terminals, but with the same features (e.g. mountains, valleys). With a single technique designers will be able to evolve very different kinds of TPs, from real looking terrains to more exotic ones with an alien semblance. Those TPs can be inserted in video games to generate different terrains each time a user plays and consequently help to keep users interested in playing.

The potential shown by *GenTP* suggests several lines for future developments. One of them is to augment the GP terminal set in order to try obtain a wider range of realistic terrain types with fewer generations. After that we plan to incorporate the Genetic Terrain Programming technique in a real video game. More features could be added to our technique so that whole scenarios, including vegetation and buildings, can be generated. On a later stage, we will try to involve a large number of users, by means of volunteer computing, to create a whole set of different TPs.

Acknowledgements. This work was partially funded by National Nohnes project TIN2007-68083-C02-01 Spanish Ministry of Science and Education. We also would like to deeply thank Nuno Monteiro for his helpful assistance with 3D Studio Max.

References

1. Yannakakis, G.N., Hallam, J.: A scheme for creating digital entertainment with substance. In Aha, D.W., Muñoz-Avila, H., van Lent, M., eds.: Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, Edinburgh, UK, IJCAI (2005) 119–124
2. Duchaineau, M., Wolinsky, M., Sigeti, D., Millery, M., Aldrich, C., Mineev-Weinstein, M.: ROAMing terrain: Real-time optimally adapting meshes. In: VIS '97: Proceedings of the 8th conference on Visualization '97, Los Alamitos, CA, USA, IEEE Computer Society Press (1997) 81–88
3. DEM, U.G.S.: Digital elevation model standards. Website (accessed on 2007/11/06) http://rockyweb.cr.usgs.gov/nmpstds/demstds.html.
4. Wind Erosion Research Unit, K.S.U.: Wind erosion simulation models. Website (accessed on 2007/11/06) http://www.weru.ksu.edu/weps.html.
5. Kelley, A., Malin, M., Nielson, G.: Terrain simulation using a model of stream erosion. In: SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, NY, USA, ACM (1988) 263–268
6. Olsen, J.: Realtime procedural terrain generation - realtime synthesis of eroded fractal terrain for use in computer games. Department of Mathematics And Computer Science (IMADA), University of Southern Denmark (2004)
7. Peitgen, H.O., Jürgens, H., Saupe, D.: Chaos and Fractals - New Frontiers of Science. 2nd edn. Springer (2004)
8. Voss, R.: Fractals in nature: characterization, measurement, and simulation. SIGGRAPH (1987)
9. Ong, T.J., Saunders, R., Keyser, J., Leggett, J.J.: Terrain generation using genetic algorithms. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, NY, USA, ACM (2005) 1463–1470
10. Bentley, P.: Evolutionary Design by Computers. Morgan Kaufmann Publishers, Inc., CA, USA (1999)
11. Sims, K.: Artificial evolution for computer graphics. In: SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques, NY, USA, ACM (1991) 319–328
12. Sims, K.: Interactive evolution of dynamical systems. In Varela, F., Bourgine, P., eds.: Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, Paris, FR, MIT Press (1992) 171–178
13. Unemi, T.: A design of multi-field user interface for simulated breeding. In: Proceedings of the Third Asian Fuzzy and Intelligent System Symposium, Masan, Korea (1998) 489–494
14. Unemi, T.: SBART 2.4: breeding 2D CG images and movies and creating a type of collage. In: The Third International Conference on Knowledge-based Intelligent Information Engineering Systems, Adelaide, Australia, IEEE (1999) 288–291
15. Unemi, T.: SBART 2.4: an IEC tool for creating 2D images, movies, and collage. In: Proceedings of 2000 Genetic and Evolutionary Computational Conference, NV, USA (2000) 153
16. Machado, P., Cardoso, A.: NEvAr - the assessment of an evolutionary art tool. In Wiggins, G., ed.: Proceedings of the AISB'00 Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science 2000, Birmingham, UK (2000)
17. Silva, S.: A Genetic Programming toolbox for matlab. Website (accessed on 2007/11/06) http://gplab.sourceforge.net/.
18. Koza, J.R.: Genetic programming. on the programming of computers by means of natural selection. Cambridge MA: The MIT Press. (1992)