Generative art inspired by nature, using NodeBox

Tom De Smedt[1,2], Ludivine Lechat, Walter Daelemans[1]

[1] Computational Linguistics & Psycholinguistics Research Group,
University of Antwerp, Belgium
[2] Experimental Media Group, Sint Lucas Antwerpen, Belgium
tom@organisms.be, ludivinelechat@gmail.com, walter.daelemans@ua.ac.be

NodeBox is a free application for producing generative art. This paper gives an overview of the nature-inspired functionality in NodeBox and the artworks we created using it. We demonstrate how it can be used for evolutionary computation in the context of computer games and art, and discuss some of our recent research with the aim to simulate (artistic) brainstorming using language processing techniques and semantic networks.

Keywords: computer graphics, generative art, emergence, natural language processing

1 NodeBox

1.1 Computer graphics and user interfaces

Traditionally, user interfaces in computer graphics applications have been based on real-world analogies (e.g., a pen for drawing, scissors for slicing). This model raises creative limitations. First, the features can only be used as the software developers implemented them; creative recombination of tools is impossible when not foreseen. Second, there is little room for abstraction: users will tend to think along the lines of what is possible with the built-in features (buttons, sliders, menus), and not about what they want [5].

In 2002 we released NodeBox[1], a free computer graphics application that creates 2D visual output based on Python programming code, with the aim to overcome these limitations. By writing Python scripts, users are free to combine any kind of functionality to produce visual output. This approach has also been explored in software applications such as Processing [19] (using Java code) and ContextFree (using a context-free grammar). Over the course of two research projects the application has been enriched with functionality for a variety of tasks, bundled in intermixable Python modules—for example, for image compositing, color theory, layout systems, database management, web mining and natural language processing.

[1] NodeBox for Mac OS X version 1.9.5, http://nodebox.net

Example script. Images are downloaded using the Web module and arranged in a random composition, using the NodeBox rotate() and image() commands.

```
import web
images = web.flickr.search("flower")
for i in range(100):
    img = choice(images).download()
    rotate(random(360))
    image(img,
        x=random(800),
        y=random(600), width=200, height=200)
```

A number of modules are inspired by nature. For example, the Graph module combines graph theory (i.e., shortest paths, centrality, clustering) with a force-based physics algorithm for network visualization. The Supershape module implements the superformula [10], which can be used to render (and interpolate between) many complex shapes found in nature (ellipses, leaves, flowers, etc.). The L-system module offers a formal grammar that can be used to model (the growth of) plants and trees [18]. The Noise module implements Perlin's pseudo-random generator, where successive numbers describe a smooth gradient curve [17]. This technique is used in computer graphics to generate terrain maps, clouds, smoke, etc. Finally, two modules provide functionality for working with agent-based AI systems. The Ants module can be used to model self-organizing ant colonies. The Boids module presents a distributed model for flocking and swarming [20]. "Boids" is an emergent Artificial Life program where complexity arises from the interaction between individual agents. Each boid will 1) steer away to avoid crowding other boids, 2) steer in the average direction of other boids and 3) steer towards the average position of other boids.

1.2 Generative art

In practice, NodeBox is used to create what is called "generative art". Generative art is an artistic field inspired by ideas about emergence and self-organization, and making use of techniques borrowed from AI and artificial life [2, 14]. The concept of emergence was first coined by Lewes (1875) and later described by Goldstein (1999) as "the arising of novel and coherent structures, patterns and properties during the process of self-organization in complex systems" [11]. In terms of generative art, emergence implies that the artist describes the basic rules and constraints, and that the resulting artwork is allowed a certain amount of freedom within these constraints to self-organize.

In this sense NodeBox is for example useful for: a graphic designer producing a 200-page document in one consistent visual style but with variations across pages, information graphics based on real-time data, evolutionary art installations that react to input (e.g., sound), customized wallpaper based on e-mail spam [16], and so on. In section 2 we discuss one such project, which demonstrates how the software can be used for evolutionary computation in the context of the visual arts. In section 3 we show three example works of generative art.

An approach using programming code leads to new opportunities, but it also introduces a problem: many people active in the arts (e.g., art students) are not trained in programming. In section 4 we briefly discuss our attempts to alleviate this problem with a natural language processing approach, and by using a node-based interface.

2 Evolutionary computation in NodeBox

2.1 Genetic algorithms and swarming

In 2007 we created Evolution,[2] a NodeBox art installation based on boid swarming and a genetic algorithm (GA). A starting set of creatures is randomly designed from a pool of components – heads, legs, wings, tails, etc. Different components have a behavioral impact. For example: the type of head allows a creature to employ better hunting strategies (ambush, intercept), better evasive strategies (deception, hide in the flock), or better cooperative skills. Larger wings allow a creature to fly faster.

Central in a GA's design is the fitness function, which selects optimal candidates from the population for the next generation. Here, the fitness function is an interactive hunting ground where creatures are pitted against each other. Survivors are then recombined and evolved into new creatures. Evolution's GA uses a Hierarchical Fair Competition model (HFC) [12]. HFC ensures that a population does not converge into a local optimal solution too quickly, by ensuring a constant supply of new genetic material (i.e., new random creatures to fight). Interestingly, when correctly tweaked this produces an endless crash-and-bloom cycle of 1) creatures that are exceptional but flawed and 2) mediocre all-rounders. Random newcomers will eventually beat the current (mediocre) winner with an "exceptional trick" (e.g., very aggressive + very fast), but are in turn too unstable to survive over a longer period (e.g., inability to cope with cooperating adversaries). Their trick enters the gene pool but is dominated by generations of older DNA, leading to a very slow overall evolution.

2.2 City in a Bottle – a computer game on evolution by natural selection

Later, we expanded this prototype into a computer game project (City in a Bottle) based on the principles of emergence and evolution by natural selection. The project is currently in development. In short, the game environment is procedural, i.e., lacking a predefined landscape or storyline. Organisms (plants and insects) are described in terms of their basic behavioral rules: *"if attacked, flee"*, *"when cornered, fight"*. Complex game mechanisms then arise as organisms interact. If the most nutritious food is found in tall-stemmed flowers, creatures with wings will thrive— and in turn the spores from this kind of flower will spread. The game mechanisms are inspired by complex systems [13]: neither the designers nor the players of the game

[2] Evolution prototype, http://nodebox.net/code/index.php/Evolution

control the environment in full; only local actions such as planting a seed or catching and domesticating an insect are allowed.

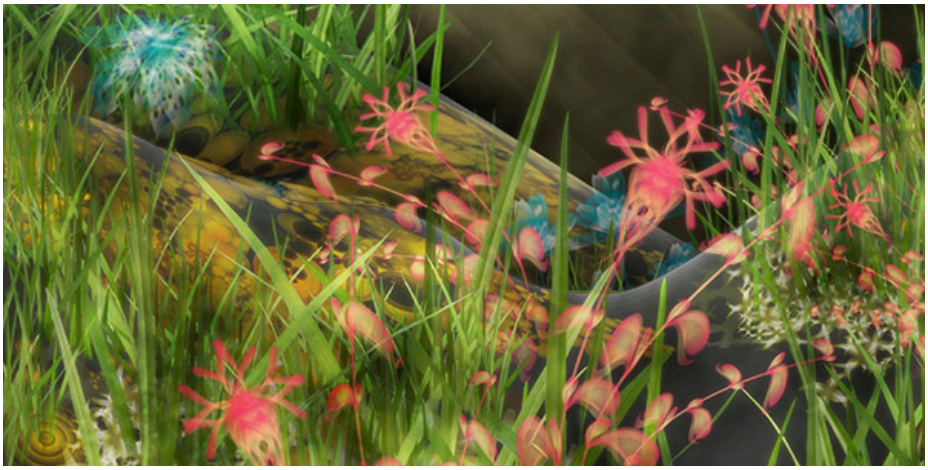

Fig. 1. Prototype of the City In A Bottle game world, with two kinds of flowers thriving.

The game music adheres to the same principle. Typically, computer games use a predefined library of sound effects that accompany an event (a weapon that is fired, a spell that is cast) and music tracks that are looped in specific situations (the haunted mansion music, the magic forest music). However, audio in City In A Bottle is composed in real-time, resulting in an emergent, swarm-based music score [1] where individual audio samples are composed based on the creature's wing flap velocity, the clicking of mandibles and the rushing of leaves. The overall music composition then arises as an interaction of creatures flocking together near food or wind rustling the leaves of various plants.

On a final note, the project uses a spin-off of NodeBox called NodeBox for OpenGL,[3] which uses hardware-acceleration on the computer graphics card for better performance.

[3] NodeBox for OpenGL, version 1.5, http://cityinabottle.org/nodebox

3 Examples of generative art created with NodeBox

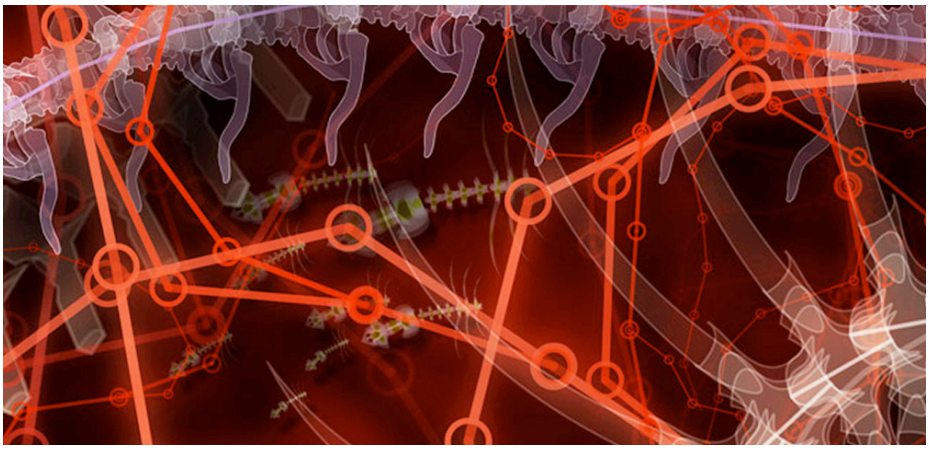

Fig. 2. "Creature": 350x150cm panel created for the department of Morphology, University of Ghent. It was realized using a recursive approach to simulate veins and skeleton structures.

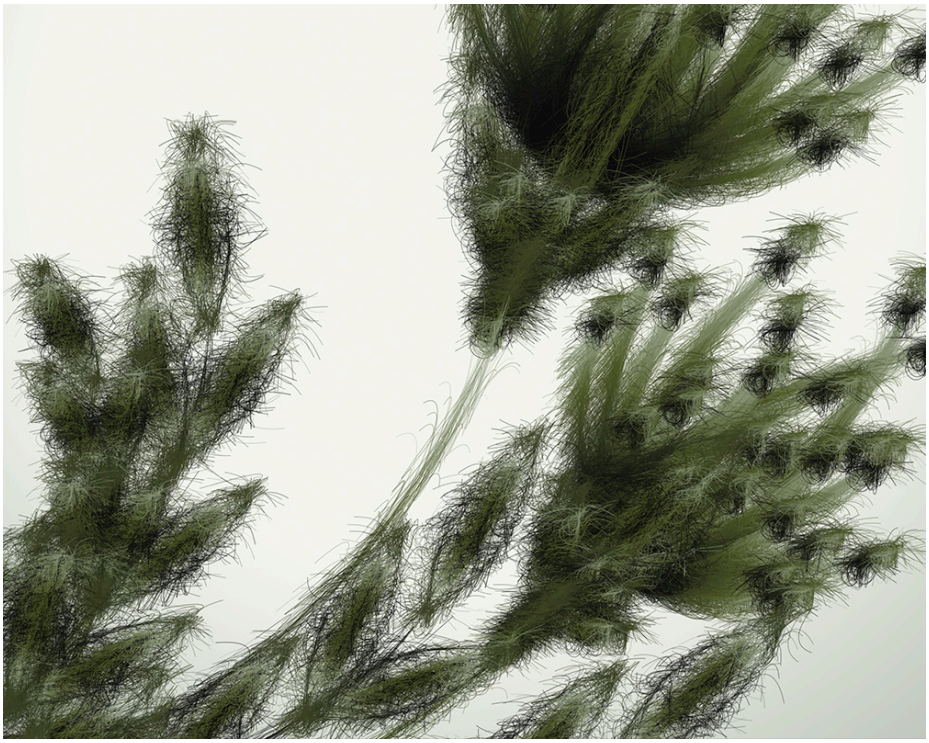

Fig. 3. "Superfolia": 6 panels 70x150cm realized using an agent-based approach (a single blade of grass responds to its neighbors).

Fig. 4. "Nanophysical": 66.5x2.5m wall design at IMEC (European institute for nanotechnology). The work regards the hall window as a source of energy and then evolves along the walls, using (among other) a force-based physics algorithm.

4 Computational creativity

A user interface with programming code introduces a steep learning curve for users not trained in programming. In a recent research project ("Gravital"), we have attempted to alleviate this shortcoming by providing a node-based interface.[4] Visual building blocks (nodes) can be connected in the interface to create interesting visual effects. Building blocks can be opened to examine and edit their source code.

Furthermore, a programming tool for the visual arts is useful in terms of production-intensive tasks, but it does not provide leverage on *what* to make—what ideas are "interesting" from a creative standpoint. The second aim in the Gravital project was to develop a set of algorithms to find creative associations and analogies between concepts (i.e., words), to help users discover interesting new ideas. The system uses a memory-based shallow parser [8], a semantic network of commonsense [22] and heuristic search techniques. We hypothesize that this system can be used to simulate conceptual brainstorms based on natural language input.

4.1 Memory-based shallow parser

The first task in the system is to transform information in natural language sentences to a meaning representation language. This task has a long history in AI, and in practice the translation of natural language into a deep, unambiguous representation (i.e., understanding) turned out to be impossible (except for small domains where all

[4] NodeBox 2, beta version, http://beta.nodebox.net/

relevant background knowledge was explicitly modeled, see for example [23]). Natural language processing (NLP) has since switched to robust, efficient and reasonably accurate methods that analyze text to a more superficial partially syntactic and partially semantic representation (shallow parsing), using machine learning and statistical methods trained on large annotated corpora.

The shallow parser used by our system is MBSP; a memory-based shallow parser implemented as memory-based learning modules using the Machine Learning package TiMBL [7]. Memory-based learning is a form of exemplar-based learning that is based on the idea that language processing involves specific exemplars of language use, stored in memory. With MBSP we can process user input in the form of natural language (i.e., English) and mine relevant concepts from it. These are then processed further with the Perception[5] solver: a semantic network traversed with heuristic search techniques.

4.2 Semantic network of commonsense

For example, assume we have a drawing machine that can draw either circles or rectangles, in any color. The task "draw a circle" is trivial and can be solved by the user himself without having to rely on NLP algorithms. The task "don't draw anything except an ellipse preferably of equal width and height" is quite complex to solve in terms of NLP, and perhaps not worth the effort. However: "draw the sun" poses an interesting challenge. What does the sun look like? Given the possibilities of our drawing machine, a human might translate the "sun" concept to an orange circle. This kind of conceptual association is a form of human creativity [15], which we attempt to simulate using a semantic network of related concepts. When given the word "sun", Perception will propose colors such as orange and yellow, and shapes such as a circle or a star.

To illustrate this further, say we are looking for images of creepy animals. The system could search the web for images named `creepy-animal.jpg`, but that is not very creative. What we want is a system that imitates an artistic brainstorming process: thinking about what animals look like, what the properties of each animal are, which of these properties can be regarded as creepy, and look for pictures of those animals. In this particular example the Perception solver suggests such animals as octopus, bat, crow, locust, mayfly, termite, tick, toad, spider, ... No frolicking ponies or fluffy bunnies here! For the octopus the logic is obvious: the semantic network has a direct *creepy is-property-of octopus* relation. The bat (second result) has no *is-creepy* relation however, only a set of relations to *black*, *cave*, *night* and *radar*. What happens here is that many aspects of a bat are inferred as a strong causal chain [21] leading to creepiness. Let us clarify the meaning of "many aspects".

In [4], Hofstadter argues that AI-representations of human high-level perception require a degree of flexibility (or fluidity), where objects and situations can be comprehended in many different ways, depending on the context. To reflect this, Perception's solver uses clusters of concepts as its basic unit for reasoning, instead of a single concept. Concepts are surrounded by other concepts that reinforce meaning.

[5] Perception module, beta version, http://nodebox.net/code/index.php/Perception

A concept cluster is the concept itself, its directly related concepts, concepts related to those concepts, and so on, as deep as the representation requires (we used depth 2). This is called spreading activation [6]. Activation spreads out from the starting concept in a gradient of decreasing relatedness. What defines the bat concept are its directly surrounding concepts: *black*, *cave*, *night*, *radar*, and concepts directly related to these concepts: *Darth Vader*, *dark*, *dangerous*, *deep*, *evil*, *cat*, *airplane*, *sky*, *nothing*, ... Several of these have a short path [9] in the network to *dark*, and *dark* is directly related to *creepy*. The sum of the shortest path length to *creepy* is significantly less than (for example) the path score of the cluster defining *bunny*. A bat has many dark aspects, and dark is pretty creepy.

Note that different concepts in a cluster have a higher or lower influence on the final score. For the bat concept, the distance between *dark* and *creepy* is more essential than the distance between *deep* and *creepy*. This is because *dark* is more central in the bat cluster when we calculate its betweenness centrality [3]. More connections between concepts in the cluster pass through *dark*. We take *dark* as a sort of conceptual glue when reasoning about bats.

Using conceptual association, we think the system can be useful for human designers to come up with more creative ideas, or to find visual solutions for abstract concepts (e.g., *jazz = blue*).

Future work

Section 4 presents a preliminary computational approach to simulate brainstorming. In future research, we will investigate if this is indeed how human brainstorming works, and if the analogies the system comes up with are “good” or “bad” creative finds. This will introduce new challenges, since what is “good” or what is “bad” appears to involve numerous cultural and personal factors.

Acknowledgements

The Gravital research project was funded by the agency for Innovation by Science and Technology (IWT), Belgium, 2007-2009. City In A Bottle is funded by the Flemish Audiovisual Fund (VAF), Belgium, 2008-2011.

References

1. Blackwell, T. M., Bentley, P.: Improvised music with swarms. In: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress. vol. 02, pp. 1462-1467 (2002).
2. Boden, M.: What is generative art? In: Digital Creativity, vol. 20, pp. 21-46 (2009).
3. Brandes, U.: A Faster Algorithm for Betweenness Centrality. In: Journal of Mathematical Sociology (2001).

4. Chalmers, D. J., French, R. M., Hofstadter, D. R.: High-Level Perception, Representation, and Analogy: A Critique of Artificial Intelligence Methodology. In: Journal of Experimental & Theoretical Artificial Intelligence (1991).
5. Cleveland, P.: Bound to technology - the telltale signs in print. In: Design Studies, vol. 25, no. 2, pp. 113-153 (2004).
6. Collins, A. M., Loftus, E. F.: A Spreading-Activation Theory of Semantic Processing. In: Psychological Review, vol. 82, no. 6, pp. 407-428 (1975).
7. Daelemans, W., Zavrel, J., van der Sloot, K., van den Bosch, A. (2004). Timbl: Tilburg memory-based learner. In: Tech. Report ILK 04-02, Tilburg University, December 2004.
8. Daelemans, W., van den Bosch, A.: Memory-based language processing. In: Studies in Natural Language Processing, Cambridge University Press (2005).
9. Dijkstra, E. W.: A note on two problems in connexion with graphs. In: Numerische Mathematik, vol. 1, pp. 269-271 (1959).
10. Gielis, J.: A generic geometric transformation that unifies a wide range of natural and abstract shapes. In: American Journal of Botany. vol. 90, pp. 333-338 (2003).
11. Goldstein, J.: Emergence as a Construct: History and Issues. In: Emergence: Complexity and Organization, vol. 1, pp. 49-72 (1999).
12. Hu, J., Goodman, E.: The hierarchical fair competition (HFC) model for parallel evolutionary algorithms. In: Proceedings of the Congress on Evolutionary Computation, pp. 49-54 (2002).
13. Kauffman, S. A.: The origins of order: self-organization and selection in evolution. Oxford University Press (1993).
14. McCormack, J., Dorin, A.: Art, Emergence and the Computational Sublime. In: A conference on generative systems in the electronic arts, CEMA, Melbourne, Australia, pp. 67-81 (2001).
15. Mednick, S.: The Associative Basis of the Creative Process. In: Psychological Review, vol. 69, pp. 220-232 (1962).
16. Olivero, G.: Spamghetto. In: Data Flow 2, Visualizing Information in Graphic Design, pp. 165. Gestalten, Berlin (2010).
17. Perlin, K.: Improving noise. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques (SIGGRAPH '02), pp. 681-682 (2002).
18. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer-Verlag (1990).
19. Reas, C., Fry, B.: Processing: a programming handbook for visual designers and artists. MIT Press (2007).
20. Reynolds, C. W.: Flocks, Herds, and Schools: A Distributed Behavioral Model. In: Computer Graphics, vol. 21, no. 4 (1987).
21. Schank, R. C., Abelson, R. P.: Scripts, plans, goals, and understanding: an inquiry into human knowledge structures. Lawrence Erlbaum Associates (1977).
22. Sowa, J. F.: Semantic Networks. In: Encyclopedia of Artificial Intelligence, edited by S. C. Shapiro, Wiley, New York, 1987; revised and extended for the second edition (1992).
23. Winograd, T.: Understanding Natural Language. Academic Press (1972).