P&D
DESIGN 2012

# Fluidiom, inovação através de processo evolutivo em ambiente computacional

*Fluidiom, innovation through evolutive process in computational environment*

Alão, Rui S. D.; Ms; Universidade Anhembi Morumbi e
doutorando; FAUUSP
ruialao@gmail.com.br

## Resumo

O artigo discute o caráter inovador dos resultados obtidos através do uso de algoritmos computacionais na área de design. O objeto de estudo que utilizamos é o projeto Fluidiom, que simula aspectos evolutivos em seus algoritmos. O caráter inesperado das soluções obtidas traz a questão da inovação levada a cabo por processos computacionais. O projeto é descrito em cada uma de suas fases e algumas conclusões são apontadas a partir da análise dos resultados do experimento.

**Palavras Chave:** design**,** inovação; algoritmo.

## Abstract

*The paper deals with the innovative role of the results obtained through computational algorithms in the field of design. Our object of study is Fluidiom, an internet initiative that simulates evolutive aspects in its algorithms. The unexpected character of the obtained solutions brings the issue of the innovation created through computational processes. The project is described in each of its phases and some conclusions are drawn from the analysis of the experiment results.*

***Keywords:*** *design, innovation; algorithm.*

## Introdução

Por mais que se afirme que se articulam equipes multidisciplinares para dar conta de projetos complexos na área de design, o resultado final de tais processos ainda se encontra atrelado às vontades de seus componentes. Como enfrentamos problemas cada vez mais complexos, com múltiplas variáveis interdependentes, entendemos que a dependência de um projetista é uma fragilidade, já que há limites humanos para a compreensão e tratamento de realidades complexas. Esta dependência do projetista nos leva a uma questão: o quanto algoritmos computacionais podem atuar para modificar esta relação de dependência? Ou ainda, se seria possível contar com algum tipo de contribuição por parte dos algoritmos no sentido de criar novas soluções de projeto? É possível falar com inovação em relação ao uso de algoritmos? Existem algoritmos que podem exercer algum tipo de "criação"?

Este tipo de pergunta foi abordado algumas vezes, com diferentes respostas.

É frequente, por exemplo, a resposta de que o computador só faz o que está no programa, o que levaria à conclusão de que não há terreno para inovação por parte do algoritmo, mas somente por parte do programador. No entanto, existem casos em que a relação tende mais a uma espécie de co-criação (JOHNSON, 2011, p. 95).

Assim, entra em cena a possibilidade de delegar parte do nosso trabalho a algoritmos computacionais, de forma a criar uma gama de possíveis soluções sobre as quais podemos exercer nosso juízo quanto às suas adequações na solução do problema proposto.

Estas soluções levam o designer a uma posição de afastamento em relação à solução, na medida em que passamos a exercer nossos critérios de adequação sobre os passos do algoritmo, direcionando-o a satisfazer nossas espectativas quanto aos objetivos que devem ser alcançados. Esta postura, na qual o designer se ocupa mais de organizar as estratégias de articulação de um ambiente do que em resolver ele mesmo o problema se aproxima dos procedimentos típicos do metadesign. Passamos então a 'projetar o projeto' da solução que pretendemos alcançar e não a projetar a solução diretamente.

Neste cenário, nos interessa em especial o caráter desta inovação computacional, nos interessa detectar um caminho através do qual é possível chegar ao 'novo' através destes algoritmos.

O projeto que vamos apresentar a seguir, do qual participamos e que foi alvo de nosso estudo no curso de mestrado na área de design (ALÃO, 2008), trouxe consigo discussões acerca destes caminhos, e que desembocou em soluções inesperadas e imprevisíveis. A seguir, pretendemos apresentar o projeto de forma detalhada para, posteriormente, discutir o caráter de suas soluções no que diz respeito à inovação.

## O projeto Fluidiom

O projeto Fluidiom foi criado por Gerald de Jong, matemático canadense radicado na Holanda, em meados de 2001 com o intuito de simular o processo evolutivo em ambiente computacional. Ele chamou o projeto de um "exercício em ciberbiologia". O projeto se insere na iniciativa Darwin At Home (*www.darwinathome.org*).

> Darwin at Home é um projeto *open source* que almeja trazer o processo evolutivo para dentro de seu computador em casa, para que você possa vê-lo em funcionamento. (de JONG, 2005, online)[1]

[1] No original: Darwin at Home is an open source software project that aims to bring the process of evolution into your computer at home so that you can see it working.

Pode-se dizer, portanto, que se configura como uma iniciativa de a-Life, vida artificial, já que simula a complexidade de alguns aspectos da vida biológica através de algoritmos computacionais.

O experimento se dá através da criação e evolução simulada de estruturas tridimensionais que se configuram como criaturas, a partir da articulação entre tetraedros. Estes, por sua vez, são compostos de vetores elásticos que se comprimem e se esticam repetidamente. Cada vetor elástico tem o seu próprio ciclo, com um timing independente dos outros. Combinados, os diferentes timings de compressão e distensão dos vários vetores fazem emergir na criatura um certo movimento. Inicialmente a criatura tem seus ciclos de compressão e distensão todos em assincronia, o que faz com que ela se debata, quase sem sair do lugar.

O processo evolutivo aplicado às criaturas envolve a variação dos timings dos ciclos de cada vetor elástico para que, em conjunto, surja um movimento eficaz no corpo como um todo. O objetivo primário do projeto é fazer com que estas estruturas encontrem, através de um processo de auto-organização, formas inovadoras de resolver o problema deste deslocamento. Ou seja, o objetivo maior do experimento é fazer surgir espontaneamente na criatura uma forma de deslocamento eficiente.

Conforme define seu autor:

> Fluidiom é um programa construído em Java para investigar a evolução por sobrevivência do mais apto. Usuários criam estruturas de corpos 3D feitas de intervalos elásticos e músculos aleatórios, e Fluidiom os sujeita à evolução com a intenção de criar novas gerações.(de JONG, 2005, online)[2]

O interessante da iniciativa é que a estratégia que cada objeto (ou criatura) deve usar para deslocar-se não é programada *a priori*, mas emerge a partir de um mecanismo de controle que tenta emular o processo seletivo muito semelhante ao descrito por Darwin.

O processo gera uma grande diversidade de soluções para o problema do deslocamento, como iremos demonstrar. Ao final do processo, algumas criaturas fazem seu corpo todo se contorcer de modo sincronizado como uma cobra, outras pulam, outras se arrastam, outras ainda caminham apressadamente. E estes comportamentos surgem a partir de mudanças aleatórias na intensidade e na sincronia dos ciclos de seus vetores elásticos.

> Os corpos das criaturas de Fluidiom consistem em uma coleção de intervalos elásticos, articulados pelas pontas uns com os outros. Cada "mola" é, na verdade, apenas a relação entre dois pontos no espaço, chamados "juntas", de modo que elas tendem a ficar a uma determinada distância uma da outra. Quando as juntas estão muito próximas, o intervalo as empurra e quando estão muito distantes, ele puxa. (de JONG, 2005, online)[3]

Na figura abaixo vemos um objeto com seus vetores elásticos. Os representados em vermelho estão comprimidos, isto é, menores que seu tamanho original, e os azuis estão

[2] No original: Fluidiom is a Java program built to experiment with evolution by survival of the fittest. Users create 3D body structures made of elastic intervals and random muscles, and Fluidiom subjects them to evolution with the intent of having future generations

[3] No original: The bodies of Fluidiom creatures consist of a collection of individual springy elastic "intervals", joined at their ends with others. Each spring is actually just a relationship between two points in space, called "joints", such that they tend to stay at a particular distance from each other. When the joints are too close together the interval pushes, and when they are too far it pulls.

esticados. Os representados em cinza estão próximos de seu comprimento original e, portanto, permanecem em repouso.

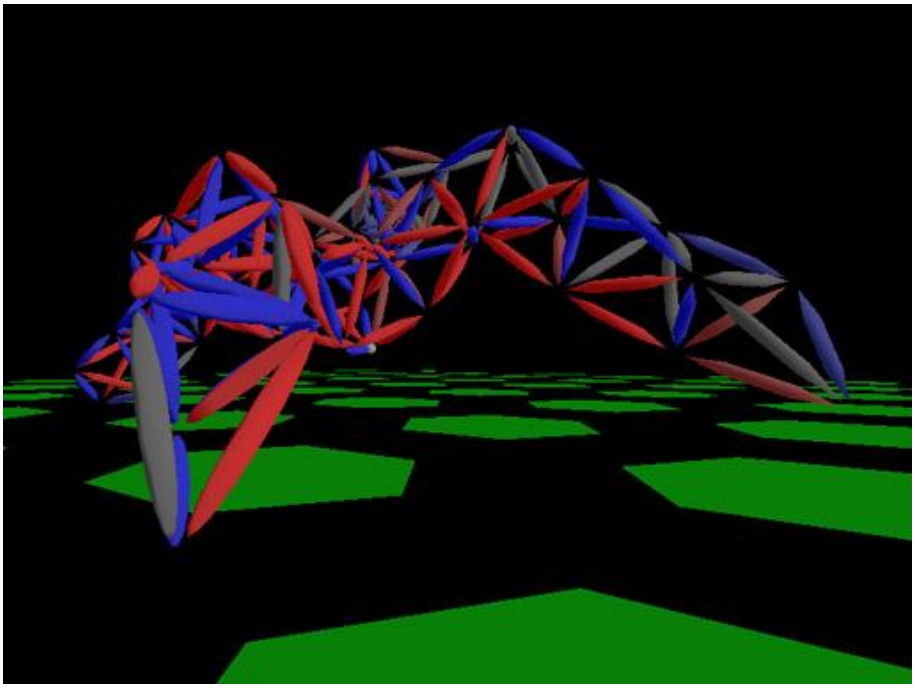

Figura 1: Criatura dentro do ambiente Fluidiom. Os vetores comprimidos são mostrados em vermelho e os esticados, em azul.
(fonte: captura de tela feita pelo autor)

O projeto Fluidiom foi feito para que usuários possam projetar suas criaturas a partir destes tetraedros com arestas elásticas. A idéia é que a compressão e distensão destas arestas resultem em movimento para a criatura como um todo. O projeto é montado de forma a selecionar as versões de intervalos que resultem em maior eficiência no movimento global da criatura. Ao fim de várias gerações de versões da criatura, ela deverá se movimentar de forma eficiente. Este processo evolutivo da criatura segue algumas etapas, as quais iremos descrever a seguir.

## Primeira etapa: criação da geometria do ambiente

Assim que o usuário baixa o programa Java em sua máquina, ele tem a opção de usar uma das criaturas já feitas por algum outro usuário ou criar a sua própria. Se optar por usar uma criatura pronta, pula para a segunda etapa, descrita mais à frente. Caso contrário, é chamado a se cadastrar e a criar geometricamente a sua própria criatura. Abre-se então uma interface com um editor que simula o espaço tridimensional. Nela, ele define tetraedros que se justapõem. Em cada aresta de cada tetraedro se encontra um vetor elástico. Pode-se girar a criatura, criar tetraedros ou octaedros, selecionar, acrescentar, modificar ou eliminar nós ou arestas, e mesmo gerar um espelhamento tendo como base uma face selecionada.

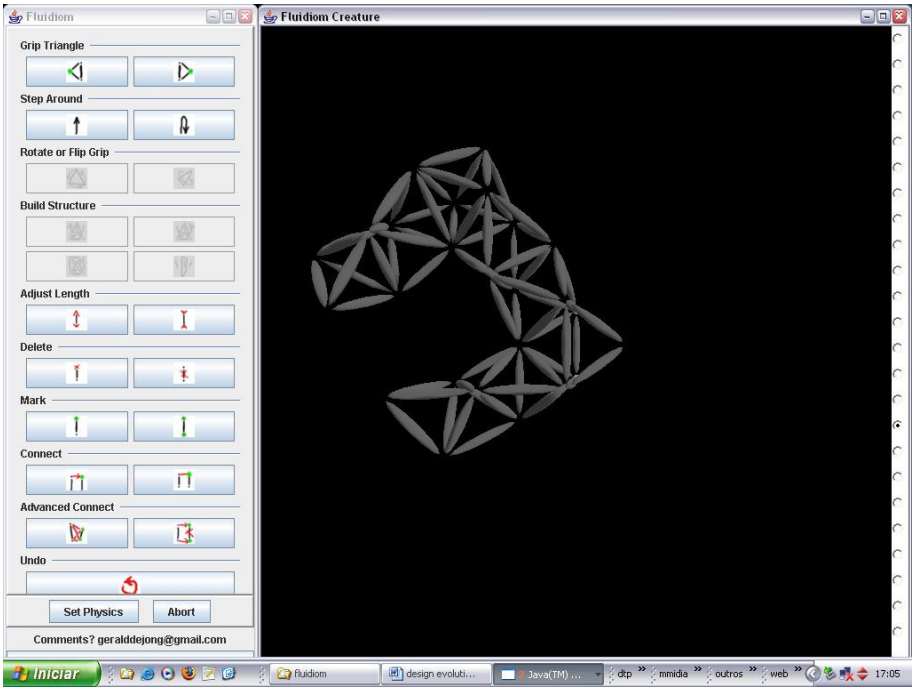


Figura 2: Interface de edição tridimensional da criatura através de tetraedros. (fonte: captura de tela feita pelo autor)

À esquerda vemos os botões que comandam a criação e a edição dos vetores que irão compor a estrutura da criatura, que é representada à direita. Todo o corpo da criatura pode ser rotacionado livremente em ambiente tridimensional para melhor visualização.

Depois de criada geometricamente, a criatura deve ser inserida num ambiente de simulação. Nele, o usuário pode configurar algumas de suas características: a gravidade, a resistência, a rigidez e o tempo do ciclo a ser imposto à criatura, etc.

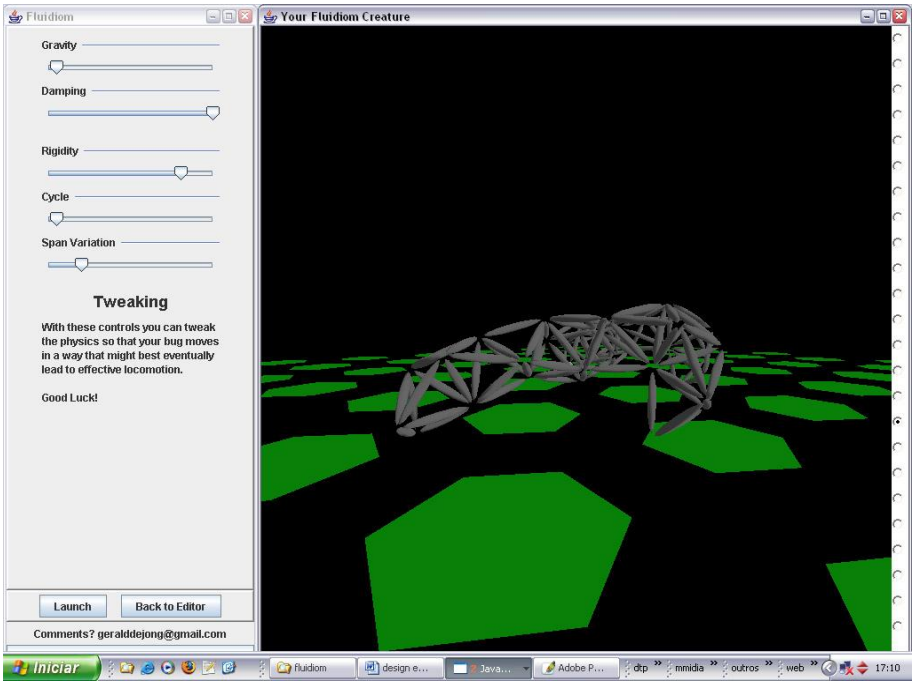


Figura 3: Interface de configuração dos parâmetros ambientais (fonte: captura de tela feita pelo autor)

Definidos estes parâmetros, a estrutura está pronta para ser batizada.

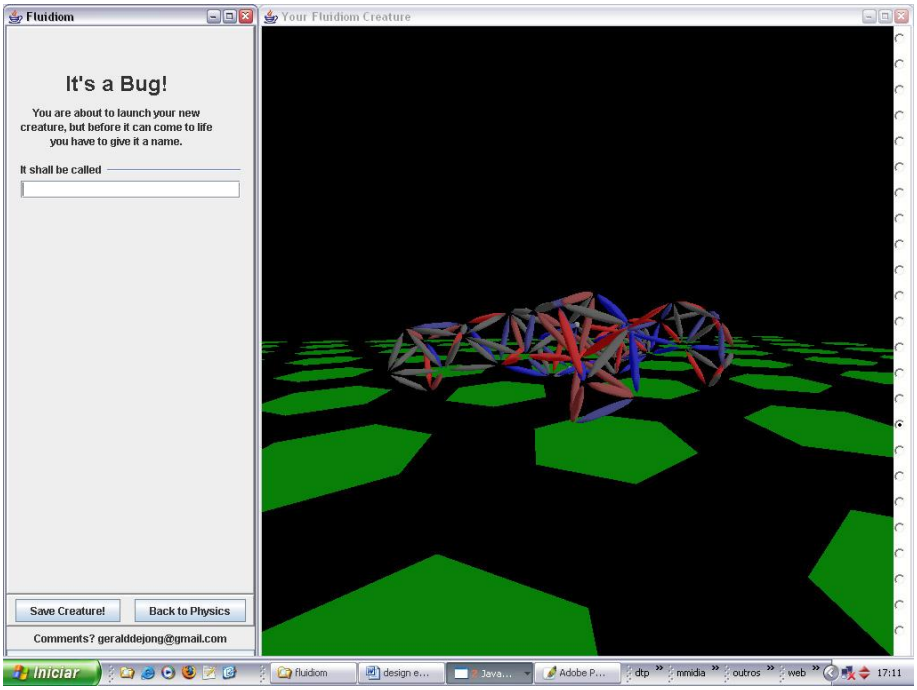


Figura 4: Interface de batismo da criatura. Fim do ciclo de criação
(fonte: captura de tela feita pelo autor)

## Segunda etapa: variação e seleção

Em seguida, o usuário pode colocar a sua criatura num ambiente de simulação do processo evolutivo.

As arestas da criatura são vetores elásticos e sua geometria é definida na sua criação e não pode ser mudada. O movimento da criatura, no entanto, é dado pela sincronia entre as contrações e distensões de suas arestas.

É fácil perceber que algumas combinações de sincronias entre as arestas dão à criatura algum movimento e outras sincronias só a fazem debater-se sem sair do lugar. A ideia é fazer com que a criatura ganhe locomoção através da reprodução de características que fazem a criatura se locomover. As variações se dão nas arestas, mas o movimento se dá na escala da criatura como um todo. Assim, a locomoção emerge das variações impostas às arestas.

Embora o usuário tenha acesso a alguns parâmetros que influenciam nestes ajustes, ele não tem acesso à sincronia de cada vetor elástico. O usuário altera parâmetros ambientais, e o ambiente interfere no algoritmo evolutivo que faz a seleção de novas instâncias de criaturas.

Os parâmetros disponíveis ao usuário são:

- Tamanho da população
- Taxa de mutação
- Escala entre mutação global e local
- Duração do ciclo de mutação
- Mutações devem seguir uma simetria ou não
- Interações
- Se o objetivo é andar, trotar ou correr

Uma vez configurados os parâmetros evolutivos, o processo seletivo começa.

Em primeiro lugar, são gerados, aleatoriamente, diferentes ciclos para diferentes vetores elásticos. Nas primeiras gerações de criaturas, quando a sincronia entre os movimentos dos vetores é ainda bastante primária, o movimento que emerge na criatura é bastante desengonçado, muito pouco eficiente. O sistema, no entanto, não gera apenas uma solução, isto é, uma combinação de ciclos para uma criatura, mas várias combinações, várias candidatas a soluções a partir de variações aleatórias de suas sincronias. Destas soluções, como seria de se esperar, algumas terão sincronias que serão mais eficientes que outras.

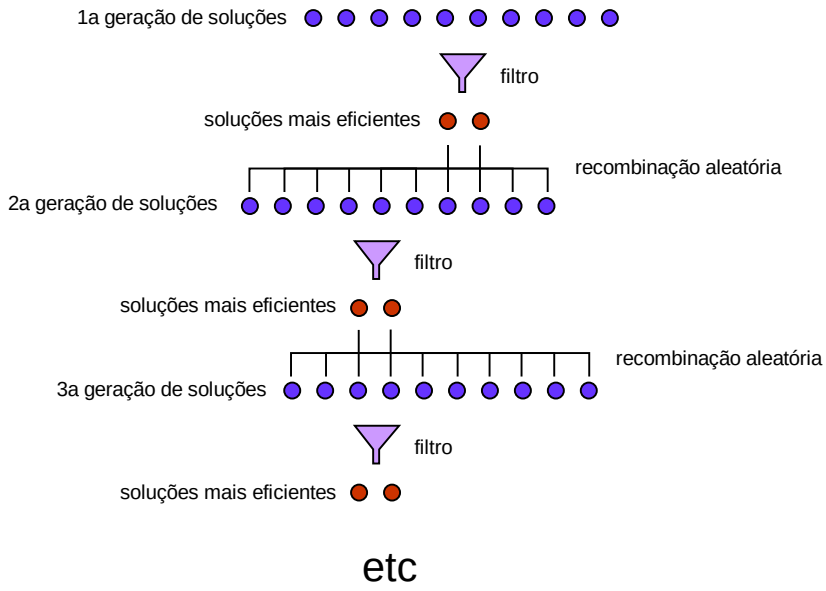


Figura 5: Modelo esquemático do funcionamento de Fluidiom, mostrando a dinâmica da sucessão de gerações de soluções.
(fonte: ilustração do autor)

Num segundo momento, o sistema promove a aplicação de um filtro sobre as várias soluções: as que obtiverem maior grau de deslocamento são selecionadas para se tornarem "mães" da próxima geração de soluções; as que não obtiveram bom desempenho são descartadas. Estas "mães" dão origem a uma segunda geração de soluções também através de variações aleatórias de si mesmas, tanto no que diz respeito à intensidade dos alongamentos e contrações quanto do timing de cada ciclo. Quando todas as criaturas da segunda geração tiverem sido geradas, promove-se novamente a aplicação de um filtro, e novas 'mães" são escolhidas, e o ciclo recomeça. Este processo é repetido inúmeras vezes automaticamente pelo software.

Desse modo, alternam-se etapas de geração aleatória e de filtragem das soluções. Naturalmente, como as novas gerações são escolhidas a partir de uma avaliação de desempenho, as gerações subsequentes têm grande chance de ter desempenho melhor que as anteriores. O desempenho médio das soluções aumenta a cada geração.

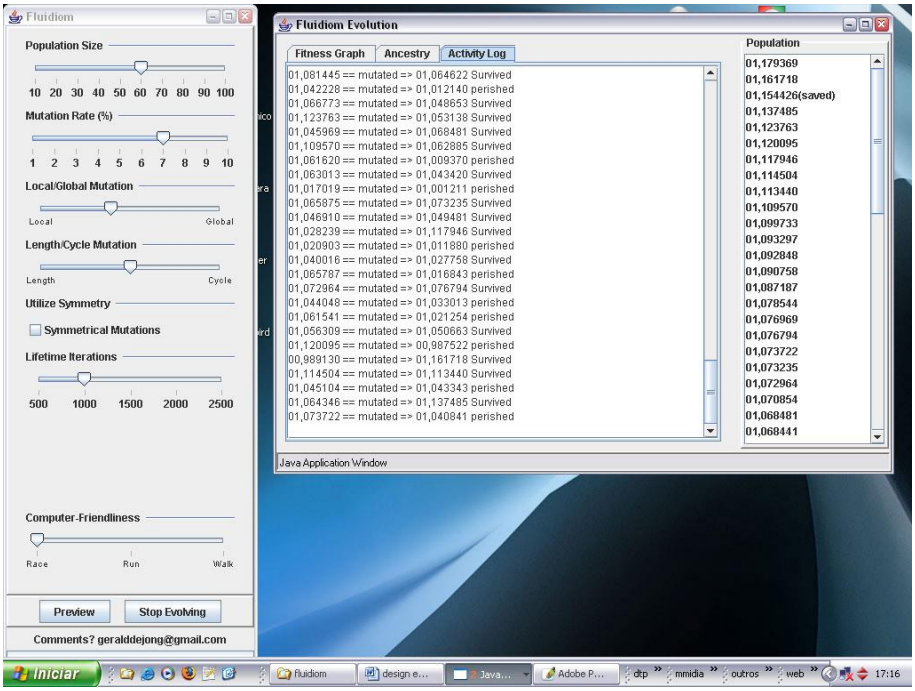


Figura 6: Interface de configuração de parâmetros evolutivos, listados à esquerda.
(fonte: captura de tela feita pelo autor)

Na figura acima pode-se ver os parâmetros disponíveis para o usuário: tamanho da população, taxa de mutação, mutações globais ou locais, etc. À direita vê-se o *activity log*, ou seja, o registro de atividades com a sucessão de gerações. Algumas instâncias sobrevivem e geram descendência (na lista, marcados como *survived*), enquanto outras morrem (marcados como *perished*).

Note-se que os parâmetros usados dizem respeito não diretamente à sincronia, mas à como deve se dar a evolução dessa sincronia.

Esta fase lida com o cerne do mecanismo evolutivo do aplicativo. Jong o descreve assim

> Cada trecho de músculos alterado pela mutação resulta em algum jeito diferente de fazer o movimento de puxar, e cada um vai mover a criatura a alguma distância em relação ao seu ponto de partida durante sua vida. Tudo o que temos que fazer é favorecer os mais rápidos sobre aqueles que não são tão eficientes em sua locomoção. As criaturas mais rápidas devem deixar mais filhos (ou mutações, no caso), como o guepardo mais rápido o faz. (de JONG, 2005, online)[4]

[4] No original: Each mutated set of muscles results in some kind of twitching motion, and each kind of twitch will move the creature some distance away from the starting point during its lifetime. All we have to do is favor the faster ones over those who are not so proficient at locomotion. The faster creatures must somehow produce more children (or mutations, in this case), like the faster cheetah might.

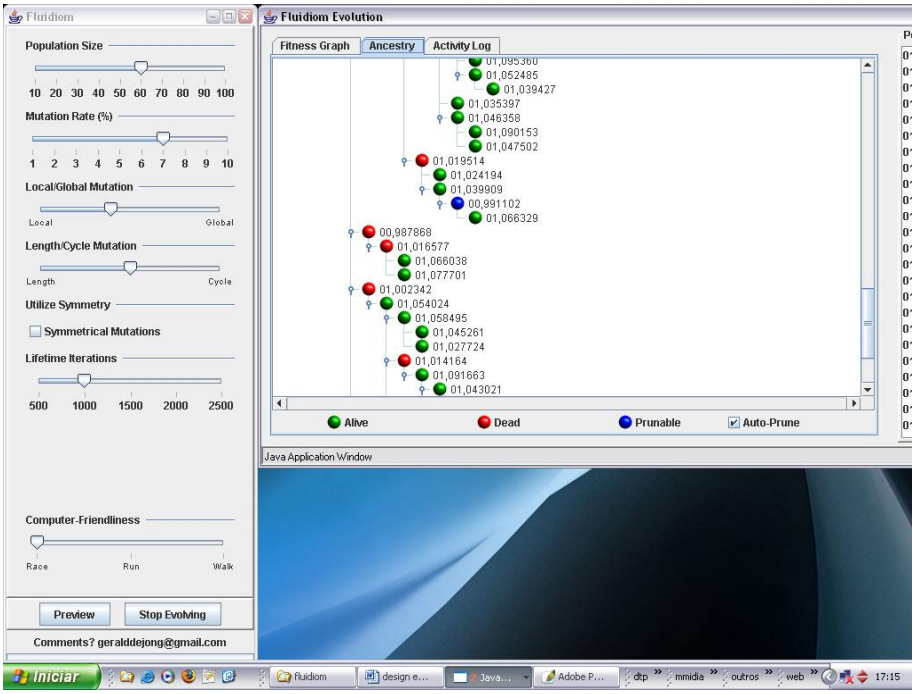


Figura 7: Mapa de ancestralidade. Os números que aparecem ao lado das criaturas representam a velocidade de cada exemplar. Em geral, este número tende a crescer com a sucessão das gerações. (fonte: captura de tela feita pelo autor)

E, finalmente,

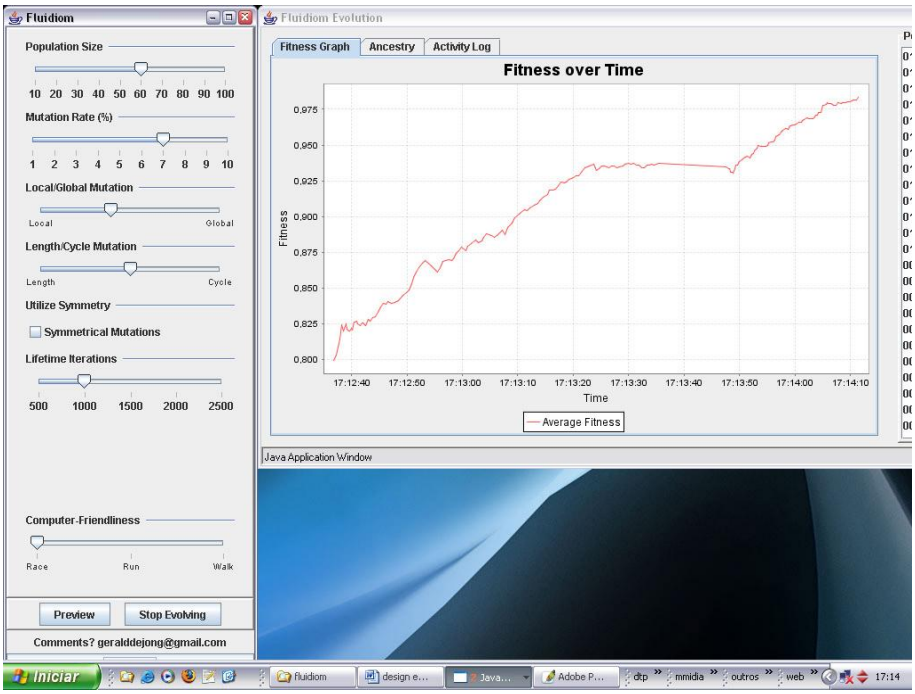


Figura 8: Gráfico de eficácia (Fitness Graph). A medida da velocidade da criatura crescendo ao longo do tempo do experimento.
(fonte: captura de tela feita pelo autor)

Na figura anterior podemos ver o gráfico de eficácia (*fitness graph*). No eixo horizontal, o tempo, no vertical, a medida de eficácia da criatura em termos de seu deslocamento. Vê-se que, em geral, a tendência da evolução simulada é melhorar a eficácia de deslocamento da criatura, muito embora às vezes essa medida possa diminuir um pouco para mais tarde voltar a subir.

## Terceira etapa: publicação

Uma vez que o usuário esteja satisfeito com o resultado de sua criatura, ele pode publica-la. Outros usuários podem ter acesso à sua criatura em particular e podem apreciar seu deslocamento (as criaturas tendem a desenvolver várias formas de locomoção como resultado de sua evolução artificial) ou podem também utilizar a criatura para evoluir novamente, alterando os parâmetros de evolução, com o intuito de obter um resultado melhor ainda.

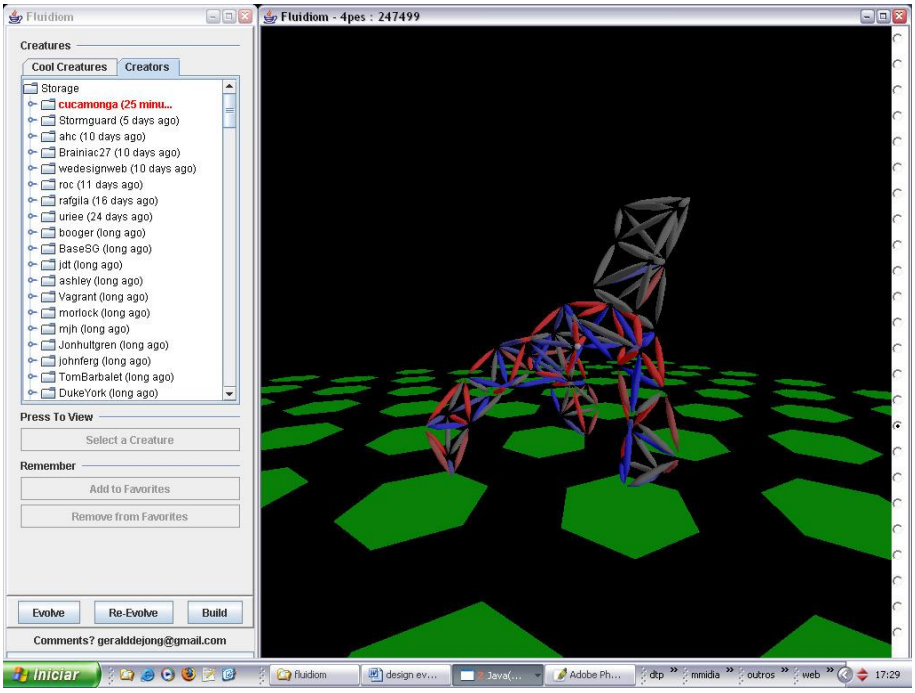


Figura 9: Publicação da criatura na web. À esquerda, vê-se a lista de usuários do projeto. (fonte: captura de tela feita pelo autor)

O fato do sistema ser aberto e permitir este tipo de apropriação faz com que criaturas "fracas" possam evoluir ajudadas por usuários mais experientes, dando-lhes, assim, uma segunda (ou terceira, quarta) chance evolutiva.

A publicação da criatura, portanto, não finaliza a sua evolução, mas apenas a coloca à disposição de outros usuários (e, portanto, a novas estratégias evolutivas) para ganhar novas configurações.

Assim como acontece com outras iniciativas de vida artificial, não há propriamente um fim do ciclo de vida de uma criatura dentro do experimento. Ele está acontecendo a cada vez que algum usuário o usa em sua máquina e publica uma nova criatura.

## Análise de Fluidiom: fenômenos emergentes e inovação

O que nos interessou na iniciativa Fludiom foi o uso intensivo de fenômenos emergentes para o processo projetual. Em nossa opinião, após análise, foi esta característica que viabilizou a ocorrência de soluções inesperadas no experimento.

Detectamos três níveis diferentes nos quais enxergamos a ocorrência de fenômenos emergentes no experimento.

Em primeiro lugar, o projeto é *open source* e, embora administrado por de Jong, se encontra à disposição de outros programadores para que estes possam incluir código no projeto ou mesmo apropriar-se dele e criar o seu próprio experimento. Temos, assim, todas as características do paradigma projetual *open source,* que se configura como um sistema complexo adaptativo (um SCA): quebra do ciclo de lançamento, diluição das fases de desenvolvimento, uso da inteligência coletiva e auto-organização. Neste caso, os agentes do SCA são os programadores e os códigos que geram para o projeto.(ALÃO, 2008, p. 41)

Em segundo, é permitido ao usuário (e não só ao programador) utilizar-se do trabalho de outro usuário — no caso, tanto a configuração tridimensional da criatura quanto a evolução já feita anteriormente — para obter sucesso no desenvolvimento de sua criatura. Pode-se, por exemplo, ao consultar a lista de criaturas, deparar-se com alguma que tenha uma forma instigante e se apropriar dela para redirecionar seu desenvolvimento. É comum trabalhar numa criatura que já tenha passado por vários outros usuários antes. Na verdade, é virtualmente impossível descobrir quantos usuários trabalharam numa dada criatura através da interface, embora se possa traçar uma estimativa na medida em que é possível identificar semelhanças morfológicas entre criaturas da lista. Nesse sentido, a noção de autoria simplesmente não se aplica, ou antes, é substituída pela de co-autoria, uma das características do metadesign que Giaccardi (2004) propõe. Neste caso, os agentes do SCA são os usuários e as criaturas das quais se apropriam.

Em terceiro, há ainda a emergência gerada pelo código de programação responsável pela evolução das criaturas. Como dito anteriormente, a sincronia entre os vários vetores elásticos é, em si, um processo emergente de caráter computacional. É a partir das múltiplas variações provocadas nesta sincronia que são obtidas novas instâncias de criaturas mais eficientes. Neste caso, os agentes dos SCA são os *timings* dos vetores elásticos e as instâncias evolutivas de cada criatura.

É interessante notar que cada um destes níveis dá margem à inovação: o código aberto, o compartilhamento das criaturas (inovação através da inteligência coletiva e co-criação) e também a inovação que emerge da aplicação do código evolutivo por si só. O fato de se usar variações aleatórias dá margem ao inesperado que, de fato ocorre no experimento.

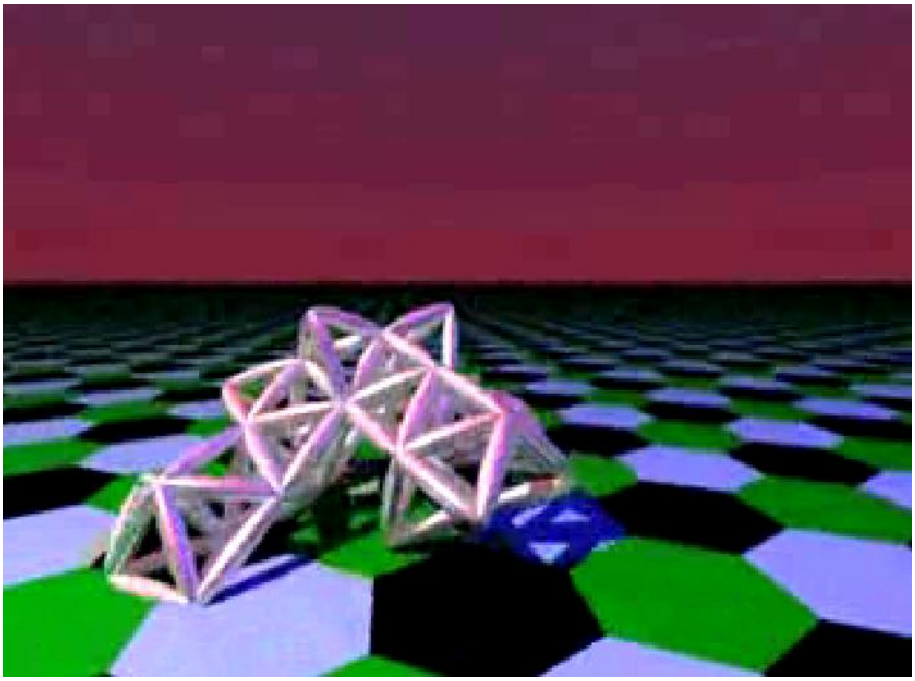

Figura 10: Criatura que se locomove caminhando.
(fonte: captura de tela feita pelo autor)

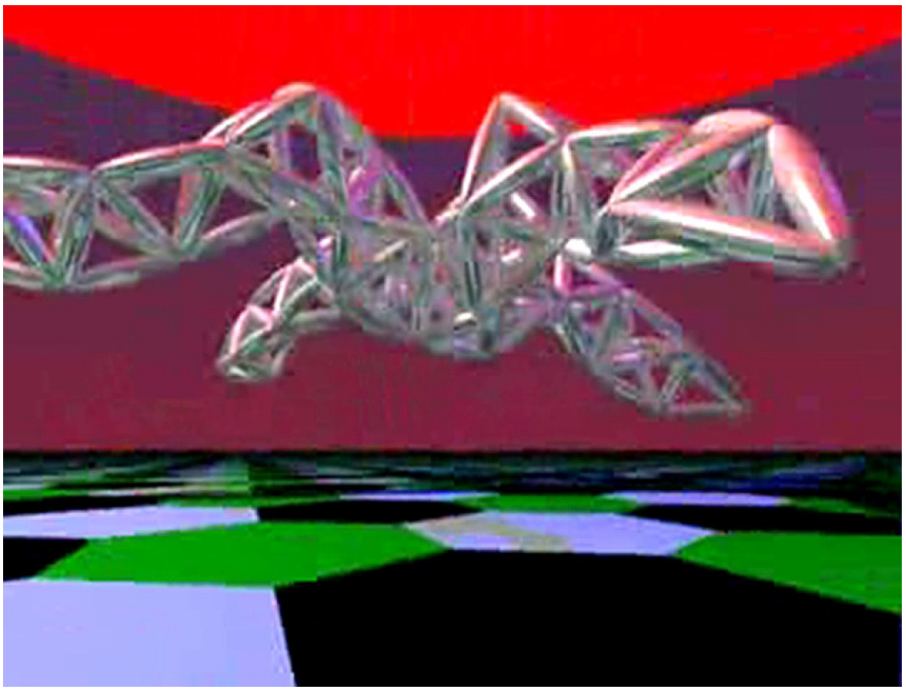

Figura 11: Criatura que se locomove dando saltos.
(fonte: captura de tela feita pelo autor)

Algumas criaturas ‘aprenderam’ a caminhar, outras a se arrastar, outras a pular, muito embora nenhuma delas tenha recebido qualquer tipo de instrução neste sentido. Estes comportamentos emergiram do algoritmo cujo único critério de avaliação era: as criaturas que se deslocarem mais continuarão no experimento e serão base para outras criaturas futuras.

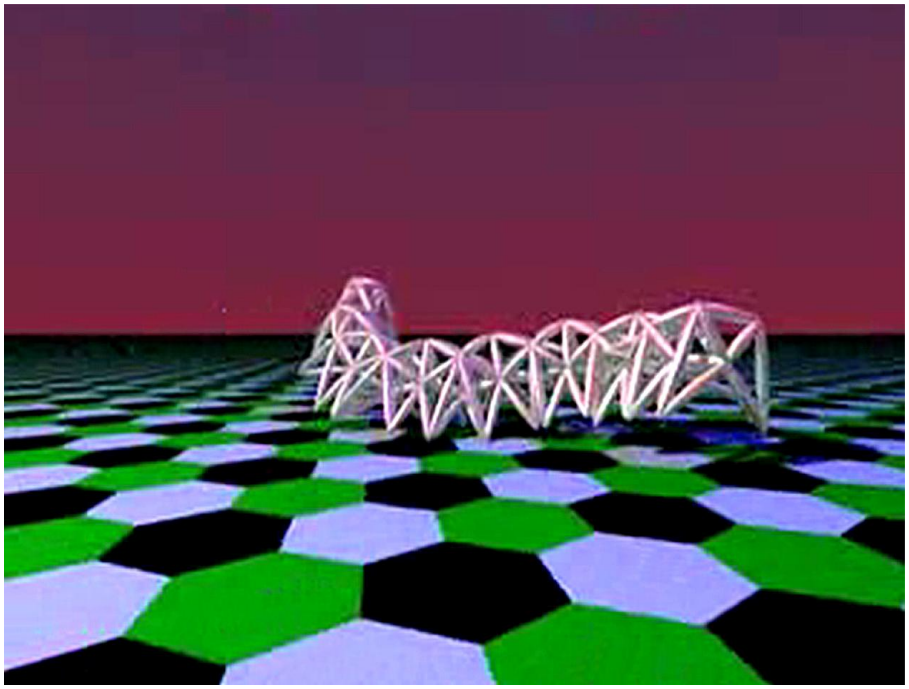

Figura 12: Criatura que se locomove se arrastando.
(fonte: captura de tela feita pelo autor)

Como foi dito, a intenção do projeto é fazer com que as criaturas “aprendam” a se deslocar num plano. Não é fornecido ao sistema nenhum método pré-estabelecido de como será este deslocamento; cada criatura deve “aprender” na medida em que seu critério de sobrevivência é o da eficiência em seu deslocamento. Assim, gerações mais novas tendem a ser mais eficientes que suas antecessoras. A eficiência do sistema, assim, emerge a partir das seleções de seus indivíduos.

Cabe-nos dizer que o experimento foi estabelecido tendo como pano de fundo várias características dos sistemas complexos: muitos agentes, grande interação entre eles, muita diversidade e não-linearidade nos processos entre agentes, de acordo com as bases estabelecidas por Holland (1998) e Hazen (2005) em seus trabalhos. Assim, o uso dos ambientes de simulação computacional foi decisivo para que se pudesse criar o número de agentes necessários para o surgimento de um ambiente propício à emergência. Os agentes desta evolução foram as sucessivas versões de criaturas.

O projeto como um todo é gerenciado tanto pelos programadores (que geraram o código do experimento), pelos usuários (que criaram as criaturas e as fizeram evoluir segundo seus parâmetros) quanto pelo próprio algoritmo (que executou a evolução de cada uma checando sua aderência aos critérios de eficiência em relação ao problema de deslocamento proposto). Não se pode, portanto, dizer que todo o crédito da inovação deve ser atribuído aos usuários, programadores, nem só aos algoritmos computacionais. Apesar disto, só os algoritmos têm a capacidade de influenciar o deslocamento da criatura tridimensional diretamente. E é justamente neste fator que o experimento parece gerar soluções mais inesperadas. Assim, acreditamos que se trata se uma espécie de co-criação entre estes três perfis de agentes: programadores, usuários e algoritmos.

Por fim, gostaríamos de ressaltar que, ao final do processo, as soluções atingidas através de métodos permeáveis a fenômenos emergentes levam a resultados, além de inesperados, coerentes com as regras e o problema que tiveram que resolver. Estes resultados são dinâmicos e tendem a se modificar na medida em que o problema proposto também se

modifica. A dinâmica própria de um sistema complexo assegura esta amarração entre problema e solução. Se, por exemplo, fosse dado um terreno acidentado às criaturas de Fluidiom, certamente a evolução deles se daria em outras direções diferentes das que foram obtidas em um terreno plano. Este tipo de solução é particularmente interessante na resolução de problemas altamente dinâmicos (metabolismo biológico, fluxos da cidade, fluxos de informação em redes etc).

A inovação das soluções, portanto, não ocorre em prejuízo do descolamento em relação ao problema proposto, mas se mantém vinculada a ele ao longo de todo o processo.

Estas soluções, advindas de fenômenos emergentes, muito provavelmente, não poderiam ser atingidas sem o uso dos métodos próprios dos sistemas que acabamos de descrever.

> No domínio da emergência, supõe-se que tanto os sistemas reais quanto os modelos operam com a seleção a partir de um espaço imenso e a variabilidade do mundo do possível, e elaborando esta seleção, propriedades novas e surpreendentes emergem. [...] Emergências ocorrem tanto em modelos quanto em situações do mundo real. Se os modelos são bem feitos, os dois tipos de emergência se espelham mutuamente. Eles ressoam um no outro. Em ambos os casos a emergência leva a novidades: o todo é, de algum modo, diferente da soma das partes. O resultado não pode ser conhecido sem rodar o programa de computador. (MOROWITZ, 2002, p. 20)[5]

Esta constatação nos leva a esboçar a hipótese de que existe uma certa qualidade de inovações que é própria do caráter imprevisível dos fenômenos emergentes. Faz-se necessário perceber que essa imprevisibilidade é própria e indissociável do projeto que incorpora fenômenos emergentes.

Acreditamos, portanto, na capacidade de inovação dos modelos dos projetos apresentados, bem como na articulação dos desejos de seus envolvidos (programadores, usuários) e na capacidade de obtenção de altos níveis de desempenho. Para cada tipo de problema, estes fatores devem ser avaliados para que se possa montar um sistema capaz de fazê-los aflorar.

Acreditamos também que muitos outros problemas podem tirar vantagem de propriedades emergentes que podem existir nos problemas abordados no campo do design, e que também podem ser inseridas artificialmente, quando se tratar de simulações computacionais. Assim, toda uma gama de problemas de design podem virtualmente se beneficiar da combinação entre inovação provenientes de processos de simulação computacionais e inovações dos processos projetuais tradicionais, gerenciados pelo designer.

[5] No original: In the domain of emergence, the assumption is made that both actual systems as well as models operate from the selection from the immense space and variability of the world of the possible, and in carrying out this selection, new and unanticipated properties emerge. (...) Emergences thus occur both in model systems and real world situations. If the models are well chosen, the two kinds of emergences map onto each other. They resonate with each other. In both cases emergence leads to novelties: the whole is somehow different from the sum of the parts. The outcome cannot be known without running the computer program.

## Referências


ALÃO, Rui S. D. **Design e emergência: concepção de projeto no design contemporâneo**. Dissertação (Mestrado em Design) - Universidade Anhembi Morumbi, São Paulo, 2008

GIACCARDI, Elisa. **Metadesign as an Emergent Design Culture**. 2005, disponível em <http://x.i-dat.org/~eg/research/pdf/Giaccardi_Leonardo05.pdf> acessado em out/2007.

_____. **Principles of metadesign. Processes and levels of co-creation in the new design space**. Tese de doutorado. 2003, disponível em <http://x.i-dat.org/~eg/research/pdf/Giaccardi_PhD04.pdf> acessado em out/2007.

GIACCARDI, Elisa e FISCHER Gerhard. *Metadesign: a framework for the future of end-user development*. In: **End user development: empowering people to flexibly employ advanced information and communication technology**. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2004. disponível em <http://x2.i-dat.org/~eg/research/pdf/FischerGiaccardi_EUD.pdf> acessado em maio/2008.

HAZEN, Robert. **Origins of life.** TTC - The Teaching Company. 2005. Manual do Curso em áudio em 24 capítulos, Disponível em <http://www.teach12.com/ttcx/coursedesclong2.aspx?cid=1515&pc=Professor18> acessado em mar/2008.

HOLLAND, John**. Hidden order. How adaptation builds complexity**. New York: Basic Books, 1995.

_____. **Emergence. From chaos to order**. New York: Basic Books, 1998.

JOHNSON, Steven. **De onde vêm as boas idéias**. Rio de Janeiro: Zahar. 2011.

De JONG, Gerald. **Introducing Fluidiom**. 2005. Disponível em <http://fluidiom.v2.nl/introduction.html>. Acessado em mai/2007.

De JONG, Gerald de. **Fluidiom: Evolution@home.** 2001. Disponível em <http://sourceforge.net/projects/fluidiom>. Acessado em out/2008.

MOROWITZ, Harold. **The emergence of everything: how the world became complex**. New York: Oxford University Press, 2002.