Relatório de projeto computacional

Geometria Computacional e Vida Artificial

Thiago A. Lechuga
RA079699
Instituto de Computação, Universidade Estadual de Campinas
Campinas, São Paulo, Brasil
thiago.lechuga@students.ic.unicamp.br

RESUMO

O objetivo desse trabalho computacional é utilizar conceitos e ferramentas vistos em sala para tratar problemas de interesse prático.

O trabalho é composto por um sucinto referencial teórico seguido por aplicações praticas de Jogo da Vida, Autômatos celulares para simulação de processos naturais (CAFUN), Starlogo, Biomorphs, Sistema de funções iterativas e concluindo com uma proposta de algoritmo computacional inspirado em um fenômeno natural. Para cada aplicação o trabalho demostra as técnicas, ferramentas e dados utilizados, seguidos pelos resultados.

1. JOGO DA VIDA

O jogo da vida é um autômato celular desenvolvido pelo matemático britânico John Horton Conway em 1970. É o exemplo mais bem conhecido de autômato celular.

O jogo foi criado de modo a reproduzir, através de regras simples, as alterações e mudanças em grupos de seres vivos, tendo aplicações em diversas áreas da ciência.

As regras definidas são aplicadas a cada nova geração; assim, a partir de uma imagem em um tabuleiro bi-dimensional definida pelo jogador, percebem-se mudanças muitas a cada nova geração, variando de padrões fixos a caóticos.

As regras utilizadas são extremamente simples:

1. Qualquer célula viva com menos de dois vizinhos vivos morre de solidão.
2. Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação.
3. Qualquer célula com exatamente três vizinhos vivos se torna uma célula viva.
4. Qualquer célula com dois vizinhos vivos continua no mesmo estado para a próxima geração.

1.1 Implementação

O programa foi implementado na linguagem C utilizando pthreads e é encontrado no arquivo "jogo.c" . São utilizadas duas matrizes (*table* representando o tabuleiro atual para o jogo, e *old-table* sendo uma referência ao tabuleiro anterior) representando o tabuleiro do jogo. O tabuleiro pode ter tamanhos arbitrários (não existe restrição de tamanho), mas essa configuraçãodeve ser feita em tempo de compilação nas constantes *ROWS* e *COLUMNS* (ambas iniciadas com tamanho 20 por padrão). O número de threads também não possui limitantes. O tabuleiro será dividido entre as threads tendo uma célula como unidade mínima da divisão. Apenas uma das threads é responsável pela impressão do tabuleiro a cada geração. Um vetor de variáveis que indicam se o tabuleiro pode ser impresso ou se a próxima geração já pode ser produzida é utilizado para manter a impressão em sincronia.

Na variável *initTable* é possível configurar, também em tempo de compilação, o estado inicial do tabuleiro (O valor padrão para inicialização randômica). É possível configurar para uma inicialização randômica ou algum dos padrões mais conhecidos do jogo:

- BLOCO.
- BLINKER.
- SAPO.
- GLIDER.
- LWSS.

Para rodar o jogo basta entrar na pasta do jogo e executar o comando *make* para compilar e em seguida rodar o executável *jogo* criado.

1.2 Resultados

Rodando as principais formas encontradas na literatura conseguimos comportamentos bem distintos e interessantes. Temos algumas formas de vida parada como o **bloco**, visto na figura 1, que se mantém constante em todas as gerações. Vemos também osciladores como o **blinker** e o **sapo**,vistos nas figuras 2, 3, 4 e 5 , que oscilam em duas configurações. Também verificamos formas que passeiam pelo tabuleiro como a *glider* e a *lwss*, vistas nas imagens 6 e 7 respectivamente.

Figura 1: Imagem de um bloco

Figura 2: Primeira etapa do blinker

Figura 3: Segunda etapa do blinker

Figura 4: Primeira etapa do sapo

Figura 5: Segunda etapa do sapo

Figura 6: Imagem Glider

Figura 7: Imagem LWSS

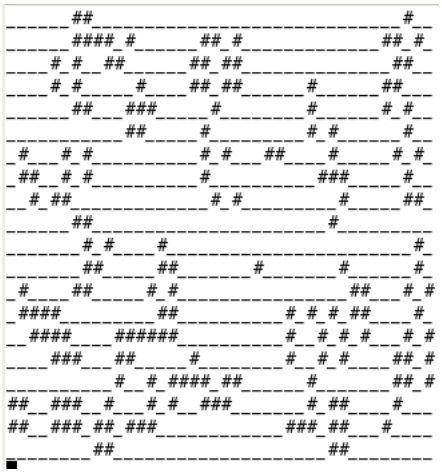

Figura 8: Execução randômca com 1 iteração

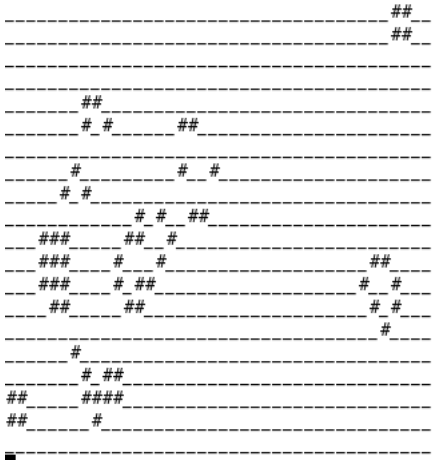

Figura 9: Execução randômca com 30 iterações

Nas imagens 8, 9 e 10 vemos uma execução randômica em 1, 30 e 60 iterações respectivamente.

2. CAFUN

O Cafun é um programa que permite a criação de simulações de processos naturais complexos de uma forma fácil utilizando autômatos celulares. Ele auxilia estudantes e pesquisadores em ensaios ou em suas teorias visualizando um complexo sistema para um melhor entendimento. Além disso, o mais interessante é que pode ser utilizado como simulador de modelos reais.

A palavra Cafun significa “cellular automata fun”, como o nome revela, Cafun pega a idéia de autômatos celulares para desempenhar as simulações. Autômatos celulares, além do que já foi visto no item 1 deste trabalho, são utilizados como importante ferramenta para investigar sistemas complexos pois utiliza composição, descentralização e paralelismo.

O Cafun adota o conceito dos autômatos celulares e introduz algumas idéias. Ele usa ao invés de regras, mutações pois

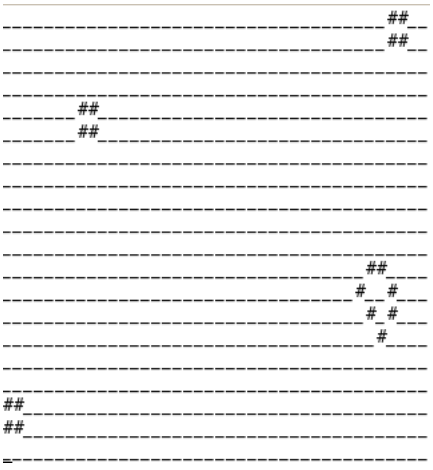

Figura 10: Execução randômca com 60 iterações (Estabilizado)

leva em conta diferentes tipos de células. A cada geração, o Cafun aplica uma mutação às células e essa mutação pode ter várias condições atribuídas. Por exemplo, a uma célula pode ser aplicada um condição quanto ao número de vizinhos, quanto à forma das células da vizinhança, à posição da vizinhança, e estar ainda sujeita a uma certa probabilidade, além de levar em conta a condição atual de todas as outras células.

O Cafun utiliza linguagem XML, um formato padrão com a vantagem de ser lido facilmente e transformada em outros formatos.

2.1 Implementação

Nesse trabalho foram testados alguns exemplos implementados no Cafun.

Para rodar o programa basta executar o comando "*java -jar cafun.jar*" e em seguida abrir a simulação desejada.

2.2 Resultados

Na figura 11, o programa Cell, é um exemplo de crescimento descentralizado de uma célula. É possível ver o citoplasma e as partículas se desenvolvendo e difundindo através dele. Não se trata de uma simulação do crescimento de células biológicas, mas mostra como são descentralizados e auto-organizados os processos de reação química.

Na figura 12, a demonstração do programa Whirl. Ele simula o movimento de quanto jogamos uma pedra em um rio, e as ondas vão se formando do lugar onde caiu a pedra e se espalhando por todo o rio.

Na figura 13 o programa simula o jogo da vida de Conway, como o implementado na parte 1 desse projeto.

Na figura 14 o programa cria padrões similares a cristais de neve.

O programa Bush Fire é a simulação de um incêndio em

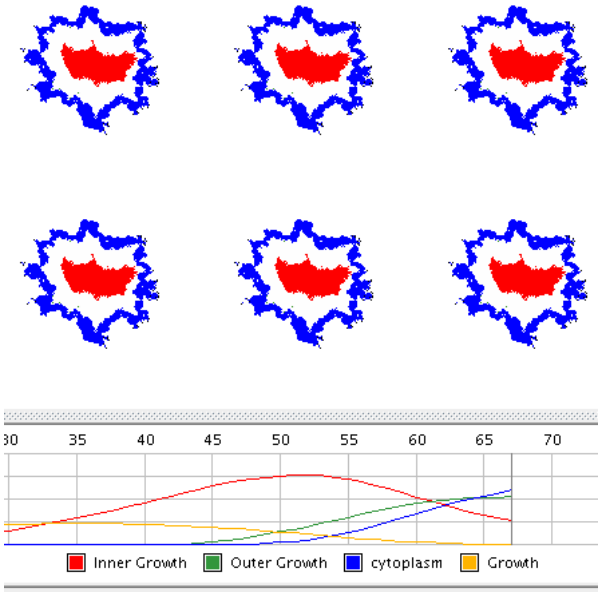

Figura 11: Execução do programa Cell no Cafun

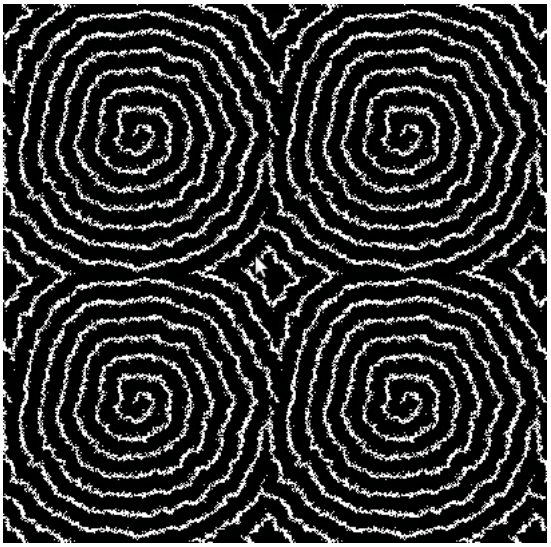

Figura 12: Execução do programa Whirl no Cafun

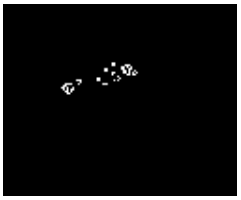

Figura 13: Execução do jogo da vida de Conway no Cafun

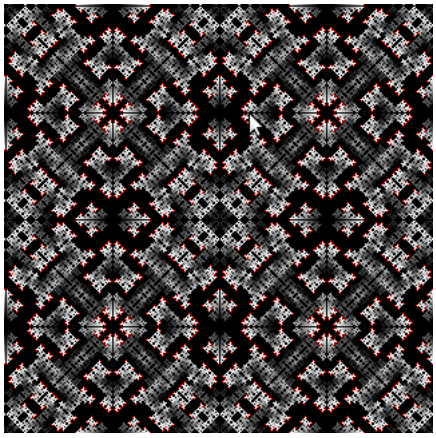

Figura 14: Cristais de neve no Cafun

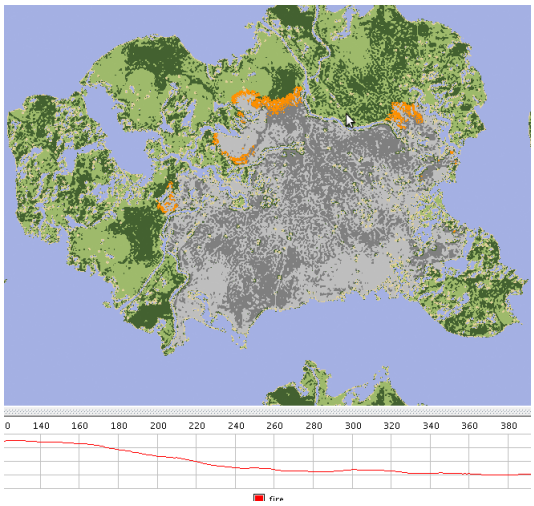

Figura 15: Resultado da queimada

uma floresta. Ele mostra a floresta queimando a partir dos pontos de início do incêndio, escolhido pelo usuário. Podemos verificar o efeito das queimadas, onde celulas com fogo tendem a ”contagiar” as células vizinhas. Na imagem 15 vemos o efeito do fogo na floresta. E na figura 16 vemos o fogo sendo contido por uma barreira de areia.

Analisando esses experimentos, podemos concluir que o Cafun é uma excelente alternativa na simulação de processos naturais, podendo ser utilizados para pesquisas em diversas áreas, além de ser um divertido entretenimento.

3. STARLOGO

StarLogo é uma linguagem de simulação baseada em agentes desenvolvida por Mitchel Resnick, Eric Klopfer no MIT Media Lab. É uma extensão da linguagem de programação Logo, um dialeto do Lisp. Projetado para a educação, StarLogo podem ser utilizados pelos alunos para modelar o comportamento de sistemas descentralizados.

Tal como a tradicional versão do Logo, permite criar desen-

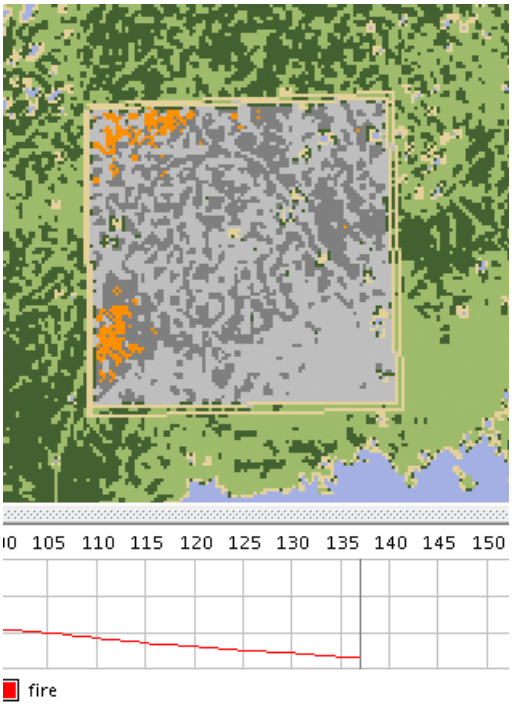

Figura 16: Queimada sendo contida por barreira de areia

hos e animações através de comandos que são dados a uma ”tartaruga” gráfica. O StarLogo permite ao utilizador controlar milhares de tartarugas em simultâneo e permite tornar o ”mundo da tartaruga” dinâmico. Tartarugas e ”patches” podem interagir - por exemplo, podemos programar as tartarugas para vaguearem no seu mundo e comportarem-se de diferentes formas perante os diferentes ”patches”. StarLogo é particularmente bem sucedido na criação de projetos de simulação.

Um dos aspectos mais famosos da linguagem Logo é o de permitir construir figuras geométricas por meio de um conjunto elementar de regras que definem a chamada “Geometria da tartaruga”. Os comandos do StarLogo dividem-se, basicamente, em comandos primitivos, que já vêm implementados na linguagem, e em nomes ou rótulos de procedimentos, escritos pelo utilizador, que, uma vez memorizados pelo computador, são executados como se fossem comandos primitivos. O atual StarLogo é escrito em Java e funciona na maioria dos computadores.

3.1 Implementação

Foram testados alguns exemplos prontos do Starlogo. Mostrando as movimentações básicas das tartarugas e o uso de interfaces com botões que permitem a um utilizador que não saiba nenhuma programação de Logo comandar a tartaruga.

3.2 Resultados

Na figura 17 vemos uma implementação do jogo da vida no Starlogo.

Exemplos muito interessantes de projetos estão implemen-

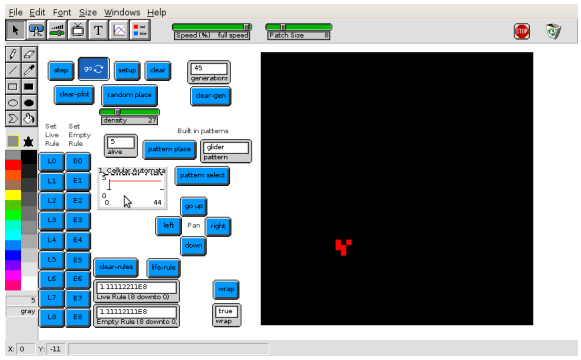

Figura 17: Execução do jogo da vida no Starlogo

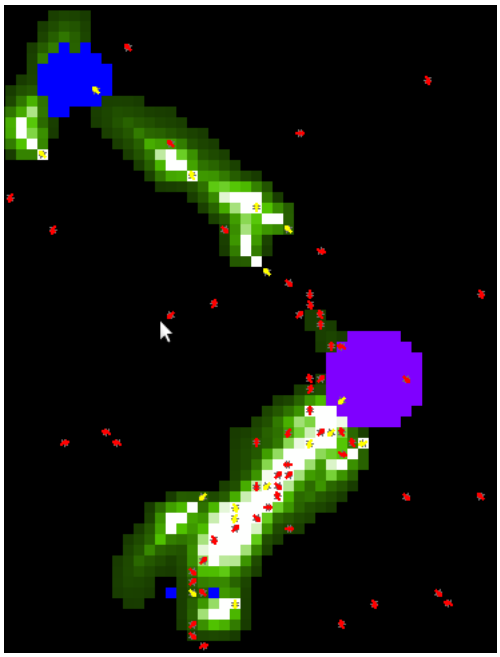

Figura 18: Aco no StarLogo

tados no Starlogo, entre eles uma demonstração da trilha de feromônio das formigas ao encontrem o alimento. Na 18 vemos um exemplo, onde o desenho central (roxo) é o ninho e os outros (azuis) são as fontes de alimentos. Os pontos em vermelho representam as formigas que saíram do ninho em busca alimento e percorrem aleatoriamente o espaço de busca. Ao encontrarem o alimento elas retornam ao ninho deixando o rastro de feromônio (em verde e branco). O programa permite ainda o controle da taxa de feromônio e a sua evaporação.

Quando alteradas as taxas de feromônio e evaporação, obtemos resultados bem variados. Quando a taxa de feromônio é bem maior que a taxa de evaporação, o resultado é obtido muito rápido pois a trilha fica bem grande, acabando com a diversidade do algoritmo. No caso da taxa de evaporação ser bem grande, a trilha de feromônio não é bem formada, aumentando de mais a diversidade e tornando o algoritmo

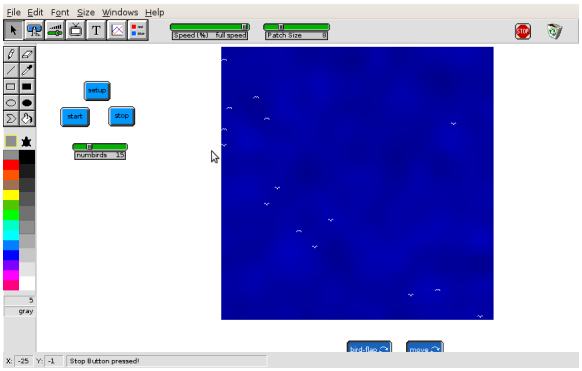

Figura 19: Vôo de pássaros no Starlogo

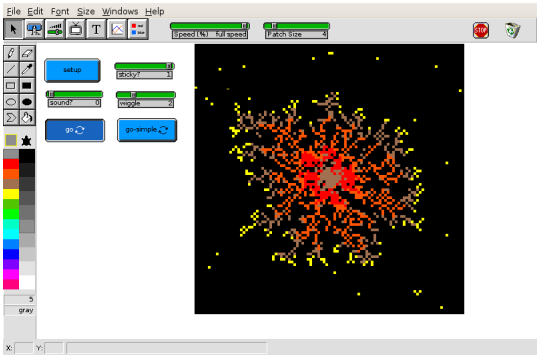

Figura 20: Diffusion-limited aggregation no Starlogo

quase que aleatório.

Ainda existem diversos outros programas como o vôo de pássaros, visto na figura 19 e simulação de explosão de partículas e simulação de crescimento de corais com movimento browniano (Diffusion-limited aggregation) (figura 20).

Também foram testados jogos implementados com o Starlogo. Nas figuras 21 e 22 vemos uma implementação de Pacman e uma de SpaceInvaders respectivamente.

Podemos ver que apesar de uma linguagem simples e com diversas limitações, o Starlogo é capaz de construir programas bem interessantes que podem ser utilizados para simulações e experimentos com eficiência.

4. BIOMORPHS

Com um programa chamado Blind Watchmaker, R. Dawkins desenvolveu um algoritmo evolutivo para gerar figuras compostas por pontos e linhas. Os organismos artificiais criados com o Blind Watchmaker foram denominados de biomorphs (bioformas).

O programa consiste basicamente de mutação aleatória seguida por seleção. A forma de cada biomorph é definida durante o desenvolvimento embrionário. A evolução ocorre porque existem pequenas diferenças, resultadas de variações genéticas, que ocorrem durante a reprodução. Assim, Blind Watch-

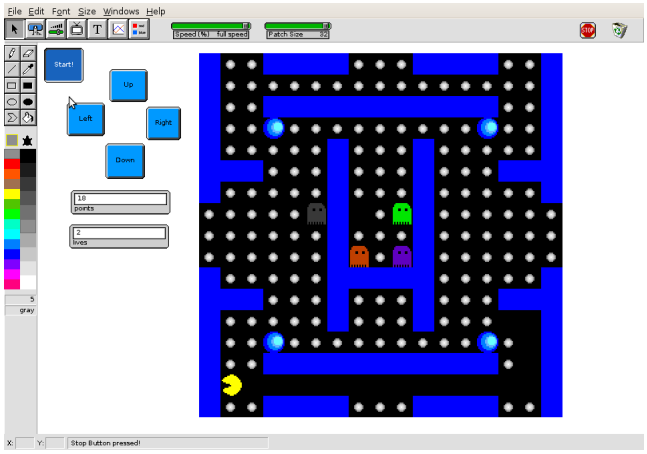

Figura 21: Jogo Pacman no Starlogo

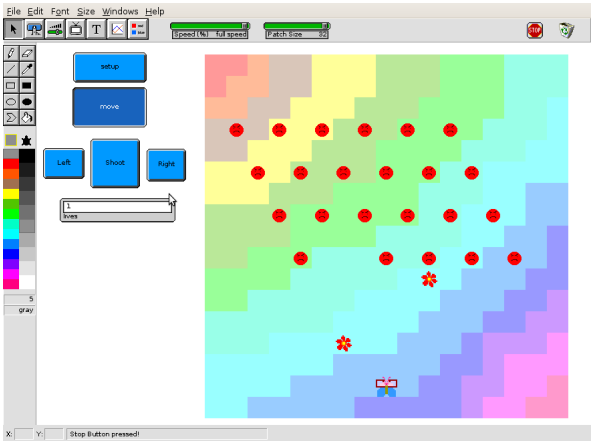

Figura 22: Jogo SpaceInvaders no Starlogo

waker também tem algo equivalente a genes que podem sofrer mutação e o desenvolvimento embrionário que realiza a transformação do genótipo para o fenótipo.

A reprodução é responsável por criar novos biomorphs, descendentes a partir dos biomorphs já existentes (pais), por meio da aplicação de um operador de mutação que alterou um gene do indivíduo.

A seleção é realizada pelo usuário que escolhe um biomorph para ser o pai na próxima geração.

Usando uma estrutura de dados semelhante a um cromossomo, como nos algoritmos genéticos, inicialmente cria-se os genes. Cada gene é representado por um número que exerce uma influência no desenho. Quanto mais genes tem o desenho mais complexo ele pode ficar.

4.1 Implementação

Neste trabalho foi implementado o Blind Watchwaker. A cada geração serão criados 6 biomorphs. Cada biomorph é formado por 8 genes, representando:

- **Genes 1, 2 e 3:** tamanho do primeiro, segundo e terceiro ramos, respectivamente.
- **Gene 4:** taxa de aumento ou diminuição do ramo.
- **Gene 5:** ângulo inicial.
- **Gene 6:** variação do primeiro ângulo.
- **Gene 7:** variação dos próximos ângulos.
- **Gene 8:** profundidade.

Uma configuração inicial de pontos e segmentos de retas é criada. O usuário informa seu biomorph preferido é escolhido como o novo cromossomo pai, e realiza a mutação nele. Para mutação foi somado 1 ao menor gene e subtraído 1 ao maior gene de cada cromossomo, criando assim uma nova geração de 6 biomorphs. Utilizando a recursividade de funções, os desenhos são criados através de pontos e segmentos de retas. A cada ponta, dois novos segmentos de retas partem de cada ponta, obedecendo os critérios estipulados nos genes. Novamente o usuário escolhe seu biomorph preferido, e assim até que este decida terminar.

Outros genes poderiam ainda ser utilizados como: cor do biomorph, número de segmentações, forma, etc.

4.2 Resultados

Na figura 23 vemos a configuração inicial. Na figura 24, após ter escolhido o biomorph 6, a nova geração. E realizando o mesmo procedimento, os biomorphs vão ganhando formas diferentes. Na figura 25 vemos décima geração.

Com um pouco de imaginação e criatividade, podemos ver vários formatos conhecidos nos biomorphs como os apresentados abaixo. Nas figuras 26, 27 e 28 vemos exemplos de biomorphs criados neste trabalho.

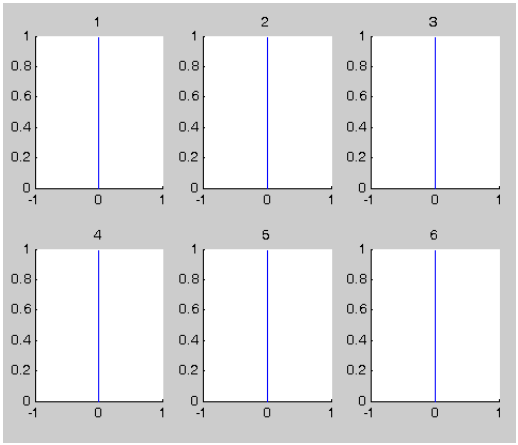

Figura 23: Configuração inicial

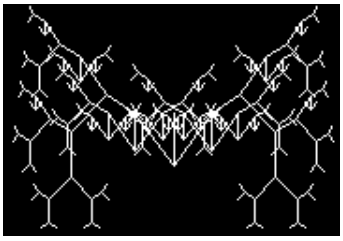

Figura 26: Biomorph Aranha

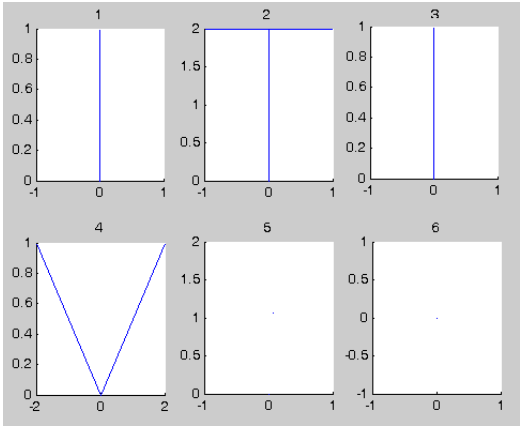

Figura 24: Segunda geração

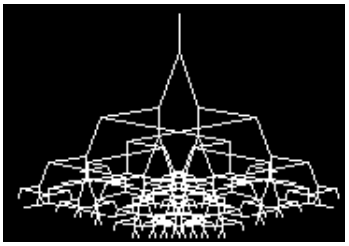

Figura 27: Biomorph Torre

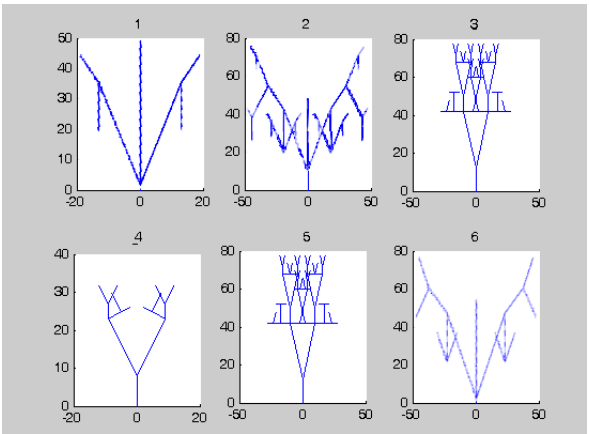

Figura 25: Décima geração

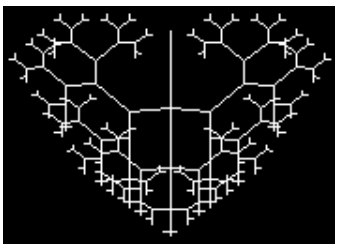

Figura 28: Biomorph Coração

5. SISTEMA DE FUNÇÕES ITERATIVAS

Sistemas de funções iterativas, também conhecidos pela sigla IFS (do inglês Iterated Function Systems) é uma técnica de se construir figuras fractais através da repetição em escala de uma mesma figura. Após um determinado número de iterações, o conjunto final, ou limite, irá definir uma certa configuração geométrica. Apesar de poder ser aplicada em qualquer número de dimensões, a técnica de sistemas de funções iterativas é mais comum em figuras bidimensionais, por questões práticas.

A técnica consiste em selecionar uma figura inicial qualquer e aplicar iterativamente a ela uma série de transformações afins (translação, escalonamento, reflexão e rotação),de onde o nome "sistemas de funções iterativas", em geral com redução de escala, que geram "cópias" menores da mesma imagem. Este procedimento é repetido infinitamente até se obter uma imagem composta de infinitas cópias cada vez menores da mesma imagem.

Os sistemas de funções iterativas podem ser determinístico ou aleatório. Os sistemas deterministicos são baseados na idéia de calcular diretamente uma sequência de conjuntos a partir de um conjunto inicial. Essa sequência convergirá para sempre para um mesmo ponto (chamado de atrator). Em sistemas aleatórios, uma probabilidade é atribuída a cada mapeamento, e um x inicial é escolhido aleatoriamente.

Figura 29: Folha com angulação 0

5.1 Implementação

Para essa etapa do projeto foi usada uma implementação de um arbusto de Barnsley usando funções iterativas e foram modificados os parâmetros: Size, angle e Shift, mas mantendo os valores das funções de geração. Em seguida foi utilizada uma implementação do triângulo de Sierpinski e uma árvore simples onde os parâmetros das funções foram alterados. Todos os sistemas de funções iterativas utilizados são não-determinísticos.

Os parâmetros para a construção do arbusto de Barnsley estão na imagem XX.

5.2 Resultados

Nas figuras 29 e 30 vemos um arbusto de Barnsley com angulação 0 e 4 respectivamente. Na figura 31 vemos a mesma folha com size, angle e shift modificados.

Para o programa triângulo de Sierpinski (figuras 36, 38) e a árvore (figuras 32, 33) foram realizados testes modificando parâmetros da função de geração.

Alterações nos parâmetros resultam em modificações na figura. Como a figura é formada por equações afins, pode ser aplicadas transformações afins na figura. A figura 37 mostra a rotação em 180o.

Podemos ver como uma sutil modificação de um parâmetro modifica o padrão da geração e pode mudar completamente a figura gerada. Nas figuras 39, 42, 39, 42 vemos modificações aleatórias que geram figuras interessantes.

Na figura 43 vemos o triângulo de Sierpinski apenas com as probabilidades modificadas. Podemos veriricar que um lado do triangulo é mais preenchido que o outro.

Figura 30: Folha com angulação 4

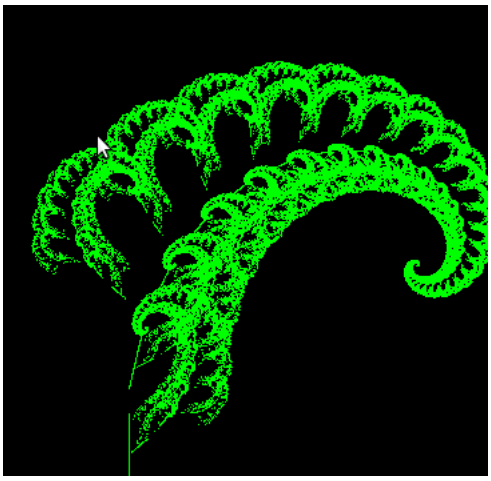

Figura 31: Folha com size, angle e shift modificados

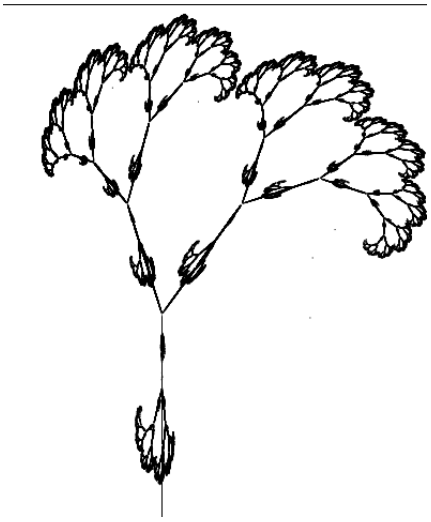

Figura 34: Arvore modificada

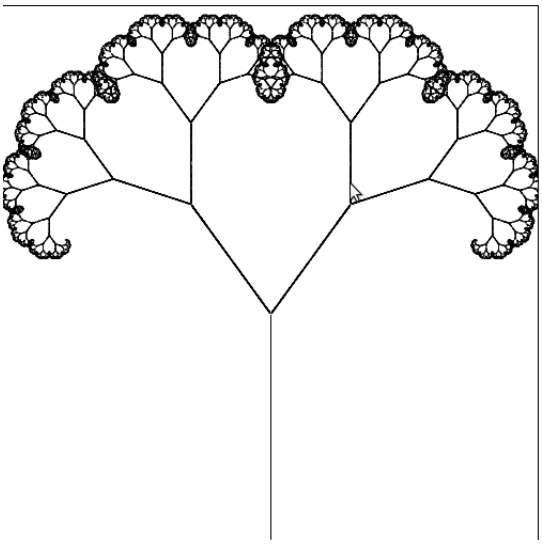

Figura 32: Arvore original

(x', y') = [(ax + by + e), (cx + dy + f)]

	a	b	c	d	e	f	prob.
função 0:	0.000(	0	0	0.424:	0	0	0.05
função 1:	-0.000	0	0	-0.424	0	0.424:	0.05
função 2:	0.485<	-0.352	0.352(	0.485<	0	0.424:	0.45

Figura 33: Função original da árvore

(x', y') = [(ax + by + e), (cx + dy + f)]

	a	b	c	d	e	f	prob.
função 0:	0.000(	0	0	0.424:	0	0	0.15
função 1:	-0.100	0	0	-0.324	0	0.424:	0.05
função 2:	0.485<	-0.152	0.352(	0.485<	0	0.424:	0.45

Figura 35: Função modificada da árvore

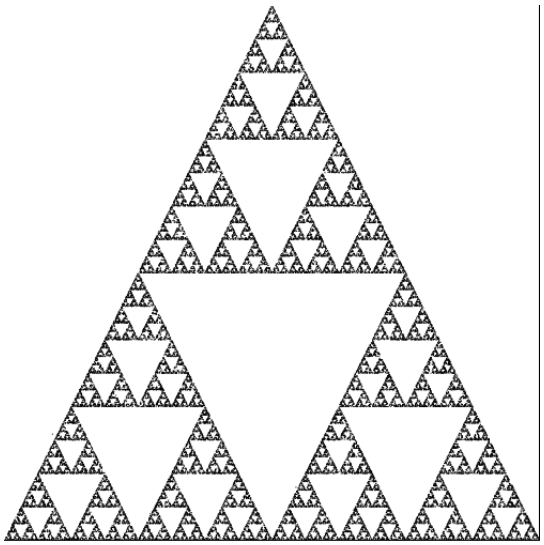

Figura 36: Triângulo original

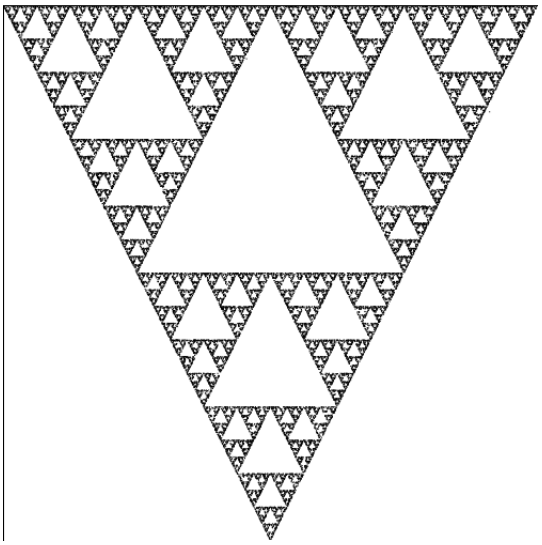

Figura 37: Triângulo rotação de 180o

(x', y') = [(ax + by + e), (cx + dy + f)]

	a	b	c	d	e	f	prob.
função 0:	0.5	0	0	0.5	0	0	0.333
função 1:	0.5	0	0	0.5	1	0	0.333
função 2:	0.5	0	0	0.5	0.5	1	0.333

Figura 38: Função original do triângulo

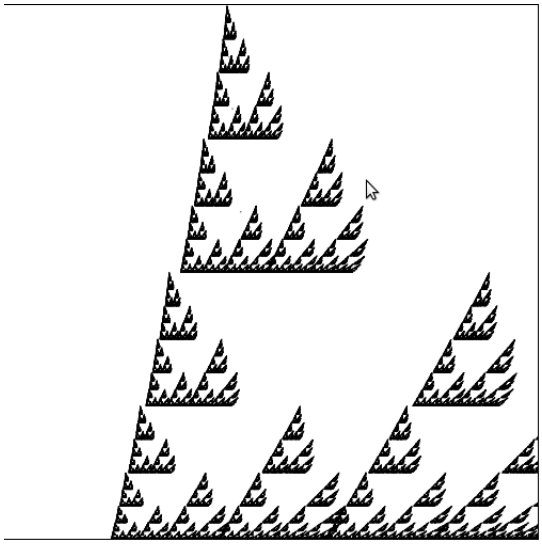

Figura 39: Triângulo modificado

(x', y') = [(ax + by + e), (cx + dy + f)]

	a	b	c	d	e	f	prob.
função 0:	0.5	0	0	0.5	0.2	0	0.333
função 1:	0.5	0.2	0	0.5	1	0	0.333
função 2:	0.4	0	0	0.5	0.5	1	0.333

Figura 40: Função modificada do triângulo

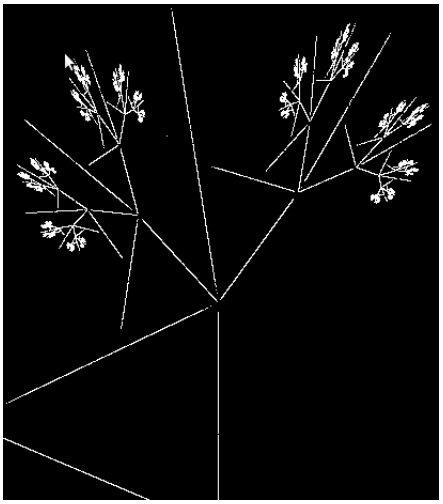

Figura 41: Árvore modificada 2

6. COMPUTAÇÃO INSPIRADA NA NATUREZA

A água da chuva pode ter vários destinos após atingir a superfície da Terra. Inicialmente uma parte se infiltra. Quando o solo atinge seu ponto de saturação, ficando encharcado, a água passa a escorrer sobre a superfície em direção aos vales. Dependendo da temperatura ambiente, uma parte da chuva volta à atmosfera na forma de vapor. Em países frios, ou em grandes altitudes, a água se acumula na superfície na forma de neve ou gelo, ali podendo ficar por muito tempo.A parcela da água que se infiltra vai dar origem à água subterrânea.

A taxa de infiltração de água no solo depende de muitos fatores:

1. **Sua porosidade:** A presença de argila no solo diminui sua porosidade, não permitindo uma grande infiltração.
2. **Cobertura vegetal:** Um solo coberto por vegetação é mais permeável do que um solo desmatado.
3. **Inclinação do terreno:** em declividades acentuadas a água corre mais rápidamente, diminuindo o tempo de infiltração.
4. **Tipo de chuva:** Chuvas intensas saturam rapidamente o solo, ao passo que chuvas finas e demoradas têm mais tempo para se infiltrarem.

A água que se infiltra está submetida a duas forças fundamentais: a gravidade e a força de adesão de suas moléculas

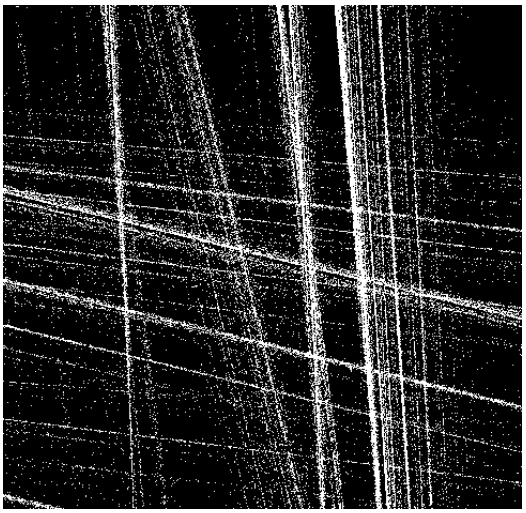

Figura 42: Triângulo modificado 2

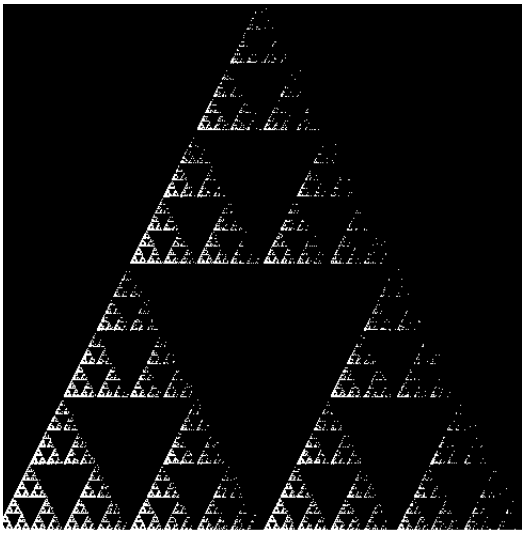

Figura 43: Triângulo com probabilidades modificadas

às superfícies das partículas do solo (força de capilaridade). Pequenas quantidades de água no solo tendem a se distribuir uniformemente pela superfície das partículas. A força de adesão é mais forte do que a força da gravidade que age sobre esta água. Como conseqüência ela ficará retida, quase imóvel, não atingindo zonas mais profundas. Chuvas finas e passageiras fornecem somente água suficiente para repor esta umidade do solo. Para que haja infiltração até a zona saturada é necessário primeiro satisfazer esta necessidade da força capilar.

A água da chuva passa por todas essas camadas antes de atingir o lençol freático e esse processo de filtragem de várias camadas, onde cada uma das camadas é capaz de filtrar niveis deferentes de impurezas, que garante a pureza da água em níveis mais baixos. Alguns níveis onde existe água subterrânea:

1. Zona de aeração.
2. Zona de umidade do solo.
3. Franja de capilaridade.
4. Zona intermediária.
5. Zona de Saturação.

Atualmente o número de spams nos e-mails é absurdo e existem investimentos muito elevados em processamento e pesquisa para resolver esse tipo de problema.

A idéia aqui seria aplicar diversas camadas de filtros (como na natureza) para diminuir a quantidade de spams recebidos. Dessa forma o filtro de spams deixa de ser um gargalo, pois mensagens mais grosseiras são filtradas primeiro e mensagens mais difíceis de se detectar são deixadas para o final. Esse processo também poderia ser feito de forma distribuída, usando o conceito de distribuição uniforme da quantidade de água no solo.

7. CONCLUSÕES

Neste trabalho pudemos ver exemplos de programas que auxiliam visualização e simulação de processos naturais, abordando a versatilidade de técnicas para e a variedade de programas. Vimos também que a visualização do processo de solução e a implementação dos algoritmos levam a uma importante contribuição para a compreensão da teoria estudada.