Instituto Nacional de Pesquisas Espaciais

Framework para simulação de agentes simples em *Java*, para estudo de comportamento de múltiplos agentes cooperativos/competitivos.

Bolsista: Francys Mainieri
IBTA-Redes de computadores/ 4ºsemestre
Orientador: Rafael D. C. dos Santos

Relatório final referente ao programa institucional de bolsas de iniciação científica – PIBIC/INPE/CNPq

São José dos Campos. Julho de 2006

Framework para simulação de agentes simples em *Java*, para estudo de comportamento de múltiplos agentes cooperativos/competitivos.

Sumário

Resumo

Este trabalho tem como objetivo o desenvolvimento de um programa simulador de agentes, que será utilizado de forma genérica, tendo como base à teoria de John Conway, denominada "The game of Life" (Jogo da vida). Sua teoria simula a vida em comunidade de determinados organismos, prevendo através de regras que cobrem desde o nascimento até a morte de cada integrante do conjunto. Para tanto, a pesquisa foi baseada em tópicos, tendo em vista o nível de necessidade de cada um, de modo a modelar a inteiração dos dados inseridos no código-fonte, com os obtidos com a execução do mesmo, reutilizando os dados atualizados no simulador para a execução da próxima geração. Para a solução dos tópicos, houve a necessidade de interação dos componentes existentes, sendo então escolhida a linguagem Java, visto sua portabilidade em qualquer sistema operacional atual. Através da comparação realizada entre os resultados obtidos e a idéia inicial do comportamento da curva referente à taxa de nascimento/morte, verificou-se que não há nenhuma discrepância entre ambas, estando esta perfeitamente normal. Os testes anteriormente citados foram obtidos através realização de 100 testes, para cada período, podendo ser estes na escala de 1%, 2 %, ou 10%, visto a necessidade de melhor verificação da área a ser observada. Para que nenhuma ocorrência, de baixa freqüência interferisse, a tabela montada com os resultados de cada faixa, separadamente, foi ordenada em ordem crescente, eliminando-se assim os 10 maiores e os 10 menores resultados, utilizando os 80 resultados restantes para se obter a média de cada faixa. A partir dos resultados obtidos ao se utilizar à faixa de 10%, um gráfico foi montado, demonstrando assim a curva realizada pela taxa de nascimento/morte.

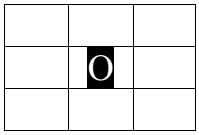
Introdução

O objetivo principal deste trabalho é o desenvolvimento de um programa simulador de agentes, que será utilizado de forma genérica, tendo como base à teoria de John Conway, denominada "The game of Life" (Jogo da vida).

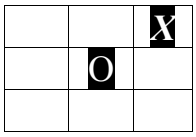
Sua teoria simula a vida em comunidade de determinados organismos, prevendo através de regras que cobrem desde o nascimento até a morte de cada integrante do conjunto.

Estas regras se baseiam nos seguintes conceitos:

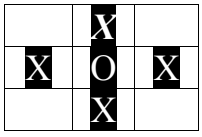
- Se uma célula ocupada tiver 0, 1, 4, 5, 6, 7 ou 8 vizinhos, o organismo morrerá;



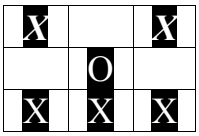
- Exemplo de célula com 0 vizinhos;



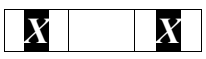
- Exemplo de célula com 1 vizinho;

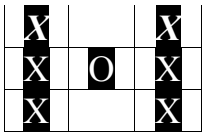


- Exemplo de célula com 4 vizinhos;



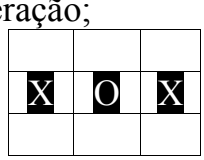
- Exemplo de célula com 5 vizinhos;



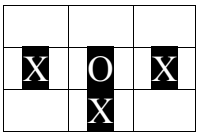


- Exemplo de célula com 6 vizinhos.

- Se uma célula tiver 2 ou 3 vizinhos, o organismo sobreviverá até a próxima geração;

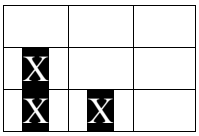


- Exemplo de célula com 2 vizinhos;

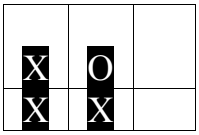


- Exemplo de célula com 3 vizinhos.

- Se uma célula desocupada tiver 3 vizinhos, esta também se tornará ocupada.



- Exemplo de célula desocupada, com 3 vizinhos;



- Demonstrativo da geração seguinte referente ao último exemplo.

Para tanto, a pesquisa foi baseada em tópicos, tendo em vista o nível de necessidade de cada um, de modo a modelar a inteiração dos dados inseridos no código-fonte, com os obtidos com a execução do mesmo, reutilizando os dados atualizados no simulador para a execução da próxima geração.

Para a solução dos tópicos, houve a necessidade de interação dos componentes existentes, sendo então escolhida a linguagem Java, visto sua portabilidade em qualquer sistema operacional atual.

Tais tópicos detêm as seguintes características:

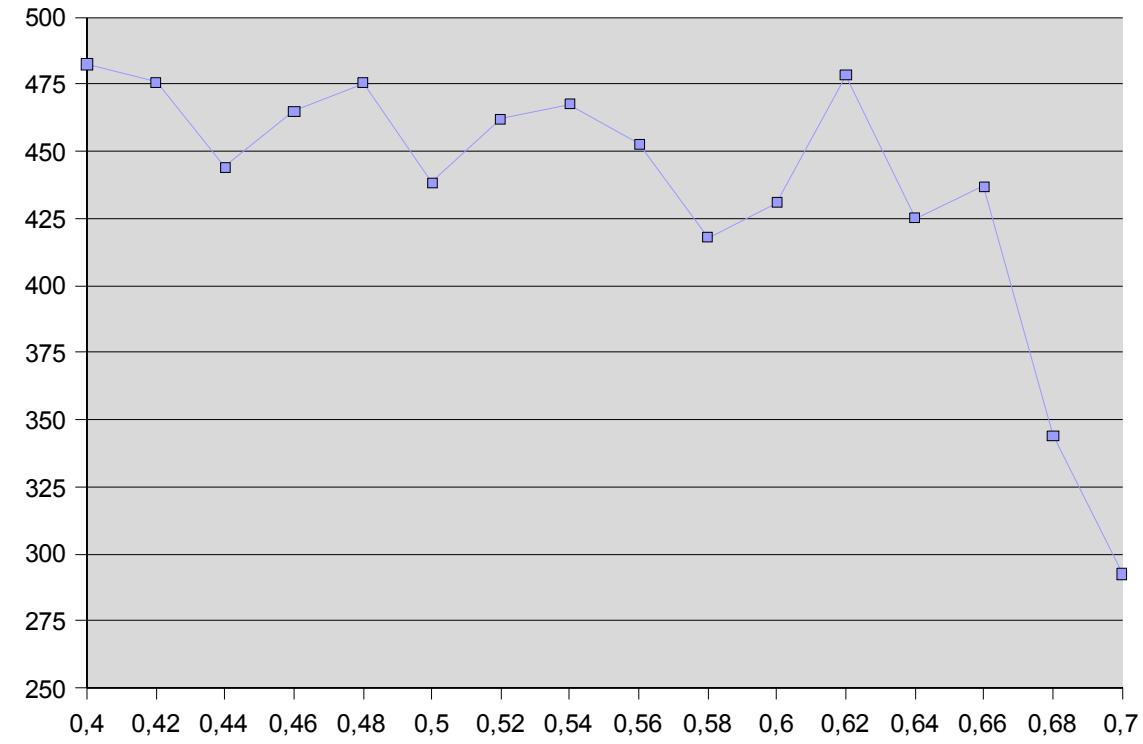
- Integração de classes, para implementação de matrizes, com classes de visualização;
- Utilização de classes que implementam outras, através utilização de herança;
- Utilização de modelos pré-estabelecidos para testes;
- Modificação, integração e desenvolvimento de novas idéias para implementação de testes.

Como continuidade do trabalho, foram desenvolvidos os seguintes tópicos:

- Interligação entre classes de exibição gráfica e de execução;
- Desenvolvimento de novos testes, para possibilitar maior abrangência de análises, de modo a possibilitar um conhecimento mais detalhado sobre comportamento do sistema estudado;
- Obtenção de resultados referentes a comunidades, em um determinado meio de cultura;
- Finalização do *framework;*
- Apresentação do *framework* e finalização do projeto.

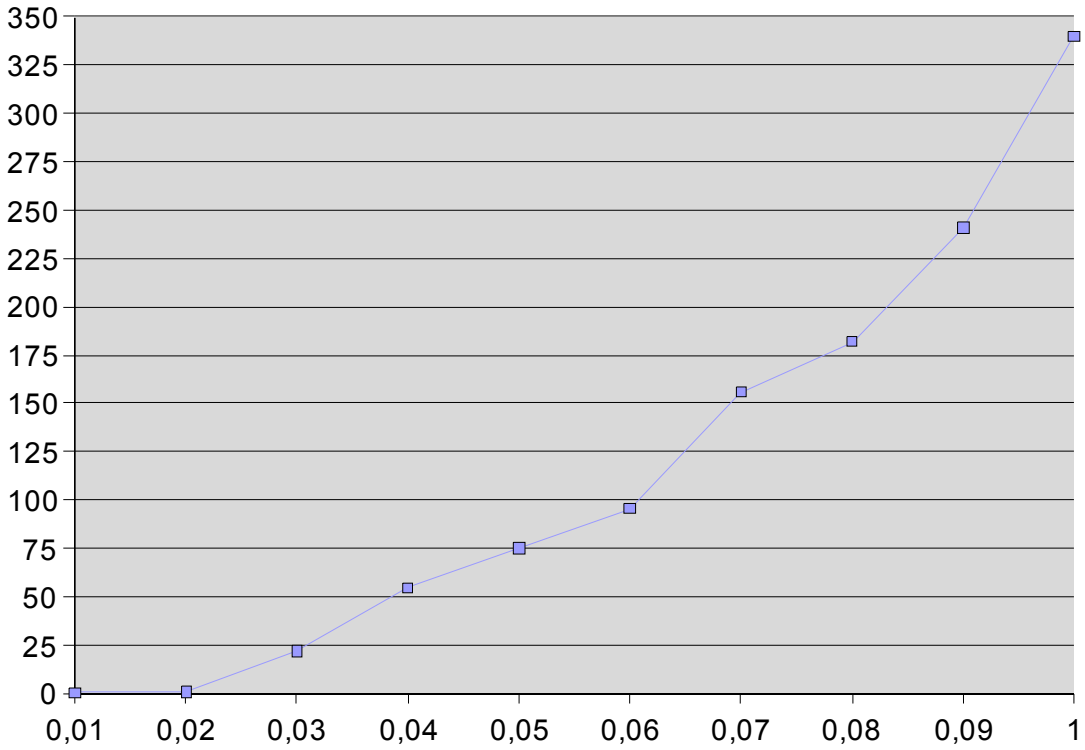
Analise dos resultados

Através dos dados obtidos nas duas fases do projeto, com a realização dos testes, conclui-se que nenhum parâmetro ao qual possa ser considerado fora do previsto foi encontrado, tendo-se como base à média de cada um dos testes realizados e com a utilização de gráficos para melhor visualização.
Para contraprova dos testes realizados, períodos foram selecionados pra estudo aprofundado, no caso a seguir utilizou-se à escala de 2% na área referente à taxa de nascimento/morte de 40 a 70%. Na qual se obteve o seguinte gráfico:



A área entre 1 a 10 % também foi verificada, havendo então a necessidade de novos testes em menor escala, sendo esta de 1%.

Após a verificação detalhada, e utilização das médias obtidas, um novo gráfico foi montado, para demonstração da curva referente ao período, embora nada discrepante fora encontrado, o gráfico em questão será a seguir demonstrado.



Código Fonte

Foi divido em tres partes sendo estas Matriz, DisplayMatriz e ComponenteMatriz :

```
public class Matriz
  {
  private boolean[][] matriz;
  private int colunas,linhas;
  private int numeroDeCelulas;
//O construtor da classe, que inicializa a matriz com os tamanhos passados como
argumentos.
  public Matriz(int l,int c,double p)
    {
    colunas = c; linhas = l;
    matriz = new boolean[linhas][colunas];
    inicializa(p);
    }

  /*
  * O método "inicializa", simplesmente inicializa a matriz com valores
  * aleatórios. Neste método deveria residir o algoritmo principal do
  * Jogo da Vida.
  */
  private void inicializa(double p)
  {
          for(int l=0;l<linhas;l++)
    for(int c=0;c<colunas;c++)
    {
          if (Math.random() <= p) matriz[l][c] = true;
          else matriz[l][c] = false;

    }
  }

  public int getNumeroDeCelulas()
  {
            return numeroDeCelulas;
  }

  public void inicializa_FlipFlop()
  {
            matriz[1][9] = true;
            matriz[2][9] = true;
            matriz[3][9] = true;
  }
```

```
public void inicializa_Glider()
{
	matriz[2][1] = true;
	matriz[3][2] = true;
	matriz[3][3] = true;
	matriz[2][3] = true;
	matriz[1][3] = true;
}

public void inicializa_Quadrado()
{
	matriz[8][1] = true;
	matriz[9][1] = true;
	matriz[9][2] = true;
	matriz[8][2] = true;
}

public void itera()
{
	numeroDeCelulas = 0;
	boolean[][] nova = new boolean[linhas][colunas];
	for(int l=0;l<(linhas);l++)
		for (int c=0;c<(colunas);c++)
		  {
			int vizinhos = 0;
			int Cmais=c+1;
			int Lmais=l+1;
			int Cmenos=c-1;
			int Lmenos=l-1;

			if(l==0)
				Lmenos = linhas-1;
			if(c==0)
				Cmenos = colunas-1;
			if(l==linhas-1)
				Lmais=0;
			if(c==colunas-1)
				Cmais=0;


			if (matriz [Lmenos][Cmenos] == true)
				vizinhos++;
			if (matriz [Lmenos][c] == true)
				vizinhos++;
			if (matriz [Lmenos][Cmais] == true)//1 linha
				vizinhos++;
			if (matriz [l][Cmenos] == true)
```

```
                        vizinhos++;
                    if (matriz [l][Cmais] == true)//2 linha
                        vizinhos++;
                    if (matriz [Lmais][Cmenos] == true)
                        vizinhos++;
                    if (matriz [Lmais][c] == true)
                        vizinhos++;
                    if (matriz [Lmais][Cmais] == true)//3 linha
                        vizinhos++;

                    if (matriz[l][c] == true) // se a célula estiver ocupada
                      {
                      if (vizinhos < 2 || vizinhos > 3) nova[l][c] = false;
                      else nova[l][c] = true;
                      }
                    else if (vizinhos == 3) nova[l][c] = true;

                    if (nova[l][c] == true) numeroDeCelulas++;
                }
  matriz = nova;
}

// Retorna o número de linhas da matriz.
public int getLinhas()
  {
  return linhas;
  }

// Retorna o número de colunas da matriz.
public int getColunas()
  {
  return colunas;
  }

// Retorna o elemento da matriz que está na linha l e coluna c.
public boolean getElemento(int l,int c)
  {
  return matriz[l][c];
  }

public String toString()
        {
         String Resultado = "";
         for (int l=0;l<linhas;l++)
                {
                        for (int c=0;c<colunas;c++)
                        {
```

```
                        if (matriz[l][c] == true) Resultado += "*";
                        else Resultado += ".";
                    }
                    Resultado +="\n";
                    }
        return Resultado;
        }
}
```

A segunda classe se refere a junção entre as classes Matriz e DisplayMatriz

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JComponent;
/*
 * Esta classe em um componente que encapsula uma Matriz, para fazer a exibição
 * gráfica da matriz de forma agradável.
 */
public class ComponenteMatriz extends JComponent
  {
  // Internamente guardaremos uma cópia da Matriz.
  private Matriz celula;
   // Qual será o tamanho da "celula" a ser desenhada?
  private final int larguraCelula = 10;
  private final int alturaCelula = 10;

  // O construtor inicializa a cópia da matriz.
  public ComponenteMatriz(Matriz c)
    {
    celula = c;
    }

  // Este método retorna o tamanho máximo do componente. Usaremos o preferido.
  public Dimension getMaximumSize()
    {
    return getPreferredSize();
    }

  // Este método retorna o tamanho mínimo do componente. Usaremos o preferido.
  public Dimension getMinimumSize()
    {
    return getPreferredSize();
    }

  // Este método retorna o tamanho preferido do componente.
```

```
  public Dimension getPreferredSize()
    {
    return new Dimension(celula.getColunas()*larguraCelula,
                                          celula.getLinhas()*alturaCelula);
    }

  // Este método pinta o componente.
  protected void paintComponent(Graphics g)
    {
    // Pintamos a matriz.
    int linhas = celula.getLinhas();
    int colunas = celula.getColunas();
    for(int l=0;l<linhas;l++)
      for(int c=0;c<colunas;c++)
        {
        // Decidimos a cor baseado no estado do elemento.
        if (celula.getElemento(l,c) == true) g.setColor(Color.RED);
        else g.setColor(Color.LIGHT_GRAY);

g.fillOval(c*larguraCelula,l*alturaCelula,larguraCelula,alturaCelula);
        }
    }
}
```

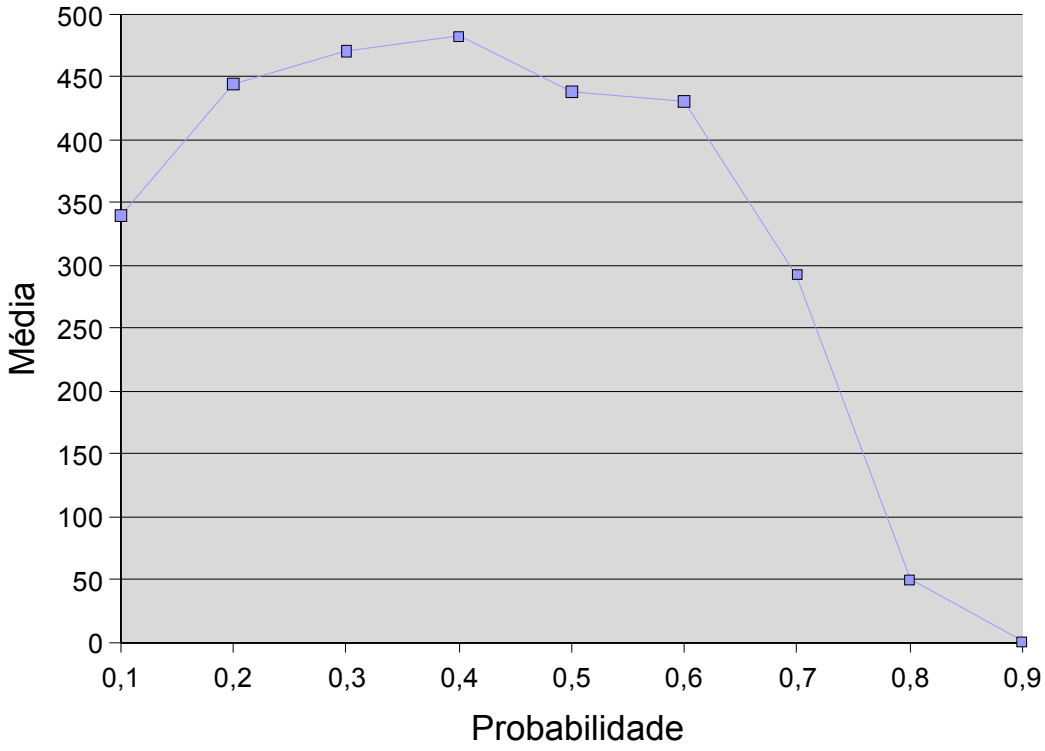
A terceira classe se refere a inicialização da matriz para visualização.

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.Timer;

public class DisplayMatriz extends JFrame implements ActionListener
  {
  private Matriz celula;
  private ComponenteMatriz componente;
  private JLabel contador;
  private Timer timer;

  public DisplayMatriz(Matriz c)
    {
    super("Matriz - Jogo da Vida"); // mostra título na barra da aplicação
    celula = c;
```

```
  // Adicionamos o componente para display da matriz.
  componente = new ComponenteMatriz(celula);
  getContentPane().add(componente,BorderLayout.CENTER);
  // Adicionamos um label.
  contador = new JLabel("Células vivas:");
  getContentPane().add(contador,BorderLayout.SOUTH);
  // Quando clicarmos no "x" a aplicação deve se encerrar.
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Ajustamos o tamanho da janela e mostramos a janela.
  pack();
  setVisible(true);
  timer = new Timer(500,this);
  timer.start();
  }
  // Este método seré executado quando clicarmos no botão avança.
 public void actionPerformed(ActionEvent e)
  {
  celula.itera();
  componente.repaint();
  contador.setText("Células vivas:"+celula.getNumeroDeCelulas());
  }
  // Ponto de entrada da aplicação, que cria a interface com o usuário.
 public static void main(String[] args)
  {
  Matriz c = new Matriz(40,40,0.6);
  new DisplayMatriz(c);
  }
}
```

Conclusões

Após obter a média das probabilidades entre 0 a 100% o gráfico resultante, mostrou que não houve discrepância com a idéia da qual a curva descrita pelo gráfico seguiria.
Com a utilização dos testes em menor escala, para contraprova, estes comprovaram a confiabilidade dos dados já obtidos.
O gráfico a seguir demonstra os dados obtidos através da comparação entre média e probabilidade (porcentagem):



Referencias bibliográfica

- Santos, Rafael. **Introdução à programação orientada a objetos usando JAVA**. Rio de Janeiro: Campus, 2003.
- http://www.math.com/students/wonders/life/life.html
- http://www.bitstorm.org/gameoflife/
- http://www.tech.org/~stuart/life/rules.html