TÉCNICAS VISUAIS PARA APOIAR O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

Tânia Martins Preto - tania@terra.com.br, tania@cesec.ufpr.br
Universidade Tuiuti do Paraná
Faculdade de Ciências Exatas e Tecnológicas
Curso de Sistemas de Informação
Av. Comendador Franco, 1860 – Guabirotuba
80815-909 - Curitiba - Paraná

Universidade federal do Paraná – UFPR
CESEC - Centro de Estudos de Engenharia Civil "Prof. Inaldo Ayres Vieira"
Curso de Pós-Graduação em Métodos Numéricos
Centro Politécnico - Caixa Postal 19011 - Jd Américas
81531-990 - Curitiba - Paraná

Resumo: *A disciplina de Programação de Computadores é de grande importância para os alunos da Engenharia, pois fornece base para futuros desenvolvimentos de aplicações computacionais relacionados a assuntos específicos em cada área. Alunos novatos costumam apresentar dificuldades na aprendizagem de linguagens de programação. Isto se deve a diversos fatores, dentre eles a falta de compreensão dos processos internos da máquina que ocorrem durante a execução de um programa e a dificuldade na elaboração de soluções na forma de algoritmos. Construir soluções algorítmicas para problemas reais, utilizando um conjunto de comandos de uma linguagem exige do aluno algumas habilidades como capacidade de abstração, organizar idéias usando as regras das linguagens e conhecimento sobre o significado dos comandos. Os sistemas de visualização de software que utilizam diversas técnicas para apoiar o ensino de programação e algoritmos, dentre elas a animação de algoritmos e a animação de dados. Estas técnicas buscam mostrar aspectos relevantes de um programa, ilustrando dinamicamente através de animações e gráficos aspectos relevantes dos algoritmos. Este trabalho introduz alguns conceitos sobre visualização de software e apresenta alguns exemplos de aplicações construídas na forma de applets Java.*

Palavras-Chave: *Linguagens de Programação, Animação de Algoritmos, Ambientes Interativos de Aprendizagem, Applets Java.*

1. INTRODUÇÃO

O ensino de Linguagens de Programação e Algoritmos, disciplina presente nos cursos de Engenharia e de Computação, nem sempre é uma tarefa fácil para alunos novatos. A utilização de métodos tradicionais de ensino, como quadro-negro e retro-projetor não atende totalmente às necessidades do professor, quando se deseja, por exemplo mostrar o efeito de um comando de repetição, a chamada de uma função ou a criação de uma conjunto de variáveis em tempo de execução.

Elaborar soluções algorítmicas para problemas reais, utilizando um conjunto de comandos de uma linguagem exige do aluno diversos conhecimentos como capacidade de abstração e organização das idéias, conhecimento a respeito do significado dos comandos e a relação entre os mesmos.

A aquisição de um novo conhecimento, muitas vezes é feita através do relacionamento da nova informação com outras adquiridas anteriormente, ou seja, a nova informação é associada à modelos existentes na memória, segundo MAYER, R. E. (1981).

A utilização de algumas estratégias de ensino como, por exemplo: (1) fornecer um modelo do computador que seja de fácil assimilação por parte do aluno ou que possa ser diretamente associado à algo que ele já conhece, (2) incentivar os alunos à descrever soluções para determinados problemas usando suas próprias palavras, (3) ilustrações gráficas detalhadas de aspectos relevantes dos algoritmos e estruturas de dados, e (4) a aprendizagem guiada, contribuem para melhorar a compreensão dos algoritmos por parte dos iniciantes.

Diversas ferramentas tem sido desenvolvidas a fim de facilitar a utilização dessas estratégias, destacando-se os sistemas de visualização de *software* e os sistemas tutoriais. Essas ferramentas buscam apoiar o ensino de programação, algoritmos e estruturas de dados e representam um grande apoio ao trabalho do professor em sala de aula, além de servir como material extra-sala. As mesmas não possuem o propósito de substituir os livros, os manuais das linguagens e tampouco o professor, mas sim contribuir na formação de modelos mentais corretos nos iniciantes, capacitando-os à resolver problemas mais complexos.

Este trabalho aborda os aspectos relevantes do processo de ensino-aprendizagem de linguagens de programação, o uso de técnicas de visualização de *software*, em especial animação de algoritmos e de dados, para o ensino de programação e descreve também alguns experimentos elaborados na forma de *applets* Java inseridos em textos *html*.

Outras experiências no ensino de Engenharia através de ambientes de aprendizagem interativa que usam *applets* Java são relatadas em SCHEER, S. (2000) e em SCHEER, S. (2001). As referencias relatam respectivamente, o ensino de diversos conceitos de mecânica de estruturas e o ensino de estruturas de aço para o curso de Engenharia Civil.

2- ASPECTOS RELACIONADOS À APRENDIZAGEM DE CONCEITOS DE PROGRAMAÇÃO

A aprendizagem de conceitos de programação, seja qual for a linguagem a ser ensinada e o método de ensino a ser utilizado, envolve a aquisição de alguns conhecimentos e habilidades específicas. A partir do momento que o aluno adquire esses conhecimentos ele passa a ser um programador perito e consegue resolver problemas de maior complexidade.

2.1 Características de um Programador Perito

Existem algumas abordagens para definir quais os conhecimentos e habilidades que caracterizam um programador como sendo perito:

Segundo du BOLAY, B. e SOTHCOTT, C. (1987), o aluno deve adquirir os seguintes conhecimentos:

(1) Sintaxe: Conhecimentos relacionados à forma, isto é, às regras de escrita. (2) Semântica: Conhecimentos referentes ao aspecto procedural que está relacionado ao que cada comando significa, ou seja, qual o comportamento produzido pelo mesmo e o aspecto declarativo, que corresponde ao conhecimento das propriedades das entidades pertencentes à linguagem. (3) Estratégias para decomposição: Saber avaliar que tipos de problemas podem ser resolvidos através do computador. A partir dai, saber decompor um problema grande em subproblemas menores, que possam ser resolvidos com mais facilidade e ainda, saber avaliar a necessidade ou não de fazer integração com outras ferramentas. (4) Estratégias para composição: Saber escolher entre diversas opções qual a melhor solução para um determinado problema.

Elaborar estratégias eficientes para união e comunicação entre as diversas partes de um programa. Especificar objetivos e tarefas. (5) Conhecimento pragmático: Saber utilizar os diversos recursos do sistema, isto é , saber editar, compilar, depurar, testar, interpretar as mensagens do sistema, etc. Outro ponto importante é saber escolher a metodologia de desenvolvimento.

Segundo PIMENTEL, A. R. E DIRENE, A., I. (1998), o aluno deve obter as seguintes habilidades: (1) Precisão Sintática e Semântica: Capacidade de escrever os comandos de forma correta e utilizá-los em contexto adequado. (2) Análise do Problema: Saber decompor o problema em sub-partes e projetar soluções adequadas para cada uma. (3) Reutilização de soluções prontas: Capacidade de adaptar soluções já conhecidas na resolução de novos problemas. (4)Simulação mental do estado do computador durante a execução de um programa: Capacidade de simular mentalmente os valores das variáveis e componentes do programa em cada etapa de sua execução.

2.2 A importância dos Modelos

Existem diversas estratégias para auxiliar o processo de aprendizagem. A apresentação de modelos concretos de situações e a apresentação do assunto de forma organizada são exemplos de estratégias que podem ser aplicadas em diversas áreas do conhecimento. No entanto, nem todas as estratégias são facilmente aplicadas ao ensino de programação.

Um dos principais motivos dessa dificuldade é que as atividades realizadas pelo computador não são visualizadas no "mundo real", como ocorre em atividades de outras áreas. Como exemplo é fácil visualizar a situação concreta que ocorre durante o cálculo de uma determinada área ou ainda a ação de uma força sobre um corpo.

Criar um modelo concreto fazendo analogia com alguma situação real em se tratando de conceitos de programação não é uma tarefa simples. É necessário que o professor use a sua imaginação para criar novos modelos não tão distantes do "mundo real" mas que de alguma maneira ilustrem o “mundo da máquina” de maneira simples.

Entender os processos que ocorrem durante a execução de programas exige um grande poder de abstração e para isso é necessário que o aluno tenha em mente uma imagem do que ocorre na máquina, essa imagem é chamada de modelo mental.

Ocorre freqüentemente que muitos programadores novatos escrevem seus programas sem saber o que cada linha de código representa fisicamente na memória do computador. O aluno tem que imaginar a situação e construir o seu modelo mental, que nem sempre corresponde a situação real, originando os problemas de aprendizagem.

2.3. A compreensão de programas

No contexto do conhecimento da sintática e semântica, compreender programas está relacionado com o entendimento de várias regras, conceitos e relações. Além disso, é necessário que ocorra o entendimento da lógica dos algoritmos e saber associar aos algoritmos as características da linguagem.

O conhecimento da lógica de construção de algoritmos envolve o entendimento de diversos aspectos. Estes aspectos podem ter algumas variações em função da linguagem que se pretende utilizar, mas em geral quem compreende bem aspectos relevantes de alguns algoritmos adquire habilidades que facilitam a compreensão e elaboração de outros algoritmo, mesmo em outras linguagens.

Alguns aspectos importantes que podem ser destacados no que diz respeito à lógica de algoritmos são os seguintes: Como dados são inseridos e manipulados, como as estruturas de dados são criadas e representadas, como é o comportamento dos comandos de repetição e

decisão e o seus respectivos efeitos sobre os dados; como ocorre a chamada de funções, passagem de parâmetros e retorno de valores.

2.4. Algumas Estratégias para Ensino

2.4.1 Criação de Modelos

Segundo MAYER, R. E.(1981), duas estratégias podem ser usadas no ensino de programação, a fim de melhorar o entendimento: (1) Incentivar o aluno a descrever informações técnicas (algoritmos) com suas próprias palavras, sem se preocupar inicialmente com as regras da linguagem. (2) Fornecer um modelo concreto dos processos que estão ocorrendo: ferramentas computacionais que utilizam recursos como a associação de imagens e animações ao processo de execução, podem ser úteis na realização dessa estratégia.

MAYER, R. E.(1976) ainda elaborou um modelo para ensinar comandos básicos em linguagens para principiantes e que é semelhante ao que muitos professores têm utilizado. Esse modelo consiste em visualizar a máquina como sendo dividida em quatro unidades: janela de entrada de dados, janela de saída de dados, Memória na forma de uma matriz com vários espaços, listagem do programa com um comando em cada linha.

A execução do processo é comandada pela listagem do programa e o efeito de cada comando é simulado passo a passo. Esse modelo, apesar de simples, serviu como base para diversos sistemas para visualização de programa utilizados hoje em dia.

2.4.2 Uso da Abordagem Conceitual

Outra contribuição de MAYER, R. E.(1981) diz respeito às diferentes abordagens que podem ser utilizadas para ensinar um determinado conceito: (1)Técnica. Ex.: *read* é um comando responsável por fazer a leitura. (2) Forma e gramática: O formato é *read(<nome>)*, coloca-se o nome de um endereço entre os parênteses. (3) Conceitual: O nome de um endereço é um espaço na memória e *read* coloca um valor digitado nesse espaço.

A abordagem conceitual produz resultados melhores, pois o aluno vai ter maior capacidade de transferir esses conhecimento adquiridos na solução de problemas novos.

2.4.3 Escolha de Exemplos Didáticos

A escolha de bons exemplos relacionados aos conceitos ensinados também auxilia a aprendizagem e contribuí para aumentar a motivação dos alunos em relação ao conteúdo. Alguns fatores a serem considerados nessa escolha são: (1) Exemplos bem elaborados podem servir de base para a resolução de problemas semelhantes e também mais complexos. (2) A apresentação de problemas deve ser feita em grau de dificuldade ascendente. Em PIMENTEL, A. R. e DIRENE, A., I. (1998) são apresentadas medidas que podem servir de base para essa ordenação.

2.4.3 O uso da Visualização

Mesmo com a utilização de várias estratégias em conjunto, a grande dificuldade dos alunos novatos é a falta de modelos que esclareçam o que ocorre durante a execução de um programa segundo MULHOLLAND, P. E EISENTADT, M. (1998). Visualizar o que está acontecendo é uma das estratégias mais eficientes, no sentido poder ajudar a vencer quase todos os obstáculos existentes na formação dos modelos que vão servir de base para o aluno resolver problemas mais complexos.

As ferramentas gráficas de apoio à aprendizagem de programação, através da visualização de diversos aspectos estão sendo responsáveis por automatizar as estratégias

citadas. Essas ferramentas devem ser de fácil utilização e devem atender às necessidades do programador novato.

É importante que as ferramentas gráficas utilizem notações gráficas adequadas ao conhecimento a ser passado. A notação não deve dar margens a dúvidas, pois é um dos principais fatores que vai influenciar na formação do modelo mental do aluno. As notações gráficas são usadas para ilustrar estruturas de dados, fluxo de controle de comandos de decisão, repetição e desvios e também chamada de funções recursivas e não recursivas.

Além dos itens acima, notações podem agregar outras características como tipo, se é dado de entrada ou saída e outros. Em FORD, L. (1993) é apresentado alguns modelos para notações. A maior parte das notações apresentadas em livros e utilizadas na ferramenta computacionais possuí diversas características semelhantes, como a utilização de figuras geométricas (quadrados, retângulos, losangos, triângulos), setas, fluxogramas, cores e outros.

Os ambientes interativos de aprendizagem utilizam diversos recursos para transmitir a informação, dentre eles, recursos de computação gráfica como animações e visualização bidimensional e em algumas aplicações, visualização tridimensional.

3- FERRAMENTAS DE APOIO

3.1 - Características Principais

Os sistemas para visualização de *software* e os sistemas tutoriais constituem as principais ferramentas de apoio ao ensino de programação. Os tradicionais ambientes integrados de programação correspondem a ferramentas mais utilizadas, no entanto, como geralmente são construídos com fins comerciais, não costumam ser de fácil utilização para os programadores novatos e nem possuem o objetivo de serem didáticos. É importante salientar que as ferramentas de apoio não substituem os ambientes integrados de programação, mas sim atendendem às necessidades dos programadores novatos. Os sistemas tutoriais por sua vez podem ser desenvolvidos em conjunto com as ferramentas para sua visualização de programas, mas sua principal característica é fornecer aprendizagem guiada onde conceitos são apresentados e repetidos de acordo com as necessidades do aluno.

3.2 - Sistemas de Visualização de *Software*

3.2.1 Características Básicas

Os sistemas de visualização de software destinam-se a executar visualização de programas computacionais e algoritmos, PRICE, B. ; BAECKER, R.; SMALL, I. (1998) , os mesmos originaram-se dos tradicionais diagramas de fluxos que nada mais eram que gráficos ilustrando o comportamento de programas. O objetivo principal dos sistemas de visualização de *software* é auxiliar programadores novatos à formar modelos mentais ou ainda abstrações visuais de alguns aspectos dos programas e algoritmos.

A área de visualização de *software* pode ser dividida em visualização de programas e visualização de algoritmos. A seguir serão descritas algumas características dessas duas áreas.

3.2.2 Visualização de Programas

Visualização de programas consistem na apresentação de características relacionadas ao código do programa, incluindo dados, comandos e estrutura geral da implementação; tentam ilustrar os processos que ocorrem durante a execução de um programa, PRICE, B.; BAECKER, R.; SMALL, I. (1998) . Os sistemas para Visualização de programas surgiram a partir da utilização de ferramentas de computação gráfica acoplada à interpretadores e

compiladores, enriquecendo a tarefa de depuração. Ao invés de mostrar apenas valores, tabelas e mensagens durante a execução, imagens com fins didáticos também são exibidas.

Algumas abordagens para visualização de programas são as seguintes:

- Visualização do Código: Tem-se o código do programa geralmente acompanhado de pequenos textos que explicam o significado e características de cada comando. Pode-se também ter a visualização da estrutura geral do código: diagramas ilustrando as diversas partes de um programa e como as mesmas se relacionam. Detalhes tais como dados e funções são também mostrados.
- Visualização de Dados: Gráficos relacionados à partes do código que podem ser caixas e diagramas contendo valores de variáveis, retorno de funções, valores de parâmetros e outros. Este tipo de visualização muitas vezes é utilizada pelos *debuggers*, que são ferramentas que realizam visualização para ajudar o programador a encontrar erros nos programas através do acompanhamento da mudança de valores das variáveis, dos parâmetros e valores de retorno de funções.
- Animação de dados e estruturas de dados: Corresponde também a visualização de dados, porém utiliza formas mais elaboradas de visualizar os dados, principalmente as estruturas de dados. Como exemplo pode-se citar a visualização de uma lista encadeada através de um conjunto de caixas com valores de dados e setas indicando quem é o próximo elemento da lista. Se estes dados mudam dinamicamente com a execução do programa, tem-se visualização de dados animada ou ainda animação de dados (e estruturas de dados).

Alguns exemplos de sistemas para visualização de programas podem ser encontrados nas seguintes referencias bibliográficas: NAPS, T. L. E BRESSLER, E. (1998) (WebGaigs), SANGWAN, R. S. (1998) (Sangwan), REZENDE, P. J. (1998) (Astral).

3.2.3 Visualização de Algoritmos

Busca descrever um nível mais alto do programa que corresponde aos seus algoritmos. Através de gráficos e animações tenta-se descrever como os algoritmos trabalham sem se prender tanto aos detalhes do código do programa.

Algumas abordagens para visualização de algoritmos são as seguintes:

- Visualização de algoritmos estática: Os diagramas de fluxo são exemplos de visualização estática de algoritmos, utiliza-se símbolos específicos para comandos simples, leitura e escrita e indicação de repetição. Porém esses símbolos não se alteram durante o programa.
- Animação de Algoritmos: Possibilita a visualização dinâmica dos algoritmos ou seja através de gráficos e animações são exibidos valores, troca de elementos, comparações. Os elementos gráficos vão sendo alterados durante a execução dos algoritmos. Alguns ambientes são constituídos de um conjunto de ferramentas *(tool-kits)* gráficas com facilidades para representar e movimentar objetos tais como vetores, listas, árvores, tabelas de valores e outros, com as quais professores e alunos podem construir animações para seus algoritmos, a fim de visualizar aspectos que consideram relevantes.

Alguns exemplos de sistemas para animação de algoritmos podem ser encontrados nas seguintes referencias bibliográficas: PIERSON, W. E RODGER, S. (1998) (JAWWA), STASKO, J. T. (1997) (Polka e Xtango), STASKO, J. T. (1997) (a) (Samba).

3.3 Sistemas Tutoriais

Os sistemas tutorias podem ou não incorporar técnicas de inteligência artificial (IA) na tutoria, sistemas que apresentam técnicas de IA são chamados de Sistemas Tutoriais Inteligentes, sendo que um dos seus principais objetivos quando usados no ensino de programação é acompanhar e orientar o aluno durante a resolução de exercícios, BINDER, F.V. e DIRENE, A.I. (1999). Para isto o sistema deve ter a capacidade de diagnosticar o comportamento do aluno ao programar e sugerir correções quando necessário.

A construção de sistemas para diagnóstico de programas não é uma tarefa fácil pois envolve diversos aspectos como por exemplo: o objetivo do programa, características da linguagem, erros comuns, comandos e estruturas de dados semelhantes, diferentes soluções para o mesmo problema.

O sistemas tutoriais que não adotam técnicas de inteligência artificial em sua concepção possuem diversos tipos de denominações em função de suas abordagens instrucionais, como por exemplo: CBT (*Computer-Based Training* ou Treinamento Baseado em Computador), CAI (*Computer- Assisted Instruction* - Instrução Assistida por Computador, micromundos, ambientes de exploração e outros.

Muitas vezes sistemas para animação de algoritmos são chamados de micromundos e vice-versa. Um exemplo típico é o sistema LOGO que ajuda o usuário a construir soluções para diversos problemas através da utilização de um conjunto de ferramentas gráficas prontas, visualizando os resultados em um ambiente de programação WENGER, E. (1987) . Esse sistema é denominado micromundo e no entanto serviu de base para vários sistemas de animação de algoritmos.

4. EXEMPLO DE UM AMBIENTE DE VISUALIZAÇÃO DE SOFTWARE

4.1 Características

Este trabalho envolve a construção de um ambiente interativo de aprendizagem que utiliza recursos de visualização de software baseado nas abordagens de animação de algoritmos e animação de dados. O objetivo principal das visualizações é propiciar ao aluno o acompanhamento de alguns casos típicos de algoritmos cuidadosamente escolhidos que servem de base para o entendimento de outros algoritmos e processos semelhante.

Os algoritmos e assuntos escolhidos fazem parte do conteúdo de algumas disciplinas relacionadas com programação, algoritmos e estruturas de dados, presentes em nos Cursos de Engenharia. Em geral a disciplina de Programação de Computadores (incluindo algoritmos) faz parte de todos os currículos de Engenharia e a disciplina de Estruturas de Dados geralmente está presente no currículo dos Curso de Engenharia da Computação.

As animações são apresentadas na forma de *Applets Java* inseridos em documentos *html* e acompanhados de textos explicativos sobre os conteúdos apresentados.

Alguns algoritmos estão prontos e outros estão em fase de desenvolvimento. O ambiente em breve poderá ser acessado através de seus endereços eletrônicos (URL) em http://www.cesec.ufpr.br/etools/AnimAlg e http://www.utp.br/tania/AnimAlg .

Os conteúdos selecionados para esta etapa do projeto são os seguintes: Vetores (Conceitos Básicos, Ordenação, Busca), Matrizes (Conceitos Básicos, Algoritmos Relacionados com Matrizes), Funções (Conceitos Básicos, Funções recursiva), Listas Encadeadas (Conceitos Básicos, Tipos de Listas Encadeadas, Algoritmos envolvendo Listas Encadeadas), Tipos Abstratos de dados (Conceitos Básicos, Listas, Pilhas e Filas).

Outras experiências no ensino de engenharia através de ambientes interativos que usam *applets* Java estão relatados em SHEER et all (2000) e SHEER et all (2001). A primeira

referência relata o ensino de diversos conceitos de Mecânica das Estruturas e a segunda refere-se ao ensino de Estruturas de Aço para o Curso de Engenharia Civil.

4.2 Apresentação de Casos

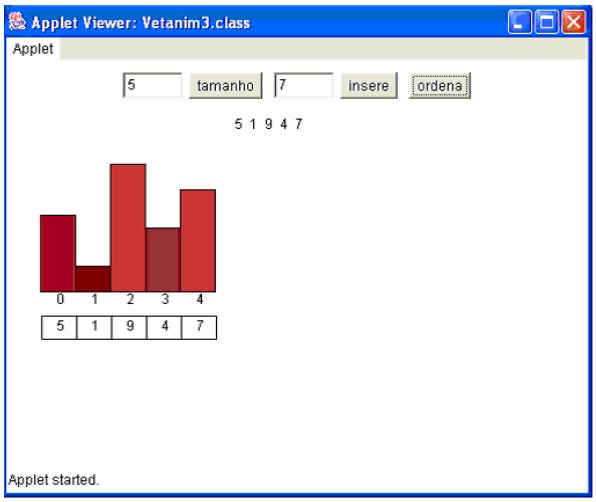
4.2.1 Algoritmo de ordenação de vetores através de Seleção

O algoritmo de seleção ordena elementos através da procura do menor elemento a cada iteração e inserção do mesmo em posições que variam do inicio para o final do vetor. A implementação do algoritmo em Linguagem C está exibida na figura 6.

O aluno pode selecionar o tamanho do vetor e entrar com os dados. Cada elemento pode ser visualizado de duas formas no vetor: Através de sua notação visual tradicional (conjunto de pequenos quadrado dispostos lado a lado) e através de barras verticais proporcionais aos valores de cada elemento do vetor.

Existe também uma variação da cor proporcional ao valor dos elementos, elementos menores são representados em tons mais escuros e os maiores em tons mais claros. A cor também serve para indicar quais elementos estão sendo trocados (amarelo com laranja). Os índices dos elementos também são exibidos abaixo das barras. A figura 1 ilustra a entrada de dados e a primeira visão dos elementos após a inserção dos dados iniciais. O tamanho do vetor do exemplo é 5.

Figura 1 - Visualização dos dados de entrada para o Algoritmo de Seleção.



A figura 2 ilustra a execução da primeira interação onde o elemento 5 era o menor (linha (5) do algoritmo) e foi substituído pelo elemento 1 após a execução dos comandos presentes nas linhas (6), (7) e (8).

Figura 2 – Escolha do menor elemento.

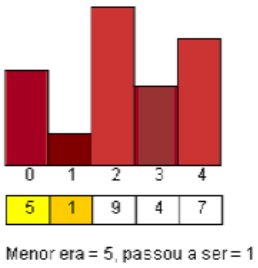
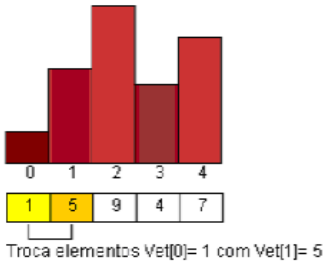


Figura 3- Troca do menor elemento.



A figura 3 mostra o vetor após a troca dos elementos 1 com 5. A figura 4 mostra o posicionamento do último elemento e a figura 5 mostra o vetor em seu estado final ordenado. As barras ficaram em ordem crescente de tamanho.

Figura 4 – Troca do maior elemento.

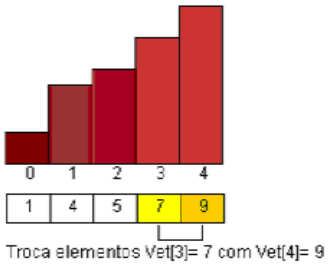


Figura 5- Vetor ordenado

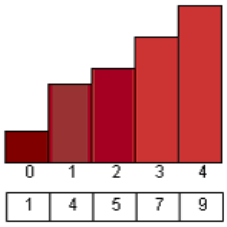


Figura 6- Algoritmo de Seleçao em Linguagem C

```
(1)  void selecao(int A[])
(2)  {
(3)   int i, j, imin,  x;
(4)   for(i=0; i<N-1; i++)   // para cada elemento i da primeira  até a penúltima posição
(5)    {      imin=i;           // escolhe elemento da vez como menor para iniciar
(6)           for ( j = i+1; j<n; j ++)     // percorre vetor da posição seguinte até o final
(7)                 if(A[j]<A[imin])   // se encontrar algum elemento menor que o escolhido
(8)                         imin=j;                 // atualiza elemento menor
(9)           x = A[i];           // realiza  troca  do elemento menor com elemento da posição i
(10)          A[i] = A[imin];
(11)          A[imin] = x;
(12)   }
(13) }
```

4.3.2 Visualização da Função Recursiva para a Seqüência de Fibonacci

A seqüência de Fibonacci é muito utilizada nas disciplinas de matemática e também é explorada nas disciplinas de programação e algoritmos devido ao seu comportamento que pode ser recursivo e iterativo.

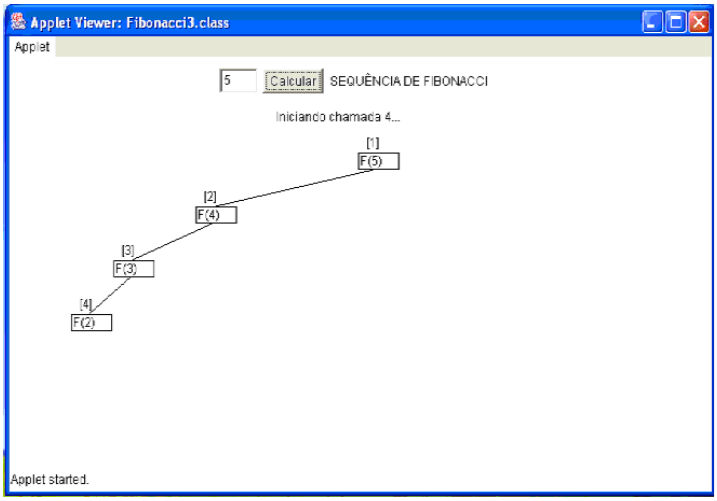
A definição matemática para calcular o n-ésimo termo da seqüência de Fibonacci, onde n é um inteiro é a seguinte:

Fib(0) = 0.
Fib(1) = 1.
Fib(n) = Fib(n-1) + Fib (n-2) para n>= 2.

O objetivo desta implementação é mostrar como são feitas as chamadas recursivas, passagem de parâmetros e retorno de valores. O calculo de cada valor da seqüência é feito em função dos dois último termos anteriores que são recursivamente calculados. Como são feitas duas chamadas recursivas, optou-se por colocar essas chamadas na forma de uma árvore binária, onde cada elemento está ligado a outros dois em um nível abaixo.

Cada chamada é ilustrada através de retângulos contendo acima o número de chamada entre colchetes e dentro o nome da função com o respectivo parâmetro. A figura 7 ilustra a entrada de dados e mostra as primeiras chamadas sendo executadas.

Figura 7: Entrada de dados e primeiras chamadas recursivas.



A medida que as chamadas vão sendo resolvidas seus valores aparecem na frente do retângulo e este fica em amarelo indicando que os dados já da chamada já foram desalocados a memória. A figura 8 mostra a visualização da execução no momento em que a 8ª chamada e´ ativada. Neste momento as chamadas 3, 4, 5, 6, 7 estão resolvidas.

A figura 10 mostra o valor de retorno da 12ª chamada e a figura 11 mostra o resultado final com todas as chamadas que foram executadas. A cor amarela indica que todas elas estão desativadas. A figura 12 mostra o algoritmo recursivo para a seqüência de Fibonacci implementado em Linguagem C.

Figura 8 – Inicio da 8ª chamada .

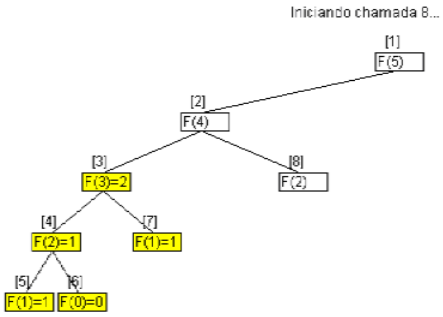


Figura 9 - Inicio da 12ª chamada .

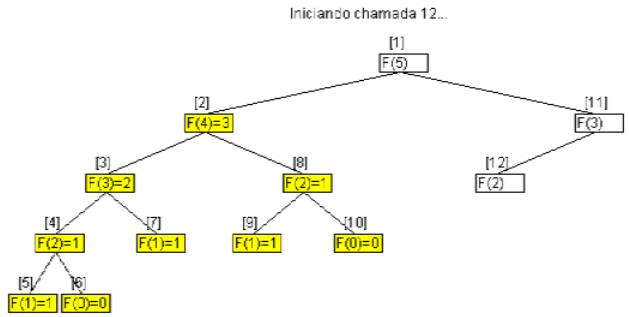


Figura 10 – Retorno da 12ª chamada.

Figura 11 - Resultado final .

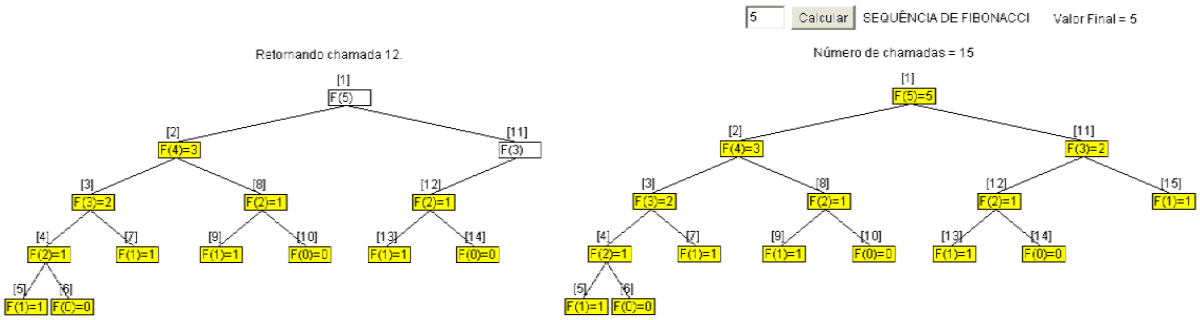


Figura 12- Algoritmo de Fibonacci recursivo em Linguagem C

```
(1)  int F (int n)
(2)  { if   (n <=1)    return n;              //  caso base para o encerramento da recursão
(3)   else   return  F(n –1 )  + F (n –2);   //  soma do resultado de duas chamadas recursivas
(4)   }
```

Um aspecto importante que a visualização mostra é a condição de parada da recursão, também chamado de caso base. No exemplo acima o processo recursivo é encerrado quando a função recebe os números 0 ou 1 como parâmetro.

5 – CONSIDERAÇÕES FINAIS

O ensino de linguagens de programação e algoritmos tem sido beneficiado pelas novas ferramentas de apoio. O uso de visualização e a construção de modelos que favoreçam a transmissão de informações e conceitos ajudam a desenvolver no estudante, a habilidade de solucionar problemas reais, pois fornece o contato com situações semelhantes àquelas que o aluno encontrará futuramente, AKAMATSU, J. e SENA, G. (2000).

A adaptação das estratégias de ensino destinadas a melhorar a aprendizagem, estão sendo facilitadas graças a grande importância que tem sido dada a automatização das tarefas de ensino e treinamento através do computador e da reunião de diversas tecnologias em um mesmo sistema, tais como computação gráfica, animação, técnicas de IA, hipermídia e outras.

Uma tendência presente nas pesquisas em sistemas para apoiar o ensino de linguagens de programação é a adequação dos sistemas à Web por atender um maior número de pessoas e usufruir de todas as facilidades que a Educação Baseada na Web proporciona.

Outra tendência está relacionada à adaptação dos sistemas às necessidades dos alunos, seja através da aprendizagem guiada, das interfaces adaptativas e das múltiplas maneiras de explicar um determinado conceito. O uso de visualização em especial favorece esse último aspecto. No exemplo apresentado sobre ordenação o mesmo vetor é apresentado de duas formas: a tradicional (seqüência de retângulos) e na forma de barras com cores e tamanhos proporcionais aos seus valores.

Segundo ROBLING. G. e NAPS, T. (2002) o uso da visualização de algoritmos facilita e promove a aprendizagem se alguns requisitos básicos são atendidos, dentre eles: (1)Apresentação do ambiente em plataforma de fácil acesso. (2) Sistemas de propósitos gerais, ou seja, modelos usados para visualizar um algoritmo devem ser aproveitados para visualizar algoritmos com objetivos semelhantes, dentro do contexto de um assunto ou disciplina. (3) Permitir que usuários forneçam entradas para o algoritmo. Este fator possibilita que sejam testados casos de valores e de comportamento extremos do algoritmo. (4)Explicações através de hipertextos com o objetivo de auxiliar os usuários à interpretar as abstrações e modelos usadas na visualização para ensinar um determinado conceito.

Os experimentos apresentados neste trabalho buscam atender os requisitos acima devido às seguintes características: (1) O uso de *applets* Java em páginas *html* representa uma plataforma de fácil acesso pois a partir de qualquer *browse* que suporte o Java é possível usar o sistema. (2) Os modelos usados para representar o vetor são facilmente inseridos na visualização de outros algoritmos de ordenação. O modelo usado para representar as chamadas recursivas também e´ facilmente adaptado para outros algoritmos recursivos, por exemplo, algoritmos para manipulação de árvores binárias. (3) Os *applets* desenvolvidos permitem ao usuário entrar com os dados. (4) Os textos explicativos contém partes do código e informações relevantes sobre os conceitos julgados mais importantes.

Outro aspecto importante presente na visualização do algoritmo de *Fibonacci* é a apresentação dos casos base que determinam o término das chamadas recursivas. Segundo HABERMAN e AVERBUSH (2002), uma das maiores dificuldades dos alunos em relação à construção de algoritmos recursivos é elaborar de forma correta os casos base, que muitas vezes são ignorados ou colocados de forma errada.

O uso de visualização gráfica auxilia a caracterizar os ambientes de aprendizagem como interativos e tem contribuído de forma positiva no ensino de programação e algoritmos. Isto se deve a diversos fatores, principalmente por contribuir na formação de um modelo mental correto sobre conceitos e aspectos que ocorrem na memória da máquina e também por utilizar imagens e animações que tornam o processo de ensino e aprendizagem mais atraente e divertido.

REFERÊNCIAS BIBLIOGRÁFICAS

AKAMATSU, J. e SENA, G. Tecnologias de Aprendizagem para o Ensino de Engenharia in Loco e a Distância. (2000) In: 28º COBENGE, Anais.

BINDER, F.V. e DIRENE, A.I. Conceitos e Ferramentas para Apoiar o Ensino de Lógica de Programação Imperativa (1999). In. X SBIE, Curitiba, Anais.

du BOLAY, B. e SOTHCOTT, C. Computers Teaching Programming: An Introductory Survey of the Field (1987)., In: Lawer, R. W and Yazdani, M. ARTIFICIAL INTELLIGENCE AND EDUCACIONAL: LEARNING ENVIRONMENT AND TUTORING SYSTEMS.

FORD, L. How programmers visualize programs. (1993), FIFTH WORKSHOP ON EMPIRICAL STUDIES OF PROGRAMMERS.

HABERMAN e AVERBUSH (2002) The Case of Base Cases: Why are They so Difficult to Recognize? Stdents Difficulties with Recursion. (2002) In: ITiCSE´02., Denmark, ACM.

MAYER, R. E. The Psychology of How Novices Learn Computer Programming (1981), ACM COMPUTING SURVEYS, vol. 13, n.1.

MAYER, R.E. Some Conditions of Meaningful Learning for Computer Programming: Advance Organizers and Subject Control of Frame Order (1976), JOURNAL OF EDUCACIONAL PSYCHOLOGY, n.68.

MULHOLLAND, P. E EISENTADT, M. Using Software to Teach Computer Programming: Past, Present and Future (1998), In SOFTWARE VISUALIZATION - PROGRAMMING AS A MULTIMEDIA EXPERIENCE. Stasko, D. and Brown, P., MIT Press.

NAPS, T. L.; BRESSLER, E., A multi-windowed environment for simultaneous visualization of related algorithms on the WWW (1998), XXIX SIGCSE, ACM Press, EUA.

PIMENTEL, A. R. E DIRENE, A., I. Medidas Cognitivas no Ensino de Programação de Computadores com Sistemas Tutores Inteligentes (1998). In. IX SBIE, Anais, Fortaleza

PIERSON, W. E RODGER, S. (1998) . Web-based Animation of Data Structures Using Jawwa, In. SIGSE'98. Proceedings. ACM Press. EUA.

PRICE, B. ; BAECKER, R.; SMALL, I. An Introduction to Software Visualization (1998), In SOFTWARE VISUALIZATION - PROGRAMMING AS A MULTIMEDIA EXPERIENCE. Stasko, D. and Brown, P., MIT Press.

REZENDE, P. J.;GARCIA, I.; CALHEIROS, F. ASTRAL: Um ambiente para Ensino de Estruturas de Dados através de Animações de Algoritmos (1997), REVISTA BRASILEIRA DE INFORMÁTICA NA EDUCAÇÃO - SBC, no. 1, ISSN: 1414-5685.

ROBLING, G. e NAPS, T. A Testbed for Pedagogical Requeriments in Algorithm Visualization (2002) In: ITiCSE´02., Denmark, ACM.

SANGWAN, R. S. A System for Program Visualization in the Classroom. (1998) SIGSE'98 In. Proceedings. ACM Press.

SHEER, S; CUNHA, M.; PINTO, F.; DAMIAN, R.; SANTOS F., J. Desenvolvendo *applets* Java para ensino de Engenharia (2000) In: 28º COBENGE, Anais.

SHEER, S; AZEVEDO, C.; WORMSBECKER, L.; ADAIME, L.; MILEK, J., Curso *online* de Estruturas de Aço (2001) In: 29º COBENGE, Anais.

STASKO, J. T. Polka Animation Designer's Package (1997), GA TECHNICAL REPORT GIT-GVU, Georgia Institute of Technology, Atlanta.

STASKO, J. T. (a) Samba Animation Designer's Package (1997), GA TECHNICAL REPORT GIT-GVU, Georgia Institute of Technology, Atlanta.

WENGER, E. Artificial Intelligence and Tutoring Systems: COMPUTATIONAL AND COGNITIVE APPROACHES TO THE COMMUNICATION OF KNOWLEDGE. (1987), Morgan Kaufmann.