A animação na aprendizagem de conceitos básicos de programação

Anabela de Jesus Gomes*, António José Mendes**
(*)Centro de Informática e Sistemas da Universidade de Coimbra Instituto Superior de Engenharia de Coimbra, (**) Centro de Informática e Sistemas da Universidade de Coimbra
e-mail : anabela@isec.pt , toze@dei.uc.pt

Resumo

É comummente aceite que a aprendizagem da programação coloca grandes dificuldades a muitos estudantes, sendo frequentes os casos de insucesso. Isto faz supor que os métodos tradicionalmente utilizados no ensino destas matérias não sejam os mais adequados. Existe um vasto conjunto de estudos que revela a valiosa contribuição que a utilização de técnicas de visualização e animação pode dar para o entendimento deste tipo de matérias. Neste trabalho é efectuada uma breve exposição sobre algumas ferramentas propostas ao longo dos anos, recorrendo às mais variadas técnicas, para apoiar o ensino da programação. É ainda feita uma reflexão sobre as qualidades que um sistema deverá possuir para apoiar este tipo de actividades. Finalmente, é apresentada uma descrição muito sumária, de uma ferramenta que se encontra em desenvolvimento, proposta para combater alguns dos problemas referidos.

Palavras Chave: Animação de algoritmos, Sistemas de Visualização, Informática Educativa, Ensino da Programação

1. Introdução

A programação é muito mais do que a escrita de um conjunto de linhas de código numa dada linguagem, é uma arte e uma ciência. Arte porque existem muitas maneiras diferentes de se codificarem instruções, com alguma criatividade. É também uma ciência, porque é constituída por um conjunto de regras orientadoras, porque é necessário o uso de lógica e porque existem alguns métodos rigorosos de programação que asseguram a eficiência, economia e utilidade dos programas gerados.

Aprender a programar é um processo difícil e tornar-se um bom programador exige vários tipos de aptidões. A experiência tem demonstrado que, uma das grandes dificuldades, apresentadas pela generalidade dos alunos, na resolução de problemas nesta área reside precisamente na compreensão e aplicação de certos conceitos abstractos de programação. Em particular, a aplicação de noções básicas, como as estruturas de controlo, à criação de algoritmos que resolvam problemas concretos, coloca desafios difíceis de vencer para muitos estudantes. Estes problemas devem-se essencialmente ao grande nível de abstracção exigido e também às metodologias tradicionais de ensino que privilegiam a aprendizagem de conceitos dinâmicos utilizando principalmente materiais de índole estática. Há ainda um outro conjunto de dificuldades que agrava toda esta situação, de que se destacam:

- a necessidade de um bom nível de conhecimentos e prática de técnicas de resolução de problemas;
- a exigência de um estudo muito baseado na prática e, por isso, bastante diferente do requerido pela maioria das disciplinas (mais baseado em noções teóricas, implicando muita leitura e alguma memorização);
- as linguagens de programação usuais apresentam sintaxes complexas e não têm representações visuais dos algoritmos, o que não contribui para uma melhor compreensão;
- a existência de turmas com alunos com *backgrounds* diversificados acerca das matérias em questão, traduzindo-se em ritmos de aprendizagem muito diferenciados;
- a impossibilidade de um acompanhamento individualizado ao aluno, devido à existência de turmas demasiado grandes e ao tempo dentro do currículo escolar ser demasiado apertado.

O facto de os programas computacionais serem inerentemente dinâmicos sugere que as operações e interacções dos algoritmos sejam, em geral, melhor descritas por meio de representações visuais dinâmicas. Assim, a visualização deve consistir no mapeamento de representações computacionais em representações perceptuais ou visuais, permitindo concretizar o abstracto, escolhendo técnicas de codificação que possam maximizar a compreensão e comunicação humana. Desta forma é possível permitir que os alunos observem e percebam visualmente a forma como um dado algoritmo funciona.

Ao aluno pede-se também para elaborar soluções para problemas que lhe são postos. Esta é, claramente, a grande dificuldade para muitos alunos. A utilização de sistemas de visualização pode, também neste caso, ser muito útil, permitindo ao aluno visualizar a forma como o seu algoritmo se comporta, detectar eventuais erros, corrigi-los e através disso aprender.

2. Importância dos sistemas de visualização e animação de algoritmos

Existem diversos estudos empíricos que permitem comparar a eficácia da utilização de animações e representações visuais em contraposição às linguagens textuais, para a compreensão de algoritmos e programas computacionais.

Um estudo conduzido no Georgia Institute of Technology (Lawrence et al., 1994) tinha como finalidade medir a eficácia da utilização de algoritmos animados, em várias estratégias de ensino. Essas estratégias variaram no nível de controlo e envolvimento activo dos alunos. A experiência foi conduzida num ambiente de sala de aula e os objectos do estudo foram alunos de ciências da computação que frequentavam o primeiro curso de programação. As animações algorítmicas utilizadas foram criadas com os Sistemas XTANGO (Stasko, 1992) e POLKA (Stasko y Kraemer, 1993). Foram estudadas diversas variáveis, incluindo o estilo de apresentação do algoritmo animado (animação ou transparências), o uso de uma sessão laboratorial para clarificar conceitos algorítmicos e a interacção dos alunos com a animação durante a sessão laboratorial (onde um grupo recebeu conjuntos de dados preparados e outro grupo criou os seus próprios conjuntos de dados). O algoritmo estudado foi o "Kruskal's Minimum Spanning Tree". Após as sessões a que foram sujeitos, incluindo várias estratégias de ensino, os diversos grupos realizaram um teste *on-line* de questões tipo verdadeiro/falso e de escolha múltipla cuja finalidade era a de testar a compreensão acerca do algoritmo e sua aplicação. Para além disso, os alunos também efectuaram um teste de respostas livres no papel, cuja finalidade era a de verificar a sua capacidade em articular conceitos relacionados com a compreensão do algoritmo. As conclusões obtidas pelos autores do estudo foram as seguintes:

- Nos testes de resposta livre os grupos sujeitos a condições laboratoriais activas (onde os alunos participaram na construção das animações) obtiveram pontuações significativamente mais elevadas do que todos os outros grupos, incluindo os sujeitos a condições laboratoriais passivas (onde os alunos se limitavam a observar as animações, com reduzida participação e intervenção). Isto foi mais visível nas questões que exigiam conhecimentos a nível mais profundo (testes de resposta livre). As questões neste tipo de teste requeriam a chegada a conclusões a partir das questões colocadas, bem como a demonstração de uma outra versão do algoritmo;

- Os alunos que receberam a sessão laboratorial (activa ou passiva) obtiveram melhores resultados nos testes *on-line* com questões do tipo verdadeiro/falso e de escolha múltipla, mas sem que tal diferença se revele significativa. Isto indica que a participação laboratorial dos alunos é mais relevante em questões que requerem mais conhecimento conceptual do que em questões mais básicas, como as que apenas requerem o reconhecimento dos passos individuais do algoritmo;
- A utilização de exemplos quer através de animações ou transparências não faz muita diferença no processo de ensino do algoritmo. Isto poderá ser explicado, pelo facto de ambos estes métodos usarem técnicas visuais passivas, uma vez que em qualquer dos casos não há interacção do aluno com a representação do algoritmo.

Outros estudos corroboram estes resultados, por exemplo Shih y Alessi (1994) mostraram que os benefícios e contributos da animação, para uma melhoria da capacidade dos alunos resolverem problemas procedimentais e conceptuais acerca de algoritmos são, na sua maioria, significativos, constatando que existe alguma evidência indirecta de que a animação faz com que o conhecimento seja adquirido mais rapidamente.

Outros estudos analisados indicam também que as animações, para serem eficazes, têm de permitir interacção e vários níveis de controlo pelo aluno. É o facto de os alunos intervirem activamente numa animação, participando na sua construção e reconstrução, explorando várias estratégias de resolução de problemas que provavelmente os ajudará a resolver novos problemas. Os estudos publicados por Stasko et al. (1993), Ramani y Rao (1994) e Kann y al. (1997) também demonstraram a influência da dimensão activa *versus* passiva na aprendizagem, reafirmando a ideia de que as animações para serem eficazes têm de ser usadas como parte de um conjunto educacional mais vasto, em que os alunos podem beneficiar da construção do algoritmo que é uma aprendizagem activa em oposição à sua simples visualização, que é uma aprendizagem passiva.

Com base nestes resultados, pode-se afirmar que, para o tipo de matérias referidas é vantajoso utilizar meios animados em detrimento de materiais não animados, essencialmente devido aos seguintes aspectos:

- Possibilidade de inclusão, numa animação, de elementos que podem fornecer informação mais detalhada ao aluno, não facilmente discernível de outra forma;
- Possibilidade de os alunos verificarem como é que determinados conjuntos de dados afectam o funcionamento do algoritmo;
- Possibilidade de execução e simulação das resoluções do problema em tempo real, traduzindo-se num melhor estudo e compreensão do comportamento do algoritmo;
- Os alunos são normalmente seduzidos por projectos que envolvam animações em detrimento de materiais estáticos.

Porém, para que a animação seja mais efectiva ela não deverá ser usada isoladamente, mas como uma parte integrante da experiência de aprendizagem. Na realidade, também pensamos que o valor real de uma animação reside não na sua simples visualização, que permite um reduzido nível de interactividade, mas na capacidade de permitir que o aluno intervenha, construa, explore, modifique, experimente e teste as suas teorias, ou seja, permita vários níveis de interacção.

3. Alguns sistemas propostos na literatura

Diversas técnicas educacionais e ferramentas de *software* têm sido desenvolvidas ao longo dos últimos 30 anos para ajudar as actividades de aprendizagem sobre

funcionamento de algoritmos e programas. Os sistemas de animação/visualização de *software* têm sido usados com o propósito de apelar ao potencial do sistema visual humano. Assim, uma vez que os programas computacionais podem ser pouco claros quando apresentados num formato textual, é esperado que o formato gráfico animado contribua para uma melhor compreensão.

Os sistemas de visualização de programas variam desde programas de baixo nível de detalhe, que mostram manipulações de estruturas de dados, até programas com um nível mais global de detalhe, como os que mostram o seu propósito e metodologias. Os sistemas que apresentam uma visualização abstracta global de mais alto nível são designados de sistemas de visualização de algoritmos. Muitas vezes, estes sistemas apresentam uma visualização contínua do programa durante a sua execução, adquirindo o nome de animação de algoritmos.
O BALSA (**B**rown University **Al**gorithm **S**imulator and **A**nimator) (Brown y Sedgewick, 1985), foi o primeiro e um dos mais importantes sistemas interactivos de animação de algoritmos. Foi escrito em C, mas os algoritmos que animava eram geralmente codificados em Pascal. Neste sistema, foi desenvolvida uma metodologia pioneira para a produção de objectos geométricos que representavam actividades significativas do programa (estados do programa) e não apenas os estados das estruturas de dados. A fundamentação desta metodologia consistia na identificação de pontos estratégicos num algoritmo (*interesting events*), transportando as suas características fundamentais. O BALSA permitia que os utilizadores controlassem as animações (parar, iniciar, controlar a velocidade, fazer a sua repetição, entre outros). Muitas das animações podiam também ser executadas ao contrário (*backward*). Suportava múltiplas visualizações simultâneas das mesmas estruturas de dados, para o mesmo algoritmo em execução, e foi o primeiro sistema que podia mostrar vários algoritmos em execução no mesmo ecrã a fim de comparar os seus desempenhos. Normalmente fornecia também uma visualização do código, mostrando o procedimento corrente com a linha actual salientada.

Também bastante interessante, o sistema de animação de algoritmos ZEUS (Brown, 1991), foi escrito em e para Modula-3, explorando a hereditariedade de objectos do Modula, de forma a que o utilizador pudesse tirar partido das classes e métodos pré-definidos, a fim de construir uma animação sofisticada e eficiente rápida e facilmente. Tal como o BALSA suporta múltiplas visualizações sincronizadas permitindo que os utilizadores as editem. O utilizador pode alterar representações de dados, assim que a animação é parada, através da edição de texto, manipulação directa ou invocando uma função. Este sistema permite ainda ao utilizador facilidades de configuração e controle. As primeiras permitem que o utilizador seleccione que algoritmo executar, que visualizações abrir, que dados fornecer ao algoritmo seleccionado ou ainda a possibilidade de gravar para ficheiro determinados instantes do estado do sistema ou restaurar o sistema a partir de um determinado instante previamente registado. As facilidades de controle permitem iniciar, parar, executar passo-a-passo um determinado algoritmo ou controlar a velocidade da animação.

Mais recentemente, surgiu o sistema MRUDS (***M**ultiple **R**epresentation for **U**nderstanding **D**ata **S**tructures*) proposto em Hanciles et al. (1997). Trata-se de um protótipo computacional com o objectivo de melhorar a aprendizagem de conceitos abstractos de programação, a alunos com pouco ou nenhum conhecimento sobre estruturas de dados, utilizando para isso múltiplas representações visuais. É aplicado dentro do domínio das estruturas de dados lineares: tabelas, pilhas, filas e listas ligadas. O protótipo desenvolvido integrou duas estratégias de aprendizagem nomeadamente, a elaboração e a meta cognição. As estratégias de elaboração são incluídas através da criação de imagens, a partir de informação textual e também pela criação de analogias relevantes para o domínio alvo. As estratégias de meta cognição, permitem a

manipulação interactiva das representações do domínio, fornecendo múltiplas representações gráficas.

O protótipo é constituído por três módulos: Analogia, Representação e Algoritmo. No módulo Analogia, tentam-se estabelecer analogias entre as estruturas de dados a aprender e problemas concretos da vida real. Assim, os alunos poderiam tirar conclusões e formar imagens e modelos mentais sobre a forma de funcionamento das estruturas de dados. A teoria patente neste módulo é a de que os alunos, apesar de não familiares com as estruturas de dados, seriam familiares com as suas experiências quotidianas. Por exemplo, para representar uma tabela, este módulo estabelece uma analogia com uma rua com filas de casas, onde cada casa representa um elemento da tabela. Cada casa possui um endereço único podendo ou não ter ocupante.

No módulo representação, os alunos são orientados no sentido de adquirirem um conhecimento mais detalhado sobre o funcionamento das estruturas de dados. Foram usados diagramas típicos, presentes em livros de texto, para ilustrar as estruturas de dados. Embora mantendo os conceitos essenciais adquiridos no módulo analogia, as estruturas de dados são aqui apresentadas como entidades que armazenam dados. Neste módulo, também se introduz uma ilustração gráfica de aspectos mais abstractos das estruturas de dados como, por exemplo, ponteiros. A implementação do módulo representação foi pensada de forma a alcançar um nível máximo de mapeamento sintáctico entre o conceito e a representação.

O módulo Algoritmo apresenta as estruturas de dados como sendo manipuladas via algoritmos. Assim é efectuada uma ligação entre a implementação textual do algoritmo e a implementação diagramática anterior, ilustrando o efeito de cada linha de código, usando os mesmos diagramas utilizados no módulo de representação. Assim, os processos ilustrados no módulo representação são particionados em sub-processos, cada um dos quais representados numa linha de pseudo código.

Há que referir outros sistemas, que recorrem a representações visuais/animações, como por exemplo: SEE (Baecker y Marcus, 1986), BALSAII (Brown, 1988), VIP (Mendes y Mendes, 1988), Pascal Genie (Miller y Chandhok, 1989), GAIGS (Naps, 1990), Tango (Stasko, 1990), ANIM (Bently y Kerninghan, 1991a), Pavane (Roman et al., 1992), Xtango (Stasko, 1992), POLKA (Stasko y Kraemer, 1993), DRUIDS (Whale, 1994), FLAIR (Ingargiola et al., 1994), POLKA-RC (Stasko y McCrickard, 1995).

Baseados na crença de que existem vantagens cognitivas em utilizar metodologias gráficas em vez de textuais, outras propostas, para ensinar programação, foram para a criação de linguagens de programação baseadas em ícones, de que se destacam o BACII (Calloni y Bagert, 1993) e o BACII++ (Calloni et al., 1996) e de *design languages* de que é exemplo o G2 (Ellis y Lund, 1994).

Para apoiar o ensino da programação foram também desenvolvidos diversos sistemas recorrendo a diversas técnicas existentes em inteligência artificial, de que são exemplo o *Lisp-Tutor* (Anderson y Reiser, 1985), PMS (Tomek et al., 1985), Proust (Johnson et Soloway, 1985), TPM (Eisenstadt y Brayshow, 1988), Ceilidh (Benford y Burke, 1993), COACH (Selker, 1994), ADAPT (Fix y Wiedenbeck, 1996), *Loop Tutor* (Tyerman at al., 1996) e C-*Tutor* (Song *et al.*, 1997).

Outras propostas de aplicações nesta área são os micro mundos, seriamente influenciados pelos gráficos da tartaruga do LOGO (Papert, 1980). O LOGO não foi concebido especificamente com o propósito de ensinar programação, mas surge como um bom instrumento para introduzir conceitos de programação a alunos sem experiência neste domínio.

Existem diversos exemplos de micro mundos com aplicações na área de programação. A primeira e ainda a mais popular das mini linguagens foi "Karel the Robot" (Pattis, 1981), concebida com o objectivo de ensinar estudantes universitários no

seu curso introdutório à programação em Pascal. Este sistema contém todas as estruturas de controlo importantes do tipo das do Pascal ensinando os conceitos básicos sobre noções de execução sequencial, abstracção procedimental, execução de condições e repetição. No Karel, a sobrecarga das linguagens de programação completas de alto nível é reduzida, na medida em que não há variáveis, tipos ou expressões. O actor, *robot* Karel, realiza tarefas num mundo que consiste na intersecção de ruas e avenidas, paredes e *beepers*. Karel também transporta alguns *beepers* na sua "mala". As principais acções de Karel são movimentar-se, virar-se à esquerda ou direita, levantar e pousar o *beeper*. Um conjunto de dezoito predicados permite que Karel verifique o estado do seu micromundo. Por exemplo, Karel pode desempenhar diversas tarefas, entre as quais, determinar a presença de paredes próximas, se há ou não *beepers* na sua mala ou em determinado sítio. Através da escrita de programas que fazem com que Karel realize cuidadosamente as tarefas seleccionadas, os alunos adquirem experiência com os aspectos fundamentais, enquanto usam uma metáfora agradável e persuasiva.

Posteriormente, surgiram uma série de ambientes inspirados no Karel, usando algumas das suas características, entre elas *Martino* (Olimpo *et* al., 1985) e *Marta* (Calabrese, 1989), *Karel-3D* (Hovrecky, 1992) *Darel* (Kay y Tyler, 1993) e *Karel Genie* (Miller et al., 1994). *Josef the Robot* (Tomek, 1982), *Robot Brothers* (Olimpo, 1988), *Playground* (Fenton y Beck, 1989), *Turingal* (Brusilovsky,1991), Gravitas (Sellman, 1992), KidSim (Smith et al., 1994), *Tortoise* (Brusilovsky,1994), e *TurtleGraph* (Jehng et. al, 1994), são exemplos de outras mini linguagens e sub conjuntos de linguagens que surgiram com o intuito de contribuir para uma melhoria da compreensão e realização de tarefas de programação.

4. O que faz a diferença?

Dos estudos efectuados, parece haver um conjunto de requisitos essenciais que uma boa ferramenta deve possuir para facilitar a aprendizagem da programação, entre eles:

- Ser Interactiva, gerando *feedback* contínuo com o aluno permitindo-lhe interactuar, controlar e desempenhar um papel activo no processo de aprendizagem;
- Ser Configurável, a fim de permitir a adição, remoção e alteração dos exemplos, exercícios propostos e soluções apresentadas, sem que tal acarrete quaisquer alterações no seu código;
- Permitir Representações alternativas, de forma a possibilitar ao aluno diferentes pontos de vista, estilos de raciocínio e soluções para os mesmos problemas;
- Ser Retroactiva, a fim de apresentar uma capacidade de resposta imediata, não apenas positiva mas também negativa, evitando a edificação de conceitos errados sobre bases incorrectas;
- Ser Animada, para melhor expressar as ideias, transmitindo a dinâmica envolvida e a informação patente no problema com mais realismo, facilitando a apreensão de conceitos, sem contudo gerar confusão visual.

Porém, para que a ferramenta construída não se resuma a mais um trabalho académico ou protótipo sem utilização prática, é necessário que possua um conjunto de características que a tornem utilizável, entre elas, ser:

- Simples, óbvia e intuitiva, exigindo pouco tempo de aprendizagem, mas persuasiva, sofisticada e atractiva;
- Portável, para que seja facilmente transportável entre diversas arquitecturas e sistemas operativos;

◆ Económica, envolvendo baixos custos de aquisição, para alcançar um elevado número de alunos.

Porém, como já exposto, pensamos que de todas estas características, as fundamentais, sejam a interactividade conjugada com a simplicidade. Assim, as animações deveriam fazer com que fosse relativamente fácil para o aprendiz prever o que irá acontecer em cada passo do programa/algoritmo, encorajado-o a auto-explorar o comportamento do programa/algoritmo, alterando-o e participando activamente na sua construção e reconstrução, a fim de melhor integrar a nova informação com as estruturas do conhecimento existentes, fazendo com que seja mais fácil transferir informação para novas situações. Todavia, construir um sistema deste tipo é um processo complexo que implica a reflexão sobre um vasto conjunto de questões, nomeadamente:

- Que nível de interactividade é que o sistema deverá permitir?
- Como é que a animação deve ser integrada no ambiente de aprendizagem?
- Numa dada solução, quais as partes que devem ser sujeitas à animação?
- Que tipos de animação podem ser geradas?
- A animação deverá ser verídica ou dar uma falsa aparência, ocultando certos detalhes de forma a minorar a confusão visual, fornecendo uma imagem mais clara?
- A animação, no caso de um algoritmo, deverá ser complementada com características adicionais tais como pseudo código, informação textual, entre outras?
- O que é que o aprendiz necessita de saber para "apreciar" e compreender a animação?

No fundo, a questão essencial consiste em saber o que é que constitui uma boa animação de algoritmos. Na realidade, não há linhas orientadoras claras para a construção de animações de uma perspectiva psicológica sendo, geralmente, tomadas decisões que nem sempre poderão ser acertadas, mas que em regra estão relacionadas com experiências anteriores do conceptor.

5. Conclusão

A maior dificuldade com que os alunos se deparam na aprendizagem de uma linguagem de programação diz respeito à concepção e formalização de uma solução para um determinado problema e não à sua codificação. A constatação de que o processo de entendimento e resolução de problemas computacionais é um processo dinâmico leva a sugerir que os métodos de ensino tradicionais não sejam os mais adequados às necessidades apresentadas, uma vez que promovem habitualmente a aprendizagem de conceitos dinâmicos, utilizando essencialmente técnicas estáticas, com as dificuldades daí resultantes.

Ao longo dos anos têm sido efectuados diversos estudos com o intuito de verificar a eficácia da utilização de animações/visualizações, na compreensão e resolução de problemas computacionais. A maioria deles constatou a valiosa contribuição do apelo visual dinâmico para a formação de conceitos, em detrimento da apresentação de imagens estáticas, e uma vantagem significativa da animação com elevados níveis de interactividade.

Os educadores estão constantemente a procurar novas formas e estratégias para melhorar a qualidade de ensino, minorando o insucesso e taxas de reprovação. Nesse sentido e concretamente na área de programação têm havido diversos esforços e propostas para desenvolver metodologias especiais, linguagens e ferramentas com características didácticas para minorar os problemas da aprendizagem.

Mas com um conjunto tão vasto de alternativas, que segundo os seus autores e de acordo com diversos estudos efectuados, representam um contributo tão valioso para a educação, porque razão é que os problemas subsistem? Porque é que essas ferramentas não são amplamente utilizadas hoje em dia?

Na realidade, muitas destas soluções foram apresentadas no âmbito de projectos académicos, não se encontrando disponíveis para avaliação e experimentação. Além disso, muitas delas foram construídas com limitações a nível de hardware e características gráficas que hoje em dia é possível ultrapassar, possibilitando a construção de ferramentas mais apelativas e motivadoras.

Face a todo este conjunto de problemas, surgiu a ideia de conceber um sistema educativo capaz de auxiliar o ensino e aprendizagem de conceitos básicos de programação. Como, na nossa opinião, o problema principal reside na incapacidade de os alunos resolverem problemas, apresentarem soluções, ou seja, construírem algoritmos, a aplicação criada terá como preocupação principal fornecer um ambiente onde os alunos não apenas compreendam as diversas fases de um algoritmo já concebido, mas que sobretudo permita que o aluno conceba, teste, experimente, altere e corrija os seus próprios algoritmos.

Como, a nosso ver, os problemas apontados são comuns a qualquer linguagem de programação, a ferramenta em desenvolvimento não irá preocupar-se com a codificação das soluções, mas antes com a possibilidade de permitir que o aluno desenvolva um algoritmo para resolver um problema, ou seja, que o aluno, de alguma forma, descreva um conjunto de acções que quando executadas resultem numa determinada solução. A construção da solução/algoritmo, para os problemas propostos, não será feita à custa de uma especificação verbal, mas antes recorrendo ao uso de ilustrações gráficas, mais apelativas e motivadoras.

O programa permitirá a execução da solução proposta pelo aluno, de modo a que ele verifique de forma gráfica e animada a lógica do algoritmo, os seus passos individuais e o fluxo de execução. Desta forma, mesmo que a solução proposta não satisfaça as exigências do problema em causa, provavelmente o aluno aprenderá mais ao verificar e corrigir o erro lógico do seu raciocínio, do que não intervindo e assistindo apenas ao desenrolar de uma solução correcta. O erro pode ser instrutivo, e os alunos podem aprender tanto com os sucessos quanto com os insucessos.

Como também pensamos que os alunos apresentam diversas formas de raciocínio e aprendizagem, o programa deve permitir diferentes representações alternativas para o mesmo problema.

6. Referências

Anderson, J. R. y Reiser, B. J. (1985). *The LISP Tutor*. Byte. April.

Baecker, R. M. y Marcus, A. (1986). *Design Principles for enhanced presentation of computer program*. In Proc. of the Human Factors in Computing Systems (CHI'88). Pp. 51-58. New York.

Benford, S.D. y Burke, E. K. (1993). *Ceilidh: a course administration and marking system*. In Proc. of the International Conference on Computer-Based Learning in Science (CBLIS'93). pp. 364-372. Vienna.

Bently, J. L. y Kerninghan, B. W. (1991). *A system for algorithm animation*. Computing Systems, Vol 4 No. 1.

Brown, M. y Sedgewick, R. (1985).*Techniques for algorithm animation*. IEEE Software. Vol 2 No. 1.

Brown, M. (1988). *Exploring algorithms using BALSA-II*. IEEE Computer. Vol 21 No. 5.

Brown, M. (1991). *ZEUS: A System for algorithm animation and multi-view editing*. In Proc. of the 1991 IEEE Workshop on Visual Languages. Pp. 4-9. Japan.

Brusilovsky, P. L. (1991). *Turingal - the language for teaching the principles of programming*. In: Proc. of Third European Logo Conference. Pp. 423-432. Parma.

Brusilovsky, P. (1994). *Program visualization as a debugging tool for novices*. In: Proc. of INTERCHI'93 .Pp. 29-30. Amsterdam.

Calabrese, E. (1989). *Marta – The "intelligent turtle"*. In Proc. of the Second European Logo Conference. Pp. 111-127. Ghent. Belgica.

Calloni, B. A. y Bagert, D. J. (1993). *BACCII: An iconic syntax-directed system for teaching procedural programming*. In: Proc. of the 31st ACM Southeast Conference. Pp. 177-183. Birmingham.

Calloni, B. A., Bagert, D. J.y Haiduk, H. (1996). *Iconic vs. text-based programming in the introductory programming sequence*. In Proc. of the ASEE Annual Conference. (CD-ROM 6pp.). Washington DC.

Eisenstadt, M. y Brayshow, M. (1988). *The Transparent Prolog Machine: an execution model and graphical debugger for logic programming*. Journal of Logic programming. Vol 5 No. 4.

Ellis, G. P. y Lund, G. R. (1994). *G2- A design language to help novice C programmers*. Disponível em http://www2.ulst.ac.uk/misc/cticomp/gpellis.html.

Fenton, J. y Beck, K. (1989). *Playground: An object-priented simulation system with agent rules for children of all ages*. In Proc. of OOPSLA'89 Object-Oriented Programming: Systems, Languages, Applications. Pp. 123-137. New Orleans.

Fix, V. y Wiedenbeck, S. (1996). *An intelligent tool to aid students in learning second and subsequent programming languages*. Computers & Education. Vol 27 No. 2.

Hanciles, B., Shankararaman, V. y Munoz, J. (1997). *Multiple representation for understanding data structures*. Computers & Education. Vol 29 No. 1.

Hvorecky, J. (1992). *Karel the Robot for PC*. In Proc. of the East-West Conference on Emerging Computer Technologies in Education. Pp. 157-160. Moscow.

Ingargiola, G, Hoskin, N., Aiken, Duby, R., Wilson, J., Papalaskari, M, Christensen, M., y Webster, R. (1994*). A repository that supports teaching and cooperation in the introductory AI course*. In Proc. of the 1994 ACM SIGCSE Technical Symposium.

Jehng, J., Shih, Y., Liang, S. y Chan, T. (1994). *Turtle-Graph: A computer Supported Cooperative learning environment*. In: Proc. of the ED-MEDIA'94. Pp. 293-298.

Johnson, W. L. y Soloway, E.. (1985). *PROUST: An automatic debugger for Pascal programs*. BYTE. Abril.

Kann, C., Lindeman, R. y Heller, R. (1997). *Integrating Algorithm Animation into a Learning Environment*. Computers & Education. Vol 28 No. 4.

Kay, J. y Tyler, P. (1993). *A microworld for developing learning design strategies*. Computer Science Education. Vol 3 No. 1.

Lawrence, A., Badre, A., y Stasko, J. (1994). *Empirically Evaluating the Use of Animations to Teach Algorithms*. In: Proc. of the 1994 IEEE Symposium on Visual Languages. Pp. 48-54. St. Louis.

Mendes, A. J. Mendes, T. (1988). *VIP – A Tool to VIsualize programming exemples*. In Proc. Education and Aplication of Computer Technology. Pp 131-140. Malta.

Miller, P. y Chandhok, R. (1989). *The design and implementation of the Pascal Genie*. In Proc. of the ACM Computer Science Conference. Louisville, KY.

Miller, P., Pane, J., Meter, G. y Vorthmann, S. R. (1994). *Evolution of novice programming environments: The structure editors of Carnegie Mellon University*. Interactive Learning Environments. Vol 4 No. 2.

Naps, T. L. (1990). *Algorithm visualization in computer science laboratories*. In Proc. of the ACM SIGCSE Technical Symposium. Pp. 105-110.

Olimpo, G., Persico, D., Sarti, L. y Tavella, M. (1985). *An experiment in introducing the basic concepts of informatics*. In Proc. of the Fourth World Conference on Computers in Education, WCCE'85. Pp. 31-38. Amsterdam.

Olimpo, G. (1988). *The Robot Brothers: An environment for learning parallel programming oriented to computer education*. Computers & Education. Vol 12 No 1.

Papert, S. (1980). *Mindstorms, children, computers and powerful ideias*. Basic Books. New York.

Pattis, R. (1981). *Karel the Robot: A gentle introduction to the art of programming*. John Wiley & Sons.

Ramani, K. V. y Rao, T. P. (1994). *A graphics based computer-aided learning package for integer programming: the branch and bound algorithm*. Computers & Education. No. 23.

Roman, G.-C., Cox, K., Wilcox, D. y Plun, J. (1992). *Pavane: a system for declarative visualization of concurrent computations*. Journal of Visual Languages and Computing. Vol 3 No. 2.

Selker, T. (1994). *COACH: A teaching Agent that learns*. Comunications of the ACM. Vol 37 No. 7.

Sellman, R. (1992). *Gravitas: An object-oriented discovery learning environment for Newtonian graviattion*. In Proc. of the East-West International Conference on Human-Computer Interaction. Pp. 31-41. St. Petersburg.

Shih, Y. y Alessi, S. (1994). *Mental Models and transfer of learning in computer programming*. Journal of Research on Computing in Education. No. 26.

Smith, D. C., Cypher, A. y Spohrer, J. (1994). *Kidsim: programming agents without a programming language*. Comunications of the Association for Computing Machinery. Vol 37 No. 7.

Song, J. S., Hahn, S. H., Tak, K. Y. y Kim, J. H. (1997). *An intelligent tutoring system for introductory C language course*. Computers & Education. Vol 28 No. 2.

Stasko, J. (1990). *TANGO: A framework and system for algorithm animation*. IEEE Computer. Vol 23 No. 9.

Stasko, J. (1992). *Animating algorithms with XTANGO*. SIGACT News. Vol 23 No2.

Stasko, J, y Kraemer, E. (1993). *A methodology for building application specific visualizations of parallel programs*. Journal of Parallel and Distributed Computing. No 18.

Stasko, J., Badre, A. y Lewis, C. (1993). *Do algorithm animations assist learning? An empirical study and analysis*. In: Proc. of the INTERCHI'93 Conference on Human Factors in Computing Systems. Pp. 61-66. Amsterdam, Netherlands.

Stasko, J. y McCrickard, D. (1995). *Real clock time animation support for developing software visualizations*. Australian Computer Journal. Vol 27 No. 3.

Tomek, I. (1982). *Josef the Robot*. Computers & Education, Vol 6 No 3.

Tomek, I., Muldner, T. y Khan, S. (1985). *PMS – A program to make learning Pascal easier*. Computers & Education. Vol 9 No. 4.

Tyerman, S., Woods, P. y Warren, J. (1996). *Loop Tutor and HyperTutor: experiences with Adaptative Tutoring Systems*. In Proc. of ANZIIS'96. Adelaide.

Whale, G. (1994). *DRUIDS: tools for understanding data structures and algorithms*. In Proc. of the 1994 IEEE First International Conference on Multi-media Engineeing Education. Pp. 403-407. New York.