UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE UMA LINGUAGEM DE PROGRAMAÇÃO DE COMPUTADORES ORIENTADA POR FORMAS GEOMÉTRICAS, VOLTADA AO ENSINO DE PROGRAMAÇÃO

OSCAR ALCÂNTARA JÚNIOR

BLUMENAU
2003

2003/2-32

OSCAR ALCÂNTARA JÚNIOR

PROTÓTIPO DE UMA LINGUAGEM DE PROGRAMAÇÃO DE COMPUTADORES ORIENTADA POR FORMAS GEOMÉTRICAS, VOLTADA AO ENSINO DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. José Roque Voltolini da Silva- Orientador

BLUMENAU
2003

2003/2-32

PROTÓTIPO DE UMA LINGUAGEM DE PROGRAMAÇÃO DE COMPUTADORES ORIENTADA POR FORMAS GEOMÉTRICAS, VOLTADA AO ENSINO DE PROGRAMAÇÃO

Por

OSCAR ALCÂNTARA JÚNIOR

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. José Roque Voltolini da Silva, FURB
Membro:	Prof. Joyce Martins
Membro:	Prof. Carlos Eduardo Negrão Bizzotto

Blumenau, 15 de Dezembro de 2003.

AGRADECIMENTO

A Deus que meu deu forças para continuar esta batalha e saúde e por sempre estar ao nosso lado.

A meus pais queridos, que ao longo de meus estudos sempre apoiaram e incentivaram a continuar e não desistir.

A minha namorada, pela sua compreensão e ajuda durante este longo período que decorreu o trabalho.

Ao professor, amigo e orientador José Roque Voltolini da Silva, por sua persistência, conhecimento e ajuda nos momentos que precisei.

Agradeço também a Júlio Tôrres, que através do meu orientador, sugeriu o uso do Tangram no ambiente implementado.

Obrigado a todos que me ajudaram e me acompanharam, neste longo período do curso de Ciências da Computação (BACHARELADO).

RESUMO

Este trabalho descreve a implementação de um protótipo de um ambiente de programação visual, com o intuito de facilitar o ensino de programação de computadores. Este ambiente permite que o usuário crie desenhos com os 7 tipos de figuras geométricas que compõe o jogo matemático TANGRAM. Parte da implementação do protótipo é uma linguagem de programação, onde cada comando interage com as figuras adicionadas pelo usuário.

Palavras chaves: Ambiente de Programação; Linguagem de Programação; Ensino de Programação; Programação Visual.

ABSTRACT

This work describes the implementation of a prototype of an ambient of visual programming, with the intention of facilitating the teaching of programming of computers. This ambient allows the user to create drawings with the 7 types of geometric illustrations that composes the mathematical game TANGRAM. Part of the implementation of the prototype is a programming language, where each command interacts with the illustrations added by the user.

Key words: Ambient of Programming; Programming Language; Teaching of Programming; Visual Programming.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Atualmente muito se tem discutido sobre o uso dos computadores em escolas e principalmente por crianças. A utilização da informática como instrumento de aprendizagem vem aumentando de forma rápida. Nesse sentido, a educação vem passando por mudanças estruturais e funcionais. Com estas novas mudanças alguns questionamentos são mencionados em Rosemeire (1998), entre os quais destacam-se:

a) computador pode ajudar a criança a entender a realidade na qual está inserida ou, ao invés disto, se tornará um instrumento de fuga, alienação?;
b) "Poderá o computador ajudar uma criança a se comunicar e expressar melhor, sendo ele uma máquina e não uma pessoa? De que forma?";
c) com a estrutura sócia-econômica do Brasil será possível introduzir máquina tão sofisticada em escolas? De onde os recursos virão?;
d) "Como preparar os professores para usá-los, quebrando as barreiras com o ensino tradicional?";
e) será que o computador pode ser usado por crianças com dificuldade de aprendizagem e excepcionais?.

Estas são algumas das perguntas entre várias que estão surgindo. Alguns destes questionamentos aos poucos estão sendo respondidos, à medida que as experiências vão difundindo-se e mostrando a versatilidade da informática. Ainda, profissionais da área de informática estão criando condições favoráveis para ajudar no desenvolvimento da criança, através de softwares especializados.

Visto o acima descrito, este trabalho sugere a criação de um ambiente que facilite o aprendizado de programação de computadores por uma criança alfabetizada. O ambiente será composto por um editor de formas geométricas pré-definidas. À medida que as formas geométricas forem utilizadas para construir desenhos, as mesmas serão convertidas para uma forma textual (em uma linguagem de programação). Uma linguagem com um paradigma parecido, que se chama *LOGO* (VALENTE, 1988), já existe. Esta linguagem tem um ambiente gráfico onde existe um desenho de uma tartaruga. O usuário informa comandos para que a tartaruga mova-se para direções diferentes na tela e, para cada movimento, um rastro é deixado, objetivando formar um desenho. Este trabalho difere do *LOGO*, visto que o usuário constrói o desenho com formas geométricas pré-definidas, e à medida que ele vai

desenhando, o código da linguagem vai sendo gerado. Este código poderá ser diretamente alterado, surtindo efeito no desenho.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho de conclusão de curso é desenvolver um protótipo de software para auxiliar na introdução do ensino de programação de computadores para crianças alfabetizadas.

Os objetivos específicos do trabalho são:

a) criar um ambiente gráfico para a criança desenhar formas geométricas (desenhos), as quais serão visualizadas estaticamente;
b) criar uma linguagem de interação com a programação gráfica, onde o agente programador poderá fazer alterações no código, e essas alterações surtirão efeito no programa gráfico (desenho);
c) permitir criar movimentos dos objetos geométricos, descrevendo-os através da inclusão de comandos no código previamente gerado a partir do desenho criado;
d) recuperar o que foi desenhado a partir do texto gerado. As formas geométricas serão armazenadas e recuperadas através do texto (linguagem). Cada desenho editado vai gerar um conjunto de comandos no código, formando um programa, o qual poderá ser salvo e depois recuperado para retornar ao desenho original.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta uma introdução sobre ambientes de aprendizagem, linguagens de programação, ambientes visuais, Tangram e computação gráfica 2D. No capítulo 3 serão abordados detalhes sobre o desenvolvimento do protótipo. Por fim, o capítulo 4 traz as considerações finais e sugestões para extensões no trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

O trabalho em questão visa facilitar o ensino de linguagens de programação de computadores. Para isto foi proposto um ambiente visual de programação que auxilie esta tarefa. Para realização e entendimento do trabalho proposto, neste capítulo são discutidos aspectos de ensino através do computador (texto fortemente baseado em Valente (1993)), linguagens voltadas ao ensino de programação de computadores, linguagens de programação, ambiente visual *LOGO*, história do Tangram e suas peças e por fim transformações 2D.

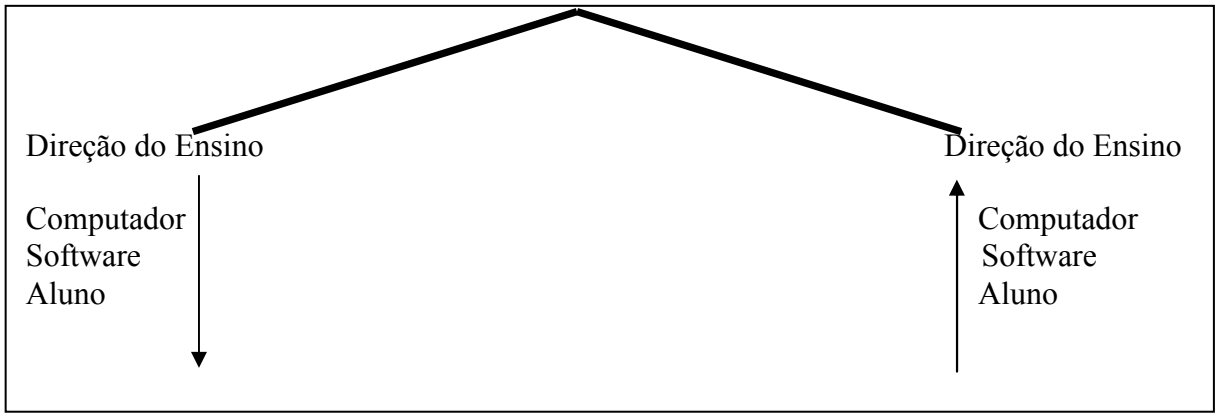
2.1 ENSINO ATRAVÉS DO COMPUTADOR

Segundo Valente (1993), para a implantação do computador na educação são necessários basicamente quatro ingredientes: o computador, o software, o professor capacitado para usar o computador como meio educacional e o aluno.

Na educação o computador tem sido utilizado tanto para ensinar sobre computação – ensino de computação ou **"***computer literacy"* – como para ensinar praticamente qualquer assunto **–** ensino através do computador.

No ensino de computação, o computador é usado como objeto de estudo, ou seja, o aluno usa o computador para adquirir conceitos computacionais, como princípios de funcionamento do computador, noções de programação e implicações sociais do computador na sociedade. Entretanto, a maior parte dos cursos oferecidos nessa modalidade podem ser caracterizados como de "conscientização do estudante para a informática", ao invés de ensiná-lo a programar. Assim, os propósitos são vagos e não determinam o grau de profundidade do conhecimento que o aluno deve ter – até quanto o aluno deve conhecer sobre computadores e técnicas de programação. Isto tem contribuído para tornar esta modalidade de utilização do computador extremamente nebulosa e facilitando a sua utilização como chamariz mercadológico. Certamente esse não é o enfoque da informática educativa, e, portanto, não é a maneira como o computador é usado em ambientes de aprendizagem.

O ensino pelo computador implica que o aluno, com auxilio da máquina, possa adquirir conceitos sobre praticamente qualquer domínio. Entretanto, a abordagem pedagógica de como isso acontece é bastante variada, oscilando entre dois grandes pólos, como pode ser visto na fig. 1.



Fonte: Valente (1993 , p. 2).

FIGURA 1 – Ensino ou aprendizado através do computador

Esses pólos (destacados na fig. 1) são caracterizados pelos mesmos ingredientes: computadores (hardware), o software (o programa de computador que permite a interação homem-computador) e o aluno. Porém, o que estabelece a polaridades é a maneira como esses ingredientes são usados. Num lado, o computador, através do software, ensina o aluno. Enquanto no outro, o aluno, através do software, "ensina" o computador.

Quando o computador ensina o aluno, o computador assume o papel de máquina de ensinar e a abordagem educacional é a instrução auxiliada por computador. Essa abordagem tem suas raízes nos métodos de instruções tradicionais, porém, ao invés do papel ou do livro, é usado o computador. Os softwares que implementam essa abordagem podem ser divididos em duas categorias: tutoriais e exercício-e-prática (*"drill-and-practice"*).

Um outro tipo de software que ensina é dos jogos educacionais e a simulação. Nesse caso, a pedagogia utilizada é a exploração auto dirigida ao invés da instrução explícita e direta. No outro pólo, para o aprendiz "ensinar" o computador, o software é uma linguagem computacional tipo BASIC, LOGO, PASCAL, uma linguagem para criação de banco de dados do tipo DBase ou mesmo um processador de texto, onde o aprendiz pode representar suas idéias através destes softwares. Nesse caso, o computador pode ser visto como uma ferramenta que permite ao aprendiz resolver problemas ou realizar tarefas como desenhar, escrever, comunicar-se.

2.2 LINGUAGENS VOLTADA AO ENSINO PROGRAMAÇÃO

Segundo Baranauskas (1993), a escolha da linguagem de programação para uma aplicação específica, principalmente no caso de contextos educacionais, requer uma atenção particular ao paradigma subjacente à linguagem. O objetivo deste item é situar algumas linguagens de programação usadas em educação, principalmente *Prolog* e *LOGO*.

O Prolog (de *Programming in Logic*) surgiu no início dos anos 70, dos esforços de Robert Kowalski, Maarten van Emden e Alain Colmerauer, e é a linguagem de programação desenvolvida em máquina seqüencial, que mais se aproxima do modelo de computação de programação em lógica (BARANAUSKAS, 1993). O enfoque do paradigma da programação em lógica para se representar um problema a ser resolvido no computador, consiste em expressar o problema na forma de lógica simbólica. Um processo de inferência é usado pela máquina para produzir resultados. Segundo o modelo de programar proposto por Prolog, o significado de um "programa" não é dado por uma sucessão de operações elementares que o computador supostamente realiza, mas por uma base de conhecimento a respeito de certo domínio e por perguntas feitas a essa base de conhecimento, independentemente. Dessa maneira, Prolog pode ser visto como um formalismo para representar conhecimento a respeito do problema que se quer resolver, de forma declarativa (descritiva). Existe por trás do programa uma máquina de inferência, em princípio "escondida" do programador, responsável por "encontrar soluções" para o problema descrito.

O *LOGO* é uma linguagem de programação proposta por Seymour Papert, cujas raízes derivam de *Lisp* (BARANAUSKAS, 1993). A grande contribuição de Papert, para o que é atualmente a linguagem *LOGO*, está na manipulação de um objeto gráfico, chamado "tartaruga", que é capaz de "andar" pela tela deixando seu rastro. Ensinar a tartaruga a fazer (por exemplo, a figura de um quadrado ou uma casinha) é uma metáfora para a atividade de programar, no contexto da tartaruga. Dessa forma, o computador é abstraído na figura da tartaruga. O resultado (rastro de tartaruga) mostrado na tela fornece um *feedback* para a criança, podendo levá-la a reformular o procedimento "ensinado". A idéia está no fato de que no processo de "ensinar" a tartaruga, a criança pode refletir o seu próprio processo de aprender e tomar consciência disso, num certo sentido, faz dela uma "epistemóloga" (PAPERT, 1980, apud BARANAUSKAS, p. 45-63).

2.3 LINGUAGENS DE PROGRAMAÇÃO

Segundo Sebesta (2000, p. 24), uma linguagem de programação deve possuir alguns critérios muito importantes para a sua avaliação. Entre esses critérios, o mais importante deles, está a facilidade de leitura e entendimento do programa. Antes de 1970, o desenvolvimento de software era imaginado em termos de escrita de código. Na década de 70, entretanto, o conceito de ciclo de vida do software (BOOCH, 1987, apud SEBESTA, 2000, p. 24) foi desenvolvido; a codificação foi relegada a um papel muito menos importante e a manutenção foi reconhecida como uma parte importante do ciclo, especialmente em termos de custo. Uma vez que a facilidade de manutenção é determinada, em grande parte, pela legibilidade dos programas, ela tornou-se uma medida muito importante na qualidade dos programas e linguagens.

Outro critério importante é a capacidade escrita que é uma medida de quão facilmente uma linguagem pode ser usada para criar programas para um domínio de problema escolhido. A capacidade de escrita das linguagens COBOL (ANSI, 1985, apud SEBESTA, 2000, p. 24) e a da APL (GILMAN; ROSE, 1976, apud SEBESTA, 2000, p. 24), por exemplo, são drasticamente diferentes para criar um programa capaz de lidar com estruturas de dados bidimensionais, ao qual é mais indica seria APL. Suas capacidades de escrita também são diferentes na produção de relatórios financeiros com formatos complexos, para os quais o COBOL foi projetado.

O terceiro critério é a confiabilidade. Um programa é confiável se ele comporta-se de acordo com suas especificações sob todas as condições. Uma linguagem de programação contribui muito para confiabilidade de algum programa que se queira implementar na suposta linguagem, como por exemplo na verificação de tipos de dados, tratamento de exceções, verificação da sintaxe, entre outros.

O quarto e último critério seria o custo final da linguagem, onde de vários fatores existentes, há três que se destacam:

a) desenvolvimento do programa: quanto mais fácil e rápido se cria o programa na linguagem, menor o custo;
b) manutenção: quanto melhor a legibilidade, menor o custo;
c) confiabilidade: se o software falhar em um sistema crítico, como uma usina nuclear ou uma máquina de raios X, o custo poderia ser muito elevado. As falhas em

sistemas não críticos também podem ser muito caras em termos de futuro comercial da empresa que os projetou.

Ainda, as linguagens de programação mais usadas atualmente são as linguagens seqüências ou imperativas, as quais serão comentadas a seguir. Também um método para especificação das linguagens é descrito na seção 2.3.2.

2.3.1 LINGUANGENS SEQUENCIAIS OU IMPERATIVAS

Segundo Gerber (2000), as linguagens imperativas são orientadas a ações, onde a computação é vista como uma seqüência de instruções que manipulam valores de variáveis (leitura e atribuição). Foram criadas principalmente pela influência da arquitetura de computadores preponderante *Von Neumann*, onde programas e dados são armazenados na mesma memória. Instruções e dados são transmitidos da CPU para a memória, e vice-versa. Resultados das operações executadas na CPU são retornados para a memória. Seus conceitos principais são as "variáveis", que representam simbolicamente células (ou posições) de memória, "declarações de atribuição" (exemplo do Pascal: *x := 10*), baseadas nas operações de transmissão de dados, e a forma iterativa de "declarações de repetição" (exemplo do C: *while*). Operandos em expressões são enviados da memória para a CPU, e o resultado da avaliação dessas expressões é transferido da CPU para a memória, representada pelas variáveis do lado esquerdo de uma declaração de atribuição. Repetição é feita por iteração, porque as instruções em uma arquitetura *Von Neumann* são armazenadas em posições adjacentes de memória, tornando a iteração mais eficiente. Exemplos de linguagens imperativas são o FORTRAN, BASIC, COBOL, Pascal, C, Python, ALGOL e Módula.

2.3.1.1 LINGUAGENS IMPERATIVAS NÃO ESTRUTURADAS

FORTRAN foi uma das primeiras linguagens de alto nível imperativas, destinada inicialmente ao desenvolvimento de aplicações científicas. As versões originais de FORTRAN possuíam a instrução *goto* necessária para determinar a repetição e a seleção de execução de instruções, o que dificultava bastante a leitura e o acompanhamento da execução de um programa escrito em FORTRAN. Assembly e BASIC são outros exemplos de linguagens imperativas que possuem o conceito de *goto*. Uso de *goto* geralmente leva ao que chamam na literatura de código *spaguetti*. Exemplo de código com o uso de instrução *goto*, pode ser visto no quadro 1.

```
read(x);
2: if x = 0 then goto 8;
writeln(x);
4: read(next);
if next = x then goto 4;
x := next;
goto 2;
8: ...;
```

QUADRO 1 – Exemplo de código com o uso de instrução “goto”

2.3.1.2 LINGUAGENS ESTRUTURADAS

Surgiram com o objetivo de facilitar a leitura e acompanhamento da execução de algoritmos. Normalmente linguagens estruturadas não fazem uso de comando *goto*. Instruções são agrupadas em blocos, os quais podem ser considerados como unidades de programa, abstraindo-se das suas estruturas internas. Blocos de instruções podem ser selecionados para execução através de declarações de seleção como *if...else*, ou repetidamente executados através de declarações de repetição como *while*. Linguagens estruturadas *procedurais* permitem a criação de procedimentos (e funções), que são blocos de instruções, que formam os elementos básicos de construção de programas. Procedimentos criam um nível de abstração, onde não é necessário conhecer todos os passos de execução de um procedimento, apenas qual a sua função e quais os seus pré-requisitos para que execute de acordo com o esperado. Linguagens estruturadas *modulares* criam um outro mecanismo de abstração, o módulo, que normalmente é composto de definições de variáveis e procedimentos, que são agrupados de acordo com critérios específicos. Exemplo de código estruturado, baseado na sintaxe do Pascal, correspondente ao mesmo exemplo do quadro 1 pode ser visto no quadro 2.

```
read(x);
while x <> 0 do begin
writeln(x);
repeat
read(next);
until next <> x;
x := next;
end;
```

QUADRO 2 – Exemplo de código estrutura

2.3.2 MÉTODO PARA ESPECIFICAÇÕES DE LINGUANGENS

Segundo Sebesta (2000, p. 115), de meados até o final da década de 50 do século passado, John Backus e Noam Chomsky, em esforços de pesquisas não-relacionados, inventaram a mesma notação, que se tornou, desde então, o método mais usado para descrever

formalmente a sintaxe das linguagens de programação. Chomsky, um famoso lingüista, desenvolveu quatro classes de gramáticas que definem quatro classes de linguagens. Duas dessas classes gramaticais denominadas livre de contexto e comum (linguagens regulares), passaram a ser úteis para descrever linguagens de programação. Linguagens inteiras, com algumas exceções, podem ser descritas pela gramática livre contexto. Pouco depois do trabalho de Chomsky sobre classes de linguagens, foi apresentado por Jonh Backus um documento decisivo descrevendo o ALGOL 58. Esse documento introduziu uma nova notação formal para especificar a sintaxe das linguagens de programação. Esse método de descrição da sintaxe tornou-se conhecido como forma de Backus-Naur, ou simplesmente BNF.

BNF é uma metalinguagem que é utilizada até hoje para especificação de linguagens de programação. Uma metalinguagem é uma linguagem usada para descrever uma outra linguagem. A BNF usa abstrações para estruturas sintáticas. Uma simples instrução de atribuição em C, por exemplo, poderia ser representada pela abstração <atribuição> (colchetes angulados freqüentemente são usados para delimitar nomes de abstrações). A definição de <atribuição> é mostrada no quadro 3.

<atribuição> → <var> = <expressão>

QUADRO 3 – Exemplo definição da regra de "atribuição"

O símbolo à esquerda da seta, adequadamente chamado de *left-hand side (LHS)* – lado esquerdo, é a abstração que está sendo definida (símbolo não terminal). O texto a direita da seta é a definição do LHS. Ele é chamado do *rigth-hand side (RHS)* – lado direito – e consiste em uma mistura de *tokens*, de *lexemas* de referências a outras abstrações (de fato, *tokens* também são abstrações), chamados também de terminais e não terminais.

Token de uma linguagem é uma categoria de seus lexemas por exemplo, o *token* identificador é um *token* que pode conter lexemas ou instâncias, um exemplo poderia ser "sum" e "total". Em alguns casos, um *token* tem somente um único lexema possível. Por exemplo, o *token* para o símbolo de operador aritmético "+", que pode ter o nome soma_op, tem apenas um lexema possível.

Ao todo, a definição é chamada regra ou produção. No exemplo da regra que foi apresentada no quadro 3, as abstrações <var> e <expressão> evidentemente devem ser definidas antes que a definição <atribuição> torne-se útil. Essa regra particular especifica que

a abstração <atribuição> é definida com uma instância da abstração <var> seguida do lexema =, seguido de uma instância da abstração <expressão>. Um exemplo de sentença cuja estrutura sintática é descrita pela regra de <atribuição> vista no quadro 3 é apresentada no quadro 4.

```
total = sub1 + sub2
```

QUADRO 4 – Exemplo sentença definida na regra "atribuição"

As abstrações em uma descrição ou em uma gramática BNF freqüentemente são chamadas símbolos não-terminais, ou simplesmente não-terminais, e os *lexemas* e os *tokens* das regras são chamados símbolos terminais, ou simplesmente terminais. Uma descrição BNF ou gramática é simplesmente um conjunto de regras.

Os símbolos não-terminais podem ter duas ou mais definições distintas, representando, duas ou mais formas sintáticas possíveis na linguagem. Múltiplas definições podem ser escritas com uma única regra, com duas definições diferentes separadas pelo símbolo |, que significa o OU lógico. Um exemplo utilizando estas regras é mostrado no quadro 5.

```
<if_stmt> → if <expr_lógica> then <stmt>
<if_stmt> → if <expr_lógica> then <stmt> else <stmt>


<if_stmt> → if <expr_lógica> then <stmt>
          | if <expr_lógica> then <stmt> else <stmt>
```

QUADRO 5 –Definição da regra "if"

Segundo Sebesta (2000, p. 115), BNF é suficientemente poderosa para descrever a grande maioria das sintaxes das linguagens de programação, mesmo sendo simples. Em particular, ela pode descrever listas de construções similares à ordem em que diferentes construções devem aparecer nas estruturas alinhadas em qualquer profundidade, a precedência de operadores e a associatividade de operadores.

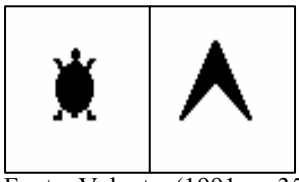
2.4 AMBIENTE VISUAL (LOGO)

Segundo Valente (1991), o *LOGO* foi desenvolvido por volta de 1968. Conta-se que a idéia surgiu durante um jantar onde estavam Seymour Papert, Wallace Feurzeig (diretor do grupo de Tecnologia Educacional da Bolt, Berananek e Newman – BBN), Cynthia Solomon (pesquisadora pertencente a BBN) e Daniel Bobrow (na época, estudante de pós-graduação do Massachusetts Institute of Technology (MIT)). Neste jantar, alguém propôs a criação de uma

linguagem de programação que fosse bastante poderosa e capaz de substituir o BASIC. Desta idéia nasceu o *LOGO* (pronunciado como logotipo), uma linguagem com capacidade de processar listas e de permitir a criação de novos procedimentos – características que o BASIC ainda não dispunha. Entretanto, nesta época o *LOGO* não dispunha de capacidade gráfica, já que os computadores de então não possuíam esta facilidade. Através da sua utilização e inúmeras pesquisas, Seymour Papert conseguiu dar aquele *LOGO* uma nova roupagem e uma estrutura filosófica, sendo por isto considerado hoje o pai do *LOGO*. As idéias de Papert são apresentadas em Papert (1980 apud VALENTE, 1991, p. 32-43).

O *LOGO* tem assim, duas raízes, uma computacional e outra filosófica. Do ponto de vista computacional, as características do *LOGO* que contribuem para que ele seja uma linguagem de programação de fácil assimilação são: exploração de atividades espaciais, fácil terminologia e a capacidade de se criar novos termos ou procedimentos. A exploração de atividades espaciais tem sido a porta de entrada do *LOGO*. Estas atividades permitiam o contato quase que imediato do aprendiz com o computador. Estas atividades espaciais facilitam muito a compreensão da filosofia pedagógica do *LOGO* por parte dos especialistas em computação. Por outro lado, elas fazem com que os aspectos computacionais da linguagem de programação *LOGO* sejam acessíveis aos especialistas em educação. Assim, o aspecto será usado neste capítulo com finalidade de apresentar-se à filosofia *LOGO*. Entretanto, é importante lembrar que o *LOGO*, como linguagem de programação, tem outras características mais avançadas, como já foi mencionado. Com as atividades espaciais a proposta é utilizar esses conceitos nas atividades de comandar uma tartaruga mecânica a se mover no espaço ou atividades de desenhar na tela do computador (atividades gráficas). Isto se deve ao fato dessas atividades envolverem conceitos espaciais que são adquiridos nos primórdios da nossa infância, quando se começa a engatinhar. Entretanto, estes conceitos permanecem em nível intuitivo. Por exemplo, a criança aprende, sem grande dificuldade, a ir da sua casa à padaria. Esta atividade é desenvolvida sem ela se dar conta que está usando conceitos como distância, ângulo reto para virar esquinas, etc. A proposta da atividade gráfica do *LOGO* é utilizar estes conceitos nas atividades de comandar a tartaruga. No processo de comandar a tartaruga para ir de um ponto ao outro, estes conceitos devem ser explicitados. Isto fornece as condições para o desenvolvimento de conceitos espaciais, numéricos, geométricos, uma vez que a criança pode exercitá-los, depurá-los, e utilizá-los em diferentes situações.

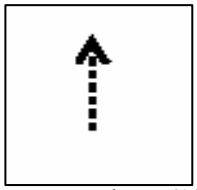
A tartaruga mecânica é um objeto que se desloca no chão, comandado pelo computador. Este brinquedo comandado pelo computador se desloca de maneira muito lenta, daí a analogia com uma tartaruga. Entretanto mais do que um brinquedo ela tem uma finalidade muito importante no *LOGO*. A tartaruga mecânica ou tartaruga de solo, como chamava Valente (1991), surgiu quando o *LOGO* foi desenvolvido, e era o meio utilizado para introduzir a linguagem a todos as pessoas – criança, adulto, deficiente físico, superdotado. Ela foi desenvolvida para facilitar a associação do movimento da tartaruga com o movimento que uma pessoa faz com o corpo quando no deslocamento no espaço. Assim, pode-se utilizar o "conhecimento do corpo" e tentar formalizá-lo através de comandos ou programas de computadores que movimentam a tartaruga. Assim, o papel da tartaruga é o de ser um objeto de transição, um objeto onde se espelha o conhecimento de deslocamento espacial. A mesma função da tartaruga de solo foi transferida para a tartaruga de tela. Na tela do computador a tartaruga é representada por um triângulo ou por um desenho de uma tartaruga como pode ser visto na fig. 2.



Fonte: Valente (1991, p. 35)

FIGURA 2 – Representação de triângulo e tartaruga

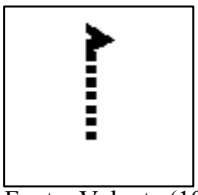
A tartaruga da tela é utilizada para a realização de atividades gráficas de grande precisão, já que é comum a tartaruga mecânica apresentar alguns problemas de ordem mecânico, como por exemplo, o de ser comandada para traçar um quadrado e este não se fechar. Os termos da linguagem *LOGO*, ou seja os comandos do *LOGO*, que a criança usa para comandar a tartaruga (tanto de solo como a da tela) são termos que a criança usa no seu dia-a-dia. Por exemplo, para comandar a tartaruga para se deslocar para frente o comando é "parafrente". Assim, "parafrente 50" desloca a tartaruga para frente 50 passos do ponto em que ela estava inicialmente, como pode ser visto na fig. 3.



Fonte: Valente (1991, p. 35)

FIGURA 3 – Deslocamento da tartaruga

Para comandar a tartaruga para girar para direita 90 graus o comando é "paradireita 90", produzindo o efeito que pode ser visto na fig. 4.



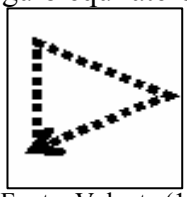
Fonte: Valente (1991, p. 36)

FIGURA 4 – Girar tartaruga

Os comandos que movimentam a tartaruga podem ser utilizados numa série de atividades que a criança pode realizar. Por exemplo, explorar o tamanho da tela ou realizar uma atividade simples, como o desenho de um triângulo eqüilátero. Para desenhar um triângulo eqüilátero cujos lados tem 30 passos da tartaruga, primeiramente desloca-se a tartaruga para frente 30 passos. Depois gira-se a tartaruga. Nota-se que a tartaruga gira o ângulo externo. Portanto, neste caso, deve girar 120 graus e não o ângulo interno 60. Assim, os comandos para desenhar um triângulo eqüilátero podem ser vistos no quadro 6 e na fig. 5.

```
parafrente 30
paradireita 120
parafrente 30
paradireita 120
parafrente 30
```

QUADRO 6 – Triângulo eqüilátero criado com “Logo”



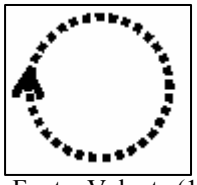
Fonte: Valente (1991, p. 36)

FIGURA 5 – Resultado dos comandos vistos no quadro 6

Nota-se que mesmo nesta atividade simples existem muitos conceitos envolvidos. Por exemplo, conceito como lateralidade, seqüência lógica de instruções e conceitos matemáticos como a diferença entre os números que deslocam e giram a tartaruga, conceito de distância, conceito de ângulo interno e externo. Todos estes conceitos, no ensino tradicional são extremamente abstratos, enquanto que no *LOGO* eles se tornam palpáveis e concretos. A criança pode experimentar o uso destes conceitos em algo prático.

O objetivo da terminologia simples e a facilidade com que os comandos podem ser juntados, é o de facilitar a interação da criança com o computador, a assimilação destes termos pela criança, bem como permitir que ela rapidamente desenvolva atividades

computacionais. Além disto, os termos sendo semelhantes ao linguajar das crianças, quando são usados por *LOGO* para dar ordens ou mesmo para descrever ações no espaço, facilita a descrição das atividades que a criança elabora através do computador. Por exemplo, para desenhar uma circunferência com a Tartaruga pode-se proceder de diversas maneiras: primeiro pode-se solicitar à criança que ande em círculo. Depois que descreva o que está fazendo. Em seguida, que descreva o seu comportamento em termos dos comandos "parafrente" e "paradireita", que são os comandos que a tartaruga entende. Quando isto é feito, é muito comum à criança deduzir que para frente um pouquinho e girar para direita um pouquinho produz uma circunferência. Esta idéia pode ser testada no computador. Eventualmente a criança acaba encontrando o número de passos e o número de graus que devem ser utilizados, bem como o número de vezes que estes comandos devem ser repetidos. E não é difícil a criança notar que "repita 360[parafrente 1 paradireita 1]" produz uma circunferência, como pode ser vista na fig. 6. Mas, a fig. 6 não é uma circunferência. Parece com uma circunferência, mas na verdade é um polígono de 360 lados.



Fonte: Valente (1991, p. 37)

FIGURA 6 – Circunferência em "Logo"

A tartaruga faz 360 vezes "parafrente 1". Então, cada 1 corresponde a um lado do polígono. Assim, esta figura na verdade é um polígono de 360 lados. Entretanto, ao invés deste fato ser um problema, pode-se explorá-lo. O que é uma circunferência? É um polígono cujo comprimento de cada lado tende para 0. Este é o conceito de circunferência segundo uma idéia de limite de cálculo diferencial. Isto serve para ilustrar que o *LOGO* não é apenas uma brincadeira de criança fazer desenho de casinhas, quadradinhos e triângulos. Ele se presta também para explorar conceitos altamente complicados dentro da matemática. Basta que o professor esteja preparado para explorar as diferentes situações que o aluno encontra ao desenvolver atividades com o *LOGO*.

Uma outra característica importante da linguagem *LOGO* é o fato dela ser uma linguagem procedural. Isto significa que é extremamente fácil criar novos termos ou procedimentos em *LOGO*. Esta característica tenta imitar a capacidade que as pessoas tem quando aprendem algo – quando uma habilidade ou conceito é dominado não se precisa pensar sobre o mesmo, ele é simplesmente utilizado em tarefas mais complicadas. Por

exemplo, andar de bicicleta. Quando aprende-se a andar de bicicleta, tem-se que prestar atenção no pedal, posição do corpo, equilíbrio. Mas uma vez que se aprende, nem se lembra mais destes fatos. Simplesmente anda-se de bicicleta. E ainda, tem-se a capacidade de fazer coisas em paralelo como, contar, olhar para os lados. Em *LOGO* isto se traduz em ter-se que atribuir um nome a um conjunto de comandos que implementa uma determinada atividade. Este conjunto de comandos pode ser utilizado em outras tarefas simplesmente fazendo referência ao nome atribuído ao conjunto. Esta facilidade permite às crianças criar os seus próprios termos e expandir a capacidade da linguagem. Depois de algum tempo usando o *LOGO*, a criança terá uma linguagem de comunicação com o computador que é bem individual, utilizando termos que têm um significado próprio, permitindo que esta linguagem expresse a sua maneira de pensar. Assim, para programar-se o computador para fazer um triângulo, a metáfora que usa-se com as crianças é a de "ensinar a tartaruga" a fazer um triângulo. Portanto, usa-se o comando aprenda e fornece-se um nome ao conjunto de comandos que produz o triângulo. Este nome pode ser qualquer nome, por exemplo, triângulo, "tri", Maria, etc. Assim o quadro 7, define o que é um "tri". Uma vez esta definição determinada, o computador indica que "aprendeu" "tri".

```
aprenda tri
para frente 50
paradireita 120
parafrente 50
paradireita 120
parafrente 50
Fim
```

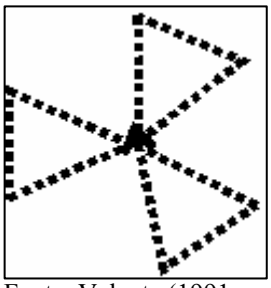
QUADRO 7 – Exemplo criando procedimento "tri" em "Logo"

Agora ao digitar "tri", o computador produz o triângulo. E, assim, pode-se usar o comando "tri" com um outro comando do *LOGO*. No quadro 8 foi criado mais um novo procedimento, "aprenda flor", que utiliza o procedimento criado anteriormente no quadro 7.

```
aprenda flor
repita 3 [tri]
fim
```

QUADRO 8 – Exemplo utilizando o procedimento "tri" dentro do "aprenda flor"

O resultado do "aprenda flor" (descrito no quadro 8) pode ser visto na fig. 7.



Fonte: Valente (1991, p. 38)

FIGURA 7 – Resultado do “aprenda flor”

À medida que a criança explora os comandos do *LOGO*, ela começa a ter idéias de projetos para serem desenvolvidos na tela. Ela pode propor para fazer o desenho de uma casa, de um vaso, etc. Neste instante a metodologia ou filosofia *LOGO* começa a se materializar.

Além dos comandos de manipulação da tartaruga, a linguagem *LOGO* dispõe também de comandos que permitem a manipulação de palavras e listas (um conjunto de palavras). Com estes comandos é possível “ensinar” a tartaruga a produzir uma frase da Língua Portuguesa, usar os conceitos de concordância verbal, criar poemas, e mesmo integrar a parte gráfica com a manipulação de palavras para produzir estórias onde os personagens são animados, um verdadeiro teatro, com as narrativas, cenários, etc. Um outro domínio de exploração pode ser música. Existem comandos que permitem a emissão de som, o que torna possível a criação de canções e o uso de conceitos musicais. Do mesmo modo, *LOGO* foi expandido para incluir comandos para desenhar figuras tridimensionais (REGGINI, 1985, apud VALENTE, 1991, p. 32-73); ou no caso do LEGO-Logo, para comandar sensores e motores, controlando objetos LEGO (RESNICK; OCKO; PAPERT, 1988 apud VALENTE, 1991, p. 32-73).

Os domínios de aplicações do *LOGO* estão em permanente desenvolvimento, com o objetivo de atrair um maior número de usuários e motivar os alunos a usarem o computador para elaborar as mais diferentes atividades. Entretanto, o objetivo não deve ser concentrado no produto que o aluno desenvolve, mas na filosofia de uso do computador e como ele está facilitando a assimilação de conceitos que permitem as diversas atividades. Portanto, é o processo de ensino-aprendizagem que é o cerne do *LOGO* e é este que deve ser discutido e explicitado. O aspecto filosófico do *LOGO* está fundamentado no grande interesse que Papert tem pelo processo de aprendizagem. Desde muito jovem, aprender tornou-se seu principal *hobby*. Este interesse tem sido a ênfase da vida de Papert como pesquisador. Durante o início dos anos 60 ele estudou com Jean Piaget no Centro de Epistemologia Genética e mais tarde, já

no Laboratório de Inteligência Artificial do MIT, pôde desenvolver as idéias principais do "pacote filosófico" *LOGO*, que incorpora aspectos das idéias piagetianas.

Piaget mostrou que, desde os primeiros anos de vida, a criança já tem mecanismos de aprendizagem que ela desenvolve sem mesmo ter freqüentado a escola. A criança aprende diversos conceitos matemáticos; por exemplo, a idéia que em um copo alto e estreito pode ser colocado a mesma quantidade de líquido do que em um copo mais gordo e mais baixo. Isso ela aprende utilizando copos de diferentes tamanhos. E com isso ela desenvolve o conceito de volume sem ser explicitamente ensinada. Assim, Piaget concluiu que a criança desenvolve a sua capacidade intelectual interagindo com objetos do ambiente onde ela vive e utilizando o seu mecanismo de aprendizagem. E isto sem que a criança seja explicitamente ensinada. É claro que outros conceitos também podem ser adquiridos através do mesmo processo.

É justamente este aspecto do processo de aprendizagem que o *LOGO* pretende resgatar: um ambiente de aprendizagem onde o conhecimento não é passado para a criança, mas onde a criança interagindo com os objetos desse ambiente, possa desenvolver outros conceitos, por exemplo, conceitos geométricos. Entretanto, o objeto com o qual a criança interage deve tornar manipulável estes conceitos, do mesmo modo que manipulando copos ela adquire idéias de volume. E isto é conseguido com o computador, através do *LOGO*.

É importante notar que a manipulação dos objetos permite a aquisição de idéias intuitivas a respeito de um determinado conceito. Neste nível de aprendizagem a criança não dispõe de argumentos formais para justificar um tipo de comportamento: por exemplo, que um copo retém mais líquido por que seu volume geométrico é maior, e mostrar que isto é verdade através de experimentos comprobatórios ou através de comparação de fórmulas matemáticas dos volumes dos respectivos copos. Isto somente acontece quando a criança já domina a representação formal, conceitos de álgebra, etc.

Entretanto, este formalismo não é nada acessível à criança pelo fato dela não dispor de objetos que funcionam como os copos funcionam no processo de aquisição de idéias intuitivas sobre volume.

A tartaruga do *LOGO* pode ser vista como sendo este objeto que torna manipulável as formalizações geométricas. Por exemplo, para deslocar a tartaruga é necessário um comando onde é explicitado o número de passos e para girá-la é necessário explicitar o ângulo. Isto permite a aquisição de idéias intuitivas sobre espaço. Entretanto, estas idéias intuitivas estão

sendo expressas segundo a linguagem que é altamente formal. O conjunto de comandos para deslocar a tartaruga é uma descrição formal do seu comportamento. Sempre que a tartaruga estiver numa situação inicial específica e for dado este conjunto de comandos ela terminará na mesma situação final. Assim, o conjunto de comandos é não ambíguo – não permite outros tipos de interpretações. Isto faz com que esta descrição do comportamento da tartaruga tenha todas as qualidades de uma descrição matemática, porém não parece com fórmulas matemáticas. Portanto, através da manipulação da tartaruga começa a desenvolver as noções de formalismo e a necessidade de ser preciso e não ambíguo nas descrições das soluções de problemas. E isto acontece como fruto da necessidade de se comunicar com a tartaruga. É assim que a tartaruga se comporta, é assim que se deve ser feito pra ela "entender" as ordens que lhe são dadas. Portanto, a tartaruga está para as noções e o formalismo geométrico, como os copos estão para as idéias intuitivas de volume.

Um outro aspecto importante da linguagem *LOGO* é que o controle do processo de aprendizagem está na mão do aprendiz e não nas mãos do professor. Isto porque a criança tem a chance de explorar o objeto "computador" da sua maneira e não de uma maneira já pré-estabelecida pelo professor. É a criança que propõe os problemas ou projetos a serem desenvolvidos através do *LOGO*. Estes são os projetos que a criança está interessada em resolver. É claro que o professor tem um papel importante a desempenhar. Por exemplo, propor mudanças no projeto para ajustá-lo ao nível da capacidade da criança, fornecer novas informações, explorar e elaborar os conteúdos embutidos nas atividades, etc. Claro que tudo sem destruir o interesse e a motivação do aprendiz.

Entretanto, o fato do processo de ensino-aprendizagem do *LOGO* ser baseado na resolução de problemas faz com que esta metodologia seja diametralmente oposta à metodologia de ensino baseada em cumprimento de pré-requisitos. No ambiente *LOGO* a criança aprende porque tem a necessidade e o interesse em aprender um conceito que vai ser imediatamente utilizado no seu projeto.

Um outro aspecto fundamental do *LOGO* é o fato de propiciar à criança a chance de aprender com os seus próprios erros. E na atividade de programar o computador existem basicamente dois tipos de erro: o erro sintático do comando e o erro conceitual. No caso do erro sintático, se o comando é erradamente escrito, como por exemplo, "pratrás70", o computador fornece uma mensagem que não entende o comando fornecido. E o comando correto deve ser reescrito. Numa outra situação de aprendizagem, como na tradicional, a

criança seria punida pelo fato de ter cometido um erro. Mas no *LOGO* não acontece nada. Se o comando está sintaticamente errado, basta fornecê-lo outra vez, por exemplo, "paratrás70" e a tartaruga se desloca para trás.

A situação de erro mais interessante do ponto de vista de aprendizagem é o erro conceitual. O programa que a criança define pode ser uma descrição do seu processo de pensamento. Isso significa que existe uma proposta de solução do problema em nível de idéia de programa. Isto permite a comparação da intenção com a atual implementação da resolução do problema no computador. Se o programa não produz o esperado, significa que ele está conceitualmente errado. A análise constitui uma grande oportunidade para a criança entender o conceito envolvido na resolução do problema em questão.

Portanto, no *LOGO*, o erro deixa de ser uma arma de punição e passa a ser uma situação onde entende-se melhor as ações e conceituações. É assim que a criança aprende uma série de conceitos. Ela é livre para explorar e os erros são usados para depurar os conceitos e não para se tornarem a arma do professor.

Finalmente, um outro aspecto metodológico é que o aluno aprende ensinando o computador, isto é, definindo novos comandos que implementam uma determinada tarefa. Portanto, é o aluno que ensina a máquina e não a máquina que ensina o aluno. E o processo de ensinar a máquina apresenta duas grandes vantagens. Primeiro, o *feedback* do computador é fiel. O aprendiz pode elaborar uma idéia e testá-la numa máquina que é "burra", onde tudo deve ser explicado segundo a sua capacidade de compreensão e cujo comportamento só pode ser fruto do que foi definido pelo aprendiz. O que o computador fornece como resultado não sofreu interpretações ambíguas ou sujeitas a emoções impulsivas. Se algo não saiu como o previsto, só pode ser culpa de quem assim definiu. Segundo, a atividade de programar em *LOGO* é uma extensão do pensamento do aprendiz. A linguagem de comunicação com o computador consiste de uma terminologia familiar e o *LOGO* permite a criação de nomes de programas que fazem sentido ao aprendiz. Estas características contribuem para que a seqüência de comandos que a criança usa seja uma descrição do processo que ela usa para resolver um determinado problema. Esta descrição pode ser analisada e depurada, servindo assim como importante fonte de aprendizado de conceitos, de idéias de resolução de problemas, e até mesmo sobre aprendizado. A atividade *LOGO* torna explícito o processo de aprender de modo que é possível refletir sobre o mesmo e entender o que o aprendiz faz para aprender e qual é o seu estilo como aprendiz: ele parte do problema e divide-o em partes mais

simples ou ataca o problema como um todo até delinear soluções globais que serão depuradas. São estes aspectos do processo de aprender que realmente falta no sistema tradicional de ensino. Assim, o uso do computador pode, além dos benefícios indicados acima, consistir numa importante ferramenta de reflexão do processo de ensino.

2.5 TANGRAM

O Tangram foi utilizado no trabalho para a representação das formas geométricas do editor gráfico. Principalmente pelo sua grande variedade de representações geométricas, além de se poder forma letras e ainda por já existir um estudo pedagógico por trás deste jogo, que é utilizado em algumas escolas para o ensino de formas geométricas, polígonos e frações. No protótipo descrito neste trabalho, não foram aplicadas todas a regras do Tangram, a fim de flexibilizar o uso do mesmo.

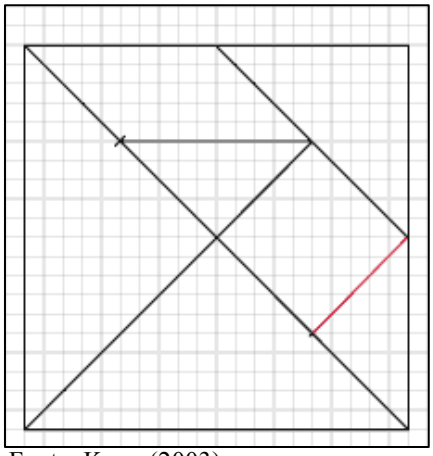
Já conhecido na China, por volta do século VII a.C. como as "Sete Taboas da Astúcia", o Tangram é um jogo figurativo para uma pessoa, do qual não se conhece o autor, nem precisamente há quanto tempo existe. Disto resultou mistérios e lendas sobre sua origem nestes últimos dois mil anos.

A história mais contada (GIRASSOL, 2003) é a de que o monge Tai-Jin chamou à sua sala o seu discípulo Lao-Tan, entregou-lhe uma placa quadrada de porcelana, um pote de tinta, um pincel e deu-lhe uma grande missão: Lao-Tan deveria percorrer o mundo e, tudo o que os seus olhos de mais belo encontrasse, deveria ser registrado na placa de porcelana. Ao sair da sala Lao-Tan deixou cair a placa quadrada de porcelana. Magicamente, a placa de porcelana quebrou-se em sete pedaços de formas geométricas simples como as peças do Tangram (fig. 8 e fig 9). Preocupado com o que acabará de acontecer, Lao-Tan imediatamente ajoelhou-se para recolher o que restava da mesma. Ao juntar os pedaços, o discípulo identificou uma figura conhecida. Trocou a posição das peças e surgiu nova figura. Assim, outras figuras foram naquele momento formando-se a cada variação de posição dos pedaços. Concluiu Lao-Tan que sua viagem não era mais necessária, pois com os sete pedaços da placa quadrada de porcelana poderia se representar tudo o que de belo existe no mundo.

2.5.1 REGRAS DO TANGRAM

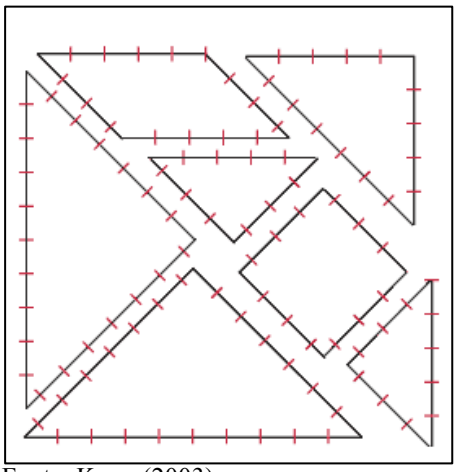
As regras básicas do Tangram são as seguintes (KONG, 2003):

a) tem de utilizar as 7 peças (tans) como pode ser visto na fig. 8 e fig. 9;
b) os tans têm que estar deitados;
c) os tans têm que se tocar;
d) nenhum tan pode sobrepor-se a outro.



Fonte: Kong (2003).

FIGURA 8 – As 7 peças do “Tangram” juntas formam um quadrado



Fonte: Kong (2003).

FIGURA 9 – Peças do “Tangram” separadas

2.6 TRANSFORMAÇÕES GEOMÉTRICAS 2D

As transformações geométricas são operações suplementares que podem ser utilizadas visando a alteração de algumas das características do objeto a ser desenhado, tais como: posição, tamanho e orientação.

Algumas soluções, para realizar translação, escalonamento e rotação de um objeto, são sugeridas por Reis (2002), as quais serão descritas a seguir.

A translação é o deslocamento do objeto a ser traçado. Nada mais é do que suas coordenadas somadas a um DY e um DX. A fórmula para se calcular a translação é mostrada no quadro 9.

```
X' = X + DX  isto para cada X do objeto a ser traçado;
Y' = Y + DY  isto para cada Y do objeto a ser traçado.
```

QUADRO 9 – Cálculo da translação

Escalonamento é a mudança de tamanho do objeto. Suas coordenadas são multiplicadas por um fator de escala FX e FY. Este processo a chamado de escalonamento. A fórmula do escalonamento pode ser vista no quadro 10.

```
X' = X * FX isto para cada X do objeto a ser traçado;
Y' = Y * FY isto para cada Y do objeto a ser traçado.
```

QUADRO 10 – Cálculo do escalonamento

Para alterar a orientação de um objeto, realiza-se uma rotação no plano das coordenadas deste objeto. É necessário saber quanto se vai rotacionar este objeto. O valor da rotação será um parâmetro informado, chamado de ANG. A fórmula de rotação pode ser vista no quadro 11.

```
X' = X * COS (ANG) + Y * SEN (ANG) isto para cada X do objeto a ser traçado;
Y' = Y * COS (ANG) + X * SEN (ANG) isto para cada Y do objeto a ser traçado.
```

QUADRO 11 – Cálculo da rotação

3 DESENVOLVIMENTO DO PROTÓTIPO

A seguir serão apresentados detalhes sobre requisitos, especificação e a implementação do protótipo.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O objetivo do desenvolvimento deste trabalho é criar um ambiente de programação gráfico que auxilie no ensino de programação de computadores. Neste ambiente tem-se um editor gráfico que servirá para que o usuário faça seus desenhos geométricos. Utiliza-se para estes desenhos, as peças do jogo matemático Tangram. Quando adiciona-se desenhos, o protótipo gera comandos em um editor de textos. Ainda, uma linguagem de programação foi criada, a qual interage com os desenhos do editor gráfico. Essa interação acontece através de comandos, os quais têm a função de criar novas peças e figuras, realizar movimentos, rotação, mudança de cor, inversão do desenho ou espelhamento. Neste editor de texto, além de ser gerado um código pelo próprio protótipo, pode-se criar um programa, utilizando a linguagem de comandos. O programa pode ser salvo em um arquivo texto, o qual posteriormente pode ser recuperado e a partir deste, gera-se o desenho novamente.

3.2 ESPECIFICAÇÃO

Os tópicos seguintes descrevem a especificação da aplicação, que visa facilitar o aprendizado de programação de computadores. Para ilustrar a funcionalidade do protótipo foram especificados 7 casos de usos os quais são: “criar figura”, “criar peça” “mover peça”, “mudar cor peça”, “rotacionar peça”, “limpar desenho” e “executar comandos”.

A especificação do protótipo foi baseada na técnica de orientação a objetos, usando-se *Unified Modeling Language* (UML), com base em Furlan (1998). Foi utilizado a ferramenta *Rational Rose* (Quatrani, 2001) para fazer a modelagem da aplicação. Esta ferramenta foi utilizada por ser uma ferramenta que dá suporte aos diagramas UML utilizados nesta especificação. Os diagramas utilizados foram: diagrama de casos de uso ou *“use cases”,* diagrama de seqüência e diagrama de classes.

3.2.1 DIAGRAMAS DE CASOS DE USO E DE SEQUÊNCIA DO PROTÓTIPO

Nas seções seguintes serão descritos, através de diagramas de casos de uso e de seqüência, o que o usuário poderá realizar com o protótipo.

3.2.1.1 PROCESSO "CRIAR FIGURA"

Este é o processo de criação de uma figura. Consiste em criar uma nova figura onde serão adicionadas peças posteriormente, no máximo 7 peças e no mínimo 1 peça. Para criar uma nova figura o usuário seleciona no protótipo o menu "Desenho" e a opção "Nova Figura". Pode-se criar quantas figuras forem necessárias para o desenho. Na fig. 10 é mostrado o diagrama de caso de uso para a criação de uma figura.

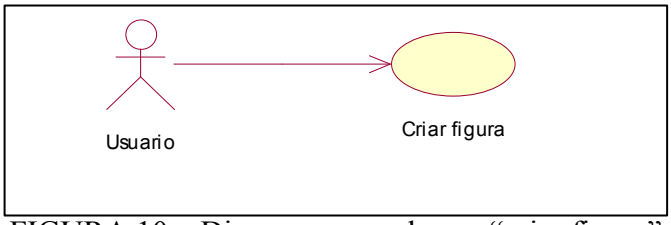


FIGURA 10 – Diagrama caso de uso "criar figura"

Quando o usuário cria uma figura, é passada a mensagem *AddFigure* para a classe *TScreenDraw*. Esta por sua vez manda a mensagem *Create* para a classe *TFigure*. A partir daí tem-se uma nova instância da classe *TFigure*. Logo após a criação dessa nova instância através da mensagem *Name* é atribuído o nome para a figura. O nome dessa figura vem como parâmetro da mensagem *AddFigure*. O diagrama de seqüência do processo "criar figura" é mostrado na fig. 11.

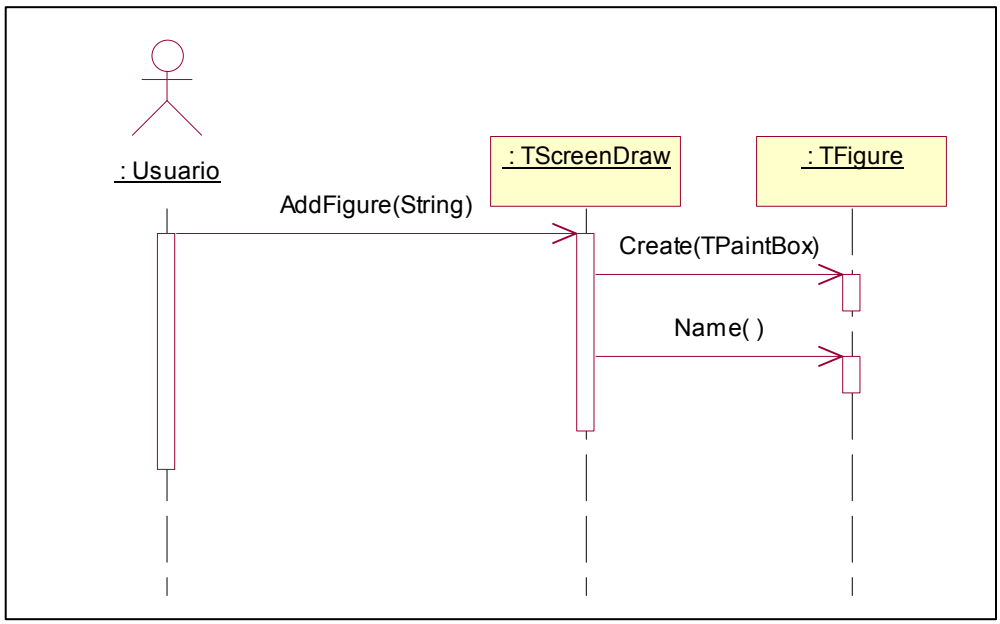


FIGURA 11 – Diagrama seqüência "criar figura"

3.2.1.2 PROCESSO “CRIAR PEÇA”

A criação de uma peça depende da existência de uma figura. O usuário pode adicionar no máximo 7 peças e no mínimo 1 peça a uma figura. Existem 7 tipos peças, os quais compõe o jogo Tangram. Para adicionar uma ou mais peças o usuário seleciona o menu “Peças” e então escolhe a peça que deseja. Após isto, esta nova peça é adicionada a figura corrente no editor gráfico do protótipo. Este processo pode ser visto no diagrama de caso de uso da fig. 12.

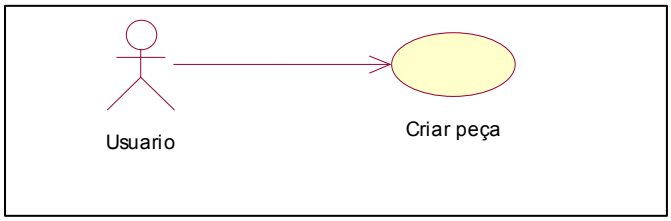


FIGURA 12 – Caso de uso “criar peça”

Quando adiciona-se uma peça, é passada a mensagem *CurrentFigure* para a classe *TScreenDraw* que retorna a instância da figura corrente onde será adiciona esta nova peça. Após isto, a partir desta instância da classe figura é passada a mensagem *AddPiece* para a classe *TFigure* com um parâmetro informando qual o tipo de peça que será criada. Na seqüência a classe *TFigure* passa a mensagem *Create* para a criação da nova instância da classe *TPiece*, que é adicionada à esta figura. O diagrama de seqüência deste processo é mostrado na fig. 13.

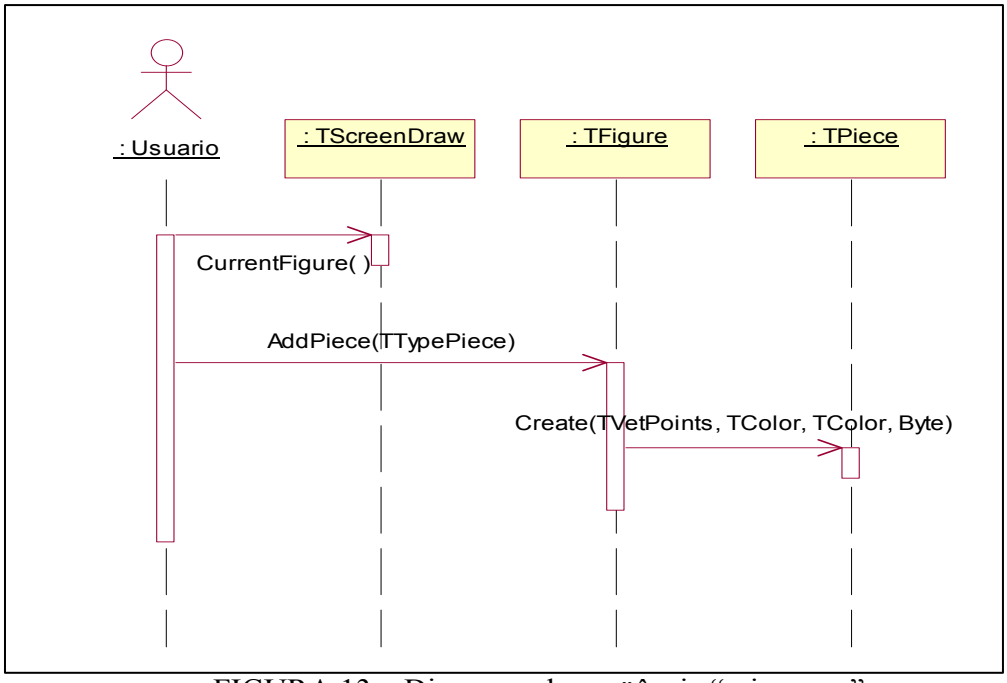


FIGURA 13 – Diagrama de seqüência “criar peça”

3.2.1.3 PROCESSO “MOVER PEÇA”

Este é o processo de movimentação de peças. O usuário no editor gráfico do protótipo clica em cima de uma peça com o *mouse*, segura o botão esquerdo do mesmo, então ainda com o botão pressionado em cima da peça arrasta a mesma sem soltar o botão até outro ponto do editor gráfico, onde soltará o botão e a peça assumira este novo ponto. A fig. 14 mostra o diagrama caso de uso para o processo “mover peça”.

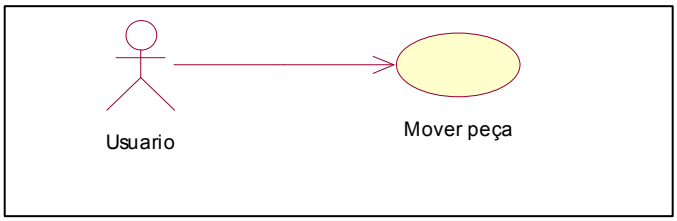


FIGURA 14 – Caso de uso “mover peça”

Quando o usuário clica em algum ponto do editor gráfico é passada a mensagem *MouseDown* para a classe *TScreenDraw* com os pontos (X, Y), o botão do *mouse* que foi pressionado e o valor da tecla, caso tenha sido pressionada alguma outra tecla de função. Logo em seguida a classe *TScreenDraw* chama a mensagem *Figure* que retorna a instância de uma figura. A partir desta instância a classe *TScreendraw* chama a mensagem *Piece*, que retorna a instância da classe *TPiece*. A classe *TFigure* chama a mensagem *PointInPoly* da classe *TPiece* que verifica se os pontos X e Y passados como parâmetro estão dentro desta peça. Logo após quando o usuário move o *mouse* é chamada a mensagem *MouseMove* com os parâmetros X e Y do novo ponto onde será arrastada a peça. Em seguida é verificado se o botão esquerdo do *mouse* esta pressionado. Caso o botão esteja pressionado, é passada a mensagem *MoveBy* com a quantidade que se deseja mover a peça. O diagrama de seqüência deste processo é mostrado na fig. 15.

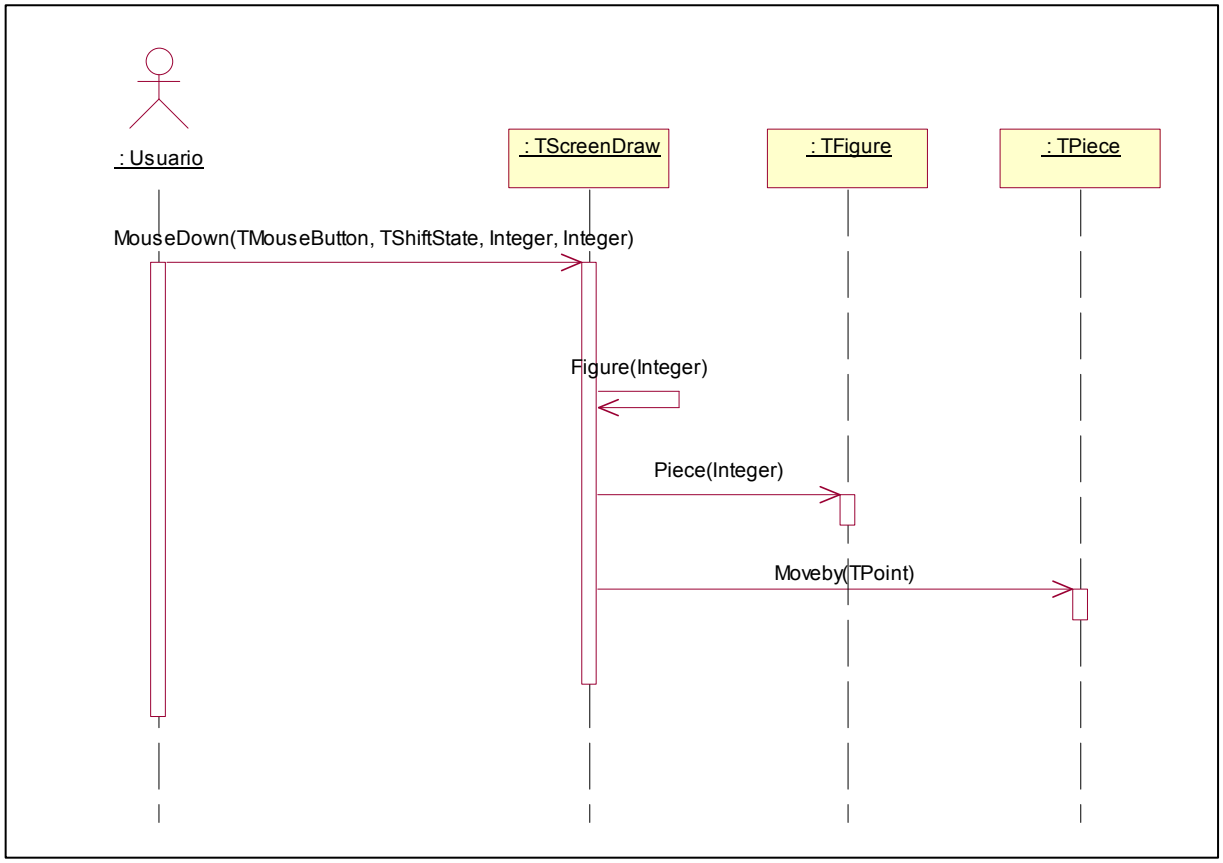


FIGURA 15 – Diagrama de seqüência “mover peça”

3.2.1.4 PROCESSO “ROTACIONAR PEÇA”

Este processo descreve como é realizada a rotação de uma peça. O usuário escolhe uma peça e clica com o botão direito do *mouse*. A peça é rotacionada 45 graus. O diagrama de caso de uso deste processo é mostrado fig. 16.

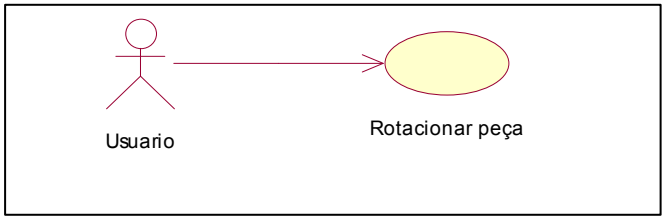


FIGURA 16 – Caso de uso “rotacionar peça”

Quando o usuário clica em algum ponto do editor gráfico é passada a mensagem *MouseDown* para a classe *TScreenDraw* com os pontos (X, Y), o botão do *mouse* que foi pressionado e o valor da tecla, caso tenha sido pressionada alguma outra tecla de função. Logo em seguida a classe *TScreenDraw* chama a mensagem *Figure* que retorna a instância de uma figura. A partir desta instância a classe *TScreendraw* chama a mensagem *Piece*, que retorna uma instância da classe *TPiece*. A classe *TFigure* chama a mensagem *PointInPoly* da

classe *TPiece* que verifica se os pontos X e Y passados como parâmetro estão dentro desta peça. Na seqüência é verificado se o botão direito do *mouse* foi pressionado. Caso o botão esteja pressionado é passada a mensagem *RotatePiece* com a quantidade em graus que será rotaciona a peça. A rotação comandada a partir do *mouse* é feita com um valor padrão de 45 gruas. O diagrama de seqüência do processo "rotacionar peça" é mostrado na fig. 17.

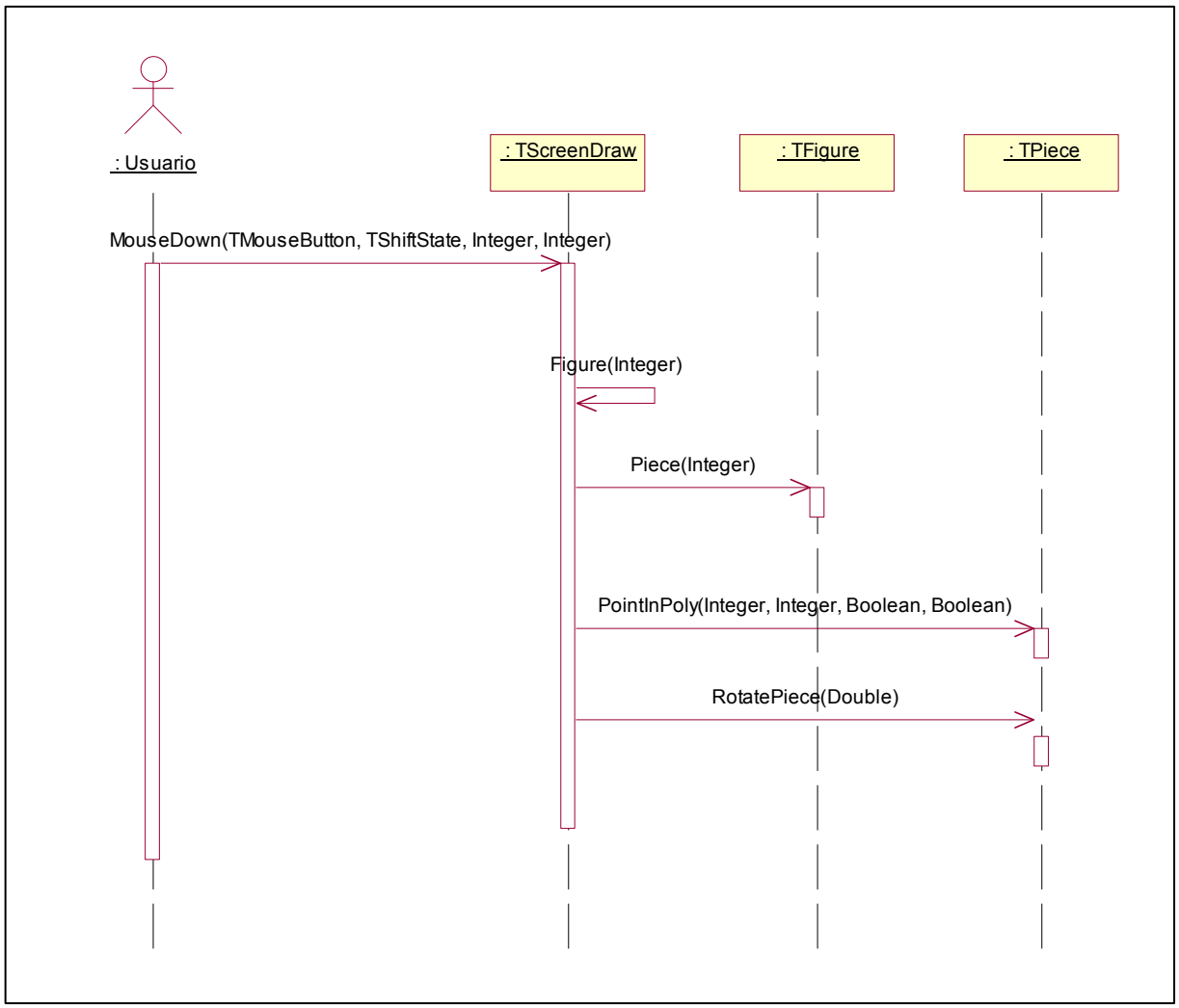


FIGURA 17 – Diagrama seqüência "rotacionar peça"

3.2.1.5 PROCESSO "MUDAR COR PEÇA"

Este processo consiste em mudar a cor de uma peça. O usuário seleciona uma cor, e logo após clicar com o botão esquerdo do *mouse* em cima de uma peça com a tecla *shift* pressionada, esta é pintada da cor selecionada. A fig. 18 mostra o diagrama de caso de uso do processo "muda cor peça".

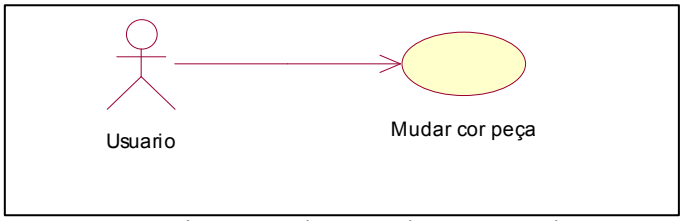


FIGURA 18 – Diagrama de caso de uso “mudar cor peça”

Quando o usuário clica em algum ponto do editor gráfico é passada a mensagem *MouseDown* para a classe *TScreenDraw* com os pontos (X, Y), o botão do mouse que foi pressionado e o valor da tecla, caso tenha sido pressionada alguma outra tecla de função. Logo em seguida a classe *TScreenDraw* chama a mensagem *Figure* que retorna a instância de uma figura. A partir desta instância a classe *TScreendraw* chama a mensagem *Piece*, que retorna a instância da classe *TPiece*. A classe *TFigure* chama a mensagem *PointInPoly* da classe *TPiece* que verifica se os pontos X e Y passados como parâmetro estão dentro desta peça. Em seguida é verificado se o botão esquerdo do *mouse* foi pressionado junto com a tecla de função *ctrl*. Caso esteja, é passada a mensagem *ChangeColor* com a cor selecionada pelo usuário na tela. O diagrama de seqüência do processo “muda cor peça” é mostrada na fig. 19.

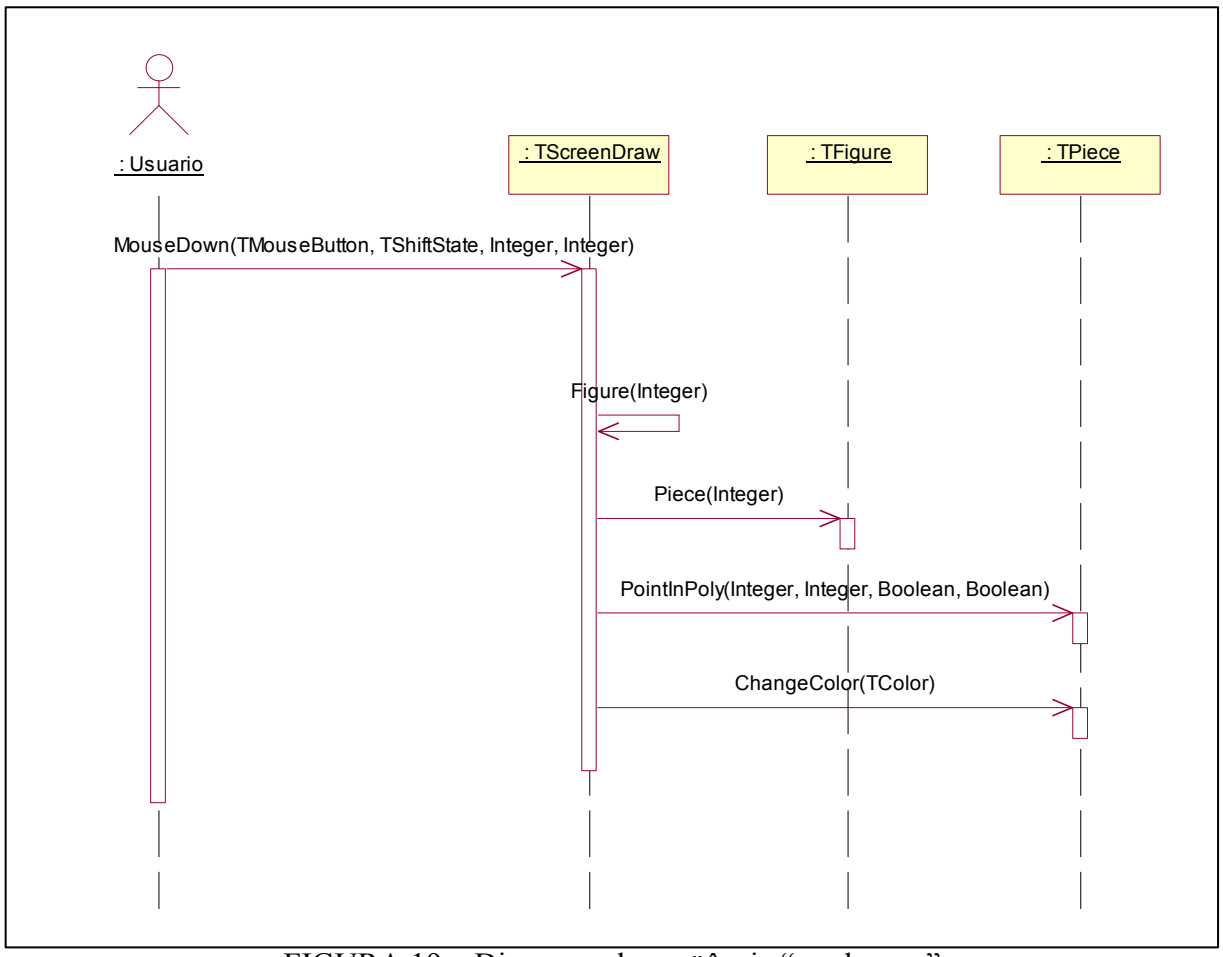


FIGURA 19 – Diagrama de seqüência “mudar cor”

3.2.1.6 PROCESSO “LIMPAR DESENHO”

O processo “limpar desenho” realiza a remoção de todas as figuras e suas respectivas peças desenhadas no editor. A fig. 20 apresenta o diagrama de caso de uso deste processo.

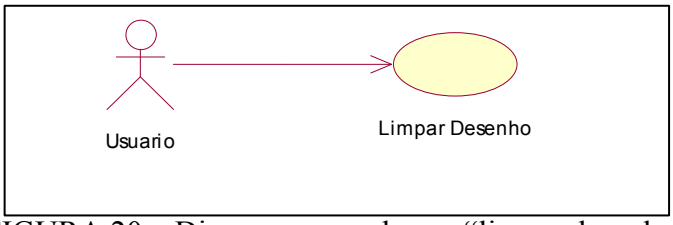


FIGURA 20 – Diagrama caso de uso “limpar desenho”

O processo “limpar figura” consiste em solicitar ao objeto da classe *TScreenDraw* para remover todas as figuras, apagando-as da tela do editor. Quando o usuário seleciona a opção “Limpar tudo” no menu desenho, é passada a mensagem *ClearAll* para classe *TScreenDraw*, que com a mensagem *RemPiece* irá chamar para todas as peças a mensagem *Destroy*, que por sua vez apagará cada peça. Em seguida será liberada a figura pela mensagem *Destroy* da figura. Isso acontecesse para cada figura, até limpar todo o desenho. O diagrama de seqüência do processo “limpar figura” é mostrado na fig. 21.

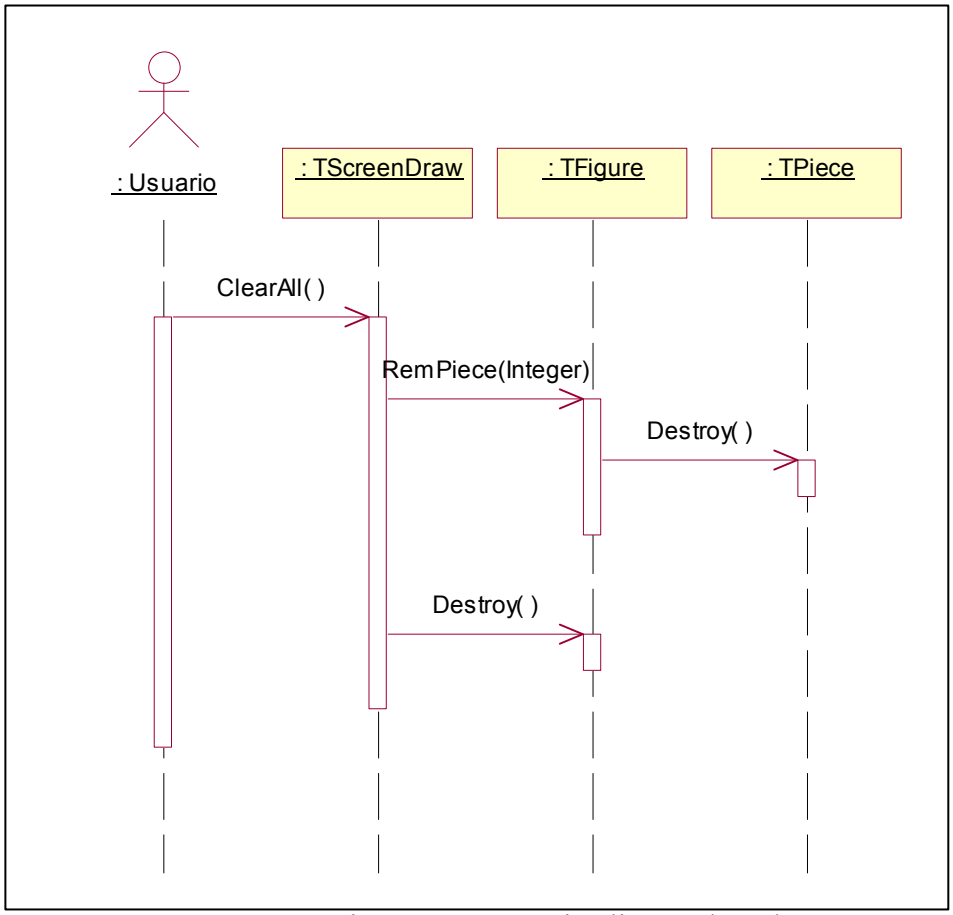


FIGURA 21 – Diagrama seqüência “limpar desenho”

3.2.1.7 PROCESSO “EXECUTAR COMANDOS”

No processo de “executar comandos”, após o usuário ter inserido comandos da linguagem através do editor de textos, estes comandos são executados e é visualizado o resultado através de figuras gráficas, no editor gráfico. A fig. 22 mostra o diagrama de caso de uso do processo “executar comandos”.

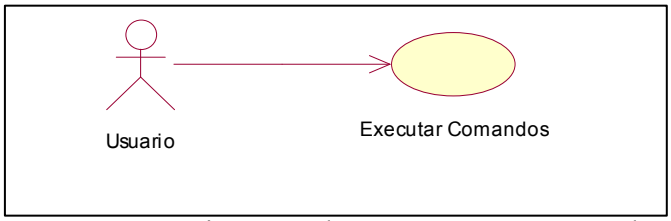


FIGURA 22 – Diagrama de uso “executar comandos”

Quando usuário executa o código da linguagem, é passada a mensagem *ProcessScript* com o parâmetro que contém o código da linguagem. Logo após é passada a mensagem *Process* para a classe *TSyntacticalAnalyser* que irá fazer toda análise sintática do código da

linguagem através das mensagens *Program*, *Comandos e Comando*. O diagrama de seqüência do processo “executar comandos” é mostrado na fig. 23.

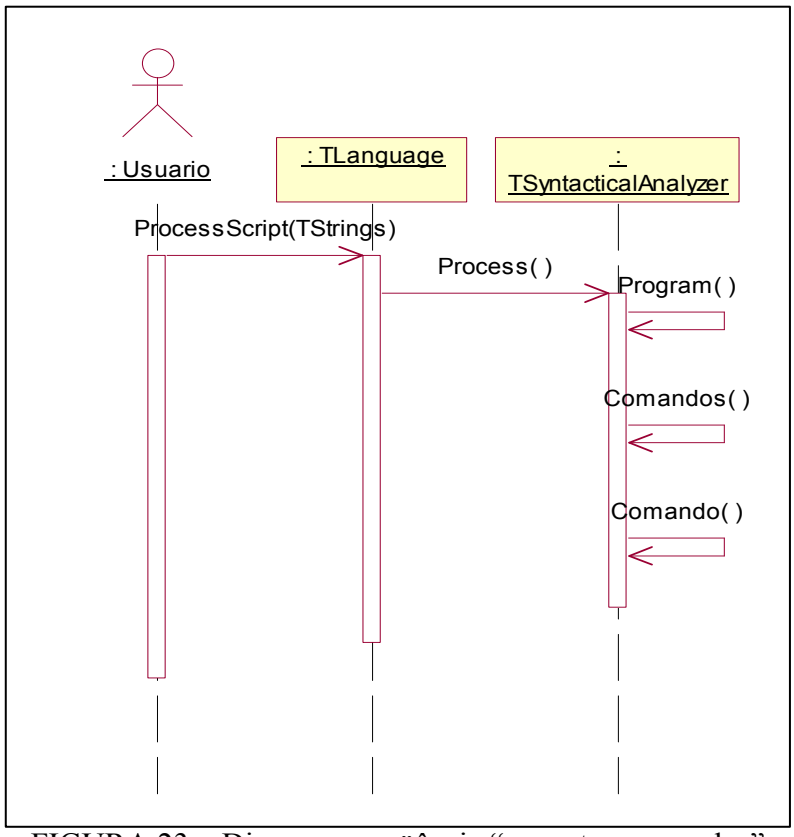


FIGURA 23 – Diagrama seqüência “executar comandos”

3.2.2 DIAGRAMA DE CLASSES DO PROTÓTIPO

A seguir são descritas as funcionalidades das classes modeladas, as quais são: TLanguage, TAnalizerLex, TSyntacticalANalyzer, TScreenDraw, TFigure e TPiece.

A classe *TLanguage* é a classe principal do sistema que agrega as classes *TAnalizerLex*, *TSyntacticalAnalyzer* e *TScreenDraw*. Esta classe é responsável em passar as mensagens para as outras classes executarem e interpretarem o código da linguagem. Também é responsável pelos comandos gerados pelo protótipo, quando é adicionada uma peça no editor gráfico da linguagem.

A classe *TScreeDraw* é a classe da tela onde são desenhadas as figuras com suas respectivas peças. Ela é a classe responsável por pintar na tela as figuras criadas. Ela agrega uma lista de figuras, que por sua vez, agrega uma lista de peças que serão mostradas na tela.

A classe *TSyntacticalAnalyzer* é responsável pela análise sintática da linguagem e a chamada das rotinas que executam os comandos.

A análise léxica da linguagem é feita pela classe *TAnalizerLex*.

As figuras são definidas pela classe de figura *TFigur*a. Cada figura contém peças. Esta classe é responsável pelos comandos que serão executados na figura como um todo, ou seja, quando todas as peças da figura são afetadas.

A classe *TPiece* define as peças que montam uma figura, onde são guardadas as coordenadas de cada peça. Esta classe é responsável por executar os comandos em cima de cada peça separadamente.

Para especificação do protótipo foram modeladas 6 classes citadas acima, as quais podem ser vistas na fig. 24.

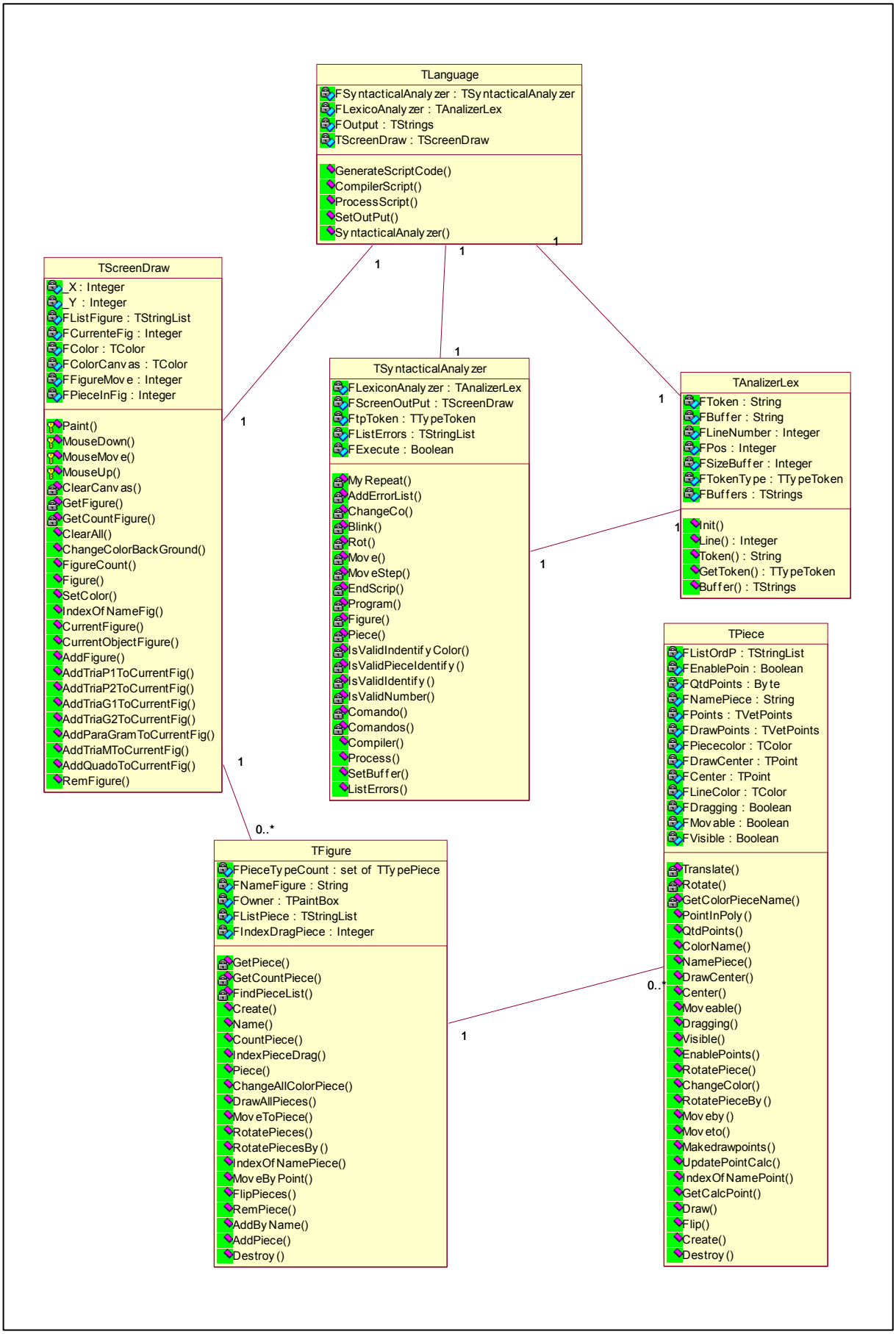


FIGURA 24 – Diagrama de classes da implementação

3.2.3 LINGUAGEM

Nesta seção são mostrados todos os comandos existentes na linguagem proposta e as especificações dos mesmos, através da BNF.

Os comandos existentes na linguagem são: cria, move, movepasso, rotaciona, piscar, mudacor, espelho e repete.

O comando "cria" cria peças e figuras, passando como parâmetro um ponto (X, Y), onde será criada a peça da figura. Exemplo da sintaxe do comando "cria" pode ser visto no quadro 12.

```
CRIA FIGURA0.P1 (0, 0)
```

QUADRO 12 – Exemplo da sintaxe do comando "cria"

O comando "move" move uma peça ou uma figura inteira para um ponto dado (X,Y). Exemplo da sintaxe do comando "move" pode ser visto quadro 13.

```
MOVE FIGURA0.P1(10,10) movendo peça
MOVE FIGURA0 (10,10) movendo figura
```

QUADRO 13 – Exemplo da sintaxe do comando "move"

Com o comando "movepasso" move-se a figura do ponto onde estava (X0,Y0) para o ponto com o deslocamento (X,Y) informado (X0+X, Y0+Y). Exemplo da sintaxe do comando "movepasso" poder ser vista no quadro 14.

```
MOVEPASSO FIGURA0.P1(10,10)  movendo peça
MOVEPASSO FIGURA0 (10,10)  movendo figura
```

QUADRO 14 – Exemplo da sintaxe do comando "movepasso"

No comando "rotaciona" é rotacionada a peça em relação a um ponto da mesma. É passado como parâmetro o quanto deseja-se rotacionar. O valor do parâmetro pode ser positivo ou negativo (anti-horária ou horária). O comando "rotaciona" também serve para rotacionar a figura inteira com relação a um dos pontos de uma peça desta figura. Exemplos da sintaxe do comando "rotaciona" podem ser vistos no quadro 15.

```
rotaciona FIGURA1.P1(1, +45) rotacionando peça
rotaciona FIGURA1(P1.1, +45) rotacionando figura
```

QUADRO 15 – Exemplo da sintaxe do comando "rotaciona"

O comando "piscar" aguarda um tempo em milisengundos antes de executar a próxima instrução. A sintaxe do comando piscar pode ser vista no quadro 16.

```
PISCAR (1000)
```

QUADRO 16 – Exemplo da sintaxe do comando “piscar”

Como seu nome já diz, “mudacor” muda a cor de uma peça, ou de uma figura inteira ou do fundo da tela do desenho. A sintaxe do comando “mudacor” pode ser vista no quadro 17.

```
MUDACOR FIGURA1.P1 preto  cor peça
MUDACOR FIGURA1 preto cor figura
MUDACOR FUNDO preto cor fundo
```

QUADRO 17 – Exemplo da sintaxe do comando “mudacor”

O comando “espelho” inverte os pontos da peça passada como parâmetro, fazendo um espelhamento da mesma. A sintaxe do comando “espelho” pode ser vista no quadro 18.

```
ESPEHO FIGURA1.P1
```

QUADRO 18 – Exemplo da sintaxe do comando “espelho”

O comando “repete” repete um bloco de comandos a quantidade de vezes indicado no mesmo. É permitido um “repete” dentro de outro “repete”. A sintaxe do comando repete pode ser vista no quadro 19.

```
repete 2 vezes
inicio
  piscar(1000)
  rotaciona FIGURA1.P1(0, +45)
  repete 2 vezes
  inicio
     piscar(1000)
   fim
fim
```

QUADRO 19 – Sintaxe do comando “repete”

3.2.3.1 BNF (BACKUS NAUR FORM) DA LINGUAGEM CRIADA

Para especificar os comandos da linguagem foi utilizada uma BNF extendida. A especificação da linguagem pode ser vista no quadro 20.

```
<LETRAMAIUSCULA>   ::=  A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | R | S | T | U | V | W | X | Y | Z

<LETRAMINUSCULA>  ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o  | p | q | r | s | t | u | v | w | x | y | z

<DIGITO>                  ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<SINAL>                   ::=  + | -

<PONTO>                   ::= .

<VIRGULA>                 ::= ,

<PONT>                    ::= <PONTO> | <VIRGULA>

<ABRE_PARENTESES> :=  (

<FECHA_PARENTESES>:= )

<SIMBES>                  ::= <ABRE_PARENTESES> | <FECHA_PARENTESES>

<LETRA>                   ::= <LETRAMAIUSCULA> | <LETRAMINUSCULA>

<ID>                      ::= <LETRA>(<LETRA>|<DIGITO>)*

<ID_COR>                  ::=  PRETO | MARROM | VERDE | OLIVA | AZULMARINHO | VIOLETA | VERDEPISCINA | CINZA
                                | PRATA | VERMELHO | VERDELIMA | AMARELHO | AZUL | ROSA | AZULPISCINA | BRANCO

<NUM_SEM_SINAL>   ::= <DIGITO> (<DIGITO>)*

<NUM>                     ::=  (SINAL | є) <NUM_SEM_SINAL>

<MOVEPASSO>          ::=  MOVEPASSO <ID> (“.”<ID> | є)  “(“<NUM>”,”<NUM>”)”

<MOVE>                    ::=  MOVE  <ID> (“.”<ID> | є)  “(“<NUM>”,”<NUM>”)”

<ROTACIONA>          ::=  ROTACIONA <ID>(.<ID> ”(“ | ”(“ <ID> ”.”)<NUM>”, “<NUM>”)”

<PISCAR>                  ::=  PISCAR “(“<NUM>”)”

<MUDACOR>           ::=  MUDACOR  ((<ID> ( “.”<ID> | є)) |  FUNDO ) <IDCOR>

<CRIA>                    ::= CRIA <ID> “.” <ID> ”(“<NUM>”,” <NUM>”)”

<ESPELHO>              ::= ESPELHO <ID> “.” <ID>

<REPETE>                ::=  REPETE <NUM_SEM_SINAL> VEZES INICIO <COMANDOS> FIM

<COMANDO>           ::= <CRIA> | <MOVE> | <MOVEPASSO> | <ROTACIONA> | <PISCAR> | <REPETE> | <MUDAR> |  <ES
PELHO>

<COMANDOS>          ::=  <COMANDO>   (<COMANDO>)*

<PROGRAMA>          ::= INICIO <ID> <COMANDOS> FIM “.”
```

QUADRO 20 – BNF extendida da linguagem criada

No quadro 21 tem-se um exemplo de programa, obedecendo à especificação do quadro 20.

```
INICIO NOVOPROGRAMA
REPETE 10 VEZES
INICIO
  PISCAR(1000)
  ESPELHO FIGURA1.P1
  CRIA FIGURA0.P1(0,0)
  CRIA FIGURA0.P2(0,0)
  ROTACIONA FIGURA0.P1(0,+45)
  ROTACIONA FIGURA0.P2(0,+45)
  MOVE FIGURA0.P2(-70,191)
  MOVE FIGURA0.P1(16,192)
  MOVE PASSO FIGURA0.P1(5, 5)
  MUDACOR FIGURA0.P2 AZUL
  MUDACOR FIGURA0.P1 AZUL
FIM
FIM.
```

QUADRO 21 – Exemplo de programa da linguagem

3.3 IMPLEMENTAÇÃO

Nesta seção são discutidos os aspectos sobre a implementação do protótipo e as ferramentas utilizadas para a construção.

3.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para implementação do protótipo foi utilizado o ambiente de desenvolvimento *Borland Delphi 7*. O Delphi é um ambiente híbrido que dá suporte tanto à orientação objetos quanto à programação procedural. A linguagem utilizada pelo ambiente é Object Pascal. No quadro 22 é mostrado parte do código do protótipo feito na linguagem Object Pascal.

```
Function TSyntacticalAnalyzer.Comandos: Boolean;
begin
 FTpToken:= FLexiconAnalyzer.GetToken;
 if (FTpToken <> ttEND) then
 begin
  if (CompareText(FLexiconAnalyzer.Token, TK_FIM) <> 0) then
  begin
   result:=True;
   while  (result)
   and    (FTpToken <> ttEND)
   and    (CompareText(FLexiconAnalyzer.Token, TK_FIM) <> 0) do
   begin
    result  := Comando;
    FTpToken:= FLexiconAnalyzer.GetToken;
   end;
   if not result then
   begin
    result:=false;
    AddErrorList(ID_TK_INVALIDO, 'COMANDO INVALIDO', FLexiconAnalyzer.Token, FLexiconAnalyzer.Line);
   end;
  end
  else
   result:=False;
  end else result:=true; end;
```

QUADRO 22 – Fonte do protótipo função TSintacticalAnalyzer.Comandos

No quadro 22 a função *TSintacticalAnalyzer.Comandos* é listada. Esta função realiza a análise e execução dos comandos da linguagem.

Um algoritmo para determinar e mostrar os pontos de uma figura foi implementado, objetivando orientar o usuário durante a manipulação do desenho. As formas geométricas disponibilizadas pelo protótipo podem ter de 3 a 4 pontos. Este algoritmo quando solicitado redefine qual será a seqüência ou ordem de cada ponto da peça, a partir do posicionamento da

mesma na tela. No quadro 23 é mostrada a estratégia adotada para a determinação da seqüência ou ordem dos pontos. Quando um ponto é classificado este não é mais verificado. Somente os pontos que não foram classificados ainda são utilizados pelo algoritmo. Um exemplo do resultado da aplicação deste algoritmo é mostrado na fig. 25.

Ponto 1 = Maior Y da peça em questão, caso exista dois Y iguais então pega-se o Y com o menor X.
Ponto 2 = Menor X da peça em questão, caso exista dois X iguais então pega-se o X com o maior Y.
Ponto 3 = Menor Y da peça em questão, caso exista dois Y iguais então pega-se o Y com o maior X.
Ponto 4 = Maior X da peça em questão, caso exista dois X iguais então pega-se o X com o maior Y.

QUADRO 23 – Algoritmo classificação de pontos de uma peça

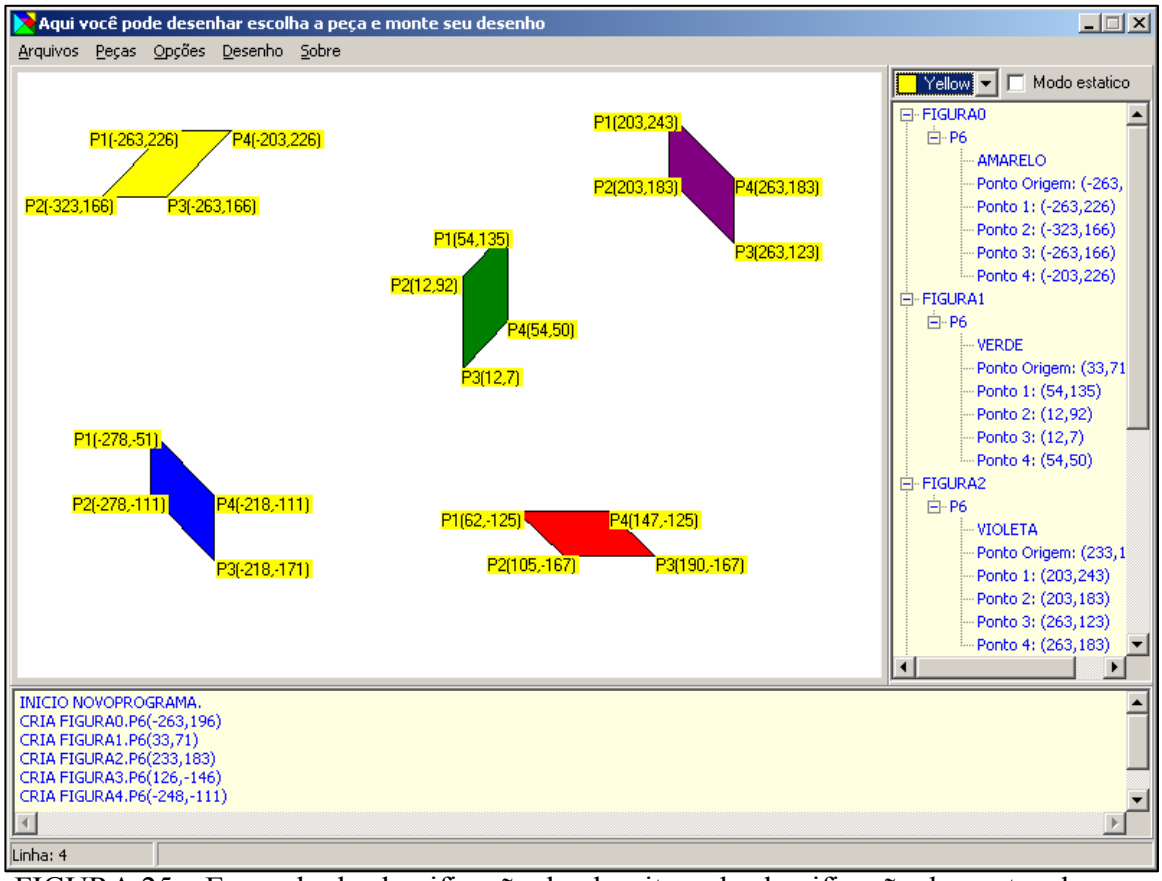


FIGURA 25 – Exemplo da classificação do algoritmo de classificação de pontos de peça

3.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Quando executado o protótipo, aparece a tela mostrada na fig. 26, onde o usuário cria seus desenhos. Conforme ele adiciona ou move peças de uma figura, o software gera linhas de código na linguagem. Este código pode ser alterado no editor da linguagem, que fica na mesma tela, logo abaixo da figura desenhada. Ainda existe nesta tela uma janela com as informações sobre estrutura do desenho como um todo, com cada figura e suas respectivas

peças, tais como: nome das figuras do desenho, o nome das peças para cada figura e para cada peça tem-se cor, ponto de origem, ponto de 1 a n, e seus respectivos valores em X e Y. Têm-se também as opções que são pintar uma peça ou espelhar uma peça. Por exemplo, escolhe-se uma peça da figura e quando pressionada a tecla “ctrl” juntamente com o botão *click* esquerdo do *mouse* em cima da peça escolhida, a mesma é pintada da cor selecionada no *combo* de cores que está no canto superior da tela . Ainda, pressionando-se “shift” juntamente com o botão *click* esquerdo do *mouse*, os pontos da peça selecionada serão invertidos, fazendo o espelhamento da peça.

Na fig. 26 pode ser visto a tela principal já com um desenho criado.

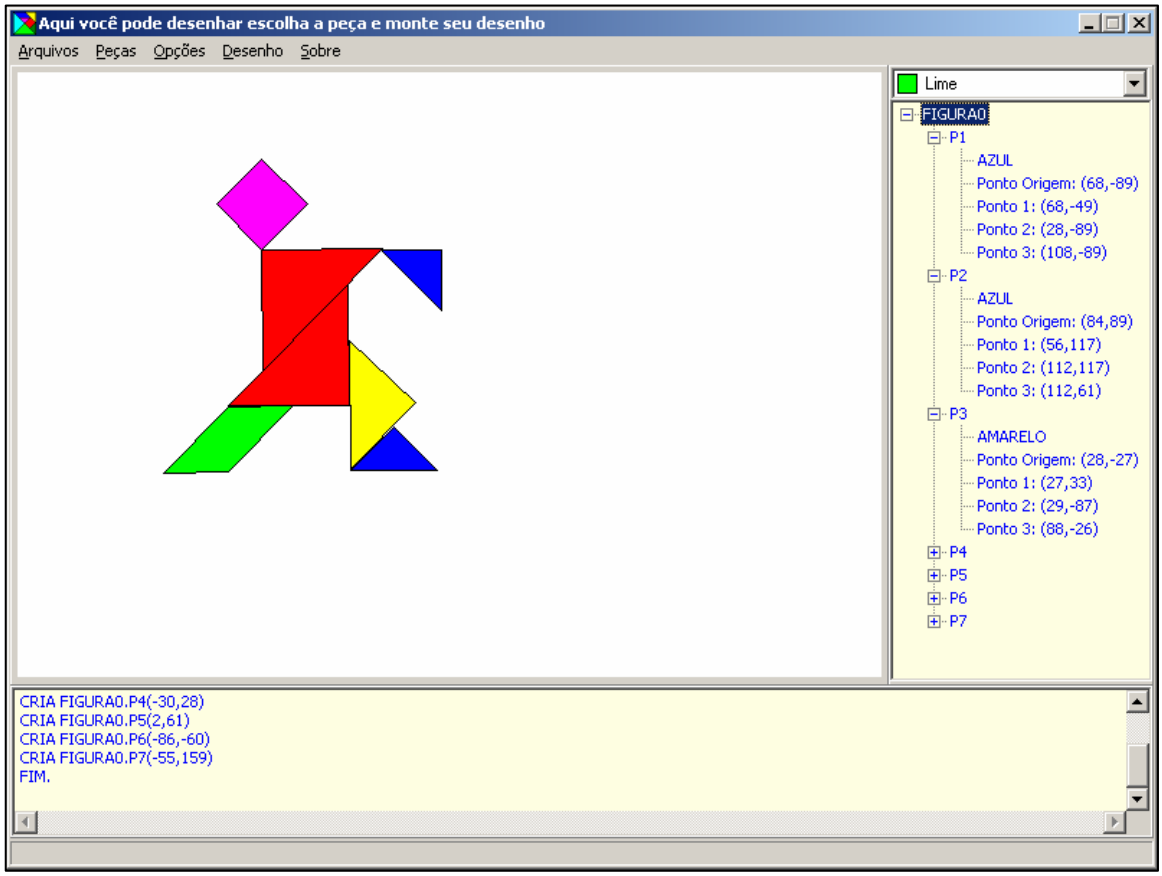


FIGURA 26 – Tela principal do protótipo

Existem no protótipo os menus “Arquivo”, “Peças”, “Opções”, “Desenho” e “Sobre”. No menu “Arquivo”, o usuário tem as opções de “Salvar Arquivo” que salva o código do editor da linguagem, para posteriormente retornar ao desenho criado. Com a opção “Abrir Arquivo”, no mesmo menu. Ainda, tem-se a opção de “Fechar Arquivo” que limpa o código do editor da linguagem, limpa a figura e por fim as informações sobre o desenho. A opção “Sair”, fecha o software. Estas opções podem ser vistas na fig. 27.

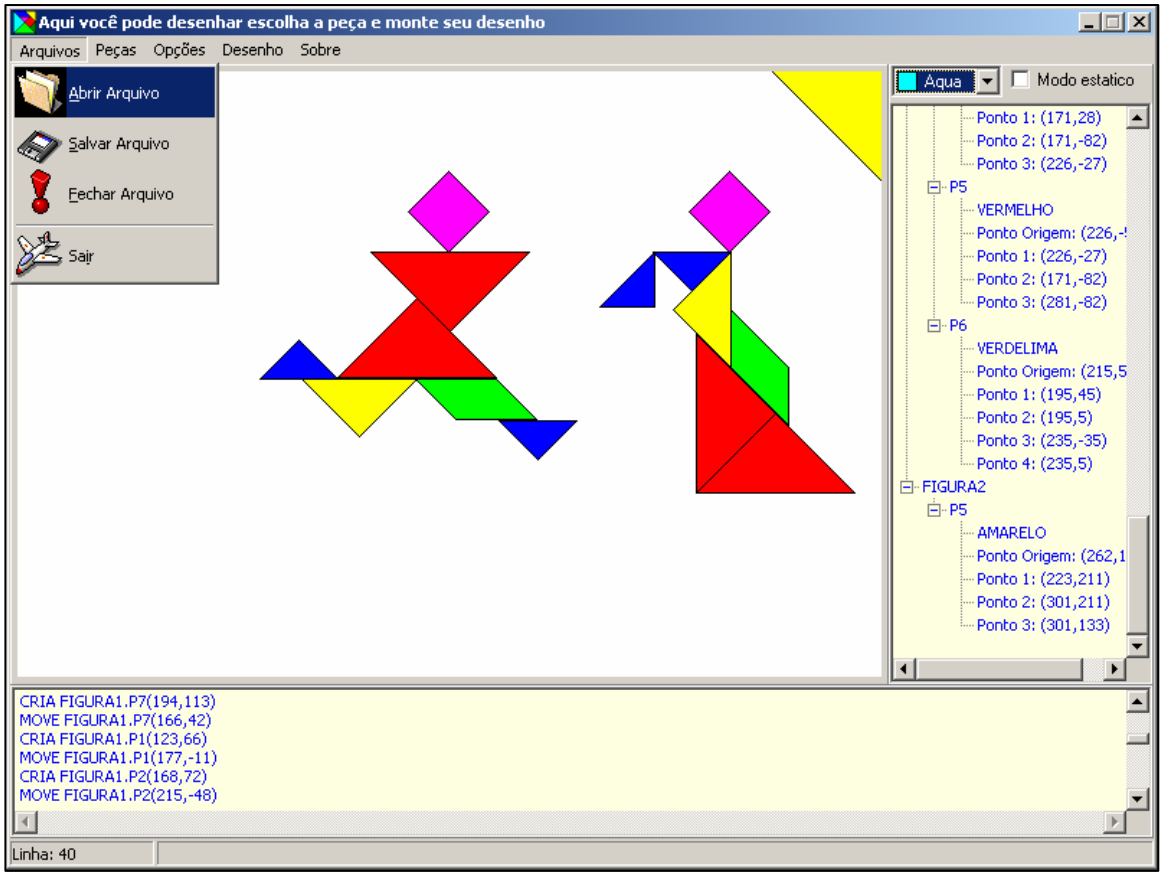


FIGURA 27 – Menu arquivos

No menu “Peças”, o usuário pode adicionar peças a uma figura, mas primeiro ele deve ter entrado no menu “Desenho” e ter criado uma nova figura. Depois de criar uma nova figura, ele pode adicionar até sete tipos de peças, as quais são mostradas na fig. 28.

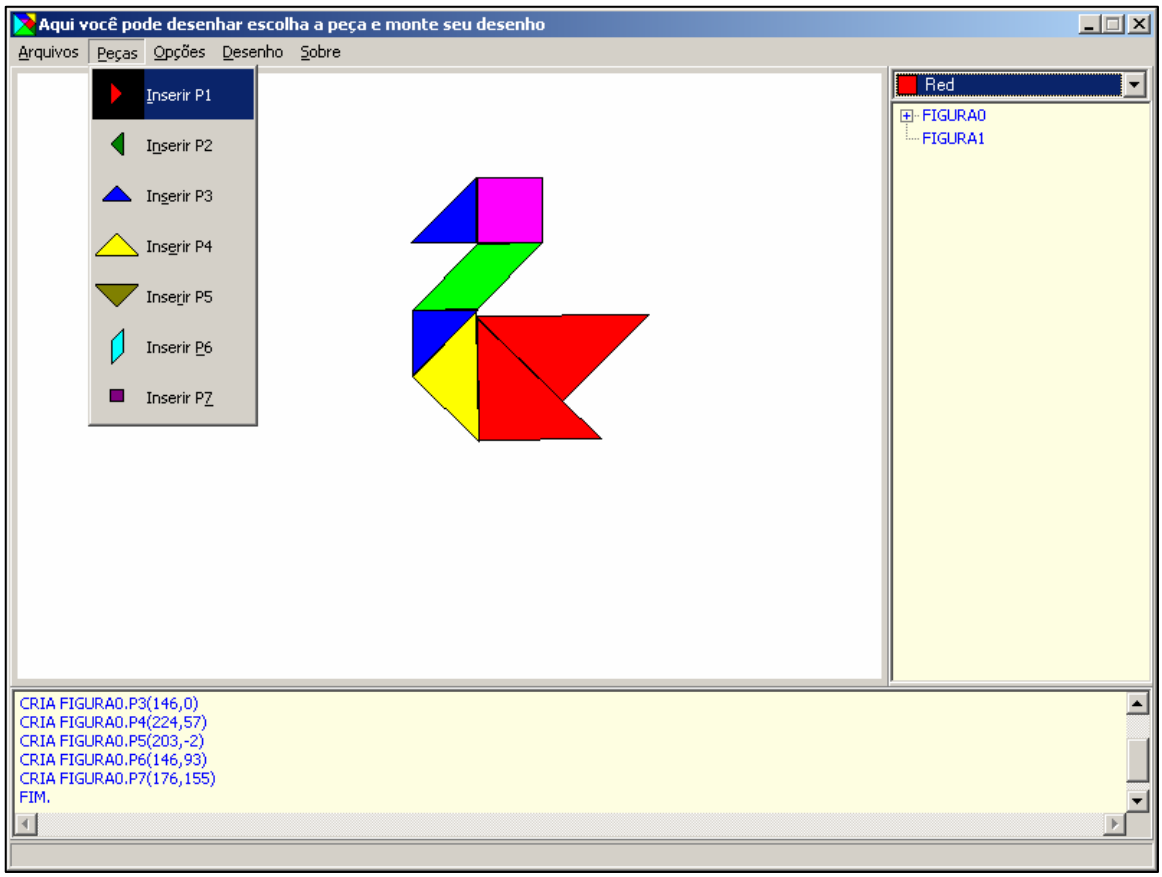


FIGURA 28 – Menu peças

Ainda tem-se o menu “Opções”, onde o usuário pode fazer a verificação do seu código, antes de executá-lo. Caso erros sejam detectados, é apresentada uma tela com os mesmos e suas respectivas linhas no código. Para fazer esta verificação, o usuário escolhe a opção “Verificar Escrita”. Tem-se ainda a opção “Executar” que executa o código que está no editor da linguagem e a opção “Mostrar Pontos”, que mostra cada ponto e seu respectivo valor, de cada peça. Os pontos são mostrados no editor gráfico. Esta opção pode ser habilitada e desabilitada cada vez que é escolhida. Este menu pode ser visualizado na fig. 29.

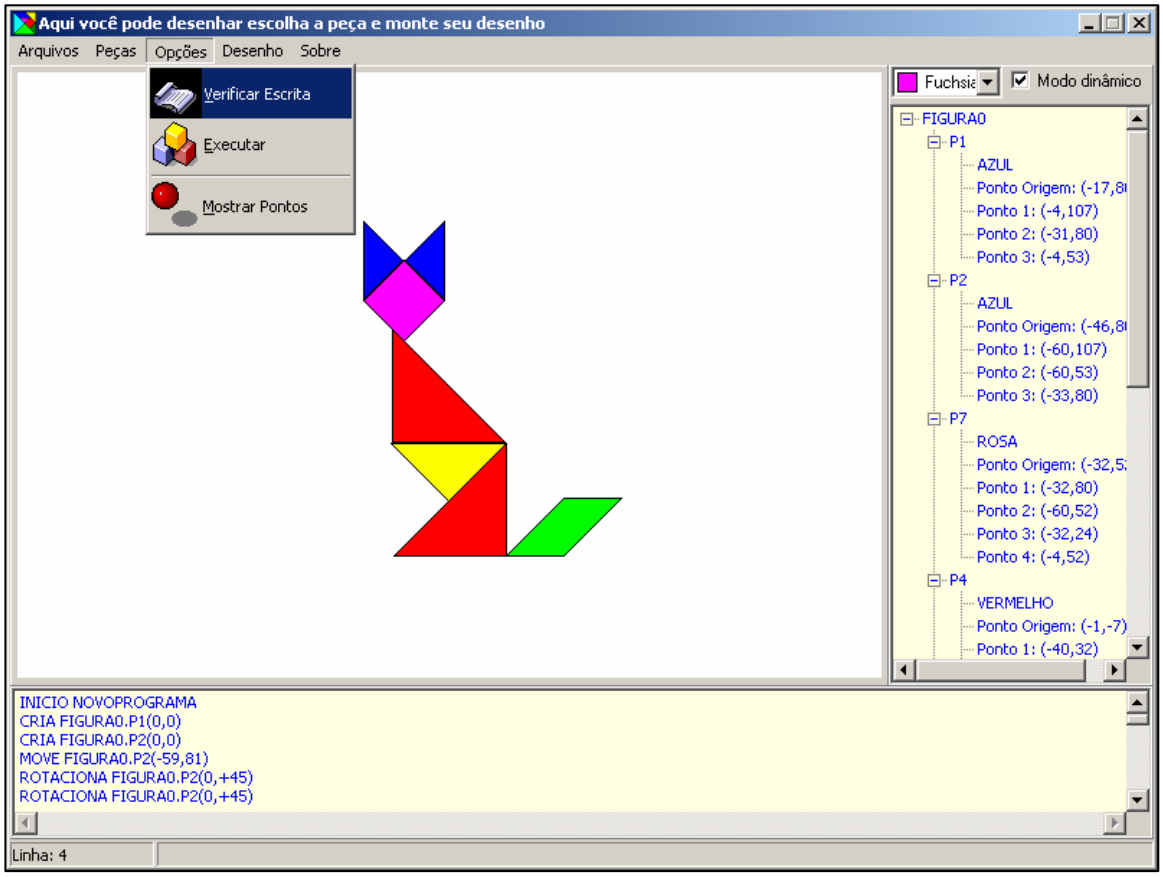


FIGURA 29 – Menu opções

No menu “desenho” tem-se mais duas opções, que são “Nova Figura” e “Limpar tudo”. A opção “Nova figura” cria uma nova figura onde poderão ser adicionadas até 7 peças, do menu “peças” (fig. 28). A opção “Limpar tudo” apaga todo o desenho e o código do editor da linguagem, ou seja, retorna ao estado inicial do protótipo. Este menu pode ser visto na fig. 30.

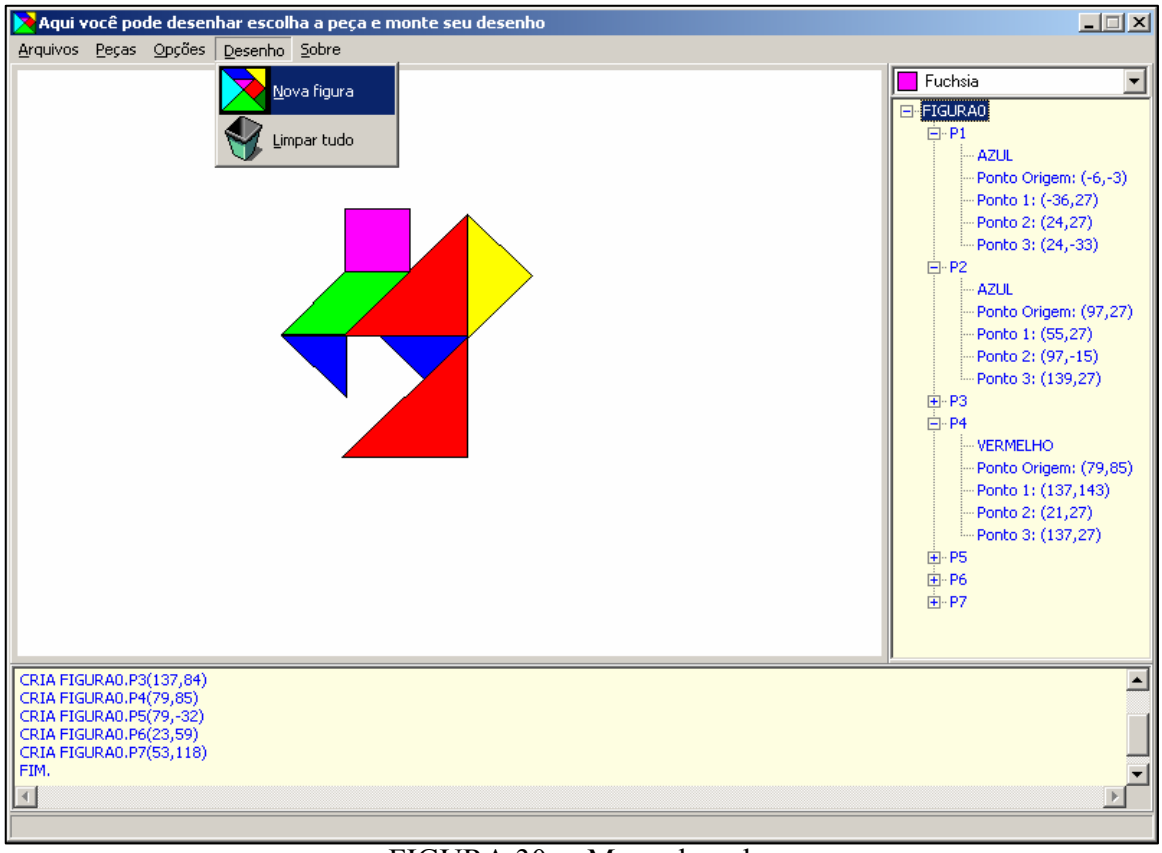


FIGURA 30 – Menu desenho

Por fim, tem-se a opção “sobre”, que contém algumas informações sobre o protótipo criado (fig. 31).

FIGURA 31 – Menu sobre

4 CONCLUSÃO

O objetivo inicial do trabalho foi atingido, o qual era a criação de um ambiente visual que possa ser utilizado no ensino de programação de computadores. O ambiente criado é composto por dois editores, um gráfico e outro de texto. Ainda, foi criada uma linguagem de programação que interage com os desenhos do editor gráfico. Para representar os desenhos no editor gráfico foram utilizadas as peças do jogo Tangram, pois a partir das 7 peças que compõem o mesmo, pode-se representar uma grande variedade de figuras do mundo real. Ainda o uso do Tangram justifica-se no trabalho pelo estudo pedagógico que já existe por trás deste jogo, muito utilizado no ensino em algumas escolas.

O ambiente de programação DELPHI utilizado na implementação do protótipo mostrou-se adequado, suprindo as necessidades da implementação.

O editor gráfico implementado tem algumas limitações, como por exemplo o tamanho da sua área de desenho hoje é fixa. A área utilizada poderia ter um tamanho dinâmico. Uma solução para aumentar a área de desenho seria separar a tela do editor gráfico do editor de textos em duas distintas.

Para facilitar o manuseio das coordenadas no editor gráfico, foi criada uma opção onde são mostrados os pontos das peças que estão desenhadas. Isto poderia ser aprimorado com a criação de um *grid,* dividindo o editor gráfico em pequenas regiões.

No protótipo atual, quando se arrasta uma peça na tela, não se tem um posicionamento preciso com o *mouse*. Sugere-se a criação de um mecanismo que facilite este procedimento. Ainda, no protótipo não existe a opção de selecionar uma figura por inteiro. Seleciona-se uma figura através da seleção individual de cada peça da mesma.

No editor de texto existem algumas limitações. Uma delas é seu tamanho reduzido. Algumas melhorias poderiam ser feitas no ambiente, como a criação de links visuais entre os comandos textuais e as peças no editor gráfico.

Os menus criados satisfazem o uso básico do protótipo nas suas principais funcionalidades, porém não foi implementado um menu de ajuda, onde o usuário poderia fazer consultas sobre os comandos da linguagem e suas funções.

REFERÊNCIAS BIBLIOGRÁFICAS

BARANAUSKAS, Maria Cecília Calani. Procedimento, função, objeto ou lógica? Linguagens de programação vistas pelos seus paradigmas. In: VALENTE, José Armando (Org.). **Computadores e conhecimento**: repensando a educação. Campinas, SP: Gráfica Central da Unicamp, 1993. p. 45-63.

CANTÚ, Marco. **Dominando o Delphi 7:** a bíblia. São Paulo: Makron Books, 2003. 896 p.

FURLAN, Davi. **Modelagem de objetos através da UML** – the unified modeling language. São Paulo: Makron Books, 1998. 329 p.

GERBER, Luciano. **Paradigmas de linguagens de programação**.[Cachoeira do Sul], [2000]. Disponível em: <http://www.ulbra.tche.br/~lgerber/Paradigmas/>. Acesso em: 10 out. 2003.

GIRASSOL, ESCOLA. Salvador, [2003]. Disponível em: <http://www.escolagirassol.com.br/>. Acesso em: 30 set. 2003.

KONG, Ana Maragareth. **Pititi,** Portela, [2003]. Disponível em: <http://www.pititi.com/sobrepititi/pititi.htm>. Acesso em: 20 set. 2003.

PRICE, Ana Maria de Alencar; TOSCANI, Simão Sirineo. **Implementação de linguagens de programação**: compiladores. Porto Alegre: Sagra Luzzatto, 2001. 195 p.

QUATRANI, Terry. **Modelagem visual com Rational Rose 2000 e UML**. Tradução Savannah Hartmann. Rio de Janeiro: Ciência Moderna, 2001. 206 p.

REIS, Adriano. **Pesquisas em computação gráfica 3D**, São José dos Campos, [2002]. Disponível em: <http://www.compgrafica3d.eng.br/adrhpg.htm>. Acesso em: 10 ago. 2003.

ROSEMEIRE. **Rose home page**, Londrina, [1998]. Disponível em: http://www.geocities.com/athens/ithaca/8750. Acesso em: 23 jul. 2003.

SEBESTA, Robert W. **Conceitos de linguagens de programação.** Tradução José Carlos Barbosa dos Santos. Porto Alegre: Bookman, 2000. 624 p.

VALENTE, José Armando. Diferente usos do computador na educação. In: VALENTE, José Armando (Org.). **Computadores e conhecimento**: repensando a educação. Campinas, SP: Gráfica Central da Unicamp, 1993. p. 1-23.

VALENTE, José Armando. Logo: mais do que uma linguagem de programação. In: VALENTE, José Armando (Org.). **Liberando a mente**: computador na educação especial. Campinas, SP: Gráfica Central da Unicamp, 1991. p. 32-43.

VALENTE, José Armando; VALENTE, Ann Berger. **Logo**: conceitos, aplicações e projetos. São Paulo: ITAUTEC Informática, 1988. 292 p.