UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

LINGUAGEM VISUAL ORIENTADA POR FIGURAS GEOMÉTRICAS VOLTADA PARA O ENSINO DE PROGRAMAÇÃO: VERSÃO 2.0

FABRÍCIO JOSÉ THEISS

BLUMENAU
2006

2006/2-11

FABRICIO JOSE THEISS

LINGUAGEM VISUAL ORIENTADA POR FIGURAS GEOMÉTRICAS VOLTADA PARA O ENSINO DE PROGRAMAÇÃO: VERSÃO 2.0

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 Ciências da Computação — Bacharelado.

Prof. José Roque Voltolini da Silva - Orientador

BLUMENAU
2006

2006/2-11

LINGUAGEM VISUAL ORIENTADA POR FIGURAS GEOMÉTRICAS VOLTADA PARA O ENSINO DE PROGRAMAÇÃO: VERSÃO 2.0

Por

FABRÍCIO JOSÉ THEISS

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. Paulo César Rodacki Gomes, FURB
Membro:	Prof. Jomi Fred Hübner, FURB

Blumenau, 13 de dezembro de 2006

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

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

À minha noiva, pelo amor, carinho, compreensão e incentivo.

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

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

RESUMO

Este trabalho descreve um processo de manutenção corretiva e adaptativa feita na ferramenta implementada por Alcântara Jr (2003), denominado *Language Tangram Draw* (LTD). O objetivo desta ferramenta é facilitar o ensino de programação para crianças alfabetizadas. A ferramenta contém um ambiente gráfico, denominado editor de figuras. As figuras são formadas pelas peças do jogo Tangram. Ainda, disponibiliza uma linguagem textual de ligação com a programação visual. Uma nova especificação e implementação do LTD foi feita, utilizando a técnica de orientação à objetos. Na implementação foi utilizada a biblioteca gráfica OpenGL. Novas funcionalidades foram acrescentadas, citando-se como principal a inclusão da terceira dimensão.

Palavras-chave: Linguagem visual. Ensino de programação.

ABSTRACT

This work describes a process of corrective and adapted maintenance done in the tool implemented by Alcântara Jr (2003), denominated Language Tangram Draw (LTD). The objective of this tool is to facilitate the programming teaching for alphabetized children. The tool contains a graphic atmosphere, denominated editor of illustrations. The illustrations are formed by the pieces of the Tangram game. Still, it availated a textual language of connection with the visual programming. A new specification and implementation of LTD was made, using the orientation technique to objects. In the implementation was used the graphic library OpenGL. New functionalities were increased, being mentioned as main the inclusion of the third dimension.

Key-words: Visual language. Teaching of programming.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

BNF – *Backus-Naur Form*

FURB – Universidade Regional de Blumenau

GALS – Gerador de Analisadores Léxicos e Sintáticos

LTD – *Language Tangram Draw*

OO – Orientação à Objetos

OpenGL – *Open Graphics Library*

RGB – *Red Green Blue*

TCC – Trabalho de Conclusão de Curso

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Andrade, Hoffmann e Wazlawick (1998) enfatizam que “[...] o computador deve ser utilizado não apenas como ferramenta pedagógica, ou como substituto do professor: a proposta é criar ambientes enriquecidos com a tecnologia [...]”, através dos softwares educacionais.

Um fator essencial na visão de software educacional é que este seja interativo, isto é, o aluno deve estar em plena comunicação com o software e vice-versa. Esta interação pode ocorrer através de resolução de problemas, representações gráficas e participação ativa no próprio ambiente.

Os softwares educacionais para a área de computação apresentam várias ferramentas voltadas para o ensino de programação de computadores. Dentre essas, destacam-se as que usam linguagem de programação visual, como o *Language Tangram Draw* (LTD).

O LTD, descrito em Alcântara Jr (2003), tem como objetivo facilitar o ensino de programação para crianças alfabetizadas. A ferramenta contém um ambiente gráfico, onde as figuras são formadas pelas peças do jogo Tangram, e além disso, disponibiliza uma linguagem textual de ligação com a programação visual. O ambiente é composto de duas áreas de trabalho, um editor de figuras e um de texto. Através do editor de figuras, textos (comandos) são gerados, sendo que estes podem também serem alterados, refletindo no editor de figuras.

Conforme o acima descrito, este trabalho propõe a re-implementação da ferramenta descrita em Alcântara Jr (2003) utilizando novas tecnologias, ou seja, a especificação passa a ser orientada a objetos e não mais estruturada como era a anterior e o uso da biblioteca OpenGL, visto que na ferramenta anterior são utilizados recursos próprios do Ambiente de Programação Delphi para implementar a parte gráfica. Além disso, este trabalho pretende corrigir erros existentes na versão descrita em Alcântara Jr (2003) e incluir novas funcionalidades como: mecanismos de seleção de figuras e peças; permitir que o editor de figuras tenha tamanho dinâmico e seja separado do editor de texto e a inclusão da terceira dimensão.

O ambiente, através destas novas funcionalidades, pretende facilitar a formalização da descrição de situações do mundo real, objetivando melhorar o apoio ao ensino de programação. Ainda, fatores relacionados à interface são verificados, objetivando a melhoria da comunicação entre o usuário e a ferramenta.

A relevância deste trabalho está relacionada com a interdisciplinaridade que o software

envolve, abrangendo diversas áreas como: educacional; de interface homem computador; linguagens de programação e computação gráfica, envolvendo o uso da biblioteca OpenGL. Justifica-se ainda a importância deste trabalho pelo fato de ser uma continuidade de outro já desenvolvido no curso de Ciências da Computação da FURB.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é re-implementar a ferramenta descrita em Alcântara Jr (2003), denominado LTD, incluindo novas funcionalidades.

Os objetivos específicos do trabalho são:

a) incluir a terceira dimensão no LTD, permitindo a visualização em perspectiva (profundidade);
b) definir comandos para manuseio da terceira dimensão;
c) corrigir funções que não funcionam de acordo com o esperado, citando como exemplo a movimentação de uma peça após sua rotação, o que ocasiona um problema de posicionamento não exato da peça;
d) disponibilizar mecanismos de seleção de figuras e de peças;
e) permitir que o editor de figuras tenha tamanho dinâmico e seja separado do editor de texto.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta uma introdução sobre alguns assuntos importantes para a realização deste trabalho, tais como: linguagens de programação, linguagens voltadas ao ensino de programação, Tangram, computação gráfica usando a biblioteca OpenGL, interface gráfica com o usuário, ferramenta GALS, LTD e por fim apresenta dois (2) trabalhos correlatos. No capítulo 3 são abordados detalhes sobre o desenvolvimento dessa nova versão da ferramenta. Por fim, o capítulo 4 traz as conclusões e sugestões para extensões no trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

O trabalho em questão visa facilitar o ensino de linguagens de programação de computadores. Para realização e entendimento do trabalho proposto, neste capítulo são apresentados alguns aspectos relacionados, os quais são: linguagens de programação, linguagens voltadas ao ensino de programação, Tangram, computação gráfica usando a biblioteca OpenGL, interface gráfica com o usuário, ferramenta GALS, LTD e por fim são apresentados dois (2) trabalhos correlatos.

2.1 LINGUAGENS DE PROGRAMAÇÃO

As linguagens de programação podem ser classificadas em: imperativas, funcionais, lógicas (declarativas), orientadas a objeto e visuais.

Linguagens imperativas são aquelas que possuem algoritmos bem detalhados, onde a execução dos seus comandos ou instruções, que servem para realizar transformações sobre os dados, seguem uma seqüência/ordem (SEBESTA, 2000, p. 278). Como exemplo desse tipo de linguagem cita-se o Formula Translator (Fortran). Fortran foi uma das primeiras linguagens de alto nível imperativas, destinada inicialmente ao desenvolvimento de aplicações científicas.

O propósito de uma linguagem funcional, segundo Sebesta (2000, p. 541), é imitar as funções matemáticas em seu maior grau possível. A mais antiga das linguagens funcionais é o List Processor (Lisp).

Para Sebesta (2000, p. 38-40), as linguagens de programação lógicas baseiam-se em regras e não possuem uma ordem de execução dos comandos ou instruções. Um exemplo de uma linguagem lógica é o Programming in Logic (Prolog).

Uma linguagem puramente orientada a objeto, de acordo com Sebesta (2000, p. 418), deve oferecer três recursos essenciais: tipos de dados abstratos, herança e um tipo particular de vinculação dinâmica. Toda computação em uma linguagem orientada a objeto pura é feita por uma mesma técnica, que é enviar “[...] uma mensagem a um objeto para invocar um de seus métodos. Uma resposta a uma mensagem é um objeto que retorna o valor da computação do método.” (SEBESTA, 2000, p. 422-424). Portanto, a essência da programação orientada a objeto é identificar os objetos do mundo real, criando as comunicações necessárias entre eles.

Um exemplo de linguagem orientada a objeto é a Smalltalk (SMALLTALK-80, 2006).

As linguagens de programação visual partem do princípio de que gráficos são mais fáceis de serem entendidos do que textos. Especificar um programa por meio de diagramas e outros recursos gráficos tende a tornar a própria programação mais fácil, permitindo a geração de programas por usuários sem muitas habilidades. Nem todo uso de gráficos e diagramas em programação pode caracterizar a linguagem como linguagem visual. Em alguns casos, o mais adequado é chamá-las de ferramentas de programação visual. As linguagens de programação visual podem ainda ser classificadas em híbridas (somente parte de sua especificação é determinada de forma visual) e puras (especificada exclusivamente por meio de gráficos e diagramas) (GUDWIN, 1997, p. 13-15).

2.1.1 Especificações de linguagens

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 (SEBESTA, 2000, p. 115).

Segundo Sebesta (2000, p. 115), “BNF é suficientemente poderosa para descrever [...] listas de construções similares à ordem em que diferentes construções devem aparecer nas estruturas alinhadas em qualquer profundidade [...]”.

A BNF usa abstrações para estruturas sintáticas. Uma simples instrução de atribuição na linguagem C, por exemplo, poderia ser representada pela abstração <atribuição>. A definição de <atribuição> é mostrada no quadro 1.

```
<atribuição> → <var> = <expressão>
```

Quadro 1 – Exemplo de definição da regra de atribuição

Ao todo, a definição é chamada regra ou produção. No exemplo da regra que foi apresentada no quadro 1, as abstrações <var> e <expressão> evidentemente devem ser definidas antes que a definição <atribuição> torne-se útil.

Um exemplo de sentença cuja estrutura sintática é descrita pela regra de <atribuição> vista no quadro 1 é apresentada no quadro 2.

```
total = sub1 + sub2
```

Quadro 2 – Exemplo de sentença definida na regra atribuição

2.2 LINGUAGENS VOLTADA AO ENSINO DE PROGRAMAÇÃO

Uma linguagem voltada ao ensino de programação deve produzir um código de fácil leitura e ao mesmo tempo preciso, ou seja, devem ser apresentadas as construções lógicas básicas sem gerar dúvidas de interpretação. Selecionar a linguagem mais adequada para o ensino de programação não é uma tarefa simples. Baranauskas et al. (1999, p. 54) cita Prolog, Logo e Pascal como linguagens que possuem características destinadas ao ensino.

A idéia do Prolog é incentivar os usuários a pensarem, deixando de lado o formalismo das linguagens de programação. Um programa em Prolog divide-se em um fato ou uma regra. Um fato apresenta uma verdade incondicional (cita-se como exemplo que Maria é mulher e mãe de Pedro que é homem), enquanto que as regras são condições para que certa declaração (fato) seja considerada verdadeira (cita-se como exemplo que Marcos é homem e é casado com Maria, Pedro é filho de Marcos) (SEBESTA, 2000, p. 38-40).

O quadro 3 apresenta a relação “genitor” (por exemplo Tom é genitor de Bob). O Prolog permite fazer vários tipos de perguntas sobre estes fatos. Para cada pergunta, é mostrado “Yes” na tela caso consiga uma prova para a pergunta ou “No” caso contrário. Por exemplo, para saber se existe uma prova de que Pam é genitora de Bob, basta digitar a pergunta vista no quadro 4.

```
genitor(pam,bob). % Pam é mãe de Bob
genitor(tom,bob). % Tom é pai de Bob
genitor(tom,liz).

genitor(bob,ana).
genitor(bob,pat).

genitor(liz,bill).

genitor(pat,jim).
```

Quadro 3 – Exemplo de um programa em `Prolog`

```
?- genitor(pam,bob).

    Yes.
```

Quadro 4 – Consulta em um programa `Prolog`

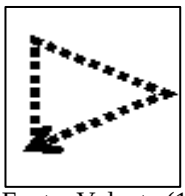
A resposta será “Yes”, pois há uma prova trivial (Quadro 3) de que Pam é genitora de Bob (é uma das premissas).

O Logo utiliza a figura de uma tartaruga que caminha e deixa rastros, que são representados graficamente. Disponibiliza comandos que permitem determinar a posição e direção da tartaruga, além de utilizar uma sintaxe simples que possibilita um fácil entendimento do usuário sobre o uso da ferramenta (VALENTE, 1991, p. 32-43). 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 3 e o resultado da execução na figura 1.

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

Fonte: Valente (1991, p. 36).
Quadro 5 – Triângulo eqüilátero criado com Logo



Fonte: Valente (1991, p. 36).
Figura 1 – Resultado dos comandos vistos no quadro 3

O Pascal foi desenvolvido por Niklaus Wirth objetivando o ensino de programação de computadores. O objetivo de Wirth era disponibilizar uma nova linguagem voltada para o ensino de programação que fosse simples e capaz de incentivar/auxiliar a criação de programas simples e bem legíveis, visando a utilização de boas técnicas de programação. Programar em Pascal é descrever a solução de problemas obedecendo suas regras de sintaxe e utilizando-se de um conjunto de recursos tais como repetição, seleção e atribuição (FARRER et al., 1999, p. 1-10). O Quadro 4 mostra um exemplo de programa em Pascal, com comandos de atribuição e de repetição. Um exemplo de atribuição pode ser visto na linha cinco (5) e um comando de repetição na linha sete (7) do quadro 6.

```
 1  Program Exemplo;
 2  var
 3   a,b : integer;
 4  begin
 5    a:= 1;
 6    b:= 2;
 7    while a < 5 do
 8    begin
 9        b:= b * 2;
10        a:= a + 1;
11    end;
12  end.
```

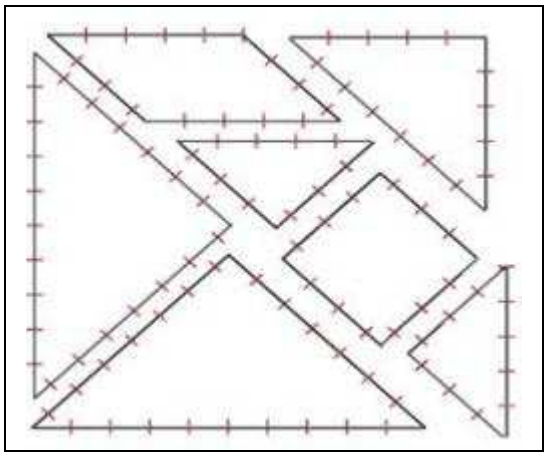
Quadro 6 – Exemplo de programa em Pascal

2.3 TANGRAM

Pouco se sabe a respeito do criador ou até mesmo sobre a origem do Tangram. Existem porém algumas teorias onde todas têm um ponto em comum: a origem chinesa.

O Tangram, conhecido na China por volta do século VII a.C. como as “Sete Taboas da Astúcia”, é um jogo baseado em figuras. A versão mais contada sobre a sua origem é a de que o monge Tai-Jin deu uma missão ao seu discípulo Lao-Tan, que consistia em percorrer o mundo e registrar numa placa de porcelana toda a beleza que encontrasse. Muito emocionado por ter sido escolhido para essa missão, o discípulo deixou cair a placa quadrada de porcelana, que quebrou-se em sete pedaços, como as peças do Tangram (LONGHI, 2004).

O Tangram é um quebra-cabeça geométrico (bidimensional) formado por 7 peças (Figura 2), de formas geométricas simples (5 triângulos, 1 quadrado e 1 trapézio) que juntas formam um quadrado. Combinando essas peças originam-se outras figuras, podendo representar uma grande variedade do mundo real.

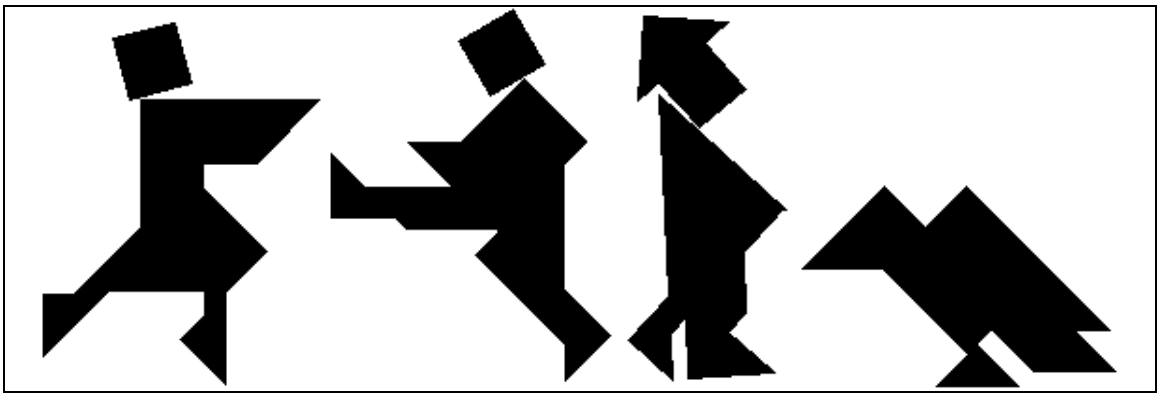


Fonte: Kong (2003).

Figura 2 – Peças do Tangram

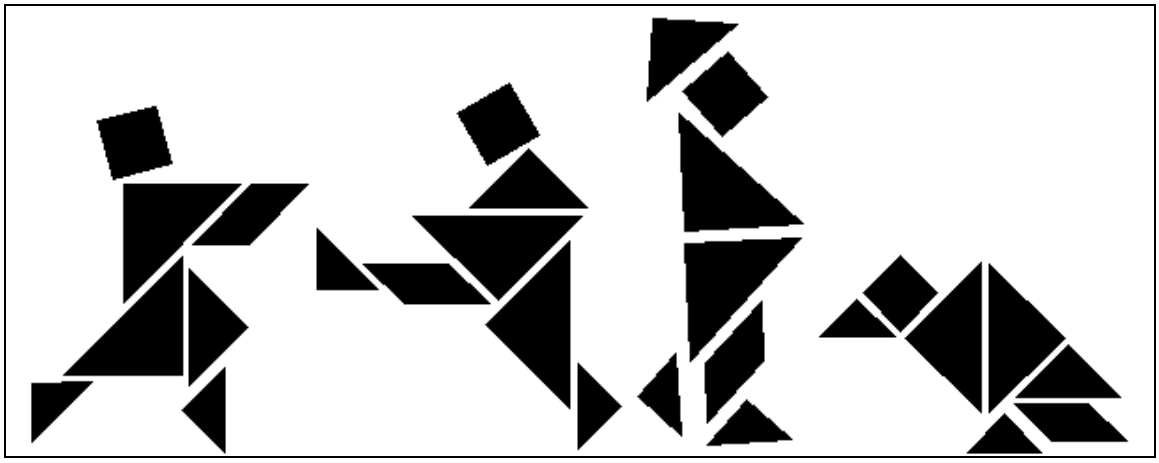
A seguir são apresentados desenhos feitos com as sete (7) peças do Tangram. Na figura 3, da esquerda para a direita tem-se respectivamente um corredor, um pontapé, um chinês e um pássaro.

Para identificar as sete (7) peças de cada desenho, observe a figura 4. Essa mostra como foi montado cada um dos desenhos da figura 3.



Fonte: Fernandes (2004).

Figura 3 – Desenhos utilizando as sete (7) peças do Tangram



Fonte: Fernandes (2004).

Figura 4 – Identificação das peças dos desenhos apresentados na figura 3

2.4 COMPUTAÇÃO GRÁFICA USANDO A BIBLIOTECA OPENGL

Segundo Persiano e Oliveira (1989, p. 1-23), a computação gráfica é a área da ciência da computação que estuda a geração, manipulação e interpretação de representações visuais, a partir das especificações geométricas.

Em computação gráfica, organizar as operações necessárias para converter objetos definidos em um espaço tridimensional para um espaço bidimensional (tela do computador) é uma das principais dificuldades no desenvolvimento de aplicações. Para isso, aspectos como transformações e definição das dimensões de *viewport* (parte da janela onde a imagem é desenhada) devem ser considerados (WANGENHEIM, 2005).

O uso de bibliotecas que dão apoio/suporte na implementação de softwares na área de computação gráfica são comuns. Cita-se como exemplo de biblioteca a OpenGL.

A Silicon Graphics introduziu em 1992 a biblioteca OpenGL, com intuito de

desenvolver uma Interface de Programação de Aplicativos, ou do inglês *Application Programming Interface* (API), gráfica independente de dispositivos de exibição. Segundo Wangenheim (2005), OpenGL "[...] é uma biblioteca de rotinas gráficas de modelagem, manipulação de objetos e exibição tridimensional que permite a criação de aplicações que usam Computação Gráfica", sendo bastante rápida e portável para vários sistemas operacionais. As suas funções utilizadas para desenhar um ponto na tela, por exemplo, possuem os mesmos nomes e parâmetros em todos os sistemas operacionais onde a mesma foi implementada e produzem o mesmo efeito de exibição em cada um destes sistemas.

Diante das funcionalidades da OpenGL, a biblioteca tem se tornado um padrão no desenvolvimento de aplicações tais como: jogos, aplicações científicas, comerciais, entre outras.

2.4.1 Projeções em OpenGL

Existem dois tipos básicos de projeções possíveis em OpenGL: ortográfica e perspectiva.

2.4.2 Projeção ortográfica

Neste tipo de projeção simplesmente descarta-se uma das dimensões. Por exemplo, um ponto (x, y, z) será mapeado para simplesmente (x, y). Este tipo de projeção é bastante utilizada em projetos arquitetônicos e de peças mecânicas, pois os desenhos assim obtidos podem ser utilizados para medir as dimensões dos objetos representados. Na projeção ortográfica, não importa a distância entre o objeto e o observador, o objeto vai sempre ser visualizado do mesmo tamanho, diferentemente da projeção perspectiva (WALTER, 2004).

Em OpenGL a chamada que especifica uma projeção ortográfica no espaço 2D é mostrada no quadro 7. No espaço 3D a chamada é mostrada no quadro 8.

```
gluOrtho2D( double left, double right, double bottom, double top );
```

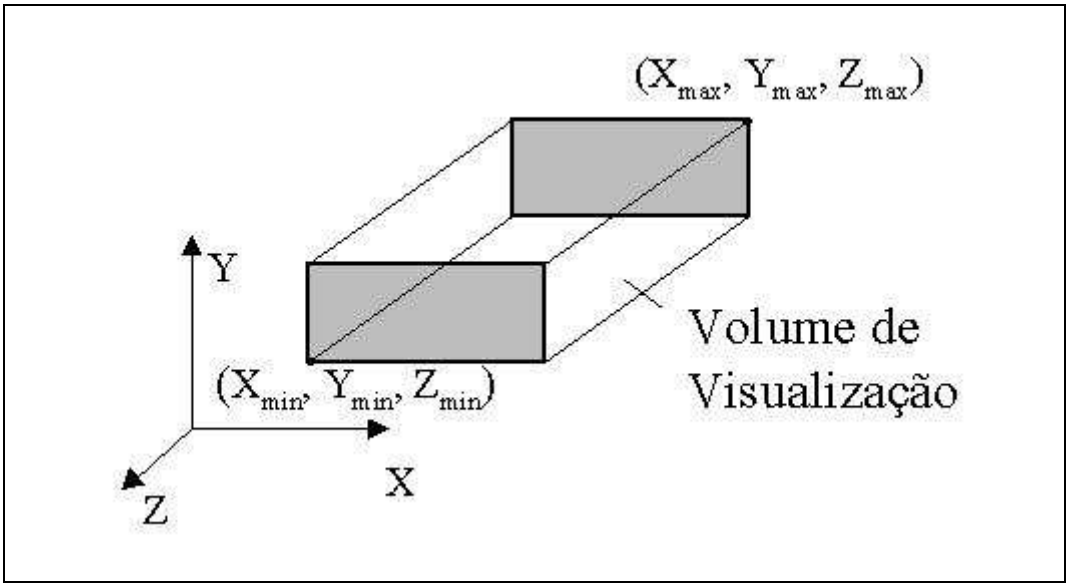
Quadro 7 – Chamada de uma projeção ortográfica em 2D

Os parâmetros do comando `gluOrtho2D`, *`left`* e *`right`*, especificam os limites mínimo e máximo no eixo X; analogamente, *`bottom`* e *`top`* especificam os limites mínimo e máximo no eixo Y.

```
glOrtho(double xmin, double xmax, double ymin, double ymax, double
zmin, double zmax);
```

Quadro 8 – Chamada de uma projeção ortográfica em 3D

A chamada mostrada no quadro 8 define no espaço 3D um paralelepípedo, ou seja, somente serão visualizados os objetos que estiverem dentro deste paralelepípedo, denominado o volume de visualização (Figura 5). Os parâmetros `xmin` e `xmax` especificam as coordenadas esquerda e direita, respectivamente, dos planos de corte verticais. Os parâmetros `ymin` e `ymax` especificam as coordenadas inferior e superior, respectivamente, dos planos de corte horizontais. Já os parâmetros `zmin` e `zmax`, por sua vez, especificam a coordenada mais próxima e mais distante do observador, respectivamente, no eixo de profundidade.



Fonte: Walter (2004).

Figura 5 – Volume de visualização (projeção ortográfica)

A figura 6 mostra um exemplo de aplicação utilizando a projeção ortográfica. Note que as coordenadas “Z” do triângulo escuro são iguais a 0 (Zero), enquanto que as coordenadas “Z” do triângulo vermelho são iguais a 1, logo quem tem o maior “Z” é o que está sobrepondo o outro, neste caso o triângulo claro.

Figura 6 – Exemplo de aplicação de projeção ortográfica (caso a)

Na figura 7, tem-se o mesmo exemplo apresentado na figura 6, só que agora as coordenadas “Z” do triângulo escuro estão iguais a 2 enquanto que as coordenadas de “Z” do triângulo claro são iguais a 1, ou seja, agora o triângulo escuro (maior “Z”) está sobrepondo o mais claro.

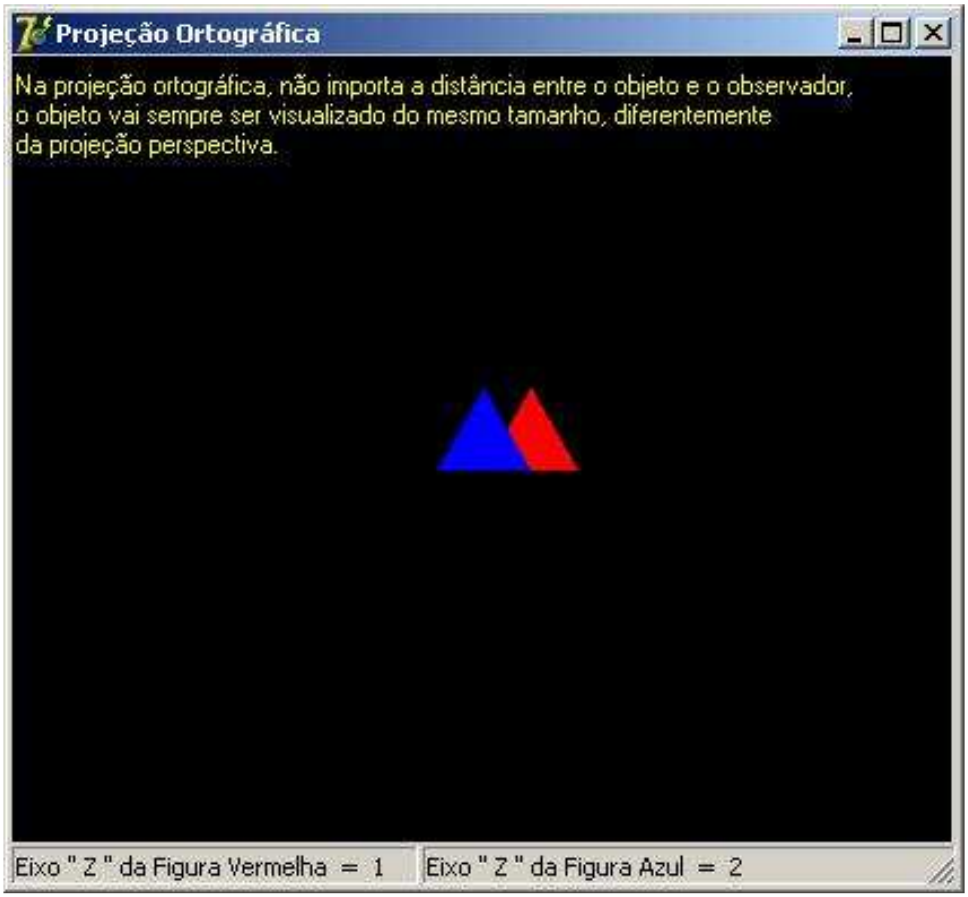


Figura 7 – Exemplo de aplicação de projeção ortográfica (caso b)

2.4.2.1 Projeção perspectiva

Segundo Walter (2004), neste tipo de projeção ocorre o fenômeno de *foreshortening*, ou seja, objetos mais longes do observador irão aparecer menores na imagem (Figura 8).

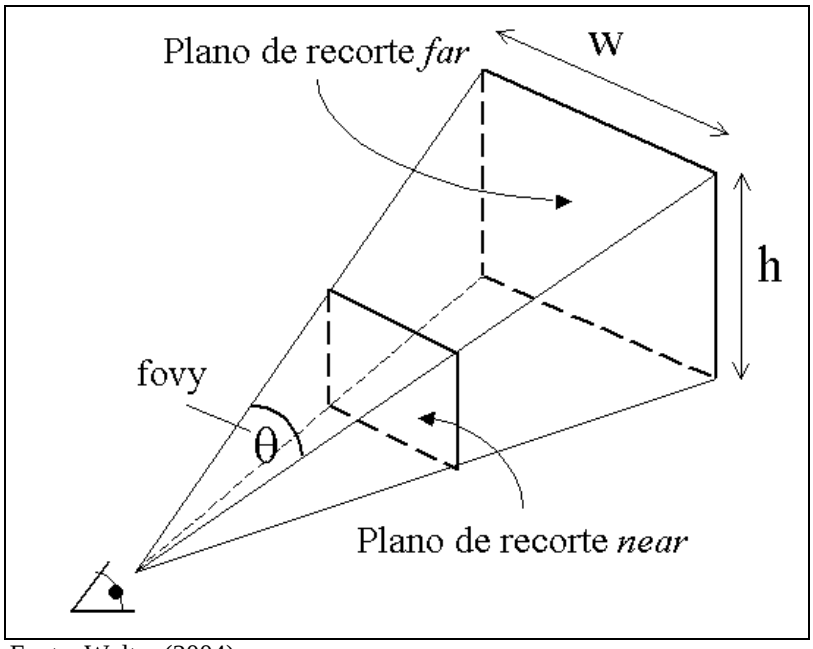
Há duas maneiras de especificar uma projeção perspectiva em OpenGL (Quadro 9).

```
gluPerspectiva( double fovy, double aspect, double near, double far);

glFrustum(double left, double right, double bottom, double top,
double near, double far);
```

Quadro 9 – Maneiras para especificar uma projeção perspectiva

Os parâmetros do comando `gluPerspectiva` descrito no quadro 9 são: *`fovy`* que especifica o ângulo do *field-of-view*, ou seja, do campo de visão da câmera sintética; *`aspect`* que especifica a razão entre a largura w e a altura h da janela; *`near`* que especifica a distância do plano de recorte próximo à câmera e *`far`* que especifica a distância do plano de recorte longe da câmera.



Fonte: Walter (2004).

Figura 8 – Plano de recorte (projeção perspectiva)

Os parâmetros do comando `glFrustum` vistos no quadro 9 são: *`left`* e *`right`* que especificam as coordenadas esquerda e direita, respectivamente, dos planos de corte verticais; *`bottom`* e *`top`* que especificam as coordenadas inferior e superior, respectivamente, dos planos de corte horizontais; e *`near`* e *`far`*, por sua vez, especificam a coordenada mais próxima e mais distante do observador, respectivamente, no eixo de profundidade.

Na projeção perspectiva existe ainda o comando `gluLookAt` (Quadro 10).

```
gluLookAt(double eyex, double eyey, double eyez, double centerx,
double centery, double centerz, double upx, upy, upz );
```

Quadro 10 – Comando que define a câmera virtual

O comando `gluLookAt` apresentado no quadro 10 define a câmera virtual. Os argumentos indicam a sua posição e para onde ela está direcionada. Os parâmetros *`eyex`*, *`eyey`* e *`eyez`* são usados para definir as coordenadas x, y e z, respectivamente, da posição da câmera (ou observador); *`centerx`*, *`centery`* e *`centerz`* são usados para definir as coordenadas x, y e z, respectivamente, da posição do ponto de foco ou alvo ou para onde o observador está olhando (normalmente, o centro da cena); *`upx`*, *`upy`* e *`upz`* são as coordenadas x, y e z, que estabelecem a "vertical" da câmera.

Um exemplo de aplicação utilizando a projeção perspectiva é apresentado na figura 9. Note que as coordenadas "Z" do triângulo escuro são iguais a -4, enquanto que o "Z" do triângulo claro é igual a 1. A idéia que se tem é que o triângulo escuro está mais longe, ou seja, tem-se a noção de profundidade, enquanto que o triângulo claro parece estar mais próximo da tela (câmera).

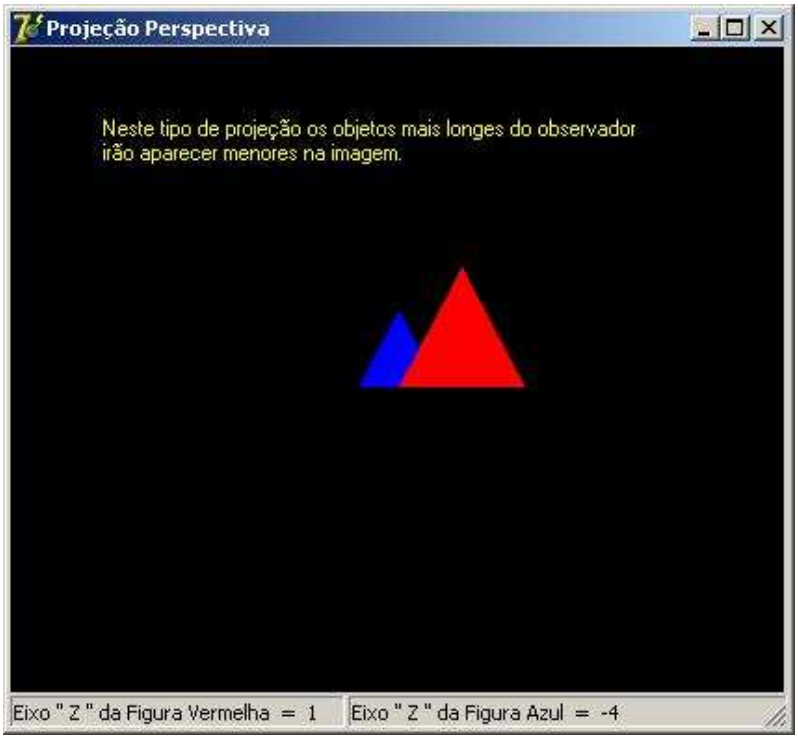


Figura 9 – Exemplo de aplicação de projeção perspectiva (caso a)

Na figura 10 é apresentado o mesmo exemplo mostrado na figura 9, só que agora as coordenadas “Z” do triângulo escuro estão iguais a 4, enquanto que as coordenadas “Z” do triângulo claro continuam iguais a 1. Note que agora é o triângulo escuro que está mais perto da câmera.

Figura 10 – Exemplo de aplicação de projeção perspectiva (caso b)

2.5 INTERFACE GRÁFICA COM O USUÁRIO

“Para os usuários comuns de computadores, uma interface é a forma de apresentação de programas ou sistemas. Com o avanço na capacidade de processamento das máquinas, aconteceu uma grande mudança na interface dos programas e sistemas” (INTERFACE, 2006).

Conforme Interface (2006), a tela (geralmente) preta e baseada em texto (Figura 11) foi trocada por janelas, botões, abas, caixas de texto ou de checagem, ícones, etc (Figura 12). Esta nova apresentação gráfica, também conhecida como "ambiente gráfico", tornou o uso do computador mais amigável.

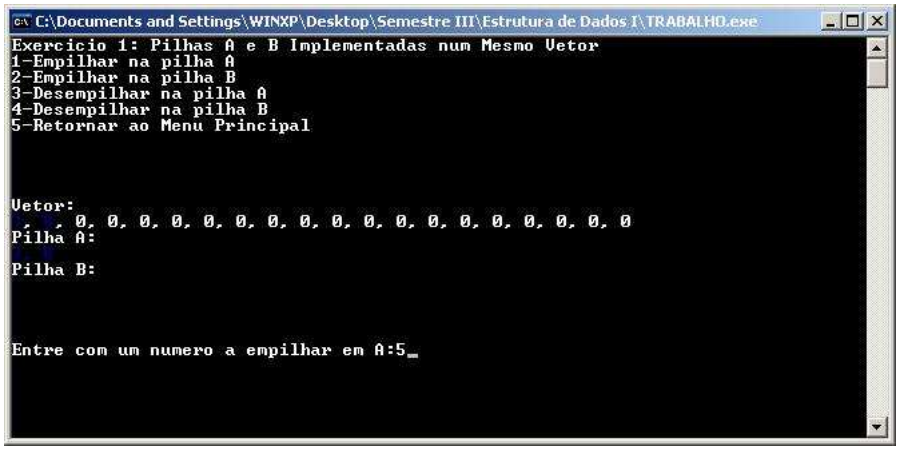


Figura 11 – Interface em tela preta baseada em texto

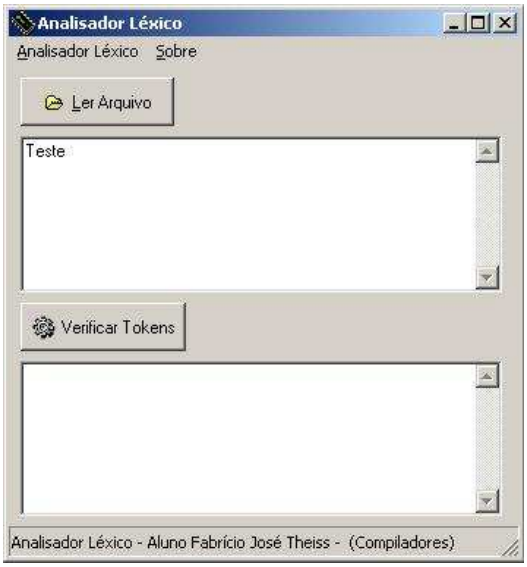


Figura 12 – Interface em ambiente gráfico

Entretanto, de acordo com Interface (2006), a forma da interface (seja gráfica ou baseada em texto) não pode ser responsabilizada pela facilidade ou dificuldade de interação com um sistema ou programa. Uma interface está diretamente ligada a funcionalidade ou comportamento do sistema. Uma mesma tela, com os mesmos componentes e imagens, pode

ter um comportamento diferente sob os mesmos "estímulos" do usuário, ou seja um aperto de botão, um arrastar e soltar.

2.6 FERRAMENTA GALS

GALS é um ambiente para a geração de analisadores léxicos e sintáticos (GESSER, 2003, p. 1-8). Os aspectos léxicos de uma especificação GALS são definidos pela declaração de definições regulares e pela declaração dos *tokens*. As definições regulares atuam como expressões auxiliares para a definição de *tokens*.

A primeira etapa na definição da linguagem consiste na especificação das definições regulares (quadro 11), que servem como expressões auxiliares para a definição dos *tokens*.

```
L  : [a-z] reconhece (a, b, c, ...z)
D  : [0-9] reconhece (0,1,...,9)

comentario : /[/][^\n]+  reconhece como comentário tudo que
está após // na mesma linha.

ws : [\t\n]     \t   reconhece a tabulação (Tab)
                \n  reconhece a quebra da Linha
```

Quadro 11 – Exemplo de definições regulares

Existem duas formas para definição dos *tokens*. Uma delas é idêntica à declaração de uma definição regular (quadro 12). Ainda, pode-se definir um *token* como sendo um caso particular de um outro *token*. Nestes casos, sempre que o analisador identifica o *token* base, ele procura pelo valor do *token* em uma lista de casos especiais. Se for encontrado, o caso especial é retornado, senão é produzido o *token* base (GESSER, 2003, p. 50-81). Esta forma de declaração é apresentada no quadro 13.

```
id : {L}*  →  reconhece zero ou mais (a, b, c, ...z)
num : {D}+  →  reconhece um ou mais (0,1,...,9)
:{comentário} →  comentários
:{ws} →  reconhece as tabulações e quebras de Linhas
```

Quadro 12 – Exemplo de definições de *tokens*

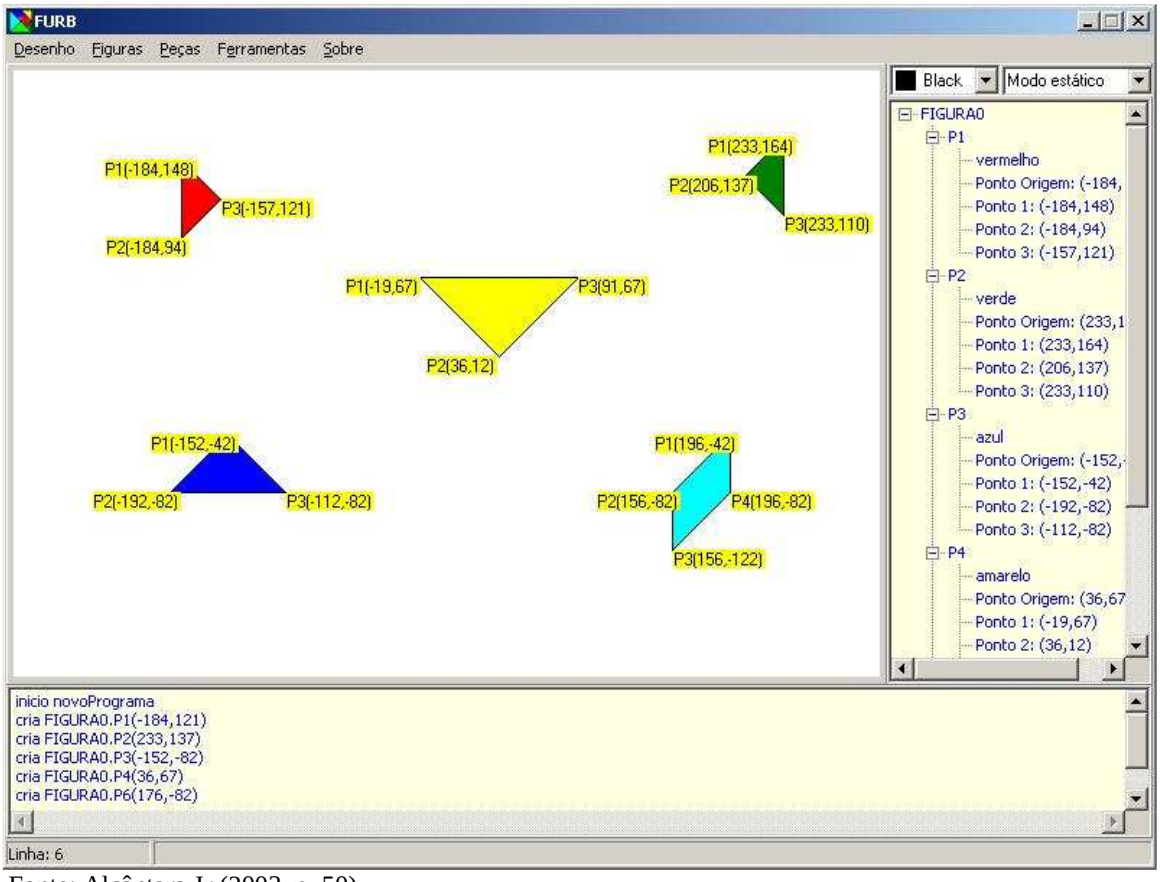
```
//Palavras reservadas
inicio = id : "inicio"
fim    = id : "fim"
```

Quadro 13 – *Token* como caso particular de outro *token*

2.7 LTD

O trabalho descrito em Alcântara Jr (2003), denominado LTD, consiste em uma ferramenta para auxiliar na introdução do ensino de programação de computadores para crianças alfabetizadas.

As principais características do LTD são: possui um ambiente gráfico (Figura 13) que permite a criação de desenhos com formas geométricas pré-definidas; disponibiliza uma linguagem textual de ligação com a programação visual; possui dois modos de operação, o modo estático e o modo dinâmico; permite a criação de movimentos dos objetos geométricos e recuperação do que foi desenhado a partir do texto gerado.



Fonte: Alcântara Jr (2003, p. 50).

Figura 13 – Ambiente gráfico do LTD

A ferramenta implementada possui algumas limitações, entre as quais são relatadas:

a) o tamanho da área do editor gráfico é fixo, assim como o tamanho do editor de texto, que permite apenas visualizar poucas linhas de programa. Para visualizar outras partes do programa, utiliza-se da facilidade de rolagem;
b) o protótipo não oferece opção para selecionar uma figura;
c) não foi implementado um menu “Ajuda”.

Além das restrições apresentadas, algumas funções não funcionam adequadamente no protótipo, criando problemas, tais como: posicionamento não exato das peças (em alguns casos) e impossibilidade de inclusão das mesmas em determinadas situações.

O problema do posicionamento não exato ocorre quando se cria uma figura com as peças (por exemplo P1 e P2), altera-se o modo para dinâmico (Figura 14). Logo após, rotaciona-se essas peças noventa (90) graus cada uma (Figura 15), e em seguida move-se as mesmas de posição, deixando-as visualmente dispostas de maneira semelhante à que estavam antes da movimentação (Figura 16). Pode-se observar na figura 17 que ao executar o código gerado, as peças não irão terminar na posição correta.

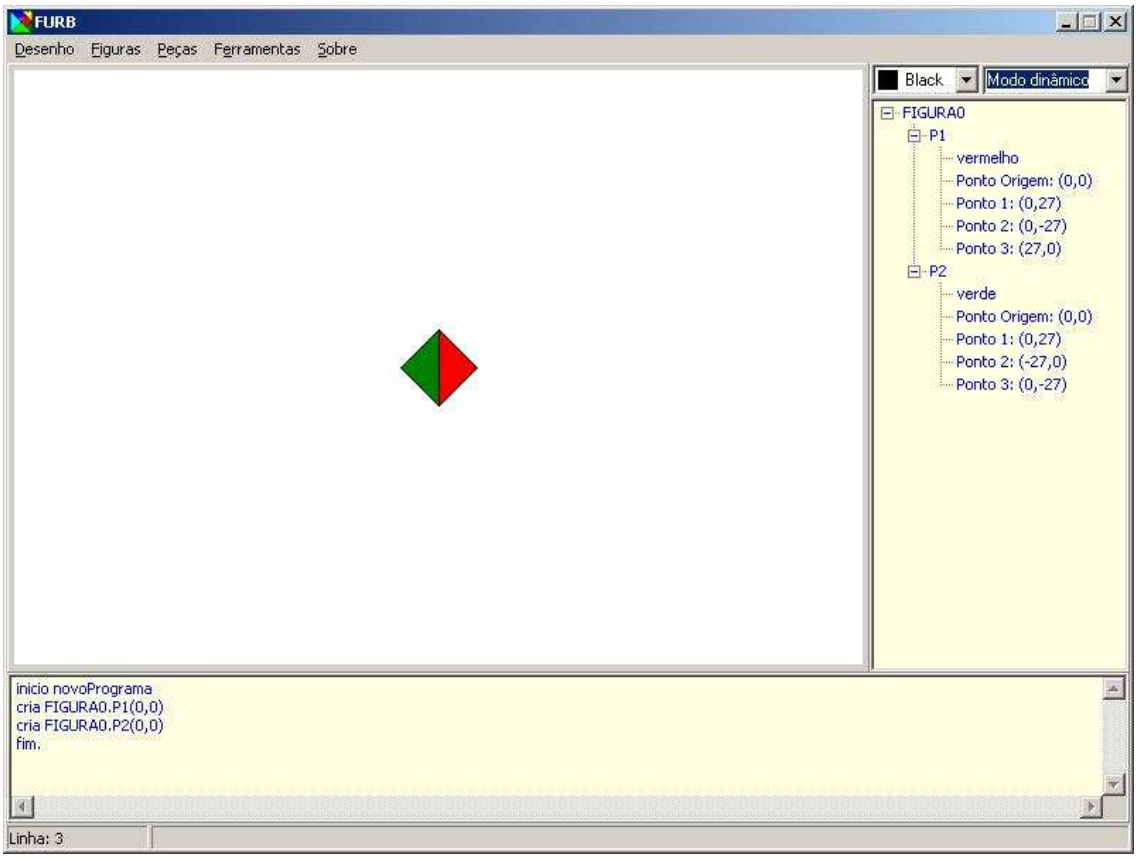


Figura 14 – Problema do posicionamento não exato das peças (caso a)

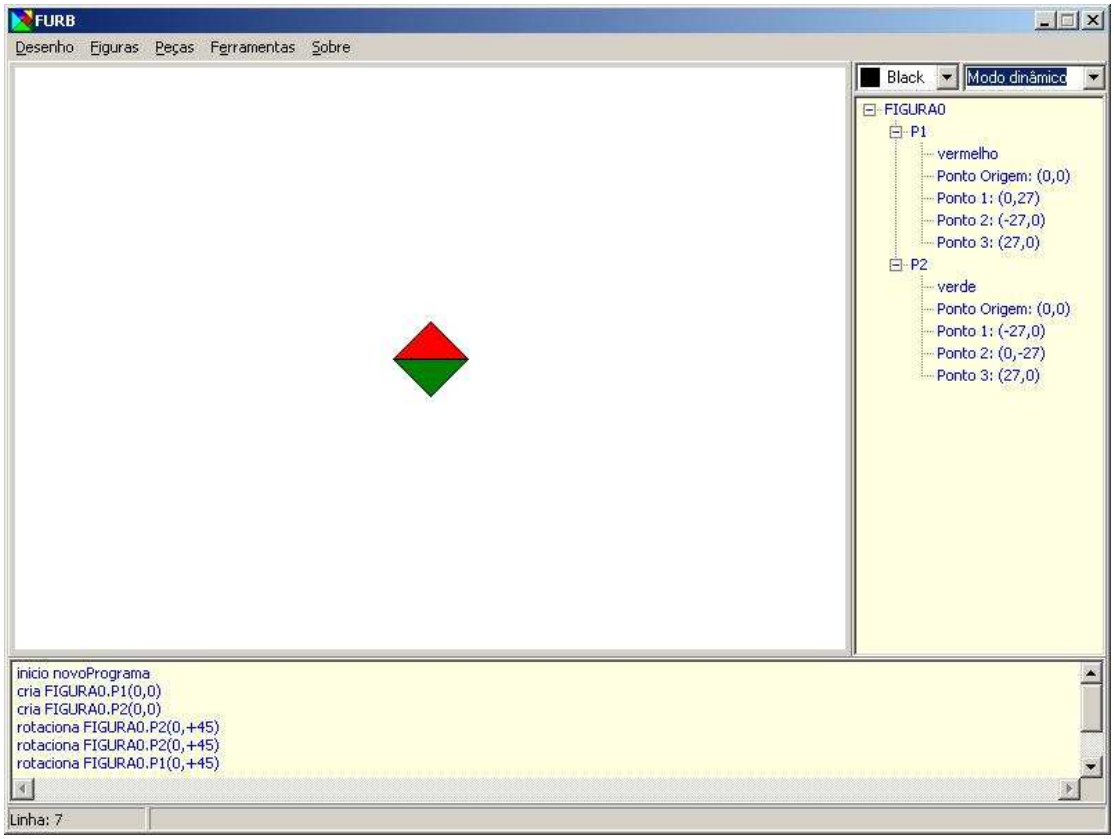


Figura 15 – Problema do posicionamento não exato das peças (caso b)

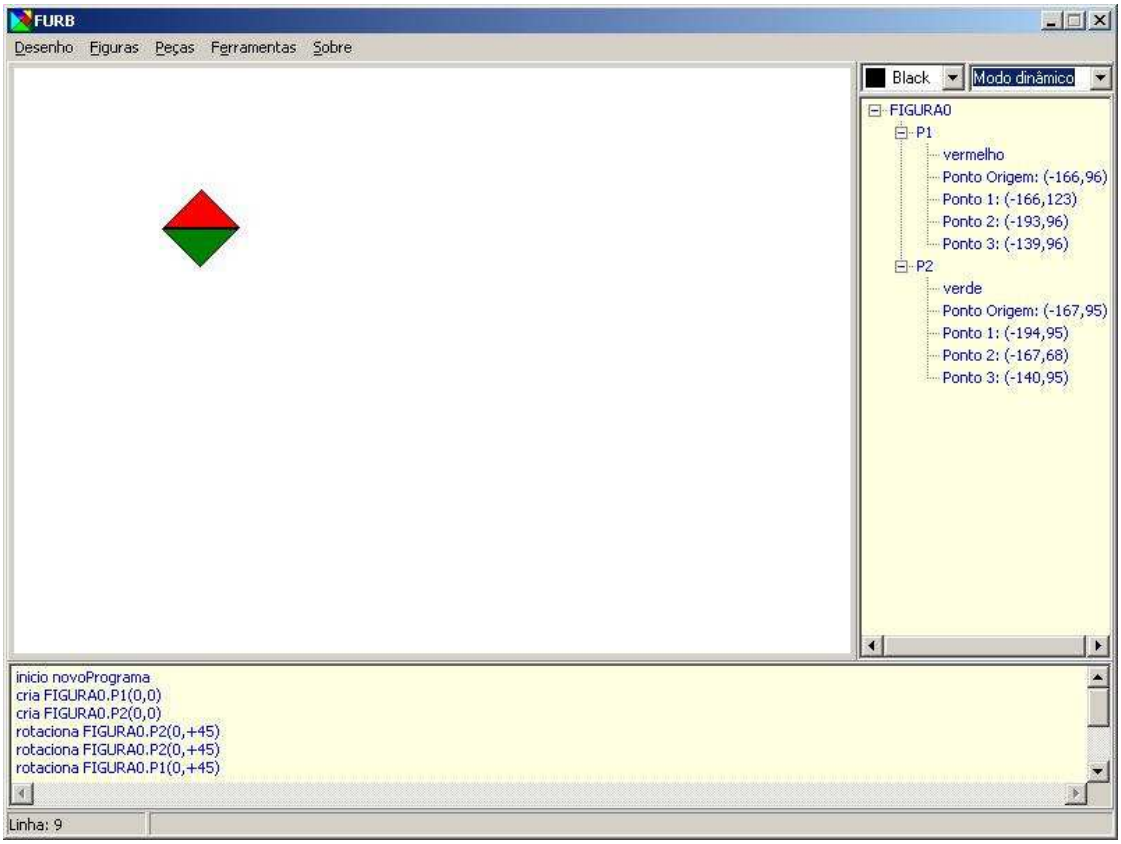


Figura 16 – Problema do posicionamento não exato das peças (caso c)

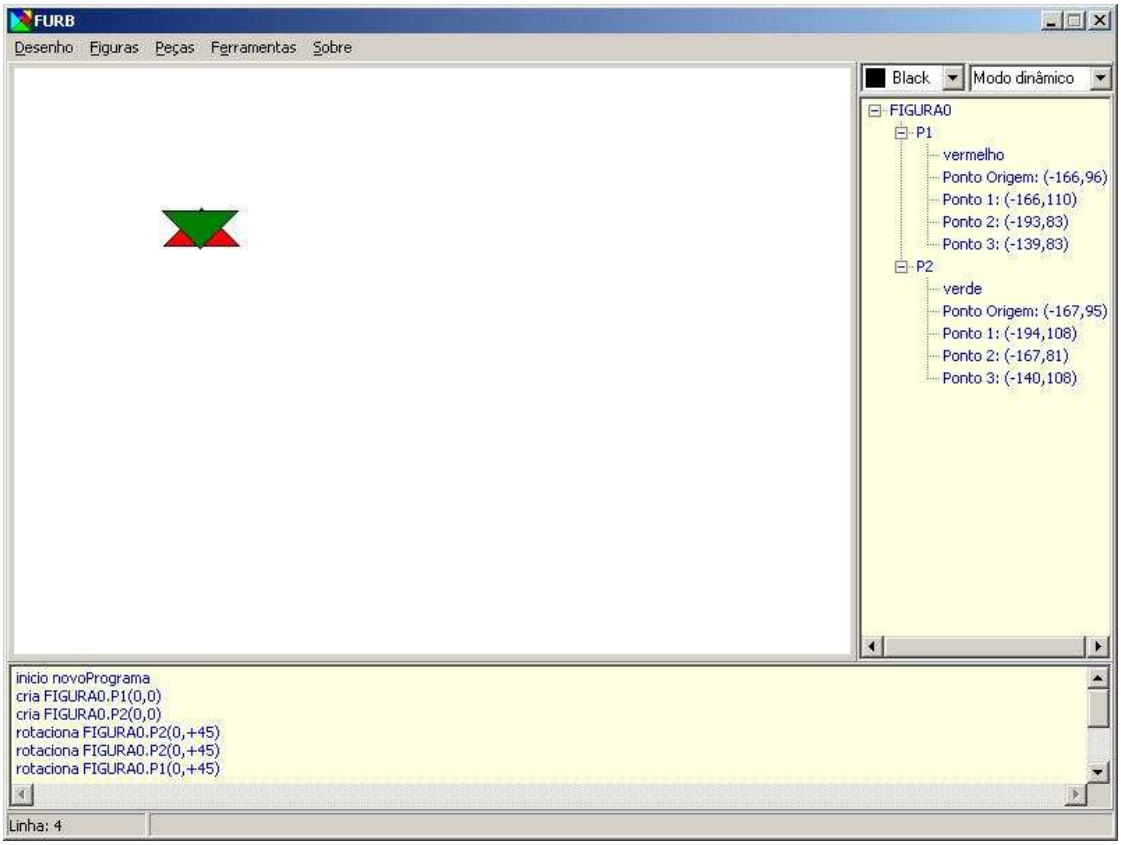


Figura 17 – Problema do posicionamento não exato das peças (caso d)

A impossibilidade de inclusão de peças acontece no modo estático e pode ser observada na figura 18 (obtida através do LTD). O usuário cria uma figura “FIGURA0” com duas peças (P1 e P3). Após, cria uma nova figura “FIGURA1” com uma peça “P5”. Neste caso, não é possível inserir uma nova peça na figura “FIGURA0”. Somente para a última figura inserida “FIGURA1” fica disponível a inserção de peças pelo Menu.

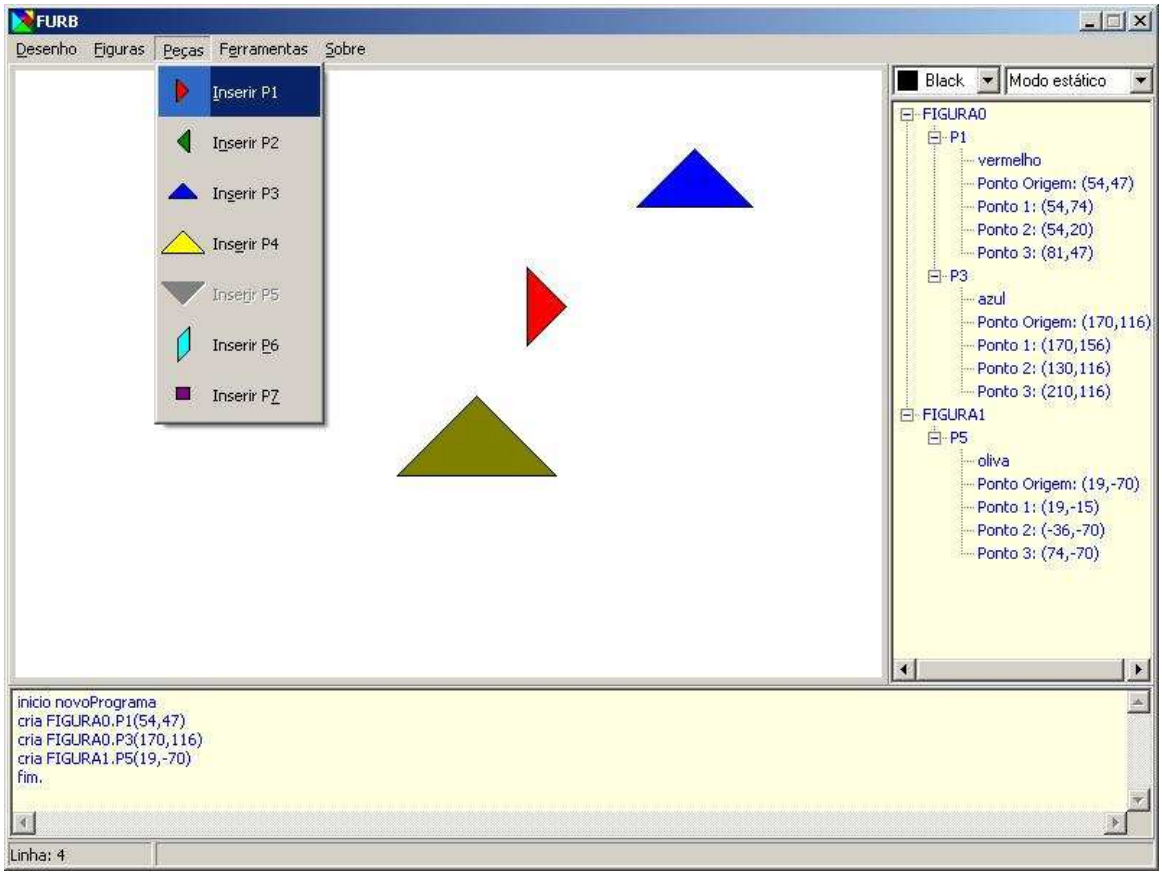


Figura 18 – Impossibilidade de inclusão de peças

2.8 TRABALHOS CORRELATOS

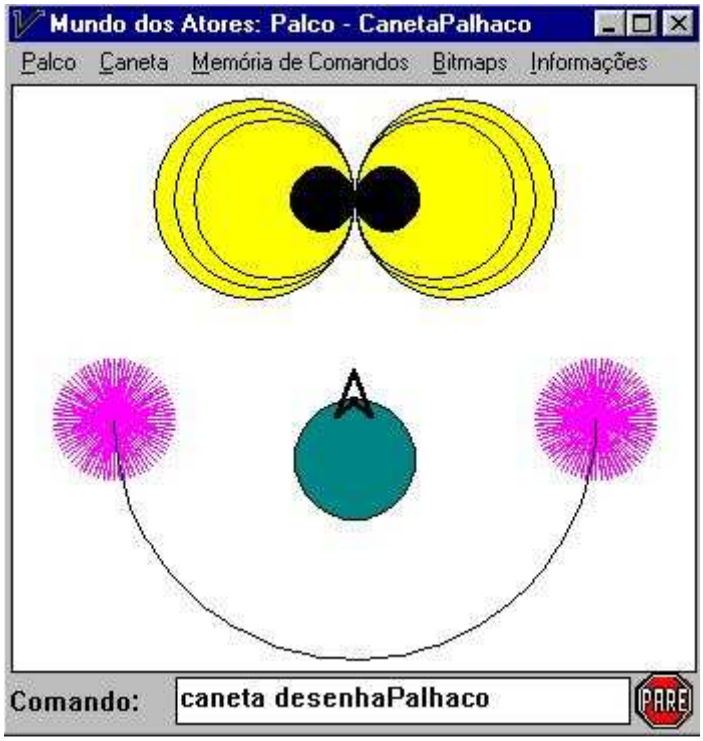
A seguir são descritos dois (2) trabalhos correlatos, os quais são: Mundo dos Atores e o Logo.

2.8.1 Mundo dos Atores

O Mundo dos Atores é uma ferramenta que tem como objetivo facilitar o ensino dos conceitos da Programação Orientada a Objetos (POO). É indicado como ferramenta prática para disciplinas de introdução à programação. Segundo Mariani (1998), a ferramenta apresenta elementos virtuais como a caneta, os atores e o palco, os quais tem relação direta com elementos concretos do mundo real.

O ambiente do Mundo dos Atores é similar ao oferecido pela linguagem Logo. O aprendiz pode controlar uma caneta (representada pela seta na Figura 19), fazendo-a traçar

diferentes desenhos no palco (área gráfica) onde ela está inserida (MARIANI, 1998). Exemplos de comandos do ambiente Mundo dos Atores podem ser vistos no quadro 14.



Fonte: Mariani (1998).

Figura 19 – Ambiente Mundo dos Atores

Comandos	Descrição
caneta anda: 50	Desloca-se 50 pontos na tela
caneta gira: 30	Rotaciona 30 graus
fixaCorDosRastros	Cor da caneta de desenho

Quadro 14 – Comandos do ambiente Mundo dos Atores

As ações `anda` e `gira` mencionadas no quadro 14 fazem parte do repertório de ações básicas conhecidas pelo Mundo dos Atores. Contudo, novas ações podem facilmente ser adicionadas a este repertório. O quadro 15 mostra um exemplo das ações que fazem parte do trabalho de desenhar o palhaço da figura 19.

```
desenhaPalhaco
      “Desenha o rosto de um palhaço”
      caneta desenhaOlhos;
                  desenhaNariz;
                  desenhaBoca.
circunferencia: raio
      "Desenha uma circunferência conforme o raio passado
como parâmetro"
      |lado|
      lado:= (2 * Float pi * raio) / 36.
      caneta gira: 5.
      36 vezesRepita: [
              caneta anda: lado;
                           gira: 10
           ].   caneta gira: -5.

desenhaOlhos
         "Desenha os olhos do palhaço"
         caneta semRastros;
         anda: 100;
          fixaCorDosRastros: Preto.
         2 vezesRepita: [
               caneta desenhaUmOlho;
                        gira: 180.      ].
         caneta semRastros;
               anda: -100;
         comRastros.
```

Fonte: Mariani (1998).

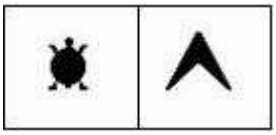
Quadro 15 – Desenhar o palhaço

2.8.2 Logo

O Logo é uma linguagem de programação que serve para comunicar-se com o computador. Segundo Valente (1991, p. 34), “[..] as características do Logo [...] são: exploração de atividades espaciais, fácil terminologia, e a capacidade de se criar novos termos ou procedimentos.”

Para as atividades espaciais, a proposta é comandar uma tartaruga mecânica. A tartaruga mecânica é um objeto que se desloca no chão, comandado pelo computador. Esse objeto desloca-se de maneira muito lenta, daí a analogia com uma tartaruga. De acordo com Valente (1991, p. 32-43), a mesma função da tartaruga no solo é repassada para a tartaruga de

tela, representada por um triângulo ou por um desenho de uma tartaruga (Figura 20). A tartaruga da tela é utilizada para a realização de atividades gráficas de grande precisão.



Fonte: Valente (1991, p. 35).
Figura 20 – Representação da tartaruga e triângulo

O objetivo da terminologia simples, ou seja a facilidade com que os comandos podem ser unidos, possibilita à criança uma maneira diferente de interagir com o computador e também permitir que ela rapidamente desenvolva atividades computacionais.

Uma outra característica importante da linguagem Logo é que ela é considerada procedural, facilitando assim a criação de novos termos ou procedimentos (VALENTE, 1991, p. 32-43).

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são apresentados os requisitos, a especificação, a implementação e a operacionalidade da ferramenta. Ainda, resultados e discussão são relatados.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os principais requisitos da ferramenta são:

a) permitir a visualização no espaço 3D (R^3), com perspectiva (profundidade) (requisito funcional - RF);
b) adaptar os comandos da linguagem do LTD para manipular a terceira dimensão (RF);
c) realizar manutenção corretiva em algumas funções (RF);
d) apresentar mecanismos para selecionar figuras e peças no editor gráfico (RF);
e) permitir que o editor de figuras tenha tamanho dinâmico e seja separado do editor de texto (o que não acontece no ambiente atual), deixando-os em locais distintos (requisito não-funcional - RNF);
f) ter um menu ajuda (*Help*) para auxiliar o usuário no manuseio da ferramenta (RNF);
g) ser reespecificada utilizando a técnica de Orientação à Objetos (OO) e implementada no Ambiente de Programação Delphi 7, utilizando a biblioteca OpenGL (RNF).

3.2 ESPECIFICAÇÃO

A especificação da ferramenta é baseada na técnica de OO, usando-se a UML, com o auxílio da ferramenta Enterprise Architect. Os diagramas utilizados são os de casos de uso, de classes e de seqüência.

3.2.1 Diagrama de casos de uso da ferramenta

O diagrama de casos de uso da ferramenta é apresentado na figura 21.

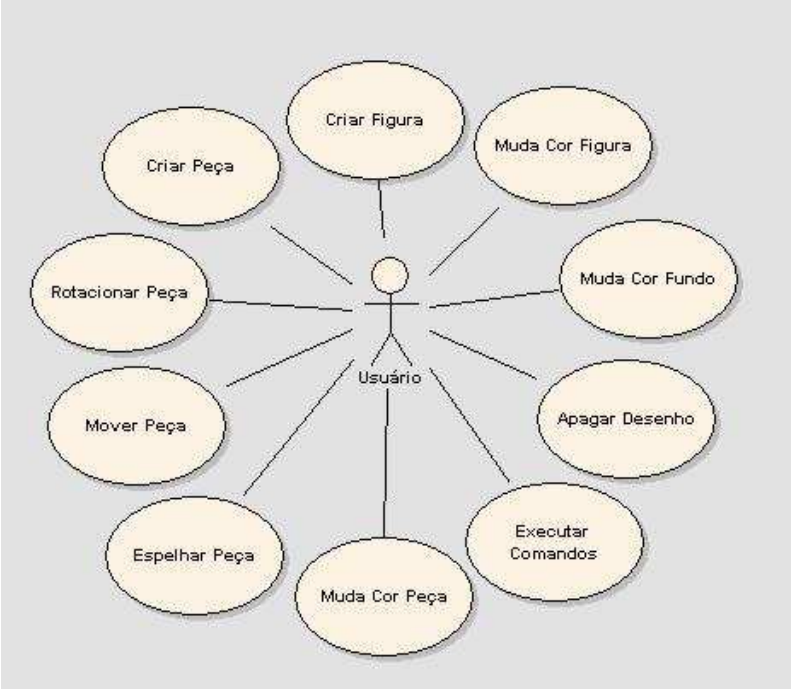


Figura 21 – Diagrama de casos de uso

A seguir são detalhados os casos de uso apresentados na Figura 21, os quais são:

a) `Criar Figura`: para criar uma figura na ferramenta, é necessário ter um menu `Figura` com a opção `Criar Figura`. Pode-se criar quantas figuras forem necessárias para o desenho. Cada figura precisa ter no mínimo uma (1) e no máximo sete (7) peças;

b) `Criar Peça`: a criação de uma peça depende da existência de uma figura. Existem sete (7) tipos de peças, as quais compõem o jogo Tangram. Para adicionar uma ou mais peças no editor de figuras, deve existir um menu `Peças` com sete (7) opções (uma para cada tipo de peça) para o usuário escolher a peça que desejar;

c) `Rotacionar Peça`: este caso depende da existência de uma peça. Para rotacionar é necessário que seja selecionada a peça clicando com o botão direito do *mouse* sobre ela. A peça selecionada é rotacionada 45 graus, no sentido anti-horário;

d) `Mover Peça`: no processo de movimentação de peças, deve existir no mínimo uma peça criada. Para mover uma peça é necessário que ela esteja selecionada. Para selecionar a peça deve-se utilizar o botão esquerdo do *mouse* clicando sobre a escolhida. Mantendo o mesmo pressionado, arrasta-se a peça até outro ponto do

editor, onde é solto o botão e a peça assumirá este novo ponto, ou seja, a peça é redesenhada de acordo com suas novas coordenadas;

e) `Espelhar Peça`: para espelhar uma peça é necessário que a mesma esteja selecionada. A seleção da peça deve ser feita clicando com o botão esquerdo do *mouse* juntamente com a tecla *`Shift`* sobre a escolhida. O comando `espelho` irá inverter os pontos da peça, fazendo um espelhamento da mesma;

f) `Muda Cor Peça`: o caso `Muda Cor Peça` consiste em mudar a cor de uma peça no editor de figuras. Para mudar a cor deve existir uma opção `Peça` no painel de controle, com todas as peças da figura, para que possa ser selecionada uma que vai ser alterada a cor; deve ter também uma opção `mudaCor` com um item chamado `Peça` identificando que será alterado a cor de uma peça e uma opção `cor` contendo todas as cores possíveis para escolher a nova cor da peça;

g) `Muda Cor Figura`: o caso `Muda Cor Figura` consiste em mudar a cor de uma figura no editor de figuras. Para mudar a cor deve existir uma opção `Figura` no painel de controle, com todas as figuras do desenho, para que possa ser selecionada uma que vai ser alterada a cor; deve ter também uma opção `mudaCor` com um item chamado `Figura` identificando que será alterado a cor de uma figura e uma opção `cor` contendo todas as cores possíveis para escolher a nova cor da figura. Todas as peças dessa figura assumirão essa nova cor;

h) `Muda Cor Fundo`: este caso consiste em mudar a cor de fundo do editor de figuras. Para mudar a cor deve existir uma opção `mudaCor` com um item chamado `Fundo` no painel de controle, identificando que será alterado a cor de fundo do editor de figuras e uma opção `cor` contendo todas as cores possíveis para escolher a nova cor;

i) `Apagar Desenho`: o processo `Apagar Desenho` realiza a remoção de todas as figuras e suas respectivas peças desenhadas no editor de figuras. Para apagar o desenho deve existir um menu `Desenho` com a opção `Apagar Desenho`;

j) `Executar Comandos`: para executar os comandos da linguagem deve ter um menu `Ferramentas` com a opção `Executar`. Esta opção executa os comandos da linguagem inseridos através do editor de texto, mostrando o resultado no editor de figuras.

3.2.2 Diagrama de classes

A seguir são descritas as funcionalidades das classes modeladas, as quais são: `TFigura`, `TPeca`, `TCor`, `TPonto`, `TFrmDesenho`, `TFrmTreeView`, `TFrmScript`, `TLexico`, `TSintatico` e `TSemantico`.

A classe `TFigura` possui informações básicas à respeito de uma figura, tais como, o identificador da figura e suas respectivas peças. Uma figura pode ter uma (1) ou no máximo sete (7) peças.

`TPeca` é a classe criada para representar instâncias de peças incluídas nas figuras. Uma peça não pode existir sem uma figura. Agregada a ela existem as classes `TCor` e `TPonto`. A classe `TCor` serve para representar as cores. Esse objeto guarda os valores RGB e o identificador da cor (por exemplo verde, azul, amarelo ou outra disponível). Uma peça tem uma cor e essa pode ser igual para várias peças. A classe `TPonto` possui informações relacionadas as coordenadas (x, y, z) do ponto. Uma peça pode ter três (3) ou quatro (4) pontos, sendo 3 pontos para os triângulos e 4 para o quadrado e o paralelograma, logo para cada ponto tem-se um objeto `Ponto` com as coordenadas (x, y, z).

A classe `TFrmDesenho` representa a tela, onde são desenhadas as figuras com suas respectivas peças. Uma lista de figuras é agregada a uma instância desta classe, que por sua vez, agrega uma lista de peças que serão mostradas na tela.

A classe `TFrmTreeView` e a classe `TFrmScript` são classes da tela, assim como a classe `TFrmDesenho`. A classe `TFrmTreeView` é responsável em montar uma árvore com informações das figuras e peças criadas e seus respectivos pontos. A classe `TFrmScript` tem como função permitir que o usuário digite os comandos no editor de texto. Esta classe é responsável em passar a mensagem para a classe `TLexico` e `TSintatico` para interpretarem e executarem o código da linguagem. Também é responsável pelos comandos gerados no editor de texto, quando é adicionada uma peça no editor de figuras.

A análise léxica da linguagem é feita pela classe `TLexico`. A classe `TSintatico` é responsável pela análise sintática da linguagem e a chamada das rotinas que executam os comandos. Esses comandos são executados conforme são disparadas as ações semânticas. Essas ações semânticas são implementadas e executadas na classe `TSemantico`, através do método `executeAction`. O diagrama das classes citadas é apresentado na figura 22.

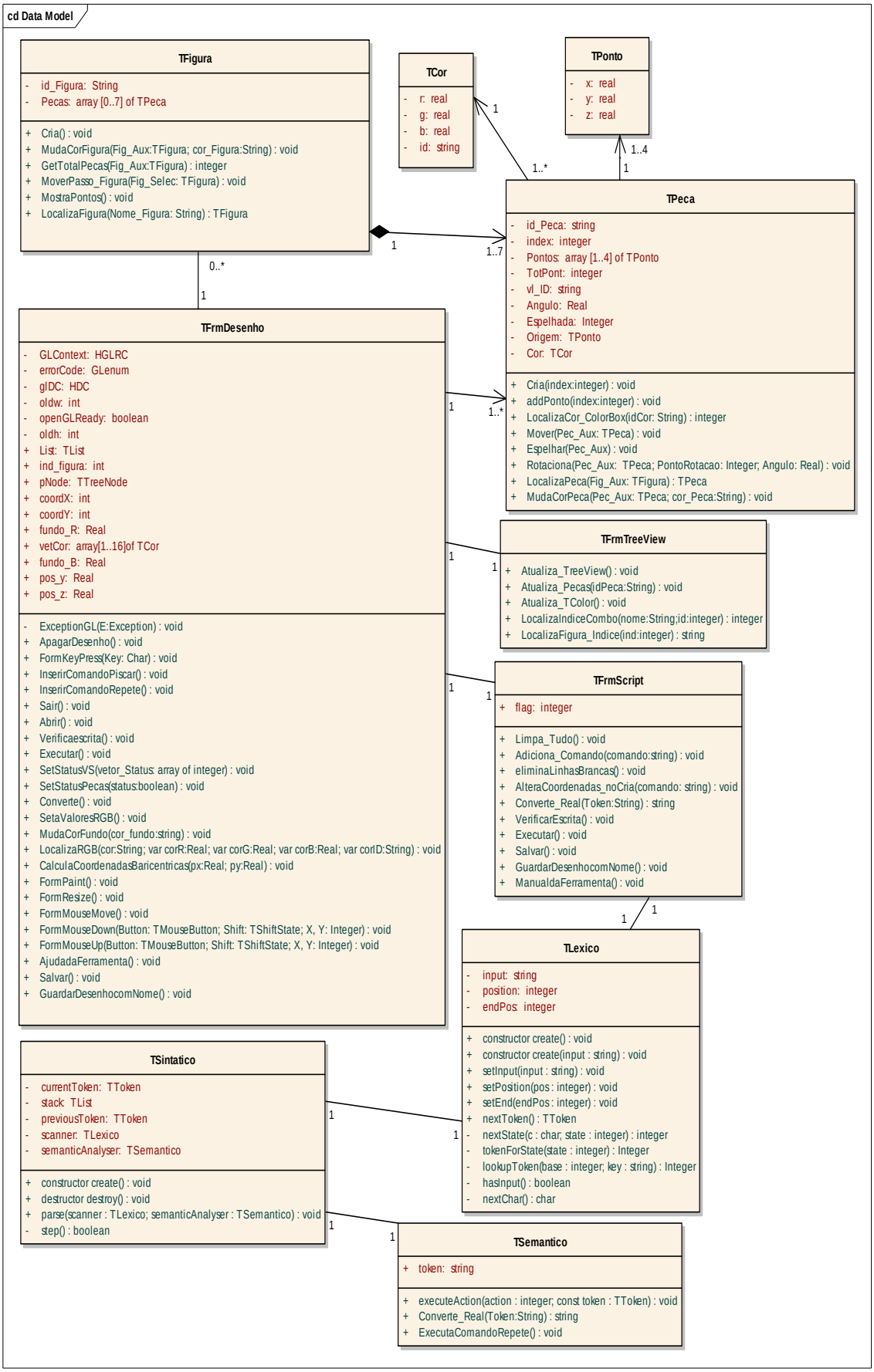


Figura 22 – Diagrama de classes

3.2.3 Diagramas de sequência da ferramenta

A seguir são apresentados os diagramas de seqüência da ferramenta.

3.2.3.1 Diagrama de seqüência `Criar Figura`

Quando o usuário cria uma figura é passada a mensagem `CriarFigura` para a classe `FrmDesenho`. Esta por sua vez manda a mensagem `TFigura.Create()` para a classe `TFigura`. A partir deste ponto, tem-se uma nova instância da classe `TFigura`. Após a criação dessa nova instância é atribuído o nome para a figura. O nome é determinado da seguinte forma: concatena-se o literal “Figura” com um índice que começa em zero (0) e vai sendo incrementado a cada nova figura. Caso o desenho seja eliminado, o índice é novamente zerado. O diagrama de seqüência do processo `Criar Figura` é apresentado na figura 23.

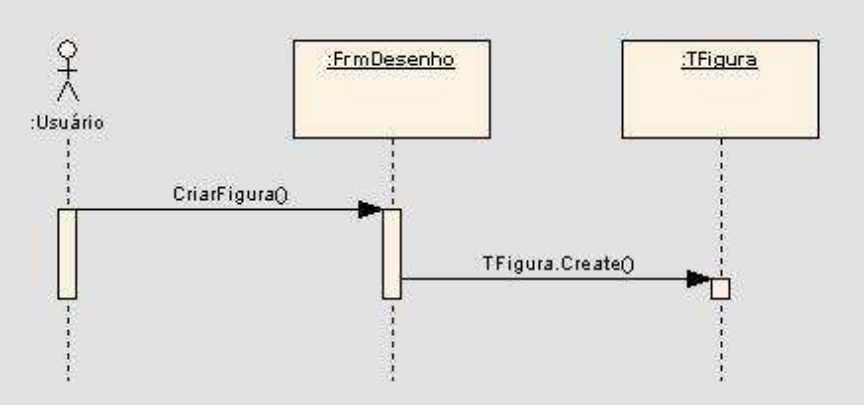


Figura 23 – Diagrama de seqüência `Criar Figura`

3.2.3.2 Diagrama de seqüência `Criar Peça`

Quando se adiciona uma peça, é passada a mensagem `InserirPeca` para a classe `FrmDesenho`. Em seguida, essa classe passa a mensagem `TPeca.Create` passando como parâmetro o índice da peca (por exemplo: Peça 1, o índice é 1) para a classe `TPeca`. A classe `TPeca` envia a mensagem `TPonto.Create` para a classe `TPonto`, criando assim um objeto ponto com os atributos x, y e z, para cada ponto da peça. Em seguida a classe `TPeca` aciona o método `addPonto` passando como parâmetro o índice da peça. Nesse momento o objeto peça

recebe os objetos ponto (correspondente a cada ponto da peça). A classe `FrmDesenho` envia o nome da figura atual para a classe `TFigura`, a qual dispara o método `Figura.LocalizaFigura`, passando como parâmetro a figura atual. Esse método retorna a instância da figura corrente onde será adicionada a nova peça. Após isto, a partir desta instância da classe figura é atribuído a nova peça na figura atual. O diagrama de seqüência deste processo é apresentado na figura 24.

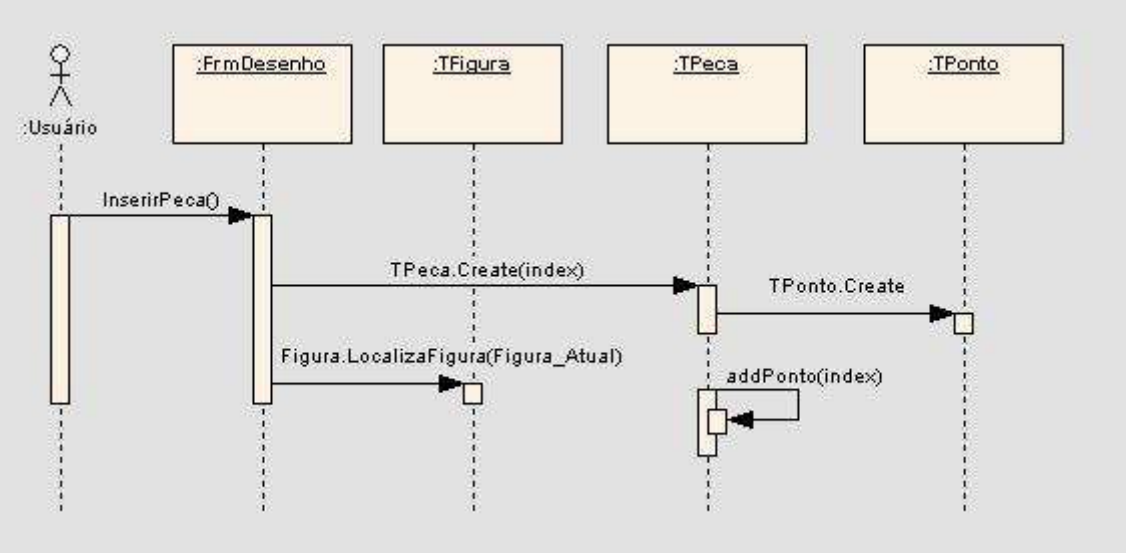


Figura 24 – Diagrama de seqüência `Criar Peça`

3.2.3.3 Diagrama de seqüência `Rotacionar Peça`

Quando o usuário clica em algum ponto do editor de figuras é passada a mensagem *MouseDown* para a classe `FrmDesenho` com 3 parâmetros: o ponto (x, y), o botão do *mouse* que foi pressionado e o valor da tecla. Em seguida, a classe `FrmDesenho` chama o método `FrmDesenho.Converte` que converte as coordenadas da tela para as coordenadas do mundo (wx, wy). Após isso, a classe `FrmDesenho` chama o método `FrmDesenho.CalculaCoordenadasBaricentricas` passando como parâmetro as coordenadas da tela que são wx e wy, o qual verifica se o ponto wx e wy está dentro da peça. Caso o ponto esteja dentro retorna a instância da peça selecionada e da figura a qual a mesma pertence. Na seqüência é verificado se o botão direito do *mouse* foi pressionado. Caso o botão esteja pressionado é passada a mensagem `Peca.Rotaciona(Peça_Selecionada,0,45)` para a classe `TPeca` com os parâmetros: peça selecionada; ponto de rotação e a quantidade em graus que será rotacionada a peça. A rotação comandada a partir do *mouse* é feita com um

valor padrão de 45 graus e com o ponto de rotação zero (0). O Ponto de rotação zero (0) significa que a peça será rotacionada tendo como ponto de referência o ponto de origem. O ponto de origem é o ponto central da figura, calculado de acordo com os pontos da peça. Por exemplo, uma peça com os pontos: p1 (0,0) ; p2 (2,0) ; p3 (2,2) ; p4 (0,2) de coordenadas (x, y) respectivamente terá como ponto de origem o ponto p0(1,1), ou seja, é o ponto central. A figura 25 apresenta o diagrama de seqüência do processo `Rotacionar Peça`.

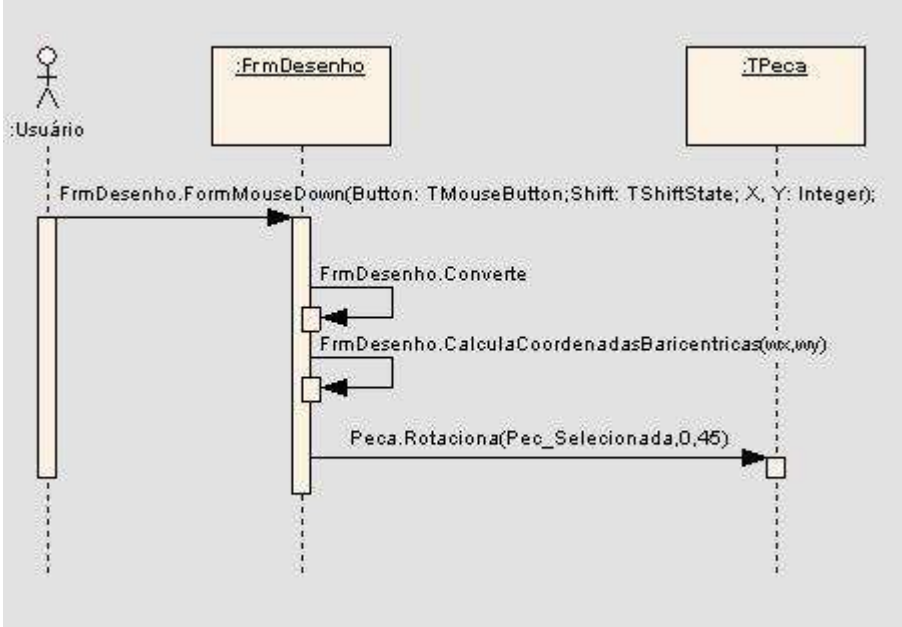


Figura 25 – Diagrama de seqüência `Rotacionar Peça`

3.2.3.4 Diagrama de seqüência `Mover Peça`

No processo de `Mover Peça` é analisado se o usuário selecionou uma peça ao clicar com o *mouse* no editor de figuras. Caso tenha selecionado, é verificado se o botão esquerdo do *mouse* foi pressionado. Se o botão estiver pressionado e houver alguma movimentação do mesmo, a mensagem *MouseMove* é passada para a classe `FrmDesenho`. Esse método verifica-se o botão esquerdo do *mouse* ainda está pressionado. Caso estiver, é acionado o método `Peca.Mover(Pec_Selecionada)`, que move a peça selecionada para as novas coordenadas, calculadas por esse método a partir da nova posição do *mouse*. O diagrama de seqüência do processo `Mover Peça` é apresentado na figura 26.

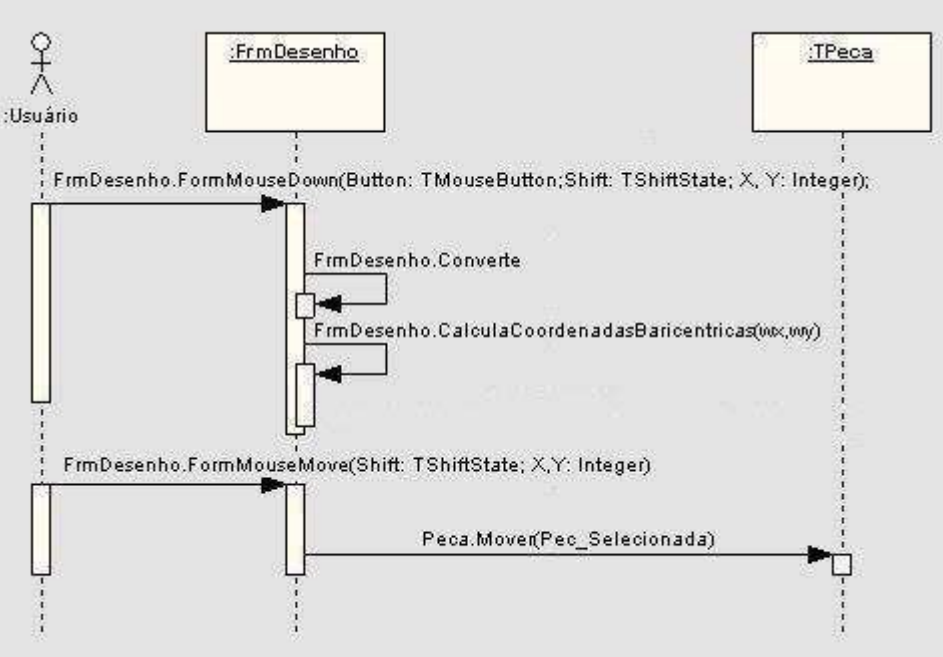


Figura 26 – Diagrama de seqüência `Mover Peça`

3.2.3.5 Diagrama de seqüência `Espelhar Peça`

No processo de `Espelhar Peça` é analisado se o usuário selecionou uma peça ao clicar com o *mouse* no editor de figuras. Caso tenha selecionado, é verificado se o botão esquerdo do *mouse* mais a tecla *`Shift`* estão pressionados. Se estiverem, é passada a mensagem `Peca.Espelhar(Pec_Selecionada)`para a classe `TPeca`. Com isso a peça é espelhada. A figura 27 apresenta o diagrama de seqüência do processo `Espelhar Peça`.

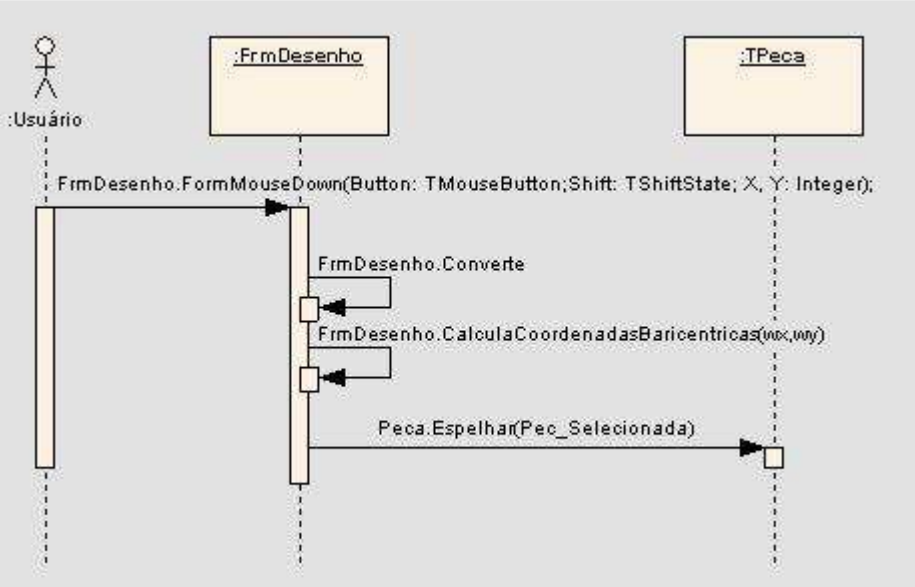


Figura 27 – Diagrama de seqüência `Espelhar Peça`

3.2.3.6 Diagrama de seqüência `Muda Cor Peça`

No processo para mudar a cor de uma peça o usuário manda a mensagem `LocalizaFigura` para a classe `TFigura`, passando como parâmetro o nome da figura. O método `LocalizaFigura` retorna a instância da figura. Em seguida é enviado a mensagem `LocalizaPeca`, sendo informado como parâmetro a instância da figura. Esse por sua vez retorna a instância da peça na qual será alterada a cor. Após é enviado a mensagem `MudaCorPeca` para a classe `TPeca`, passando como parâmetro a instância da peça e o nome da nova cor. A classe `TFigura` envia a mensagem `LocalizaRGB` para a classe `FrmDesenho`, que devolve os valores RGB da nova cor selecionada. Na seqüência, esses valores são atualizados na peça. O diagrama de seqüência do processo `Muda Cor Peça` é apresentado na figura 28.

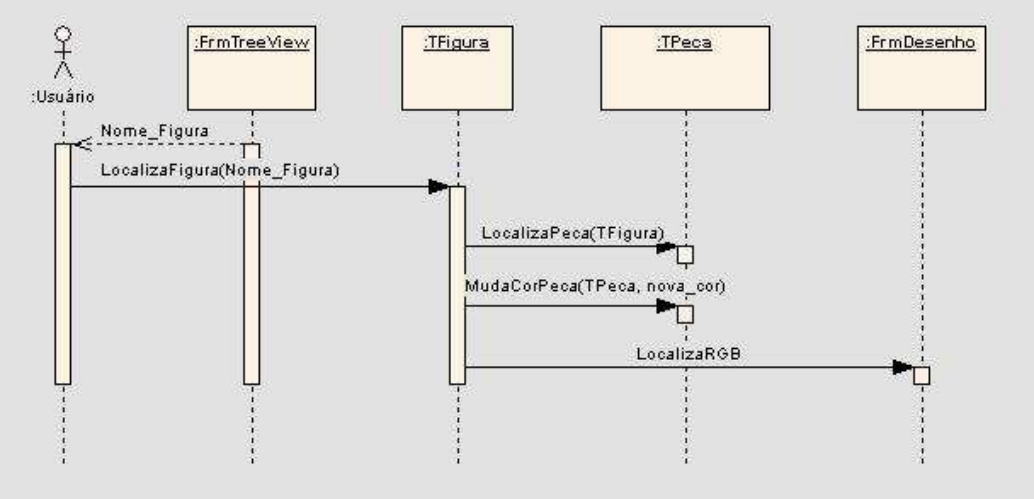


Figura 28 – Diagrama de seqüência `Muda Cor Peça`

3.2.3.7 Diagrama de seqüência `Muda Cor Figura`

No processo para mudar a cor de uma figura o usuário manda a mensagem `LocalizaFigura` para a classe `TFigura`, passando como parâmetro o nome da figura. O método `LocalizaFigura` retorna a instância da figura. Após é enviado a mensagem `MudaCorFigura` para a classe `TFigura`, passando como parâmetro a instância da figura e o nome da nova cor. A classe `TFigura` envia a mensagem `LocalizaRGB` para a classe `FrmDesenho`, que devolve os valores RGB da nova cor selecionada. Em seguida, esses valores são atualizados na figura. A figura 29 apresenta o diagrama de seqüência do processo `Muda Cor Figura`.

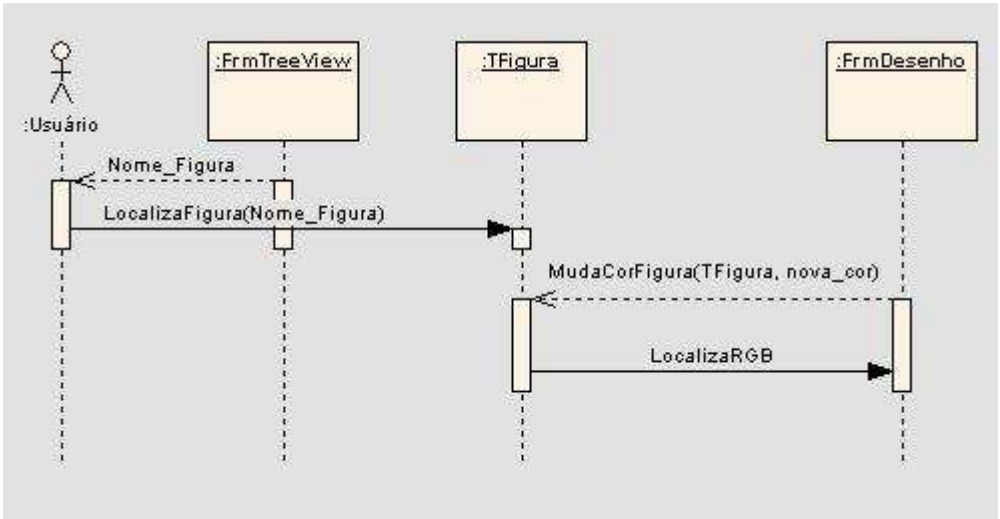


Figura 29 – Diagrama de seqüência `Muda Cor Figura`

3.2.3.8 Diagrama de seqüência `Muda Cor Fundo`

Para mudar a cor de fundo é enviada a mensagem `MudaCorFundo` para a classe `FrmDesenho`, passando como parâmetro o nome da nova cor. Em seguida a classe `FrmDesenho` aciona o método `LocalizaRGB`, que devolve os valores RGB da nova cor selecionada. Em seguida é alterado a cor de fundo. O diagrama de seqüência do processo `Muda Cor Fundo` é apresentado na figura 30.

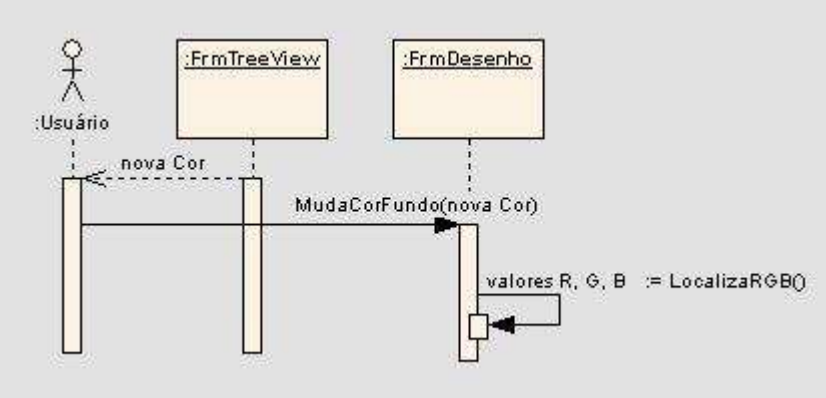


Figura 30 – Diagrama de seqüência `Muda Cor Fundo`

3.2.3.9 Diagrama de seqüência `Apagar Desenho`

Quando o usuário seleciona a opção `Apagar Desenho` no menu `Desenho`, é passada a mensagem `Limpar_Tudo` para classe `FrmScript`. Esse método aciona para todos os pontos da peça e para todas as peças da figura o método *`Free`*, o qual irá desalocar todos os objetos que foram instanciados. Em seguida é liberada a figura, também pelo método *`Free`*. Isso acontecesse para cada figura, até limpar todo o desenho. O diagrama de seqüência do processo `Apagar Desenho` é apresentado na figura 31.

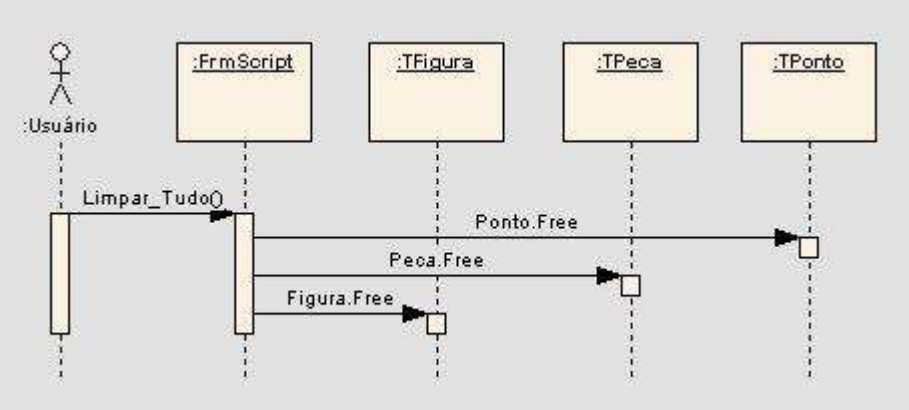


Figura 31 – Diagrama de seqüência `Apagar Desenho`

3.2.3.10 Diagrama de seqüência `Executar Comandos`

No processo de `Executar Comandos`, após o usuário ter inserido comandos da linguagem através do editor de texto, estes são executados e as ações por eles determinados são mostradas no editor de figuras. Quando o usuário comanda a execução do código da linguagem, a mensagem `lexico.setInput(MeScript.Lines.Text)` é passada para a classe `TLexico` com o parâmetro que contém o código da linguagem. Após é passada a mensagem `sintatico.parse(lexico,semantico)` para a classe `TSintatico`, passando como parâmetro os objetos léxico e semântico. Essa chamada é responsável em fazer toda análise sintática do código da linguagem. Se houver algum erro léxico ou sintático, o processo é interrompido e uma mensagem de erro é exibida, caso contrário o processo de `Executar Comandos` é efetuado. A figura 32 apresenta o diagrama de seqüência do processo `Executar Comandos`.

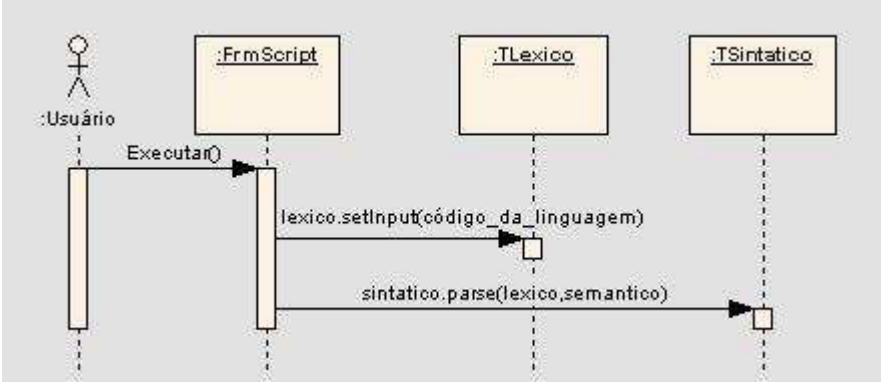


Figura 32 – Diagrama de seqüência `Executar Comandos`

3.2.4 Linguagem LTD

Nesta seção são descritos todos os comandos da linguagem e as especificações dos mesmos, através da BNF.

3.2.4.1 Especificação da linguagem LTD

O ambiente GALS foi usado para geração da parte da implementação léxica, sintática e semântica da linguagem. Para tanto foi utilizada a notação disponibilizada pelo ambiente para especificar a gramática da linguagem.

Para especificar a gramática no GALS é necessário fazer as definições regulares (quadro 16), assim como definir os *tokens* (quadro 17).

```
id: [a-zA-Z]+[a-zA-Z0-9]*
ws:[\ \t\s\r\n]+
comentario: /[/][^\n]+
multilinha:(/\*)([^\*]|\*[^/])*(\*/)
```

Quadro 16 – Definições regulares

```
//Identificador e espaços em branco

identificador:{id}+
: {ws}+

//Números

numeroComSinal: [\+\-]?[0-9]+[\.0-9]*
numeroSemSinal: [0-9]+

//palavras chave

inicio    = identificador: "inicio"
fim       = identificador: "fim"
cria      = identificador: "cria"
move      = identificador: "move"
movePasso  = identificador: "movePasso"
rotaciona  = identificador: "rotaciona"
piscar     = identificador: "piscar"
repete    = identificador: "repete"
vezes     = identificador: "vezes"
mudaCor   = identificador: "mudaCor"
fundo     = identificador: "fundo"
espelho    = identificador: "espelho"
Peca1   = identificador: "Peca1"
Peca2   = identificador: "Peca2"
Peca3   = identificador: "Peca3"
Peca4   = identificador: "Peca4"
Peca5   = identificador: "Peca5"
Peca6   = identificador: "Peca6"
Peca7   = identificador: "Peca7"

//Cores

amarelo= identificador: "amarelo"
azul= identificador: "azul"
azulMarinho = identificador : "azulMarinho"
azulPiscina = identificador: "azulPiscina"
branco = identificador: "branco"
cinza= identificador: "cinza"
marrom = identificador: "marrom"
oliva = identificador: "oliva"
prata = identificador: "prata"
preto = identificador: "preto"
rosa = identificador: "rosa"
verde = identificador: "verde"
verdePiscina = identificador: "verdePiscina"
verdeLima = identificador: "verdeLima"
vermelho = identificador: "vermelho"
violeta = identificador: "violeta"

//pontuações

";" : ;
"(" : \(
")" : \)
"[" : \[
"]" : \]
"," : ,
":" : :
"." : .
"+" :\+
"-" :\-

//comentarios
:{comentario}
:{multilinha}
```

Quadro 17 – *Tokens*

A especificação das construções existentes na linguagem (gramática) é uma extensão da apresentada em Silva, Martins e Alcântara Jr (2004). A especificação é apresentada no quadro 18.

```
<programa> ::= inicio identificador <bloco> fim ".";

<bloco>    ::= <comando> <bloco>
               |<comando>;

<comando>  ::= <cria> | <move> |<movePasso> |<rotaciona> |<piscar> |<repete>
               |<mudaCor> | <espelho>;

<cria>     ::= cria <figura> #0 "." <peca> #1 "(" <X> #2"," <Y>#3 "," <Z>#4 ","
<graus>#21 "," numeroComSinal#22")";

<move>     ::= move <figura>#5 "." <peca>#6 "(" <X>#14 "," <Y>#15 "," <Z>#24
")"          | move <figura>#5 "(" <X>#14 "," <Y>#15 "," <Z> #23")";

<movePasso>::= movePasso <figura>#5 "." <peca>#6 "(" <X>#14 "," <Y> #15"," <Z>
#16")"       | movePasso <figura>#5 "(" <X>#14 "," <Y> #15"," <Z>#17 ")";

<rotaciona> ::= rotaciona <figura>#5 "." <peca>#6 "(" <pto> #19 "," <graus>
#20")"        | rotaciona <figura>#5 "(" <peca> #6"." <pto> #19 "," <graus>
#20")";

<repete>    ::= repete numeroComSinal #11 vezes inicio #12 <bloco> fim #13;

<mudaCor>   ::= mudaCor <figura> #5 "." <peca> #6 <cor> #7
              | mudaCor <figura> #5 <cor> #8
              | mudaCor fundo <cor> #9;

<espelho>   ::= espelho <figura> #5 "." <peca> #6 #18;

<piscar>    ::= piscar "(" numeroComSinal #10 ")";

<figura>    ::= identificador;

<peca>      ::= Peca1  |Peca2  |Peca3  |Peca4  |Peca5  |Peca6  |Peca7  ;

<graus>     ::= numeroComSinal;

<pto>       ::= numeroComSinal;

<cor> ::= amarelo   | azul    | azulMarinho | azulPiscina  | branco  | cinza  |
marrom  | oliva     | prata   | preto       | rosa         | verde   |
verdePiscina        | verdeLima            |vermelho      | violeta  ;

<X> ::= numeroComSinal;

<Y> ::= numeroComSinal;

<Z> ::= numeroComSinal;
```

Quadro 18 – Gramática dos comandos da linguagem

O significado de cada ação semântica, representada pelo caractere “#” seguida de um número, é descrito no apêndice A.

No quadro 19 tem-se um exemplo de programa, obedecendo a especificação do quadro 18.

```
//Programa Exemplo
inicio tangranExemplo
        cria Figura0.Peca1(0,0,0,0,1)
        cria Figura0.Peca2(0,0,0,60,0)
        cria Figura0.Peca3(0,0,0,0,1)
        cria Figura0.Peca6(0.5,-0.5,1,0,0)
        cria Figura1.Peca1(0,0,0,0,0)
        cria Figura1.Peca2(0,0,0,45,1)
        repete 30 vezes
        inicio
            piscar(250)
            movePasso Figura0(-0.01, +0.0, 0.0)
            move Figura1.Peca2(-0.01, +0.0, 0.0)
            move Figura1(-0.01, +0.0, 0.0)
        fim
        rotaciona Figura0.Peca1(0,45)
        rotaciona Figura0(Peca1.0,45)
        /* Mudando a cor de uma peça,
         de uma figura e do fundo */
        mudaCor Figura0.Peca1 amarelo
        mudaCor Figura1 verde
        mudaCor fundo preto
        repete 15 vezes
        inicio
             piscar(250)
             movePasso Figura0.Peca1(0.00, +0.01, 0.0)
             repete 15 vezes
             inicio
               piscar(250)
               movePasso Figura0.Peca1(0.00, -0.01, 0.0)
               espelho Figura0.Peca1
               rotaciona Figura0.Peca1(1,45)
             fim
        fim
fim.
```

Quadro 19 – Exemplo de programa na linguagem LTD

3.2.4.2 Comandos da linguagem

Os comandos existentes na linguagem são: `cria`, `move`, `movePasso`, `rotaciona`, `piscar`, `mudaCor`, `espelho` e `repete`.

O comando `cria` cria peças e figuras, passando como parâmetros o ponto de origem

(x, y, z) da peça, o ângulo de rotação e um *flag* indicando se a peça está espelhada ou não. Esse *flag* pode assumir os valores 0 (não espelhada) ou 1 (espelhada). Exemplo da sintaxe do comando `cria` é apresentado no quadro 20.

```
cria Figura0.Peca1 (0, 0, 0, 0, 0)
```

Quadro 20 – Exemplo da sintaxe do comando `cria`

O comando `move` move uma peça ou uma figura inteira para uma nova coordenada (x, y, z). Essa nova coordenada é em relação a coordenada de origem. Exemplo da sintaxe do comando `move` é apresentado no quadro 21.

```
move Figura0.Peca1(0.5, 0.5, 0) movendo peça
move Figura0(0.5 ,0.5 ,0)  movendo figura
```

Quadro 21 – Exemplo da sintaxe do comando `move`

Com o comando `movePasso` move-se a figura do ponto onde estava (xi,yi,zi, sendo xi,yi,zi a coordenada inicial) para o ponto com o deslocamento (x, y, z). Portanto, tem-se a nova coordenada (xi+x, yi+y, zi+z). Exemplo da sintaxe do comando `movePasso` é apresentado no quadro 22.

```
movePasso Figura0.Peca1(0.5, 0.5, 0.1)  movendo peça
movePasso Figura0 (0.5, 0.5, 0.1)  movendo figura
```

Quadro 22 – 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 ponto de rotação e o ângulo de rotação. O ângulo de rotação pode ser positivo ou negativo (anti-horária ou horária). O ponto de rotação serve para rotacionar com relação a um dos pontos de uma peça. O ponto de rotação pode assumir os seguintes valores: 0 (origem), 1 (ponto 1), 2 (ponto 2), 3 (ponto 3) e 4 (ponto 4). Exemplos da sintaxe do comando `rotaciona` é apresentado no quadro 23.

```
rotaciona Figura0.Peca1(1, -30)  rotacionando peça
rotaciona Figura0(Peca1.1, +45) rotacionando figura
```

Quadro 23 – 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` é apresentado no quadro 24.

```
piscar(1000)
```

Quadro 24 – Exemplo da sintaxe do comando `piscar`

A linguagem dispõe de um comando para mudar a cor de uma peça, da figura e até mesmo a cor de fundo do editor de figuras. O comando é o `mudaCor`. A sintaxe do comando `mudaCor` é apresentado no quadro 25.

```
mudaCor Figura1.Peca1 azul   cor da peça
mudaCor Figura1 amarelo cor da figura
mudaCor fundo verde    cor de fundo
```

Quadro 25 – 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` é apresentado no quadro 26.

```
espelho Figura0.Peca1
```

Quadro 26 – 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 é apresentado no quadro 27.

```
...
repete 2 vezes
  inicio
     piscar(1000)
     espelho Figura0.Peca1
     repete 180 vezes
        inicio
           //vai rotacionar 360 graus a Peça1
           rotaciona Figura0.Peca1(0, +2)
           piscar(50)
        fim
  fim
...
```

Quadro 27 – Exemplo da sintaxe do comando `repete`

Ainda, a linguagem permite que sejam adicionados comentários entre os comandos. Existem os comentários de linha, onde tudo que estiver escrito após o “ // ”, na mesma linha, é ignorado pelo interpretador de comandos e os comentários multi-linhas, onde tudo que estiver escrito depois do “ /* ” e antes do “ */ ” é ignorado pelo interpretador, indiferente de que linha estiver. Exemplos desses dois tipos de comentários podem ser vistos no quadro 19.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas as técnicas e ferramentas utilizadas na implementação, a descrição do código das principais rotinas utilizadas e a operacionalidade da ferramenta.

3.3.1 Técnicas e ferramentas utilizadas

Para implementação da ferramenta foi utilizado o ambiente de desenvolvimento Borland Delphi 7. A linguagem utilizada pelo ambiente é Object Pascal.

Na implementação também foi utilizado o GALS. O GALS é uma ferramenta *freeware*, que com base em definições regulares e uma gramática, gera analisadores para três linguagens (Java, C++ ou Delphi). Tem a opção de gerar o analisador léxico, o analisador sintático e o semântico (GESSER, 2003, p. 39).

A partir da especificação feita no GALS (Quadro 18), pode-se gerar o código do analisador sintático. O quadro 28 apresenta um trecho do código gerado pelo GALS a partir da especificação apresentada no quadro 18.

Esse trecho de código é responsável em verificar a ordem (seqüência) correta dos *tokens*, por exemplo: o comando "`(150)piscar`", está errado, o correto é "`piscar (150)`". O código apresentado no quadro 28 verifica a ordem dos *tokens* (quadro 18) e exibe uma mensagem para o usuário explicando o que está acontecendo, caso encontre alguma divergência em relação à especificação do mesmo.

```
unit USintatico;

interface

uses UConstants, UToken, ULexico, USemantico, USyntaticError, UAnalysisError,
classes, ComCtrls, ExtCtrls, StdCtrls, SysUtils, Dialogs;

type
    TSintatico = class
    public
        constructor create;
        destructor destroy; override;
        procedure parse(scanner : TLexico; semanticAnalyser : TSemantico);
//raises EAnaliserError
    private
        currentToken : TToken;
        previousToken : TToken;
        scanner : TLexico;
        semanticAnalyser : TSemantico;

        function step : boolean;
    end;
var
    stack : TList;
implementation
…
procedure TSintatico.parse(scanner : TLexico; semanticAnalyser : TSemantico);
begin
    self.scanner := scanner;
    self.semanticAnalyser := semanticAnalyser;
    stack.clear;
    stack.add(Pointer(0));
    if (previousToken <> nil) and (previousToken <> currentToken) then
        previousToken.destroy;
    previousToken := nil;
    if currentToken <> nil then
        previousToken.destroy;
    currentToken := scanner.nextToken;

    while not step do
          ;
end;
…
end.
```

Quadro 28 – Código gerado pelo GALS (analisador sintático)

Para auxiliar na implementação das rotinas de desenho utilizou-se da biblioteca gráfica OpenGL.

A seguir são apresentados trechos de códigos de três (3) das principais rotinas implementadas, como: verificação da escrita dos comandos, cálculo das coordenadas baricêntricas e cálculo do apontamento das figuras com relação a câmera.

3.3.1.1 Método de verificação da escrita dos comandos

No quadro 29 é apresentando um trecho do código responsável por efetuar a

verificação dos comandos do editor de texto e exibir os erros detectados. Este código pertence ao método `VerificarEscrita` da classe `TFrmScript`.

```
procedure TFrmScript.VerificarEscrita1Click(Sender: TObject);
...
Begin
   lexico:= TLexico.create;
   sintatico:= TSintatico.create;
   semantico:= TSemantico.create;
   lexico.setInput(MeScript.Lines.Text);
   try
        sintatico.parse(lexico,semantico);
   except
     on t:ELexicalError do
       begin
          Limpa_Tudo;
          FrmErros.Show;
          FrmErros.ListBox1.Items.Add(t.getMessage+', em '+
inttostr(t.getPosition));
        end;
     on t: ESyntaticError do
      begin
         Limpa_Tudo;
         FrmErros.Show;
         FrmErros.ListBox1.Items.Add(t.getMessage+', em '+
inttostr(t.getPosition));
      end;
     on t: ESemanticError do
       begin
         FrmErros.Show;
         FrmErros.ListBox1.Items.Add(t.getMessage+', em '+
inttostr(t.getPosition));
       end;
   end;
   lexico.Destroy;
   sintatico.Destroy;
   semantico.Destroy;
   ....
end;
```

Quadro 29 – Método de verificação dos comandos

O método apresentado no quadro 29 é responsável pelo acionamento do método `sintático.parse` da classe `TSintático`, que faz a análise léxica e sintática dos comandos digitados no editor de texto. Caso algum erro léxico ou sintático seja encontrado, é exibida uma mensagem indicando o erro.

3.3.1.2 Método de cálculo das coordenadas baricêntricas

Um problema em geometria computacional consiste em determinar se um dado ponto está localizado no interior de um polígono simples. Uma forma de resolver esse problema seria implementar o algoritmo de Ponto-em-Polígono, assim como o LTD faz hoje. Outra forma de solucionar esse problema é calcular as coordenadas baricêntricas do ponto (quadro 30). Foi implementado o cálculo das coordenadas baricêntricas do ponto, devido à facilidade

e baixa complexidade na implementação.

O cálculo das coordenadas baricêntricas serve para verificar se um determinado ponto está ou não dentro da área de um triângulo. Na implementação caso a peça seja um paralelograma ou um quadrado, então divide-se essas em dois triângulos separados e faz-se então o cálculo.

```
procedure TFrmDesenho.CalculaCoordenadasBaricentricas(px:Real; py:Real);
var q1,q2,q3: Real;
    Det: array [0..3] of Real;
    i,a,p: integer;
    Fig_Aux: TFigura;
    Pec_Aux: TPeca;
    Pont_Aux: array [1..4] of TPonto;
begin
  Fig_Selecionada:= nil;
  Pec_Selecionada:= nil;
  //Cálculo do Determinante
  { q1 x1 + q2 x2 + q3 x3 = x
    q1 y1 + q2 y2 + q3 y3 = y
    q1    + q2    + q3    = 1}
  //percorrer todas as figuras
  for i:=0 to List.Count-1 do
  begin
     Fig_Aux:= List.Items[i];
     //todas as peças
     for a:= 1 to 7 do
     begin
       Pec_Aux:= Fig_Aux.Pecas[a];
       //todos os pontos
       if Pec_Aux <> nil then
       begin
        for p:= 1 to Fig_Aux.Pecas[a].TotPont do
            Pont_Aux[p]:= Pec_Aux.Pontos[p];
       //somente para Peças de 3 Pontos
       if Fig_Aux.Pecas[a].TotPont <= 3 then
        begin
          Det[0]:= (Pont_Aux[1].x * Pont_Aux[2].y * 1) + (Pont_Aux[1].y * 1 * Pont_Aux[3].x) +
                   (1 * Pont_Aux[2].x * Pont_Aux[3].y) - (Pont_Aux[1].y * Pont_Aux[2].x * 1) -
                   (Pont_Aux[1].x * 1 * Pont_Aux[3].y) - (1 * Pont_Aux[2].y * Pont_Aux[3].x);

          Det[1]:= (px * Pont_Aux[2].y * 1) + (py * 1 * Pont_Aux[3].x) +
                   (1 * Pont_Aux[2].x * Pont_Aux[3].y) - (py * Pont_Aux[2].x * 1) -
                   (px * 1 * Pont_Aux[3].y) - (1 * Pont_Aux[2].y * Pont_Aux[3].x);

          Det[2]:= (Pont_Aux[1].x * py * 1) + (Pont_Aux[1].y * 1 * Pont_Aux[3].x) +
                   (1 * px * Pont_Aux[3].y) - (Pont_Aux[1].y * px * 1) -
                   (Pont_Aux[1].x * 1 * Pont_Aux[3].y) - (1 * py * Pont_Aux[3].x);

          Det[3]:= (Pont_Aux[1].x * Pont_Aux[2].y * 1) + (Pont_Aux[1].y * 1 * px) +
                   (1 * Pont_Aux[2].x * py) - (Pont_Aux[1].y * Pont_Aux[2].x * 1) -
                   (Pont_Aux[1].x * 1 * py) - (1 * Pont_Aux[2].y * px);

          q1:= Det[1] / Det[0];
          q2:= Det[2] / Det[0];
          q3:= Det[3] / Det[0];

          //Se a peça está dentro do polígono
          if (q1 > 0) and (q2 > 0) and (q3 > 0) then
           begin
             FrmTreeView.ComboBox1.ItemIndex:= FrmTreeView.LocalizaIndiceCombo(Fig_Aux.id_Figura,1);
             FrmTreeView.ComboBox2.ItemIndex:= FrmTreeView.LocalizaIndiceCombo(Pec_Aux.id_Peca,2);
             Fig_Selecionada:= Fig_Aux;
             Pec_Selecionada:= Pec_Aux;
```

```
            FrmTreeView.Atualiza_TColor;
          end;
        end
       else
        begin
          //Peças com 4 Pontos -> quebra-se em dois Triângulos separados e compara um a um
          if  Pec_Aux.index in [6,7] then
            begin
               Det[0]:= (Pont_Aux[1].x * Pont_Aux[2].y * 1) + (Pont_Aux[1].y * 1 * Pont_Aux[4].x) +
                        (1 * Pont_Aux[2].x * Pont_Aux[4].y) - (Pont_Aux[1].y * Pont_Aux[2].x * 1) -
                        (Pont_Aux[1].x * 1 * Pont_Aux[4].y) - (1 * Pont_Aux[2].y * Pont_Aux[4].x);

               Det[1]:= (px * Pont_Aux[2].y * 1) + (py * 1 * Pont_Aux[4].x) +
                        (1 * Pont_Aux[2].x * Pont_Aux[4].y) - (py * Pont_Aux[2].x * 1) -
                        (px * 1 * Pont_Aux[4].y) - (1 * Pont_Aux[2].y * Pont_Aux[4].x);

               Det[2]:= (Pont_Aux[1].x * py * 1) + (Pont_Aux[1].y * 1 * Pont_Aux[4].x) +
                        (1 * px * Pont_Aux[4].y) - (Pont_Aux[1].y * px * 1) -
                        (Pont_Aux[1].x * 1 * Pont_Aux[4].y) - (1 * py * Pont_Aux[4].x);

               Det[3]:= (Pont_Aux[1].x * Pont_Aux[2].y * 1) + (Pont_Aux[1].y * 1 * px) +
                        (1 * Pont_Aux[2].x * py) - (Pont_Aux[1].y * Pont_Aux[2].x * 1) -
                        (Pont_Aux[1].x * 1 * py) - (1 * Pont_Aux[2].y * px);

               q1:= Det[1] / Det[0];
               q2:= Det[2] / Det[0];
               q3:= Det[3] / Det[0];
      //Se a peça está dentro do polígono
      if (q1 > 0) and (q2 > 0) and (q3 > 0) then
       begin
         FrmTreeView.ComboBox1.ItemIndex:= FrmTreeView.LocalizaIndiceCombo(Fig_Aux.id_Figura,1);
         FrmTreeView.ComboBox2.ItemIndex:= FrmTreeView.LocalizaIndiceCombo(Pec_Aux.id_Peca,2);
         Fig_Selecionada:= Fig_Aux;
         Pec_Selecionada:= Pec_Aux;
         FrmTreeView.Atualiza_TColor;
       end
      else
       begin
          Det[0]:= (Pont_Aux[2].x * Pont_Aux[3].y * 1) + (Pont_Aux[2].y * 1 * Pont_Aux[4].x) +
                   (1 * Pont_Aux[3].x * Pont_Aux[4].y) - (Pont_Aux[2].y * Pont_Aux[3].x * 1) -
                   (Pont_Aux[2].x * 1 * Pont_Aux[4].y) - (1 * Pont_Aux[3].y * Pont_Aux[4].x);

          Det[1]:= (px * Pont_Aux[3].y * 1) + (py * 1 * Pont_Aux[4].x) +
                   (1 * Pont_Aux[3].x * Pont_Aux[4].y) - (py * Pont_Aux[3].x * 1) -
                   (px * 1 * Pont_Aux[4].y) - (1 * Pont_Aux[3].y * Pont_Aux[4].x);

          Det[2]:= (Pont_Aux[2].x * py * 1) + (Pont_Aux[2].y * 1 * Pont_Aux[4].x) +
                   (1 * px * Pont_Aux[4].y) - (Pont_Aux[2].y * px * 1) -
                   (Pont_Aux[2].x * 1 * Pont_Aux[4].y) - (1 * py * Pont_Aux[4].x);

          Det[3]:= (Pont_Aux[2].x * Pont_Aux[3].y * 1) + (Pont_Aux[2].y * 1 * px) +
                   (1 * Pont_Aux[3].x * py) - (Pont_Aux[2].y * Pont_Aux[3].x * 1) -
                   (Pont_Aux[2].x * 1 * py) - (1 * Pont_Aux[3].y * px);

          q1:= Det[1] / Det[0];
          q2:= Det[2] / Det[0];
          q3:= Det[3] / Det[0];

                   //Se a peça está dentro do polígono
                   if (q1 > 0) and (q2 > 0) and (q3 > 0) then
                    begin
                      FrmTreeView.ComboBox1.ItemIndex:= FrmTreeView.LocalizaIndiceCombo(Fig_Aux.id_Figura,1);
                      FrmTreeView.ComboBox2.ItemIndex:= FrmTreeView.LocalizaIndiceCombo(Pec_Aux.id_Peca,2);
                      Fig_Selecionada:= Fig_Aux;
                      Pec_Selecionada:= Pec_Aux;
                      FrmTreeView.Atualiza_TColor;
                    end;
                end;
            end;
          end;
        end;
      end;
    end;
  end;
end;
```

Quadro 30 – Método de cálculo das coordenadas baricêntricas

A seguir é fundamentado o método de cálculo das coordenadas baricêntricas apresentada no quadro 30.

Segundo Figueiredo e Carvalho (1991, p. 28), com as coordenadas dos pontos

p, p_1, p_2 e p_3, e a equação $\lambda_1 + \lambda_2 + \lambda_3 = 1$, constrói- se um sistema de três equações e três incógnitas para encontrar as coordenadas baricêntricas, de modo que $p = \lambda_1 . p_1 + \lambda_2 . p_2 + \lambda_3 . p_3$, onde λ_1, λ_2 e λ_3 são números reais, e são denominados coordenadas baricêntricas de p em relação à p_1, p_2 e p_3 (quadro 31). Os valores λ_1, λ_2 e λ_3 podem ser obtidos usando a regra de Cramer e expressos em termos de áreas de triângulos (quadro 32) (FIGUEIREDO; CARVALHO, 1991, p. 28).

$$\begin{aligned} \lambda_1 . X_1 + \lambda_2 . X_2 + \lambda_3 . X_3 &= Xp \\ \lambda_1 . X_1 + \lambda_2 . X_2 + \lambda_3 . X_3 &= Yp \\ \lambda_1 \quad + \lambda_2 \quad + \lambda_3 \quad &= 1 \end{aligned}$$

Fonte: Figueiredo e Carvalho (1991, p. 28).

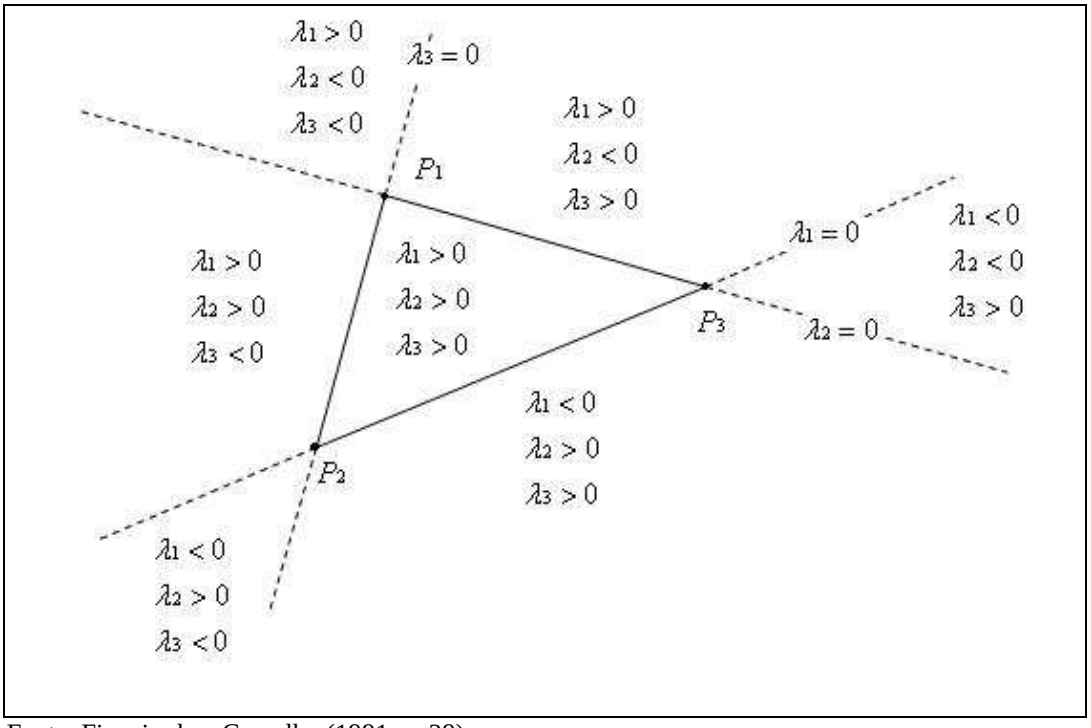
Quadro 31 – Equações para encontrar as coordenadas baricêntricas

$$\lambda_1 = \frac{S(p . p_2 . p_3)}{S(p_1 . p_2 . p_3)}, \quad \lambda_2 = \frac{S(p_1 . p . p_3)}{S(p_1 . p_2 . p_3)}, \quad \lambda_3 = \frac{S(p_1 . p_2 . p)}{S(p_1 . p_2 . p_3)}$$

Fonte: Figueiredo e Carvalho (1991, p. 28).

Quadro 32 – Valores de λ_1, λ_2 e λ_3 expressos em termos de área de triângulos

De acordo com Figueiredo e Carvalho (1991, p. 28), a análise do sinal das coordenadas baricêntricas indica a região do plano em que se encontra p, em relação ao triângulo $p_1 p_2 p_3$. Caso λ_1, λ_2 e λ_3 sejam maiores que zero (0), então p encontra-se dentro do triângulo. A figura 33 mostra como fica os sinais das coordenadas baricêntricas.



Fonte: Figueiredo e Carvalho (1991, p. 29).

Figura 33 – Sinais das coordenadas baricêntricas

3.3.1.3 Método do cálculo de seleção 3D de peças

No quadro 33 é apresentando um trecho do código responsável pelo cálculo de seleção 3D de peças com relação à posição da câmera. Este código pertence ao método `FormResize` da classe `TFrmDesenho`.

```
procedure TFrmDesenho.FormResize(Sender: TObject);
var
  Aspecto : GLdouble;
begin
  //verifica se está ativo o OpenGL
  if not openGLReady then
    exit;

  if Height = 0 then
    Height := 1;
  //cálculo do apecto (razão entre altura e largura)
  Aspecto := Width / Height;
  //ativa o modo de projeção
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity;
  //se a posição da camera for positiva
  if camera >= 0 then
    //calcula o apontamento com relação a camera
    gluPerspective(30.0, Aspecto, 1.0, 11.0)
  else
    //calcula o apontamento com relação a camera
    gluPerspective(30.0, Aspecto, camera * -1, (camera * -1)*11);
    //View Port
  glViewPort(0,0,ClientWidth,ClientHeight);
  glMatrixMode(GL_MODELVIEW);
  Invalidate;

  errorCode := glGetError;
  if errorCode<>GL_NO_ERROR then
    raise Exception.Create('FormResize:'+gluErrorString(errorCode));

  if (ClientWidth<=oldw) and (ClientHeight<=oldh) then
    FormPaint(Sender);

  oldh := ClientHeight;
  oldw := ClientWidth;
end;
```

Quadro 33 – Método *resize* (cálculo de seleção 3D de peças)

O comando `gluPerspective(30.0,Aspecto,câmera * -1, (câmera * -1) * 11)` apresentado no quadro 33 é responsável pelo apontamento das figuras com relação à câmera. O parâmetro `30.0` indica o ângulo da projeção, o `Aspecto` é a razão entre a altura e a largura e os dois parâmetros seguintes especificam a coordenada mais próxima e mais distante do observador, respectivamente, no eixo de profundidade. Esses dois últimos parâmetros são calculados com relação à posição da câmera.

3.3.2 Operacionalidade da ferramenta

Os recursos disponibilizados na ferramenta LTD (versão 2.0) ao usuário são apresentados no quadro 34.

Funcionalidade	Atalho	Descrição
Pegar Desenho	Ctrl + A	localiza um arquivo e abre no editor de texto.
Apagar Desenho	Ctrl + L	apaga todas as figuras e suas respectivas peças.
Inserir comando Piscar	Ctrl + P	o comando `piscar` funciona como um *delay*.
Inserir comando Repete	Ctrl + R	insere a estrutura do comando repete (repete n vezes o bloco componente).
Verificar Escrita	F5	verifica os comandos da linguagem antes da execução.
Executar	F9	executa os comandos caso não tenha nenhum erro léxico ou sintático.
Visualizar editor de texto	F10	visualiza o editor de texto.
Visualizar painel de controle	F11	visualiza o painel de controle.
Visualizar editor de figuras	F12	visualiza o editor de figuras.
Manual da ferramenta	F1	visualiza o manual da ferramenta.
Rotacionar Peça	-	permite rotaciona uma peça no sentido horário e anti-horário, em relação à um dado ponto da peça (ponto de rotação).
Criar Figura	-	permite criar figuras.
Criar Peça	-	permite ao usuário a criação de peças.
Move Peça/Figura	-	o comando `Mover` permite ao usuário movimentar uma peça ou uma figura, de uma posição para outra.
MovePasso Peça/Figura	-	o comando `MovePasso` permite ao usuário deslocar uma peça ou uma figura, de uma posição para outra, com deslocamento informado.
Espelho	-	o comando `Espelho` inverte as coordenadas de x da peça, efetivando um espelhamento.
Guardar Desenho	Ctrl + S	salva os comandos do editor de texto em disco.
Guardar Desenho com o nome...	Ctrl + F12	salva os comandos do editor de texto em disco solicitando o nome e o local em disco para guardar.

Quadro 34 – Relação das funcionalidades e seus atalhos

Ao iniciar o uso da ferramenta são visualizados o painel de controle, o editor de figuras e o editor de texto (figura 34).

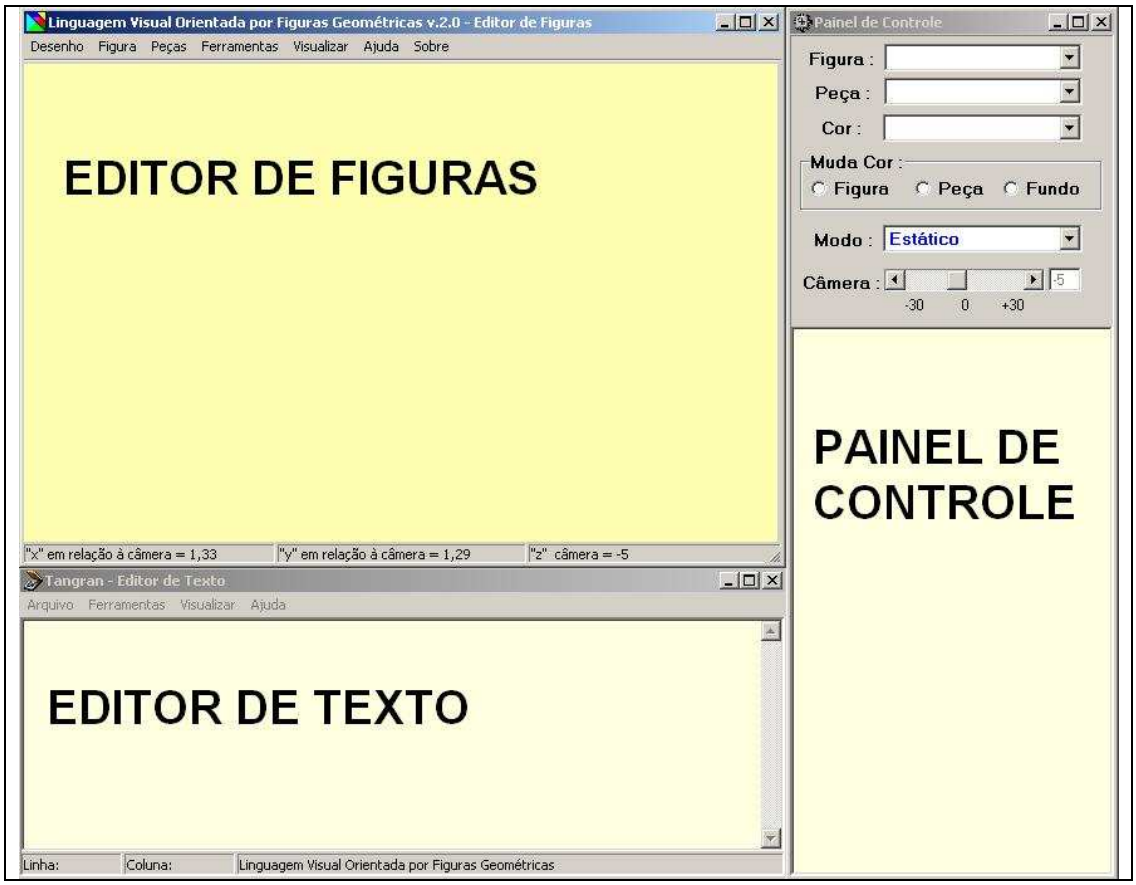


Figura 34 – Telas iniciais da ferramenta

3.3.2.1 Painel de controle

A tela do painel de controle mostra as informações sobre a 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 a cor, ponto de origem, ponto de 1 a n (sendo n = 3 ou n = 4, dependendo da peça) e seus respectivos valores x, y e z. Tem-se também as opções que são muda a cor de uma peça, de uma figura ou do fundo. Ainda, através do painel de controle, determina-se o modo de execução (estático ou dinâmico), além de controlar a posição da câmera. O modo estático é o modo de edição do comando `cria`, ou seja, qualquer movimentação, rotação ou espelhamento de alguma peça implicará em alteração nos parâmetros do comando `cria`. O modo dinâmico é responsável por apresentar outros comandos, por exemplo se o usuário desejar fazer um espelhamento de uma peça no modo dinâmico uma nova linha com o comando `espelho` é adicionado no editor de texto. A figura 35 apresenta todos os elementos do painel de controle.

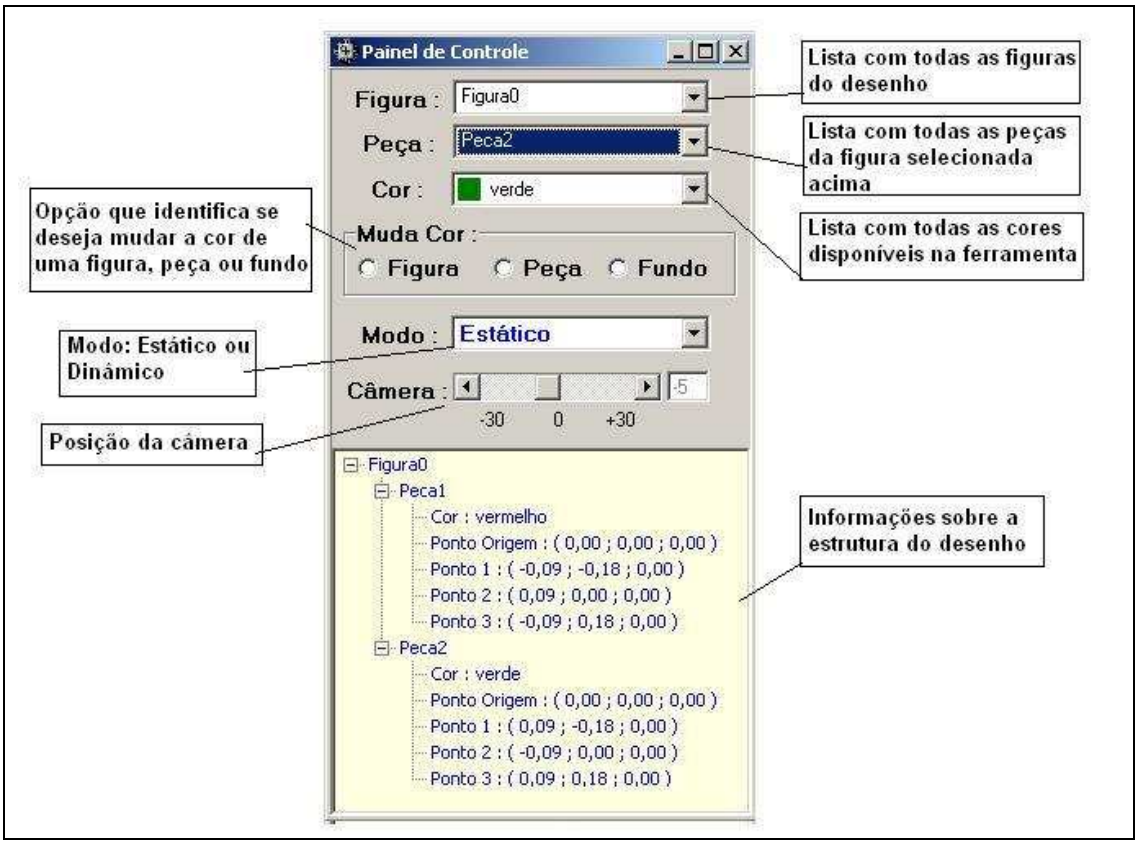


Figura 35 – Elementos do painel de controle

Para mudar a cor de uma peça, o usuário deve então selecionar na opção `Muda Cor` o item `Peça`, em seguida deve selecionar a peça que quiser alterar a cor na opção `Peça` e escolher a nova cor na opção `Cor`. Para alterar a cor da figura repete-se o mesmo processo, apenas ao invés de selecionar o item `peça` na opção `Muda Cor`, selecione o item `Figura`. Para mudar a cor do fundo, deve-se escolher o item `Fundo` na opção `Muda Cor` e então escolher a nova cor na opção `Cor`.

3.3.2.2 Editor de figuras

No editor de figuras o usuário cria seus desenhos. Conforme as operações realizadas no editor de figuras, são adicionadas ou alteradas linhas de código no editor de texto correspondente às ações do usuário, de acordo com o modo no painel de controle (se for estático, linhas são alteradas (apenas o comando cria); se for dinâmico, linhas com os respectivos comandos são adicionadas).

Ainda, o usuário pode rotacionar, espelhar e até mesmo mover as peças. Por exemplo, quando pressionado o botão *click* esquerdo do *mouse* em cima da peça escolhida, a mesma é

rotacionada 45 graus. Se o modo no painel de controle estiver estático então não é adicionada uma linha no editor de texto, apenas é alterado o parâmetro do ângulo na linha de criação da peça (figura 36). Caso o modo for dinâmico, uma nova linha com o comando rotaciona é adicionado no editor de texto (figura 37).

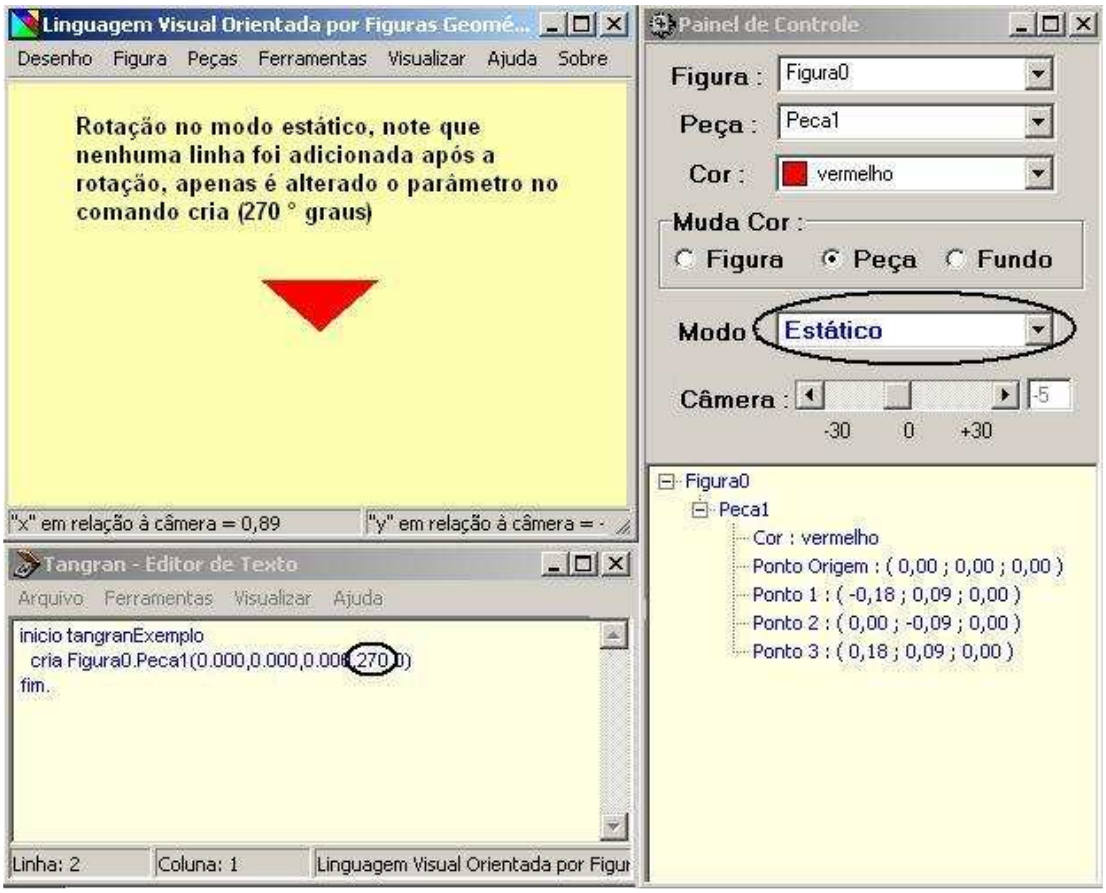


Figura 36 – Rotação no modo estático

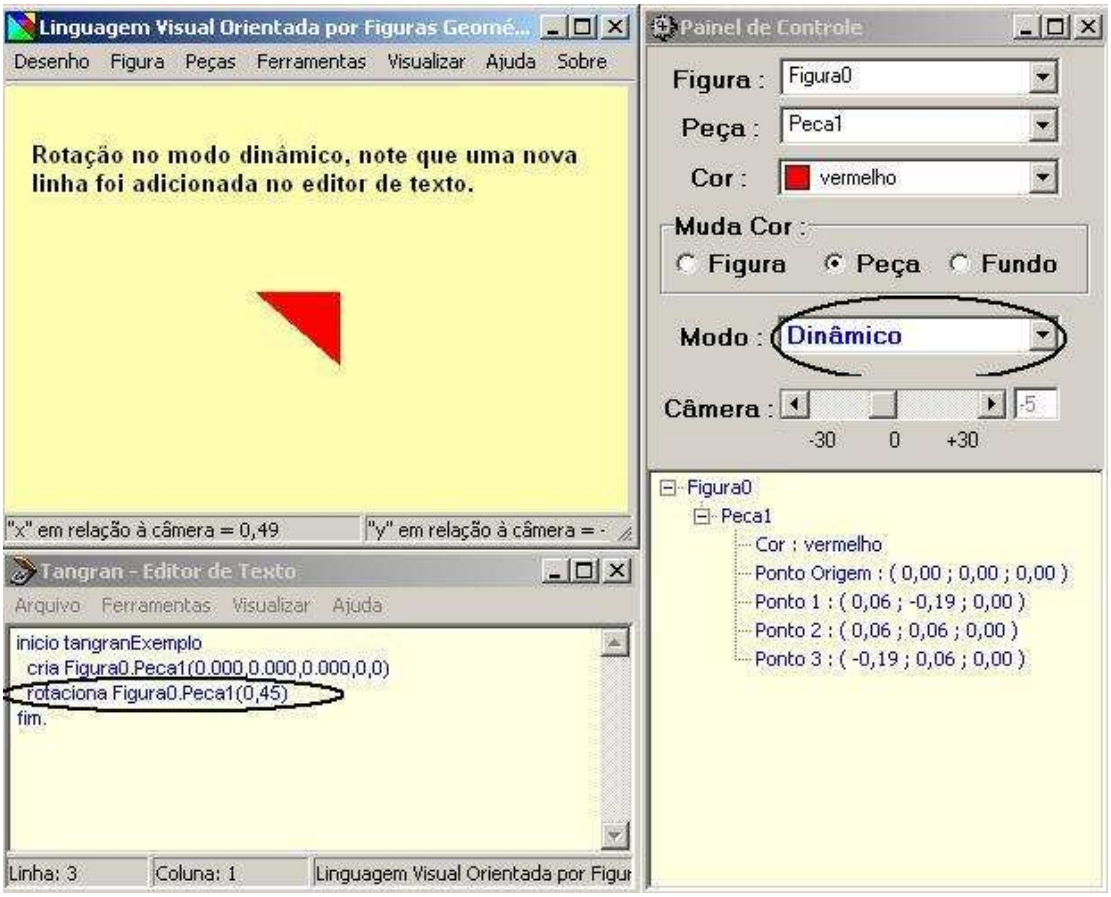


Figura 37 – Rotação no modo dinâmico

Ainda, pressionando-se `shift` juntamente com o botão *click* esquerdo do *mouse*, os pontos da peça selecionada serão invertidos, fazendo o espelhamento da peça. Se o modo no painel de controle estiver estático, então é alterado o parâmetro “espelhado” na linha de criação da peça (figura 38). Caso o modo for dinâmico, uma nova linha com o comando espelho é adicionado no editor de texto (figura 39). Para mover um figura, o usuário clica em cima de uma peça com o *mouse* no editor de figuras, segura o botão esquerdo do mesmo, em seguida arrasta a mesma sem soltar o botão até outro ponto do editor de figuras, onde soltará o botão e a peça assumirá este novo ponto, ou seja, a peça é redesenhada de acordo com suas novas coordenadas.

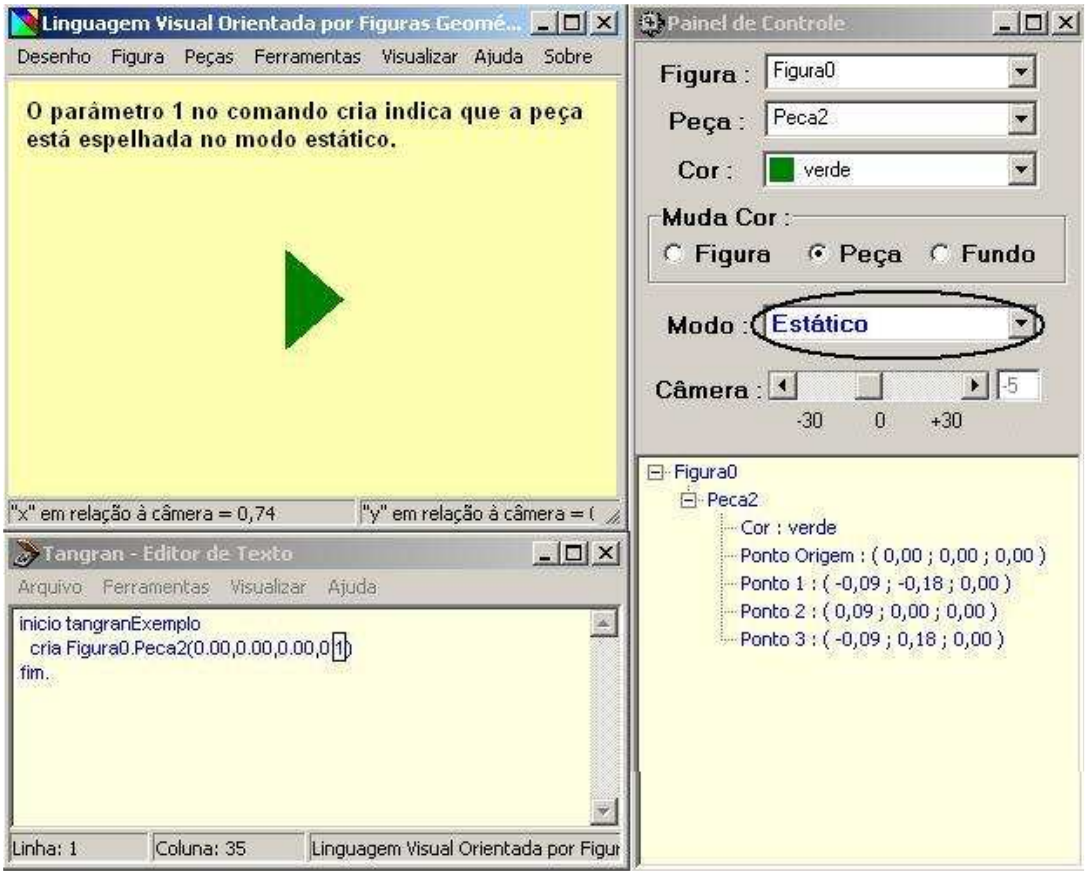


Figura 38 – Espelho no modo estático

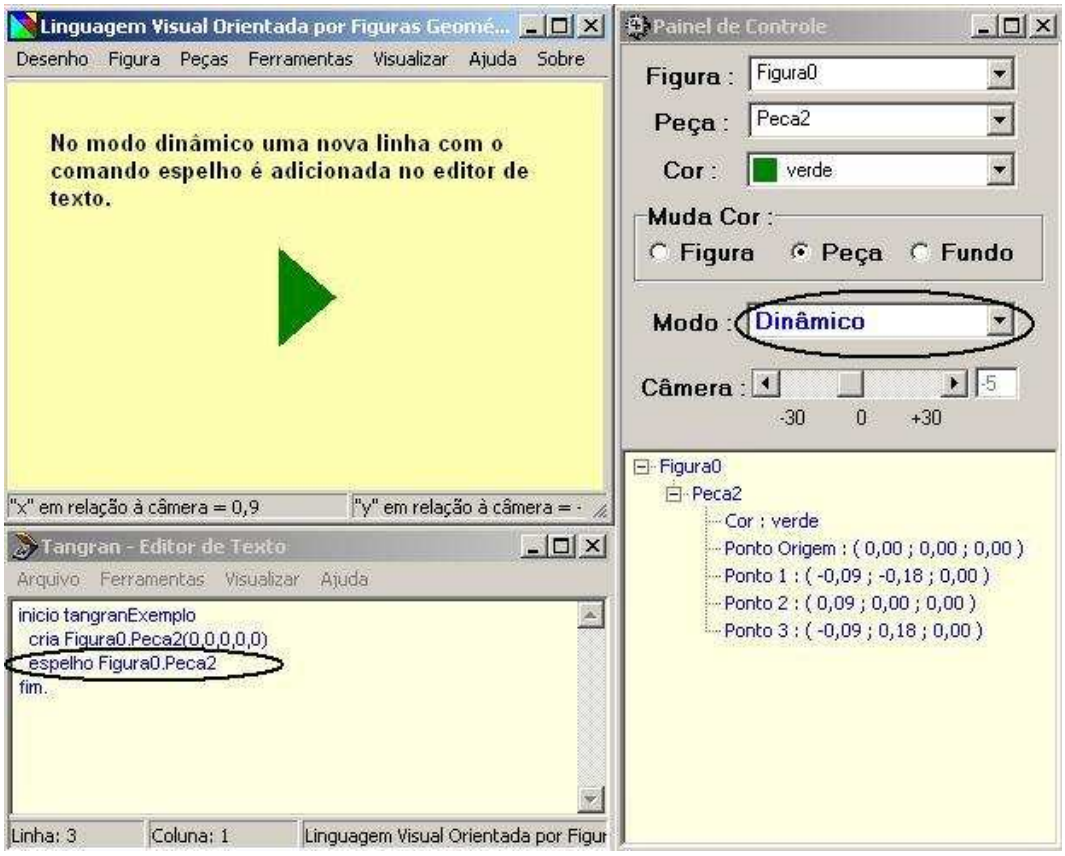


Figura 39 – Espelho no modo dinâmico

Existem na tela do editor de figuras os menus `Desenho`, `Figura`, `Peças`, `Ferramentas`,

`Visualizar, Ajuda e Sobre`.

No menu `Desenho`, o usuário têm as opções de:

a) `Pegar Desenho` que carrega um arquivo para dentro do editor de texto;

b) `Guardar Desenho` que serve para o usuário salvar os comandos que estão no editor de texto, em um arquivo em disco no local já definido;

c) `Guardar Desenho com o nome...` que serve para o usuário guardar os comandos, com o nome informado, que estão no editor de texto, em um arquivo em disco;

d) `Apagar Desenho` que limpa o editor de figuras, o editor de texto e as informações do painel de controle;

e) `Sair` que fecha a ferramenta.

Para identificar os pontos das peças no editor de figuras, deve-se posicionar o *mouse* sobre os pontos para obter a identificação do mesmo.

Estas opções são apresentadas na figura 40.

Figura 40 – Menu `Desenho`

No menu `Figura` o usuário tem a opção de `Criar Figura` (figura 41), onde poderão ser adicionadas até sete (7) peças, do menu `Peças` (figura 42).

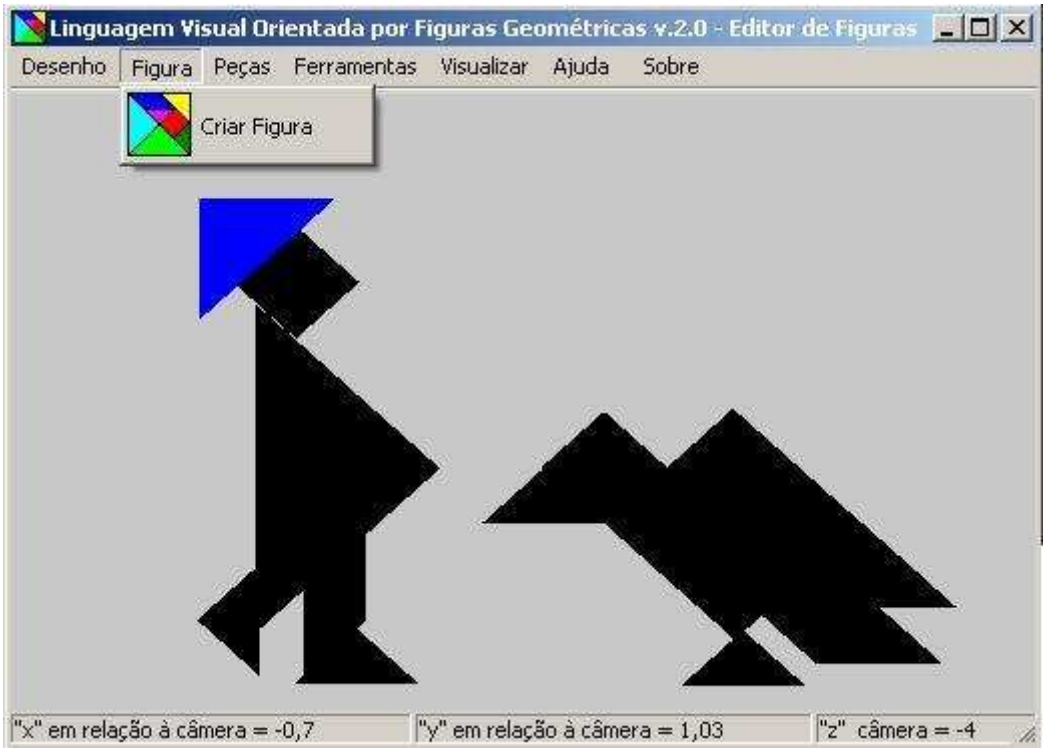


Figura 41 – Menu `Figura`

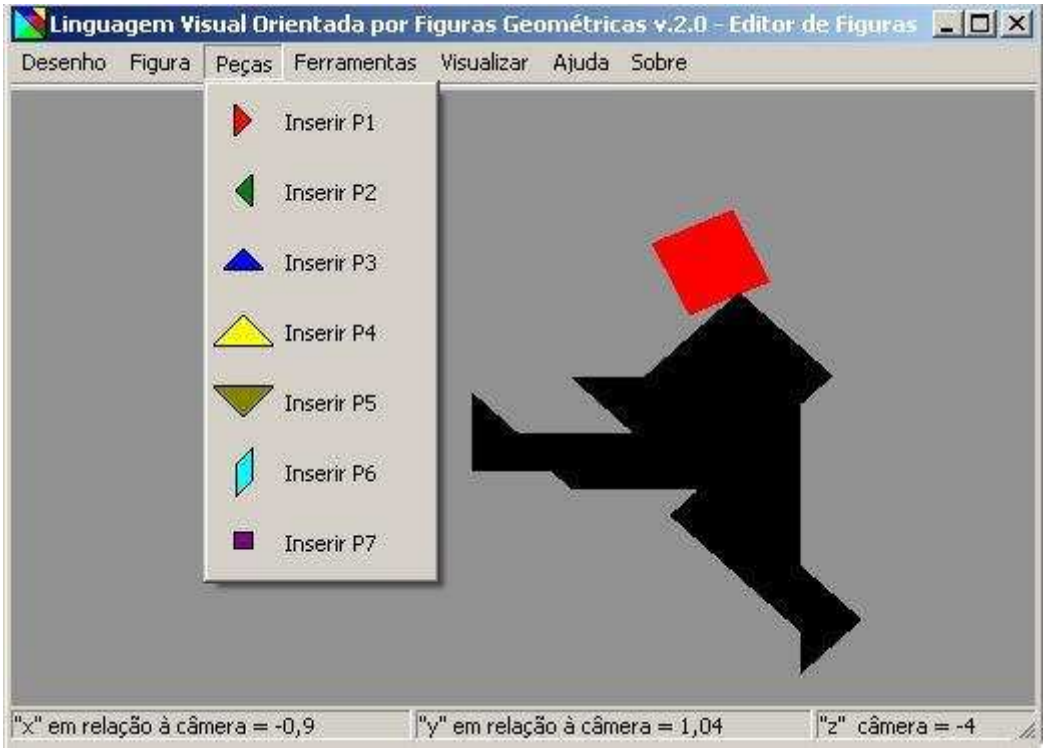


Figura 42 – Menu `Peças`

No menu `Ferramentas` o usuário pode inserir o comando `piscar()` no editor de texto, através da opção `Inserir Comando Piscar`; pode inserir o comando `repete` no editor de texto, através da opção `Inserir Comando Repete`. Além disso, através do menu `Ferramentas` é permitido comandar a verificação do código, antes de executá-lo. Caso erros sejam detectados, é apresentada uma tela com os mesmos. 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 de texto. Estas opções são apresentadas na figura 43.

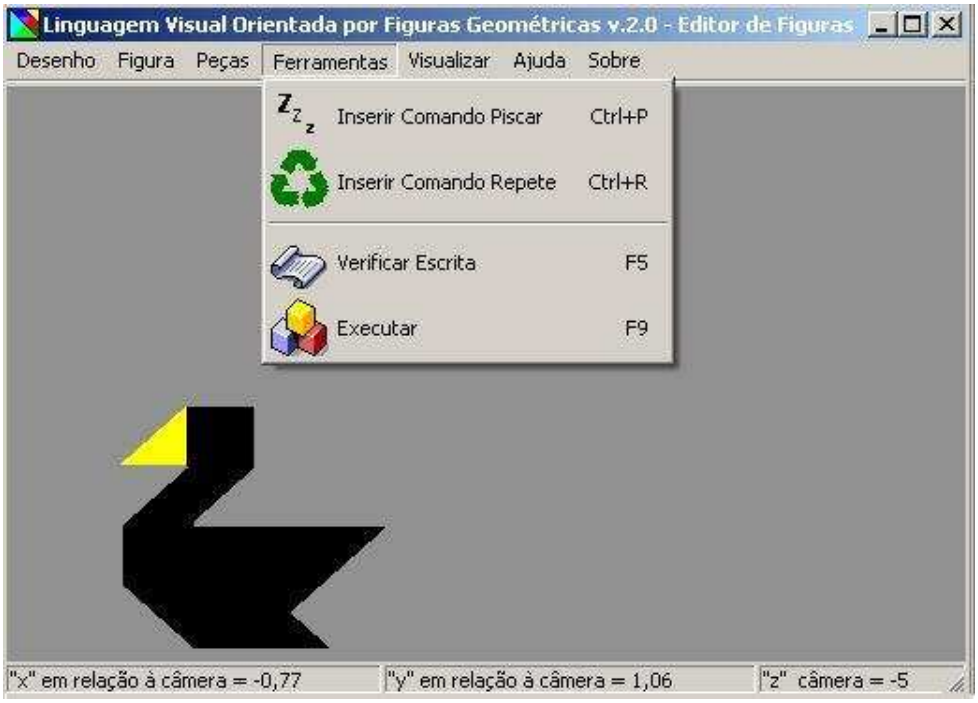


Figura 43 – Menu `Ferramentas`

O menu `Visualizar` contém as opções `Editor de Texto` e `Painel de Controle`. A opção `Editor de Texto` ativa a tela do editor de texto e a opção `Painel de Controle` ativa a tela do painel de controle. A figura 44 mostra o menu `Visualizar` com suas opções.

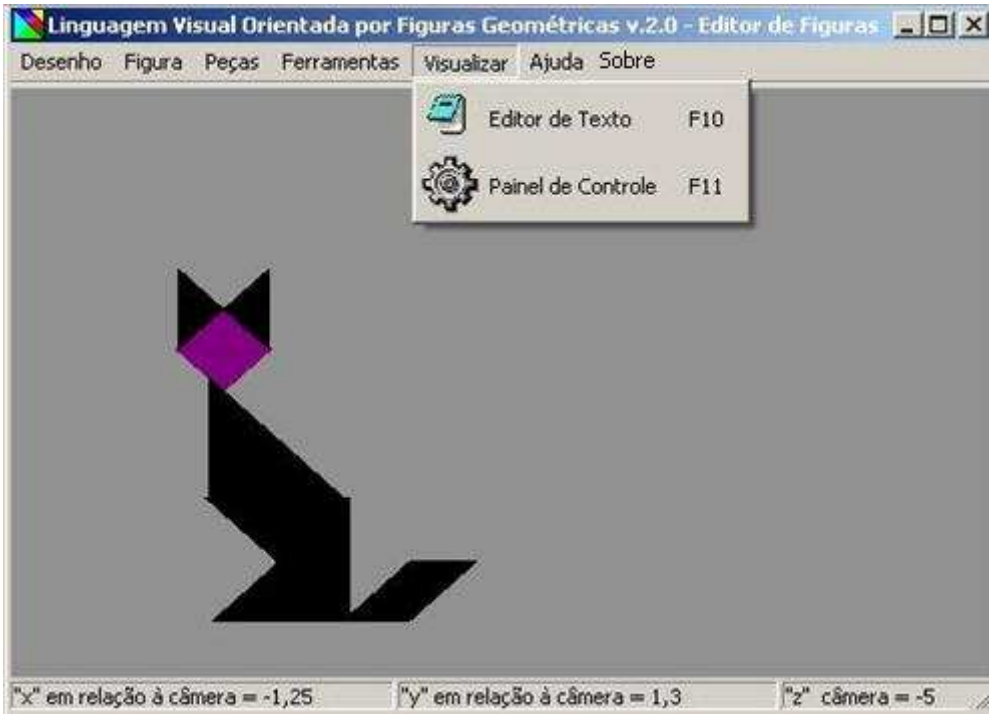


Figura 44 – Menu `Visualizar`

O menu `Ajuda` contém a opção `Manual da ferramenta` (figura 45). O manual da

ferramenta contém um exemplo dos comandos da linguagem e uma breve descrição dos mesmos (figura 46).

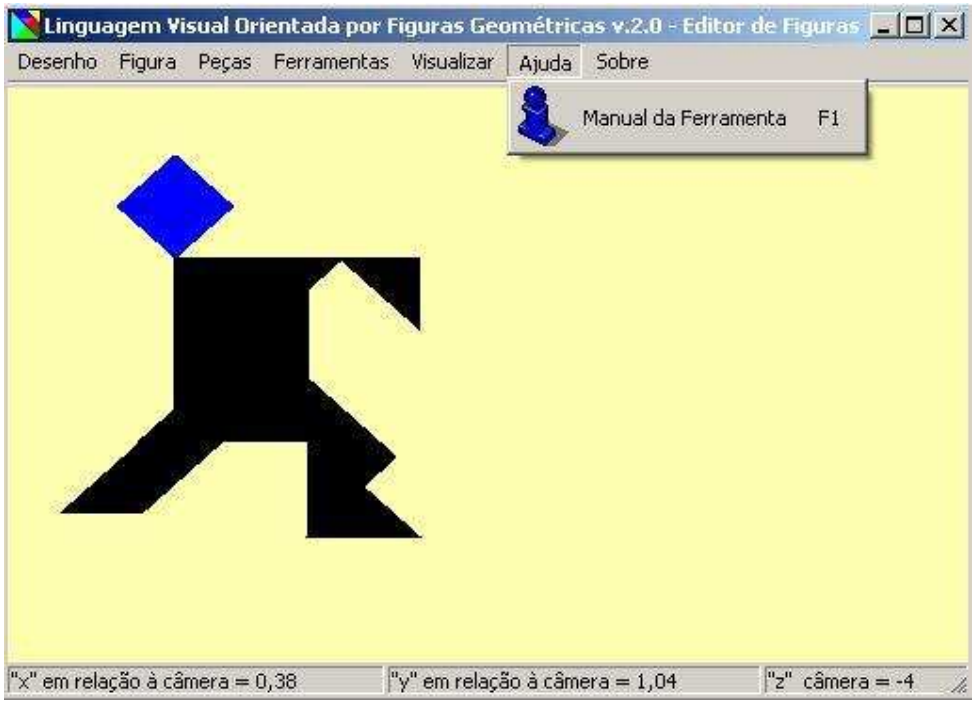


Figura 45 – Menu `Ajuda`

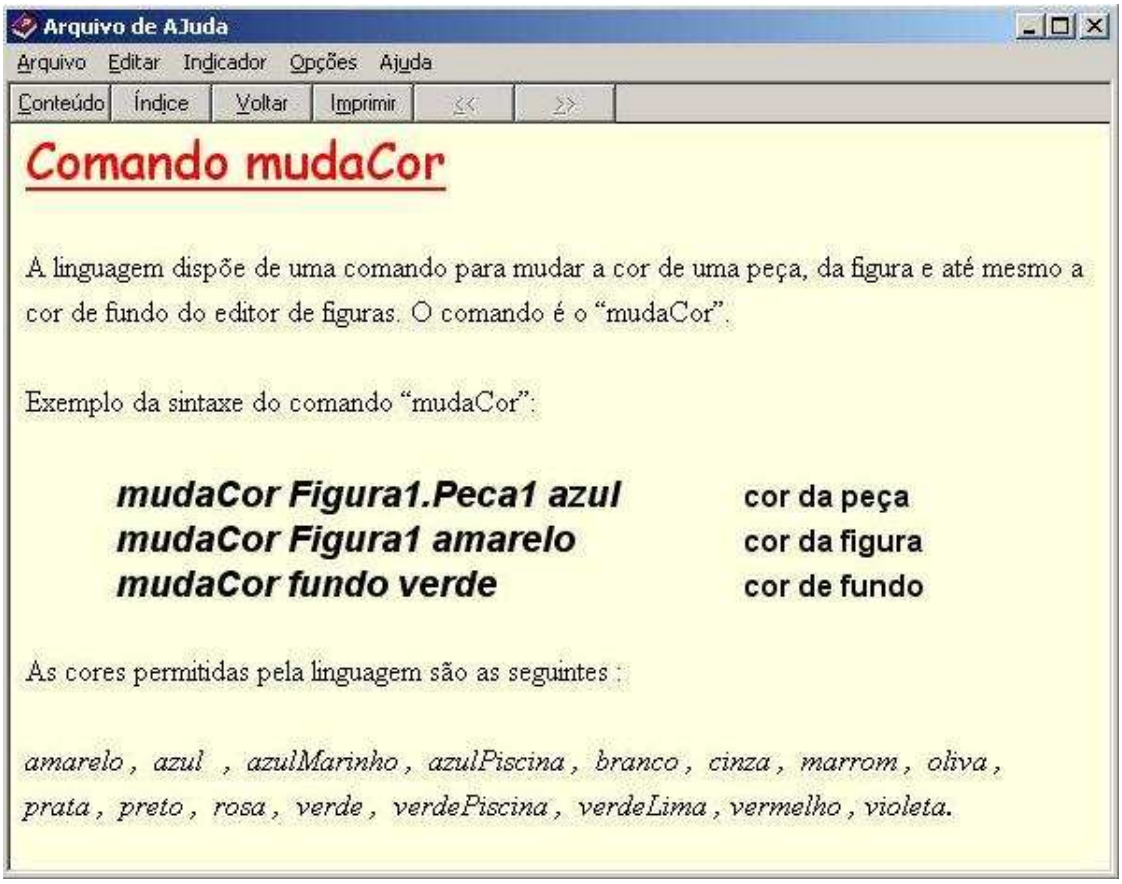


Figura 46 – Manual da ferramenta (comando `mudaCor`)

Por fim tem-se o menu `Sobre`, apresentado na figura 47.

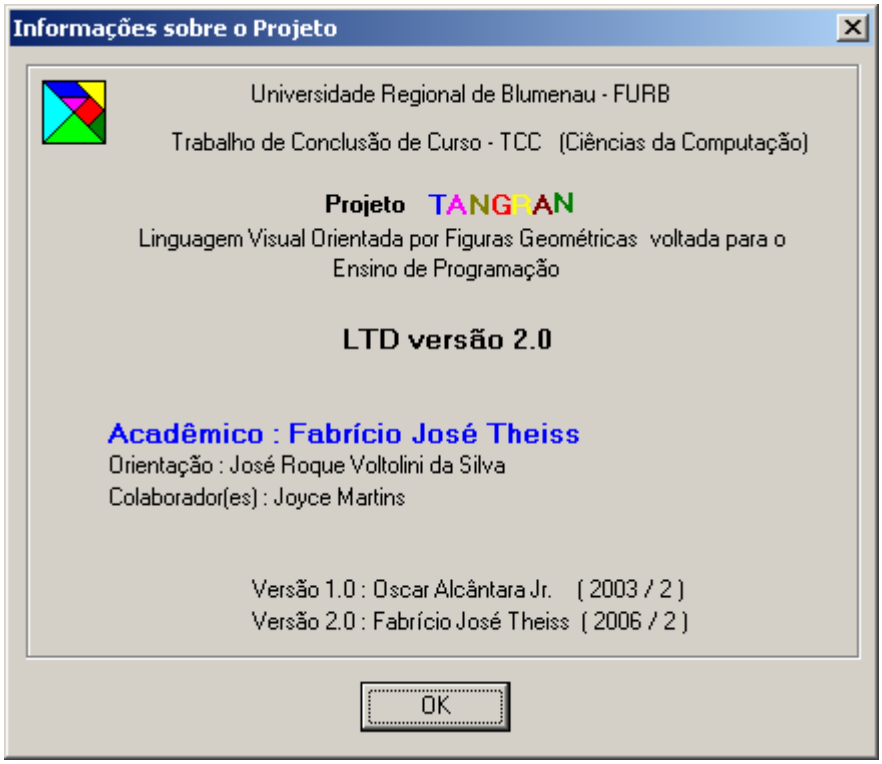


Figura 47 – Menu `Sobre`

3.3.2.3 Editor de texto

O editor de texto é local onde ficam todos os comandos utilizados no desenho, sejam eles digitados pelo usuário ou adicionados automaticamente, conforme as ações do usuário no editor de figuras. A figura 48 apresenta o editor de texto com alguns comandos informados.

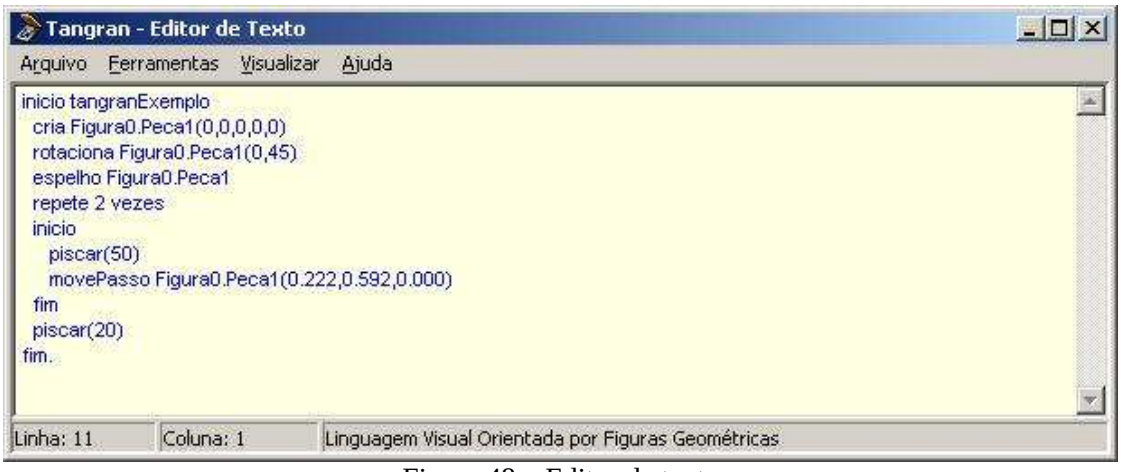


Figura 48 – Editor de texto

No editor de texto existem os menus `Arquivo`, `Ferramentas`, `Visualizar` e `Ajuda`.

O menu `Arquivo` possui as opções `Pegar Desenho`, `Guardar Desenho`, `Guardar Desenho com o nome...`, `Apagar Desenho` e `Sair`. Essas opções tem as mesmas

funcionalidades que as opções existentes no menu `Desenho` do editor de figuras. As opções do menu `Arquivo` são apresentadas na figura 49.

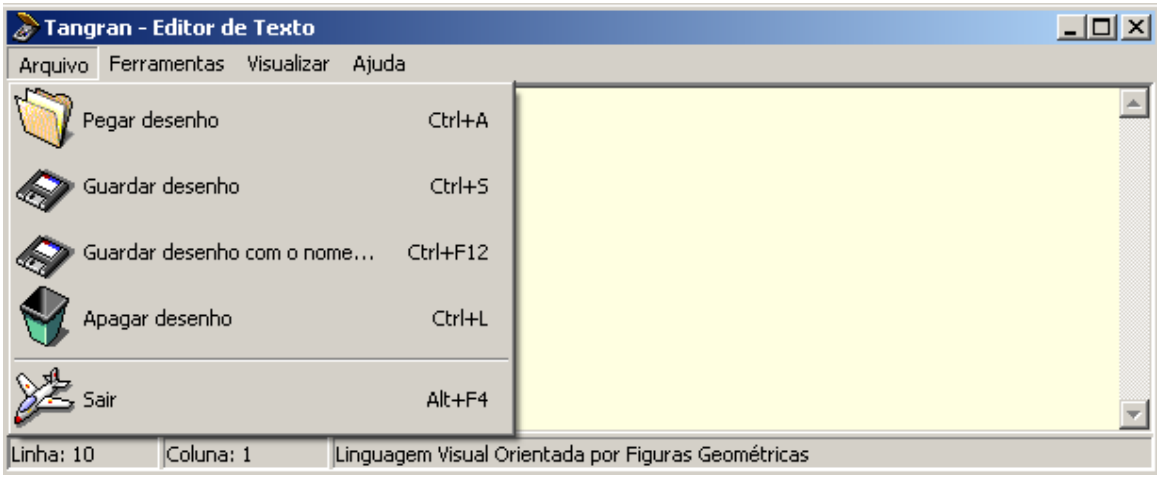


Figura 49 – Menu `Arquivo`

O menu `Ferramentas` do editor de texto é igual e com as mesmas funcionalidades do menu `Ferramentas` do editor de figuras. O menu `Ferramentas` é apresentado na figura 50.

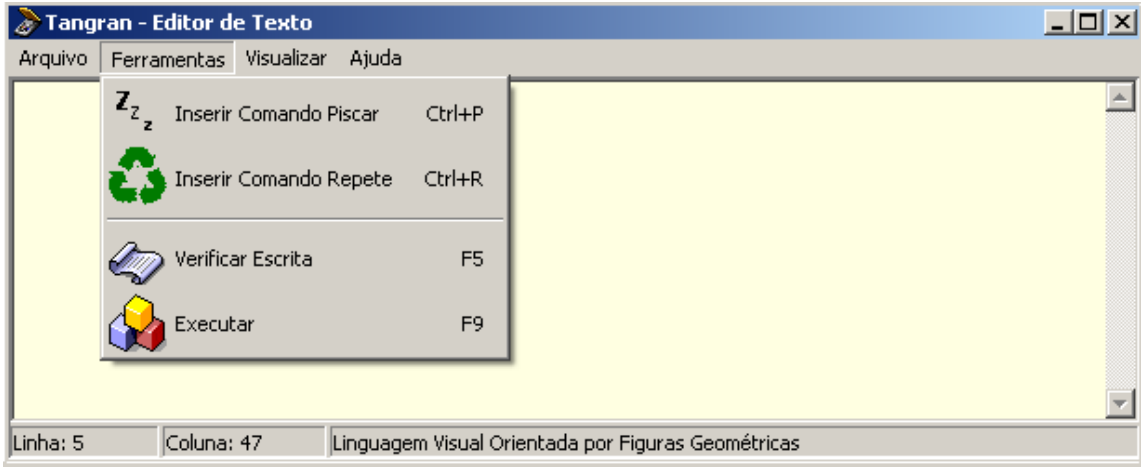


Figura 50 – Menu `Ferramentas`

O menu `Visualizar` contém as opções `Desenho` e `Painel de Controle`. A opção `Desenho` ativa a tela do editor de figuras e a opção `Painel de Controle` ativa a tela do painel de controle. A figura 51 mostra o menu `Visualizar` com suas opções.

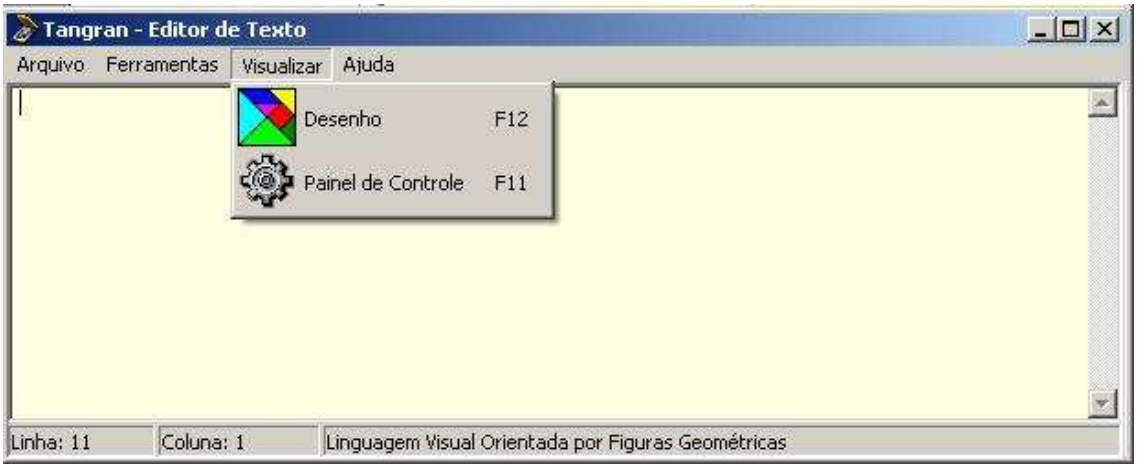


Figura 51 – Menu `Visualizar`

Por fim tem-se a opção `Manual da Ferramenta`, no menu `Ajuda` (figura 52) que contém informações sobre como utilizar a ferramenta (figura 46).

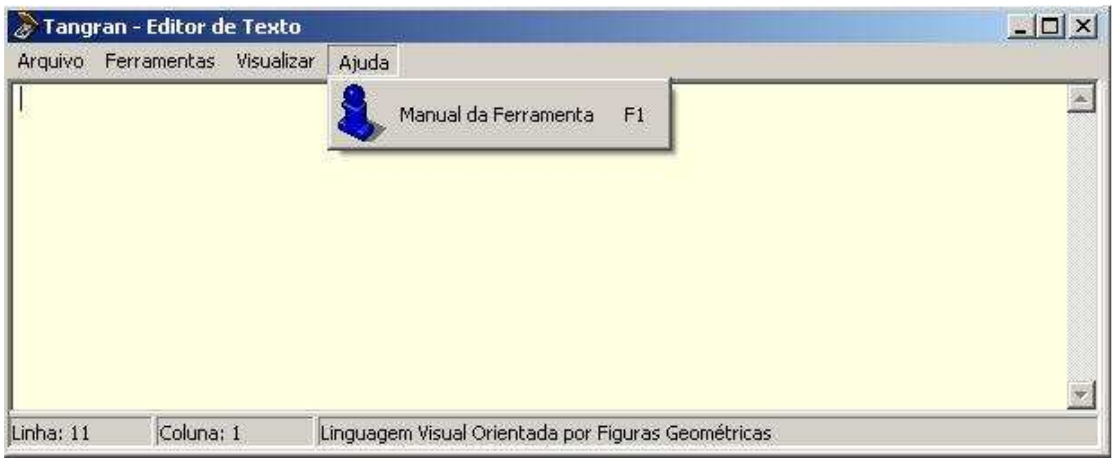


Figura 52 – Menu `Ajuda` (manual da ferramenta)

3.4 RESULTADOS E DISCUSSÃO

Nesta segunda versão do LTD, o usuário tem a possibilidade de executar comandos que movimentam as peças ou figuras do desenho em 3D (profundidade e aproximação).

A versão 2.0 do LTD difere-se do Mundo dos Atores, pois não trabalha com variáveis. Ainda, o Mundo dos Atores é voltado para o ensino de programação OO, enquanto que o LTD é voltado para o ensino de programação seqüencial.

O LTD difere-se do Logo, na forma de representação na tela. O Logo é representado por um triângulo ou por um desenho de uma tartaruga e o LTD é representado pelas peças do jogo Tangram. Ainda, o LTD difere do Logo, visto que o usuário constrói desenhos com

formas geométricas pré-definidas e, na medida que se vai desenhando, o código da linguagem vai sendo gerado. Este código poderá ser alterado diretamente no texto, surtindo efeito no desenho.

Na versão 1.0 do LTD existe uma opção onde são mostrados os pontos das peças que estão desenhadas (Figura 53). Nesta nova versão do LTD, isso foi modificado, sendo que não há mais essa opção no menu. Agora o usuário deve passar o *mouse* sobre algum ponto das peças no editor de figuras e automaticamente irá aparecer uma mensagem identificando o ponto no qual está o *mouse* (Figura 54).

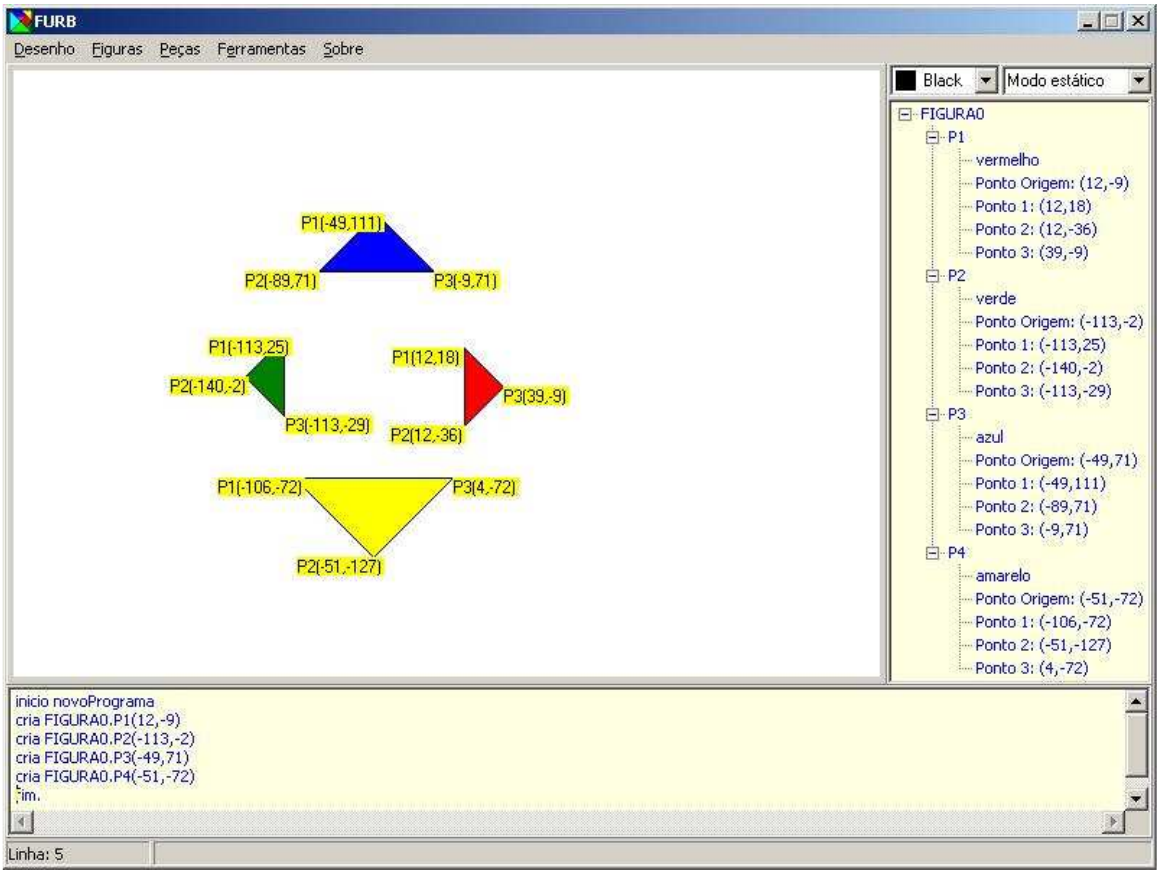


Figura 53 – Mostrar pontos v.1.0

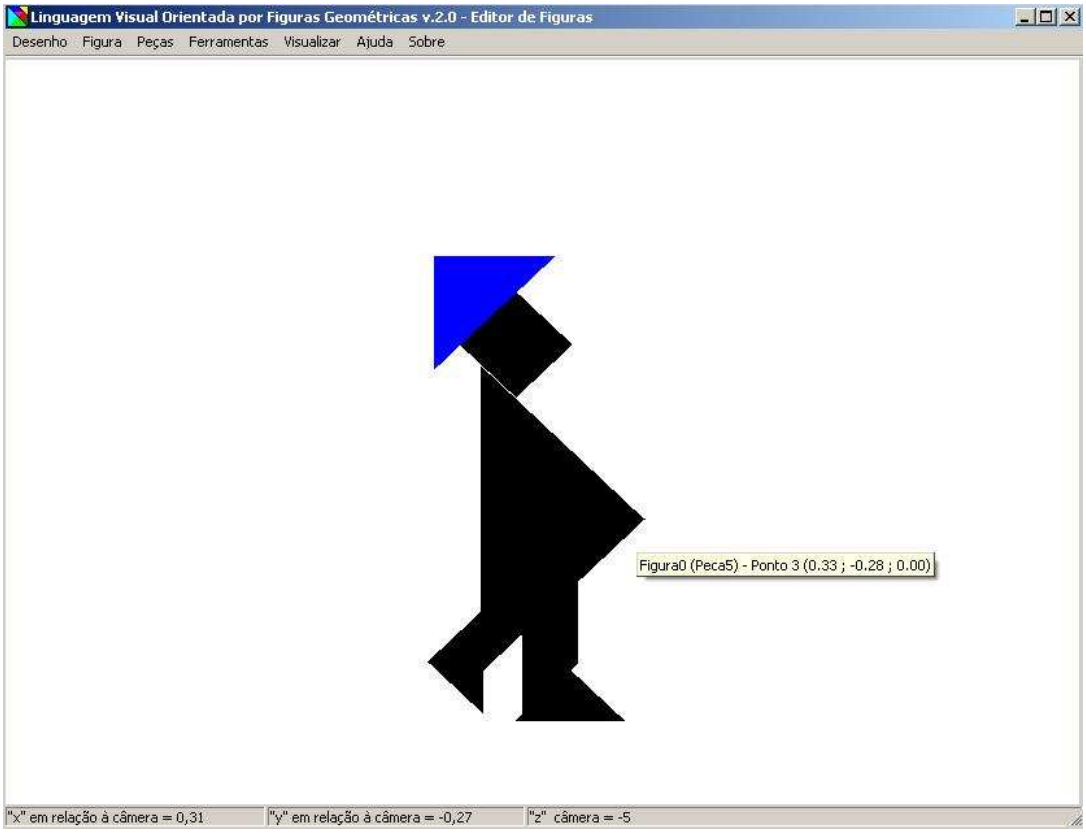


Figura 54 – Mostrar pontos v.2.0

No quadro 35 é apresentado um exemplo de como fazer uma “hélice” na versão anterior e na versão 2.0 do LTD.

Versão anterior:	***Versão 2.0:***

`inicio novoPrograma`
`  cria FIGURA0.P1(0,0)`
`  repete 2 vezes`
`  inicio`
`  rotaciona FIGURA0.P1(1,90)`
`  piscar(250)`
`  rotaciona FIGURA0.P1(2,90)`
`  piscar(250)`
`  rotaciona FIGURA0.P1(3,135)`
`  piscar(250)`
`  rotaciona FIGURA0.P1(1,45)`
`  fim`
`fim.`
	`inicio tangranExemplo`
	`  cria Figura0.Peca1(0,0,0,0,0)`
	`  repete 4 vezes`
	`  inicio`
	`  piscar(150)`
	`  rotaciona Figura0.Peca1(3,+90)`
	`  fim`
	`fim.`

Quadro 35 – Comparação entre as versões do LTD (fazendo uma hélice)

O código na versão 2.0 diminuiu, pois foi modificado o algoritmo de classificação de peças usada na versão anterior, o qual trocava a ordem dos pontos da peça após uma manipulação da mesma. Na versão atual, essa classificação é dada no momento da criação da peça, e após essa classificação os pontos não trocam de ordem. O aspecto redigibilidade nesta

versão 2.0 do LTD ficou mais valorizado, conforme pode ser visto no quadro 35. O algoritmo usado na versão anterior é apresentado no quadro 36.

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 36 – Algoritmo de classificação dos pontos de uma peça

A figura 55 apresenta o resultado da execução dos comandos do quadro 35 da versão 2.0 do LTD.

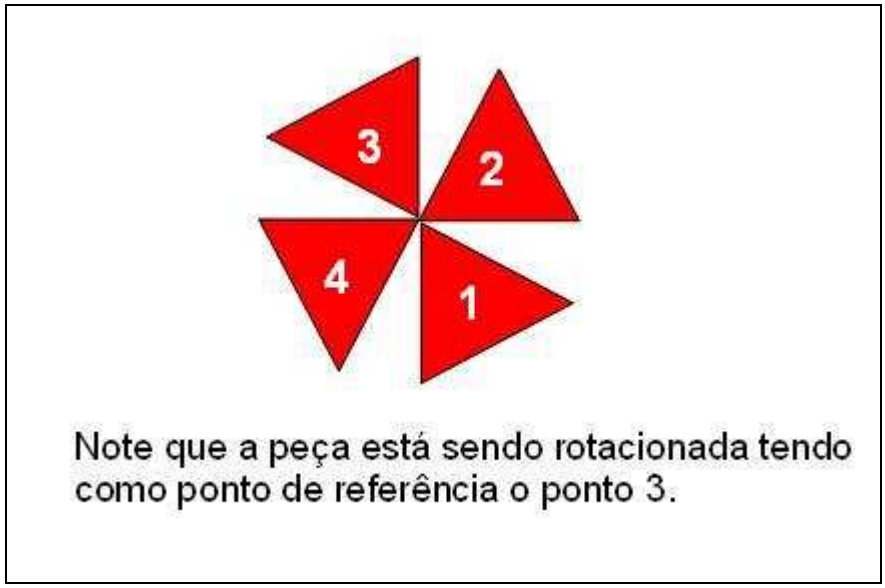


Figura 55 – Hélice (resultado da execução)

Nesta versão 2.0 do LTD, o conceito de programação visual ficou mais evidente, visto a facilidade de inserir comandos como piscar e rotacionar a partir do menu.

4 CONCLUSÕES

O objetivo de re-implementar a ferramenta (LTD) descrita em Alcântara Jr (2003) foi alcançado. Nessa re-implementação foi incluído a terceira dimensão (3D) permitindo a visualização em perspectiva (profundidade), visto que a versão anterior do LTD limita-se a apenas duas dimensões. A especificação de todos os comandos da linguagem foram revistos e alguns alterados, para dar suporte ao manuseio da terceira dimensão, ou seja, foi incluído a coordenada "z" em alguns comandos (cita-se como exemplo o comando `movePasso Figura0( x , y , z))`. Os erros existentes na versão anterior citados no item 2.7 foram corrigidos nessa nova versão. Ainda, o objetivo de re-implementar a ferramenta utilizando o conceito de O.O. foi parcialmente alcançado.

As telas do editor de figuras e do editor de texto foram separadas, permitindo assim, que os editores tenham tamanho dinâmico, facilitando a visualização de várias linhas de comandos no editor de texto. Ainda, a tela denominada painel de controle também foi separada, na qual foram adicionadas novas funcionalidades tais como: selecionar uma peça ou uma figura, alterar a cor de fundo, cor da peça ou cor da figura no editor de figuras, modificar a posição da câmera de visualização e ainda mostra uma árvore com todas as informações do desenho.

Foi desenvolvido um *help* (manual da ferramenta), visto que na versão descrita por Alcântara Jr (2003) não existia nenhuma ajuda sobre os comandos da linguagem para o usuário.

O ambiente de programação utilizado foi o Delphi 7 da Borland com o auxílio de rotinas gráficas disponibilizadas pela biblioteca gráfica OpenGL.

Ainda, alguns códigos foram revisados e reutilizados da versão 1.0 do LTD. Como exemplo, cita-se o código usado para salvar os comandos do editor de texto (`Guardar desenho`) e o salvar comandos com o nome informado (`Guardar desenho com o nome...`).

Um problema encontrado nesta versão do LTD (2.0) está no editor de figuras, quando da movimentação de peças, onde a tela pisca frequentemente durante a operação, causando um desconforto visual.

Outra limitação existente é a falta de uma função para realizar o "pan" (mover o desenho sem tirar da escala). Ainda, quando diminui-se a altura do editor de figuras, ocorre uma distorção no tamanho das peças. Esta distorção não acontece quando do redimensionamento em relação à largura.

4.1 EXTENSÕES

Como extensões para esta ferramenta, sugere-se:

a) a criação de *links* visuais entre os comandos textuais e as peças no editor de figuras, por exemplo *hiperlink's*;
b) tratar as colisões, ou seja uma peça não deve ocupar o mesmo espaço de outra;
c) implementar processos concorrentes;
d) implementar classes;
e) utilizar valores inteiros para as coordenadas (x, y e z).

REFERÊNCIAS BIBLIOGRÁFICAS

ALCÂNTARA JR, O. **Protótipo de uma linguagem de programação de computadores orientada por formas geométricas, voltada ao ensino de programação**. 2003. 58 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

ANDRADE, A. F.; HOFFMANN, A. B.; WAZLAWICK, R. S. **Aprendizagem colaborativa em mundos virtuais**. [S.l.], 1998. Disponível em: <http://www.c5.cl/tise98/html/trabajos/mundosv/index.htm >. Acesso em: 04 abr. 2006.

BARANAUSKAS, M. C. C. et al. Uma taxonomia para ambientes de aprendizado baseados no computador. In: VALENTE, J. A. (Org.). **O computador na sociedade do conhecimento**. São Paulo: USP, 1999. p. 45-68. Disponível em: <http://www.inf.ufsc.br/~edla/mec/>. Acesso em: 05 abr. 2006.

FARRER, H. et al. **Pascal estruturado**. 3. ed. Rio de Janeiro: LTC, 1999.

FERNANDES, T. **Tangram**. [São Paulo], 2004. Disponível em: <http://www.geocities.com/tania1974pt/index.html>. Acesso em: 21 set. 2006.

FIGUEIREDO, L. H.; CARVALHO, P. C. P. **Introdução à geometria computacional**. Rio de Janeiro: IMPA, 1991.

GESSER, C. E. **GALS**: gerador de analisadores léxicos e sintáticos. 2003. 150 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <http://gals.sourceforge.net>. Acesso em: 03 ago. 2006.

GUDWIN, R. R. **Linguagens de programação**. [Campinas], 1997. Notas de aula. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/ea877/lingpro.ps.gz>. Acesso em: 11 abr. 2006.

INTERFACE. In: WIKIPEDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://pt.wikipedia.org/wiki/Interface>. Acesso em: 25 set. 2006.

KONG, A. M. **Pititi**. [S.l.], 2003. Disponível em: <http://www.pititi.com/jogos/tangram/tangram.htm>. Acesso em: 29 mar. 2006.

LONGHI, D. **China on-line:** conectando você com a cultura chinesa. Caxias do Sul, 2004. Disponível em: <http://www.chinaonline.com.br/artes_gerais/tangram/default.asp>. Acesso em: 30 mar. 2006.

MARIANI, A. C. **O mundo dos atores**: uma perspectiva de introdução à programação orientada a objetos. Florianópolis, 1998. Disponível em: <http://www.inf.ufsc.br/poo/atores>. Acesso em: 30 mar. 2006.

PERSIANO, R. C. M.; OLIVEIRA, A. A. F. **Introdução à computação gráfica**. Rio de Janeiro: LTC, 1989.

SEBESTA, R. W. **Conceitos de linguagens de programação.** 4. ed. Tradução José Carlos Barbosa dos Santos. Porto Alegre: Bookman, 2000.

SILVA, J. R. V.; MARTINS, J.; ALCÂNTARA JR, O. Linguagem orientada por formas geométricas, voltada ao ensino de programação. In: CONGRESSO IBEROAMERICANO DE INFORMÁTICA EDUCATIVA, 7., 2004, Monterrey, México. **Anais**... Monterrey, 2004. p. 1176-1186.

SMALLTALK-80. In: WIKIPEDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://pt.wikipedia.org/wiki/Smalltalk-80>. Acesso em: 13 dez. 2006.

VALENTE, J. A. **Liberando a mente**: computadores na educação especial. Campinas: UNICAMP, 1991.

WALTER, M. **Projeções e câmera sintética em OpenGL**. São Leopoldo, [2004]. Disponível em: <http://www.inf.unisinos.br/~marcelow/ensino/grad/cg/projections.html>. Acesso em: 23 ago. 2006.

WANGENHEIM, A. V. **Tutorial de OpenGL**. Florianópolis, 2005. Disponível em: <http://www.inf.ufsc.br/~awangenh/CG/opengl.html>. Acesso em: 04 abr. 2006.

APÊNDICE A – Significado das ações semânticas

O significado das ações semânticas especificadas no quadro 18 são apresentadas no quadro 37.

Ação	Significado
#0	pega o nome da figura do comando `cria`, verifica se já existe essa figura, se não tem então cria.
#1	pega o nome da peça do comando `cria`, verifica se já existe essa peça na figura. Caso não exista, então cria, senão exibe mensagem de que a peça já existe na figura.
#2	pega a coordenada x da peça no comando `cria`.
#3	pega a coordenada y da peça no comando `cria`.
#4	pega a coordenada z da peça no comando `cria`.
#5	verifica se a figura existe. Se não existir, exibe mensagem informando que a figura não existe.
#6	verifica se a peça existe. Se não existir, informa que a peça não foi criada ainda.
#7	pega a cor selecionada e altera a cor da peça.
#8	pega a cor selecionada e altera a cor da figura.
#9	pega a cor selecionada e altera a cor de fundo.
#10	consiste se o parâmetro do comando `piscar` é um número positivo. Se positivo, executa o comando `piscar`, caso contrário exibe mensagem de erro.
#11	pega o número de vezes que vai repetir o bloco do comando `repete`.
#12	indica o inicio do bloco do comando `repete`.
#13	indica o final do bloco do comando `repete`.
#14	pega o deslocamento para x no comando `movePasso`.
#15	pega o deslocamento para y no comando `movePasso`.
#16	pega o deslocamento para z no comando `movePasso` e desloca a peça da figura selecionada para as novas coordenadas.
#17	pega o deslocamento para z no comando `movePasso` e move a figura selecionada para as novas coordenadas, de acordo com o deslocamento.
#18	executa o comando `espelho`, chamando o método `Espelhar` da classe `TFigura`.
#19	pega o ponto de rotação no comando `rotaciona`.
#20	pega o ângulo de rotação no comando rotaciona. Verifica se o ponto de rotação é válido para a peça selecionada. Se válido, executa a `rotação`, caso contrário exibe mensagem de erro.
#21	pega o ângulo de rotação utilizado como parâmetro no comando `cria`.
#22	indica se a figura está espelhada ou não, zero (0) indica que não espelhou e um (1) indica que tá espelhado.
#23	pega a coordenada z do comando `move` e movimenta a figura selecionada para as novas coordenadas x, y e z.
#24	pega a coordenada z do comando `move` e movimenta a peça selecionada para as novas coordenadas x, y e z.

Quadro 37 – Significado das ações semânticas