HUMBERTO YUKINORI MIYOSHI

MARCOS ANTONIO PERRI

PROGRAMA DE SIMULAÇÃO COMPUTACIONAL DE SISTEMAS RANDÔMICOS DE MANUFATURA

Relatório Final do Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro Mecânico.

Aprovados 16/12/97

São Paulo
1997

HUMBERTO YUKINORI MIYOSHI

MARCOS ANTONIO PERRI

PROGRAMA DE SIMULAÇÃO COMPUTACIONAL DE SISTEMAS RANDÔMICOS DE MANUFATURA

Relatório Final do Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro Mecânico.

Área de Concentração:
Engenharia Mecânica

Orientador:
Sérgio Rabelo

São Paulo
1997

AGRADECIMENTOS

Gostaríamos de agradecer aos professores Sérgio Rabelo e Vanderlei Bernardo, que nos apoiaram e orientaram neste trabalho. O auxílio deles foi de vital importância para a conclusão do mesmo.

DEDICATÓRIA

Dedicamos este trabalho a nossos familiares que nos apoiaram e auxiliaram durante todo o curso.

Agradecimento especial para Maria Teresa, que nos apoiou durante os momentos mais difíceis durante a elaboração deste trabalho.

S U M Á R I O

R E S U M O

Está sendo desenvolvido no Laboratório de Automação da Manufatura, do Departamento de Engenharia Mecânica da Universidade de São Paulo, um projeto de implantação de um sistema de manufatura. Este projeto visa aproveitar o atual laboratório, com suas máquinas operatrizes, implementar um sistema de transporte automatizado de material entre suas máquinas e um controle computadorizado de todo o processo de fabricação. O controle deste sistema utilizará um novo conceito desenvolvido na área de processos de fabricação: Sistema Randômico de Manufatura (*RMS - Random Manufacturing System*).

Para a implantação deste sistema, estão sendo desenvolvidos vários projetos neste laboratório, sob supervisão dos professores Sérgio Rabelo e Vanderlei Bernardo. Há um projeto voltado para o desenvolvimento do transporte de materiais entre as máquinas, outro voltado para o controle do compartimento do torno DNC, outro encarregado da implantação da tecnologia de grupo, e outro responsável pelo desenvolvimento de um simulador do sistema.

Este projeto específico tem a finalidade de desenvolver um programa computacional de simulação para o RMS, de modo que se possa saber o desempenho do sistema e determinar qual o melhor algoritmo a ser implantado no controle computacional. Para isto será necessário que o simulador possua todas as características do sistema, tanto das máquinas operatrizes quanto do sistema de transporte e das peças a serem produzidas. Para esta primeira fase, estabeleceu-se como meta a construção de um modelo básico para a análise da viabilidade do projeto.

1. PARTE 1

Apresentação e definição do Projeto

1.1 INTRODUÇÃO

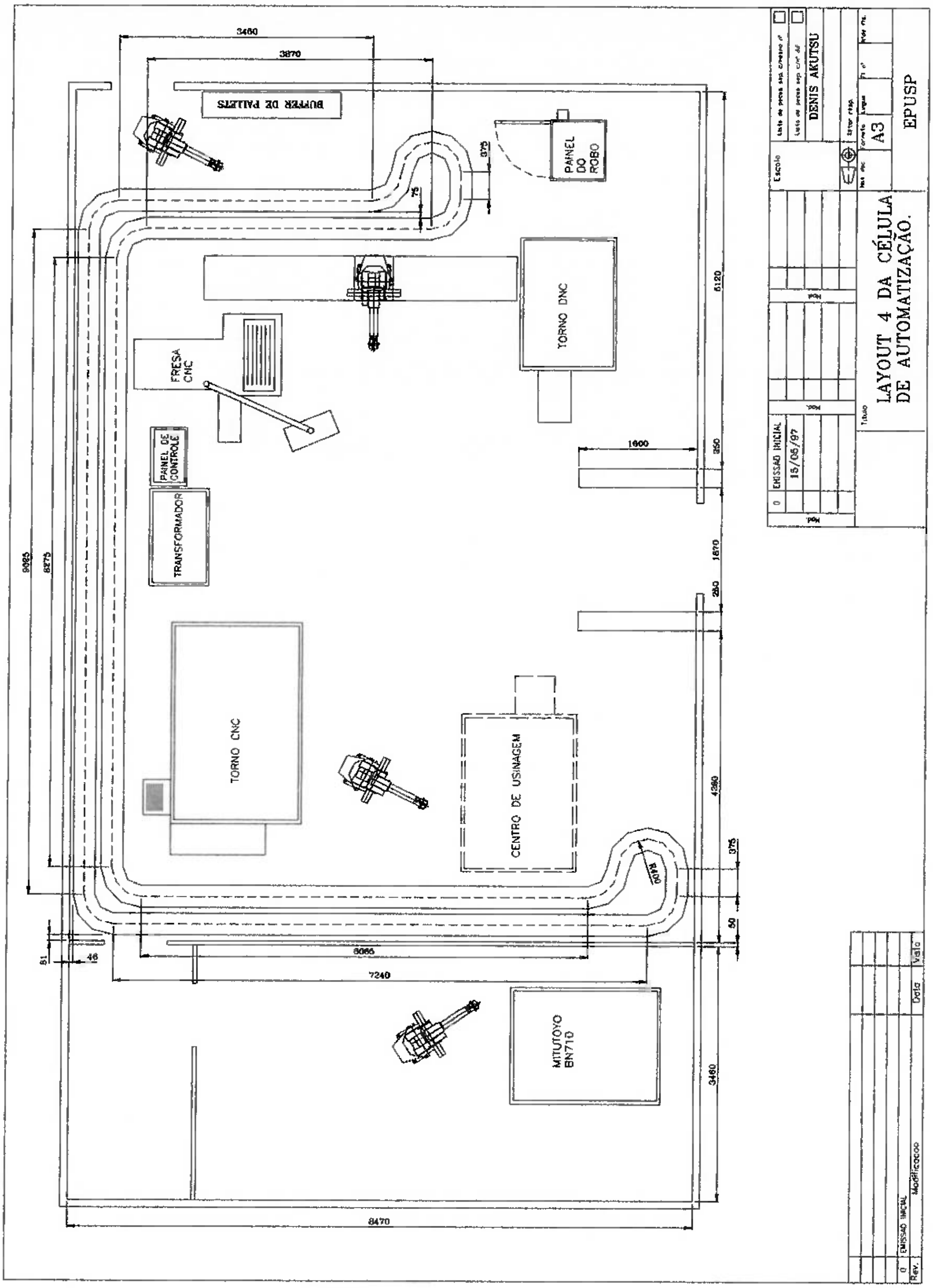
Com o grande avanço tecnológico que se tem obtido neste século, há uma constante atualização no sistema de produção das indústrias. Cada tecnologia inovadora que surge é rapidamente incorporada nas fábricas, de modo que ela otimize seu processo produtivo.

Com a globalização que vem ocorrendo no mercado mundial, estes avanços vêm sendo cada vez mais almejado pelas indústrias, de modo que estes possam se manter em um mercado de forma competitiva.

Neste contexto, os Sistemas Flexíveis de Manufatura (*FMS - Flexible Manufacturing Systems*) vêm sendo implantados e modificados nas indústrias desde a década de 70. Uma destas modificações do FMS é o Sistema Randômico de Manufatura (*RMS - Random Manufacturing System*), que visa dar maior autonomia ao sistema produtivo, de modo que ele possa se adaptar às mudanças que ocorram tanto internamente ao sistema (como por exemplo a quebra de máquinas ou ferramentas), quanto externamente (mudança no pedido de número de peças, ou mesmo de peça).

Verificou-se que não era possível desenvolver um sistema que produzisse qualquer tipo de peça sem que deixasse de ser produtivo e dispendioso. Para contornar este problema desenvolveu-se o conceito de Tecnologia de Grupo (*GT - Group Technology*), que analisa as peças a serem utilizadas e as divide em grupos que possuam algum tipo de

semelhança que possa ser aproveitada no processo produtivo.Layout do Laboratório de Automação da Manufatura do Depar-tamento de Engenharia Mecânica de Universidade de São Paulo.


BUFFER DE PALLETS
PAINEL DO ROBO
TORNO DNC
FRESA CNC
PAINEL DE CONTROLE
TRANSFORMADOR
TORNO CNC
CENTRO DE USINAGEM
MITUTOYO BN710
3460
3870
6120
1600
1670
4360
3460
8470
7240
9065
8275
LAYOUT 4 DA CÉLULA DE AUTOMATIZAÇÃO.
EPUSP
A3
DENIS AKUTSU
EMISSAO INICIAL
15/05/97

Visando a implantação destes conceitos no Laboratório de Automação da Manufatura do Departamento de Engenharia Mecânica de Universidade de São Paulo, estão em andamento vários projetos paralelos, responsáveis por partes do projeto total. O laboratório é composto atualmente pelos seguintes equipamentos:

- Torno Traub CNC - TND360, com comando TX8;
- Robô ABB - IRB62;
- Torno Mazak DNC - T32;
- Máquina de Medição de Coordenadas - Mitutoyo;
- Fresadora Traub CNC UF11, com comando TNC145C.

A seguir serão apresentados alguns dos conceitos que serão utilizados neste projeto.

1.1.1 TECNOLOGIA DE GRUPO

Com os novos conceitos de industriais que surgiram nas últimas décadas, viu-se a necessidade de organizar (*layout*) e dividir as máquinas de forma a obter um melhor rendimento delas. Para otimizar mais o processo, viu-se a necessidade de dividir as peças em grupos com algumas características em comum. Desse modo, a organização das máquinas foi facilitado e houve um melhor rendimento do processo, pois as máquinas responsáveis pela fabricação de determinados

tipos de peças passaram a ser colocadas em locais próximos, criando células de produção.

Tecnologia de grupo é uma metodologia de fabricação em que peças semelhantes são identificadas e agrupadas para que se possa aproveitar desta semelhança no processo de fabricação.

Estas peças que apresentam semelhanças são separadas em famílias de peças, onde cada família apresenta semelhanças de projeto e fabricação. Devido a isto, obtém-se uma melhora na eficiência do processo de fabricação. Eficiência obtida através do arranjo de equipamentos em grupos de máquinas ou células de produção para facilitar o fluxo de trabalho.

O projeto de peças também é melhorado, devido à facilidade de classificação e codificação das peças. A classificação e codificação das peças se preocupa com a identificação da semelhança entre as peças e listando estas semelhanças num sistema de códigos. A semelhança de peças são de dois tipos:

1. atributos de projeto - forma geométrica e dimensões;
2. atributos de fabricação - seqüência das etapas dos processos de fabricação necessários.

A codificação das peças pode ser utilizada em sistemas automatizados de planejamento de processos.

1.1.1.1 FAMÍLIA DE PEÇAS

Uma família de peças é formado por peças com semelhança geométrica e dimensional, ou com semelhança no processo de fabricação.

Geralmente são utilizados um dos três métodos seguintes para a divisão de peças em famílias:

1. inspeção visual;
2. classificação e codificação através da análise de dados do projeto e processo;
3. análise do fluxo de produção (*PFA - production flow analysis*).

O método da inspeção visual é o menos sofisticado e o menos caro. Ele implica na classificação de peças através da aparência física das peças.

O segundo método envolve a classificação de peças em famílias pelo exame individual dos atributos de projeto e/ou fabricação de cada peça.

O terceiro método analisa os dados contidos no fluxo de produção. Deste modo, peças que passam pelas mesmas máquinas em seqüências semelhantes são agrupadas numa mesma família.

1.1.1.2 CLASSIFICAÇÃO E CODIFICAÇÃO DE PEÇAS

Entre os três métodos descritos anteriormente, este é o que necessita de maior tempo e de maior complexidade. Muitos sistemas foram desenvolvidos para trabalhar com este método, mas não foi definido nenhum sistema padrão. Isto se

deve ao fator de que um sistema que funcione bem em um tipo de empresa pode não funcionar adequadamente em outra empresa, devido à variedade de produtos.

Este método é dividido em três sistemas de codificação:

1. sistemas baseados em atributos de projeto das peças;
2. sistemas baseados em atributos de fabricação das peças;
3. sistemas baseados em atributos de projeto e fabricação das peças.

O esquema de codificação consiste numa seqüência de números que identificam os atributos de projeto e de fabricação das peças. Este esquema de codificação pode apresentar dois tipos de estruturação:

1. estrutura hierárquica - nesta estrutura de códigos, a interpretação de cada dígito sucessor depende o valor do dígito predecessor;
2. estrutura em série - neste tipo de código, o significado de cada dígito é fixo, independendo do valor do dígito predecessor.

O número de dígitos necessários na codificação pode variar de 6 a 30. Em esquemas de codificação baseado apenas em atributos de projeto requerem poucos dígitos, no máximo 12. Em sistemas de codificações modernos, que utilizam

atributos de projeto e fabricação, podem se necessários de 20 a 30 dígitos.

A seguir está listada uma relação dos sistemas de codificação mais utilizados:

- Brisch System → Brisch-Birn, Inc.;
- CODE → Manufacturing Data Systems, Inc.;
- CUTPLAN → MetCut Associates;
- DCLASS → Brigham Young University;
- MultiClass → OIR - Organization for Industrial Research;
- Part Analog System → Lovelace, Lawrence & Co., Inc.

1.1.1.3 ANÁLISE DO FLUXO DE PRODUÇÃO

A análise do fluxo de produção (*PFA - Production Flow Analysis*) é um método que identifica famílias de peças e associa as com grupos de máquinas. A PFA é utilizada para analisar a seqüência de operações e a rota das peças entre as máquinas. As peças que possuem rotas iguais ou semelhantes são agrupadas em famílias que são utilizadas para formar células lógicas de máquinas em locais com layout de tecnologia de grupo.

O procedimento utilizado pela PFA é dividido nas seguintes etapas:

1. Coleta de dados - o primeiro passo na PFA é determinar o objetivo do estudo e a coleta dos dados necessários.

2. Ordenação das rotas de processo - o segundo passo consiste em dividir as peças em grupos com similaridade nas rotas de processo.
3. Diagrama de PFA - este diagrama consiste numa representação gráfica da rota de processo de cada tipo de peça.
4. Análise - é a parte mais complexa e importante da PFA. De acordo com o padrão de dados apresentados no diagrama de PFA, deve-se identificar e separar as famílias de peças.

1.1.2 SISTEMA FLEXÍVEL DE MANUFATURA

Houve um enorme avanço tecnológico na área de automação industrial nas últimas décadas. Surgiram máquinas controladas por computadores (CNC - Controle Numérico por Computador) que garantem a precisão e qualidade das peças. Há também o surgimento de robôs e outros sistemas de transportes totalmente automatizados. Aproveitando todo este avanço, e tentando diminuir a participação da mão de obra humana nos processos, surgiu o conceito de sistema flexível de manufatura, onde a influência humana fica restrita apenas ao controle geral durante o processo de fabricação.

Um sistema flexível de manufatura (*FMS - Flexible Manufacturing System*) consiste em um grupo de estações de processamento (predominantemente de máquinas CNC), interconectados por meio de um sistemas automatizado de transporte de materiais, e controlado por um sistema

computadores integrados. A origem deste nome se deve à capacidade do sistema de processar uma variedade de diferentes tipos de peças simultaneamente, sobre o controle de um programa de controle numérico em várias estações de trabalho.

O FMS é feito de três componentes básicos:

1. estações de processamento - geralmente estas estações são compostas por máquinas CNC;
2. transporte e armazenamento de material - são utilizados vários tipos de equipamentos automatizados para o transporte de material entre as estações;
3. sistema de controle por computador - o controle por computador é usado para coordenar as atividades das estações de processamento com o sistema de transporte de materiais no FMS.

Um componente adicional no FMS é o fator humano, necessário para gerenciar as operações de um sistema de manufatura flexível. Como atividades manuais no FMS temos a alimentação de matéria-prima do sistema, a retirada das peças prontas, mudança e configuração de ferramentas, manutenção e reparo de equipamentos, programação de NC, programação e operação dos sistemas computadorizados.

1.1.2.1 TIPOS DE SISTEMAS

Há vários tipos de classificação de FMS. Uma delas diz respeito à diferença entre FMS e células de manufatura.

Geralmente o termo célula é utilizado para um grupo de máquinas operadas manualmente ou automatizados, ou uma combinação dos dois. Uma célula pode possuir ou não um sistema de transporte automatizado de material e/ou controle computadorizado. Já o termo FMS é empregado em sistemas totalmente automatizados, consistindo de estações automatizadas, transporte automatizado e controle computadorizado.

Um outro sistema de classificação analisa o número de máquinas para distinguir sistema de manufatura flexível e célula de manufatura flexível. Um grupo de quatro ou mais máquinas constitui um sistema, e três ou menos máquinas constituem uma célula.

Outra forma de classificação analisa a geometria das peças a serem processadas no sistema. Existem duas categorias: prismáticas e redondas. As peças prismáticas possuem faces planas e requerem de fresamento ou operações afins. As peças redondas possuem superfícies cilíndricas e necessitam de torneamento ou outras operações do mesmo tipo.

A classificação mais importante divide o FMS em dois tipos:

1. FMS dedicado;
2. FMS randômico.

O FMS dedicado é utilizado para produzir uma variedade limitada de peças. A diferença geométrica entre as peças

deve ser pequena e o projeto dos produtos devem ser estáveis. Além disso, a seqüência de operações deve ser idêntica ou muito semelhante entre todas as peças. Neste caso, o layout em linha das máquinas é o mais apropriado, e o sistema pode ser projetado como uma série de sistemas especializados para tornar as operações mais eficientes.

O FMS randômico é o tipo mais apropriado devido às seguintes condições: a família de peças a serem processados é muito maior, as peças podem ter diferenças razoáveis, haverá novas peças projetadas para o sistema e as mudanças das peças pode ser feita no sistema. Para suportar estas variações, o FMS randômico deve ser mais flexível do que o FMS dedicado. Ele deve ser equipado com máquinas que sejam capazes de processar diversos tipos de peças em várias seqüências. Também é necessário um sistema de controle computadorizado mais sofisticado.

1.1.2.2 ESTAÇÕES DE PROCESSAMENTO

O tipo de equipamento utilizado em FMS depende do tipo de trabalho que deve ser feito pelo sistema. Em sistemas projetados para máquinas operatrizes, o principal tipo de estações de processamento são as máquinas CNC. Além deste tipo de máquina, podem ser utilizados outros tipos, como centros de usinagem, módulos de fresamento e retificação, estações de inspeção, etc.

1.1.2.3 TRANSPORTE E ARMAZENAMENTO DE MATERIAIS

O sistema de transporte e armazenamento de materiais em um FMS deve ter as seguintes características:

1. movimento independente e aleatório das peças entre as estações de trabalho;
2. possibilitar o transporte de diversos tipos de peças;
3. local para armazenamento temporário de peças;
4. local apropriado para carga e descarga de matéria-prima e peças prontas;
5. compatibilidade com o controle computadorizado.

O sistema de transporte determina o layout do FMS. Os tipos de layout encontrados atualmente em FMS podem ser divididos em cinco categorias:

1. in-line;
2. loop;
3. ladder;
4. open-field;
5. robot-centered cell.

Os equipamentos utilizados nos sistemas de transporte incluem: transportadores rolantes e de outros tipos, sistemas automatizados de veículos guiados e robôs industriais.

1.1.2.4 SISTEMA DE CONTROLE POR COMPUTADOR

Um FMS deve ser controlado através de computadores. O sistema de controle por computador deve ser responsável por oito tarefas básicas:

1. controle de cada estação de trabalho;
2. distribuição de instruções de controle para as estações de trabalho;
3. controle da produção;
4. controle do tráfego;
5. controle da entrada de material no sistema;
6. monitoramento do sistema de transporte;
7. controle das ferramentas utilizadas nas estações;
8. monitoramento da performance do sistema e geração de relatórios do sistema.

1.1.3 SISTEMA RANDÔMICO DE MANUFATURA

Nos últimos anos houve um aumento da concorrência devido à globalização. Os mercados deixaram de ser disputados apenas por empresas locais. Empresas do mundo inteiro fornecem os mais diversos produtos em qualquer localidade do planeta. Para se manter neste mercado altamente competitivo, as indústrias se viram na necessidade de melhorar seus sistemas de produção, de modo a atender seus clientes o mais rápido possível, diminuindo prazos e melhorando a qualidade.

Os sistemas flexíveis de manufatura se mostram um pouco limitados quanto às modificações de tipos de

produtos. É necessário realizar um estudo para novos pedidos e, em muitos casos, reorganizar as máquinas de forma a melhorarem o desempenho da produção. Todo este processo demanda tempo e muito dinheiro. Quando uma das máquinas do sistema sofre algum defeito, todo o processo no FMS é parado até que se sane o defeito, consertando a máquina ou substituindo-a.

Visando solucionar este problema, vem sendo desenvolvido o sistema randômico de manufatura.

O conceito de sistema randômico de manufatura (*RMS - Random Manufacturing System*) surgiu há pouco tempo. Ele consiste num sistema muito parecido com o FMS, mas difere deste por ser autônomo. Ou seja, o próprio sistema é capaz de se adaptar a mudanças necessárias para seu funcionamento. Se houver mudanças nos pedidos de peças ou alguma máquina quebrar, o sistema automaticamente muda o processo ou redireciona os serviços da máquina danificada para outra máquina. Isto é possível porque o RMS é formado por diversas máquinas que podem ser agrupadas dinamicamente para produzir determinada peça. É como se dentro do RMS fosse formado um FMS para produzir determinada peça e, terminada a produção desta peça, o FMS deixasse de existir.

Este sistema é definido por quatro características:

1. sistema de máquinas autônomas - o RMS consiste em máquinas autônomas que podem determinar o que fazer e como se auto-controlar. Desde que uma

máquina autônoma possua todas as informações necessárias para o seu funcionamento, o RMS está apto a ter alta modularidade e flexibilidade devido à sua arquitetura e ao processo de tomada de decisões;

2. agrupamento dinâmico de máquinas - o RMS não é formado por um grupo fixo de máquinas. Os grupos de máquinas são formadas dinamicamente quando chega o pedido de um determinado tipo de peça, e se desfaz automaticamente assim que a produção desta peça acaba.

3. proposta baseada em distribuição de tarefas - no RMS não há um processo que distribui as tarefas para as máquinas. Em vez disso, as atividades são apresentadas para todas as máquinas e aquelas que estão aptas a realizar as atividades formam um grupo de trabalho. Se mais de uma máquina se apresenta para realizar a mesma tarefa, apenas uma é selecionada, obedecendo-se um critério;

4. nível de controle por sistema de preferências - como o RMS é um sistema distribuído feito com máquinas autônomas, não há um controle central que comande os procedimentos de cada máquina. Para controlar o procedimento no RMS, um sistema de preferências é introduzindo no RMS, com premiação e penalidades. Cada ordem possui dois

atributos, que indicam premiação para o sucesso e penalidade para falhas. Cada máquina do RMS possui uma pontuação que levam em conta estes atributos. Desse modo, há um controle indireto baseado em preferências.

1.1.3.1 SISTEMA DE ARQUITETURA DO RMS

O RMS consiste de muitos elementos como máquinas, ferramentas e dispositivos, transportadores, computadores, rede, etc. Além desses elementos há outros quatro elementos específicos para o RMS: gerente de máquina, gerente de tarefas, task master e tender board.

1. gerente de máquina - o gerente de máquina é conectado em cada máquina autônoma na fábrica e permite que ela planeje suas atividades. O gerente de máquina possui diversas informações que o auxiliam na tomada de decisões.
2. gerente de tarefa - o objetivo do gerente de tarefas é o de organizar grupos de máquinas para realizar as atividades cooperativas entre as máquinas. No RMS, qualquer gerente de máquina pode assumir o papel de gerente de tarefas. O gerente de tarefas busca as informações sobre o planejamento do processo e a distribuição de atividades.
3. *task master* - é o responsável pela coordenação de tarefas no RMS. O task master recebe os pedidos de peças e notifica todos os gerentes de máquina. Ele também avalia

as propostas e seleciona um gerente de tarefa para o melhor planejamento. Para determinar qual o melhor planejamento, o task master utiliza o sistema de preferências, baseado em premiações e penalizações.

4. *tender board* - é o meio de comunicação entre o task master e os gerentes de máquina. Ele mostra as informações referentes aos pedidos de peças, processos planejados e possíveis alternativas, prazos, premiações e penalizações. Ele também guarda os planos de processo que os gerentes de tarefa propõem para os pedidos.

1.1.3.2 PROCEDIMENTO DE OPERAÇÕES DO RMS

A seqüência de operações do RMS pode ser divido em cinco fases:

a) fase 1 - entrada de pedidos:

O procedimento de operação do RMS inicia com a chegada de um pedido de peças. O pedido contém o planejamento de processos, incluindo processos alternativos, prazos, premiações e penalidades para cada tarefa. Os processos especificados devem estar relacionados com os processos executáveis pelas máquinas, e não com as máquinas. O task master verifica o pedido e os processos que as máquinas devem executar, e somente aceita o pedido se a célula apresentar um planejamento viável. Então o task master coloca o pedido no tender board com o tempo de proposta.

b) fase 2 - formação de grupos dinâmicos:

Gerentes de máquinas analisam o pedido no tender board. Quando um gerente de máquina encontra um pedido que necessita de vários processos que ele pode executar, e satisfaz todos os critérios de prazos, premiações e penalidades, ele se torna o gerente de tarefas para este pedido. Caso haja alguma tarefa que ele não possa executar, ele organiza um grupo dinâmico para completar todos os processos planejados. Neste caso, assumindo o papel de gerente de tarefas, consulta os outros gerentes de máquinas, passando-lhes todas as informações referentes ao processo, prazos, premiações e penalidades. Caso o gerente de máquina consultado satisfaça todos os requisitos, ele passa a fazer parte do grupo dinâmico daquele gerente de tarefas, e reserva um período em sua agenda para executar o serviço. Caso não satisfaça, o gerente de tarefas consulta outro gerente de máquinas.

c) fase 3 - propostas:

O tender boar é fechado para o pedido no qual o tempo de proposta termina. O task master verifica os planejamentos propostos para cada pedido. Caso algum pedido não receba nenhuma proposta, o task master modifica os valores de premiações, penalidades e/ou prazos e recoloca o pedido no tender board. Se algum planejamento é proposto, o task master seleciona o melhor planejamento baseado numa função de avaliação,

que pode ver o melhor prazo o número mínimo de máquinas a serem utilizadas, etc. Então o task master notifica o gerente de tarefa responsável pelo planejamento escolhido por sua opção, e avisa os outros gerentes de tarefas que o planejamento apresentado por eles não será utilizado. O gerente de tarefas escolhido avisa todas as máquinas que participaram de seu planejamento a fim de que eles programam seus processos, e os outros gerentes de tarefas finalizam os grupos formados para sua proposta.

d) fase 4 - implementação:

O grupo de máquinas selecionados para a proposta inicia a fabricação do pedido de peças de acordo com o planejamento adotado. O gerente de tarefas monitora todos os processos executados pelas máquinas de seu grupo. Se ocorrer qualquer problema ou atraso, o gerente de tarefas tenta solucionar o problema reprogramando as suas operações ou escolhendo outra máquina para substituir a máquina com problemas. Qualquer informação referente ao processo de implementação é armazenado pelo gerente de tarefas.

e) fase 5 - distribuição de premiações e penalidades:

Quando uma tarefa é finalizada ou interrompida, o gerente de tarefas relata o resultado da implementação ao task master. O task master proporciona o gerente de

tarefas com a premiação para o sucesso e com a penalização para o fracasso das tarefas. Por sua vez, o gerente de tarefas distribui as premiações ou penalidades para as máquinas do grupo, baseado em condições estabelecidas.

1.2 ESTUDO DE VIABILIDADES

A simulação da célula de manufatura randômica deverá atender os seguintes ítens:

1. deverá ser utilizado um sistema computacional para a simulação;
2. o programa deverá ser desenvolvido para microcomputadores PC-AT;
3. o programa deverá ser desenvolvido para ambiente MS-Windows;
4. permitir a criação de entidades representando cada um dos elementos da célula de manufatura;
5. permitir a atribuição de características específicas a cada elemento do RMS;
6. possibilitar a implementação de algoritmos de simulação no programa, de forma direta ou indireta.

1.2.1 SOFTWARE

Para executar a simulação da célula de manufatura randômica, será analisada a possibilidade de se utilizar uma linguagem de programação ou um programa específico de simulação.

As linguagens de programação analisadas foram escolhidas tendo como base a plataforma de desenvolvimento e a facilidade de aprendizado e utilização. As linguagens a serem analisadas são:

1. Basic;

2.Pascal;

3.C/C++.

Foram escolhidas estas linguagens devido à existência de softwares de desenvolvimento para estas linguagens que atendem às condições determinadas:

1.Basic Visual Basic, da Microsoft Corporation;

2.Pascal Delphi, da Borland International;

3.C/C++ C++ Builder, da Borland International.

Existem vários outros softwares para as linguagens escolhidas, mas estes softwares foram escolhidos especificamente por apresentarem um sistema de desenvolvimento baseado em ferramentas RAD (Rapid Application Development - Desenvolvimento Rápido de Aplicativos), que torna o aprendizado e o desenvolvimento de programas muito mais fácil.

Entre os programas de simulação disponíveis serão analisadas os seguintes:

1.Arena, da Systems Modeling Corporation;

2.AutoMod, da Forward Vision;

3.Promodel, da Promodel Corporation.

1.2.1.1 VISUAL BASIC

O Visual Basic é o pioneiro em ferramentas RAD. Utiliza a linguagem Basic para programação. Tem os seguintes prós e contras:

- o professor orientador possui o Visual Basic 4.0 Professional, o que torna o custo de aquisição do programa nulo;
- o professor orientador possui experiência com o software e pode auxiliar na solução de dúvidas referentes ao software;
- por ser um dos primeiros softwares com ferramenta RAD, possui uma vasta bibliografia e biblioteca de funções disponíveis;
- permite o acesso a bancos de dados;
* a versão 4.0 não permite a criação de programas executáveis com código nativo, necessitando a inclusão de várias bibliotecas auxiliares para seu funcionamento (a versão 5.0 lançada recentemente soluciona este problema);
* a versão 4.0 não permite a criação de novas bibliotecas auxiliares de funções (arquivos .DLL), problema solucinado na versão 5.0;

1.2.1.2 DELPHI

O Delphi é um software desenvolvido pela Borland que utiliza a linguagem Pascal para programação. Tem os seguintes prós e contras:

- o aluno Humberto possui a versão educacional do Delphi 2.0 Developer, o que torna o custo de aquisição do programa nulo;

- a aluno Humberto possui um conhecimento básico da linguagem Pascal;
- o software permite a criação de programas executáveis com código nativo, em 16 e 32 bits;
- o software permite a criação de bibliotecas auxiliares de funções (arquivos .DLL);
- permite o acesso a banco de dados;
* não possui uma vasta bibliografia como o Visual Basic;
* a falta de experiência com o software pode dificultar no desenvolvimento do programa de simulação.

1.2.1.3 C++ BUILDER

O C++ Builder é um software desenvolvido pela Borland que utiliza a linguagem C++ para programação. Tem os seguintes prós e contras:

- o aluno Humberto possui conhecimento básico de linguagem C, e o aluno Marcos possui conhecimento básico de C/C++;
- o professor Marcos Tsuzuki pode auxiliar no suporte, devido à sua experiência em programação C++;
- o software permite a criação de programas executáveis com código nativo, em 16 e 32 bits;
- o software permite a criação de bibliotecas auxiliares de funções (arquivos .DLL);

- permite o acesso a banco de dados;
- a linguagem C++ é amplamente utilizada no desenvolvimento dos mais diversos softwares, existindo uma vasta bibliografia e biblioteca de funções disponíveis;
- há a existência de uma versão de testes, que pode disponibiliza todas as ferramentas do C++ Builder por um período de 60 dias, ou até 31/12/97;

* o C++ Builder é um software novo, que ainda está para ser lançado oficialmente no mercado;
* não possui uma bibliografia apropriada;
* como a versão comercial ainda não foi lançada, haveria um custo para a disponibilização completa do software.

1.2.1.4 ARENA

Arena é um software desenvolvido pela Systems Modeling Corporation (http://www.sm.com/), empresa americana que atua no ramo de simulação computacional. Aqui no Brasil, o Arena é distribuído pela Paragon Consultoria e Tecnologia (http://www.paragon.com.br/)

O Arena é um software de simulação que permite criar modelos animados do sistema representado. Permite a criação de modelos por orientação de objetos.

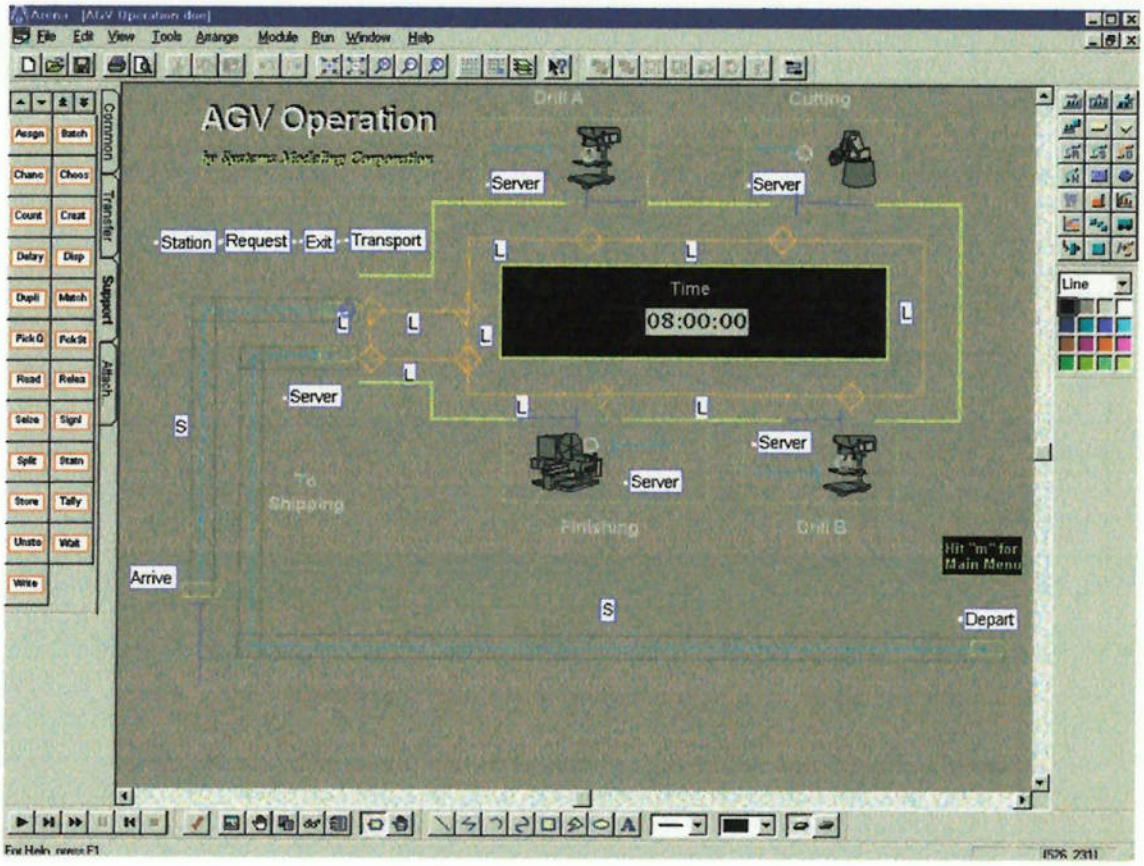
A versão Arena 2.0, de dezembro de 1995 é compatível com o sistema operacional Windows 95.

Sua arquitetura interna é formada pela linguagem de simulação SIMAN, e pelo sistema de animação CINEMA. O SIMAN, desenvolvido em 1982, é uma linguagem desenvolvida exclusivamente para a modelagem de sistemas destinados à simulação. O CINEMA, desenvolvido em 1984, é o sistema de animação utilizado pelo Arena. Quando se desenvolve algum modelo em Arena, é criado automaticamente o código do modelo em SIMAN e o layout de animação do sistema em formato CINEMA.

O Arena pode trocar informações com diversos tipos de arquivos, como planilhas e bancos de dados. Aceita também arquivos gerados em sistemas CAD com o formato DXF.

Este software foi cedido pelo Departamento de Automação e Sistemas para uma avaliação.

Abaixo pode-se ver a tela do programa com o modelo de simulação de uma célula de produção:



1.2.1.5 AUTOMOD

O Automod é um software desenvolvido pela Forward Vision (http://www.forwardvision.com/] para a criação de modelos de simulação em 3D. Foi desenvolvido para ser aplicado em modelos de manufatura e de sistemas de transporte de materiais.

Não foi possível adquirir nenhuma versão demonstrativa deste software, o que impossibilitou sua avaliação.

1.2.1.6 PROMODEL

O Promodel é um software desenvolvido pela Promodel Corporation (http://www.promodel.com/), empresa americana

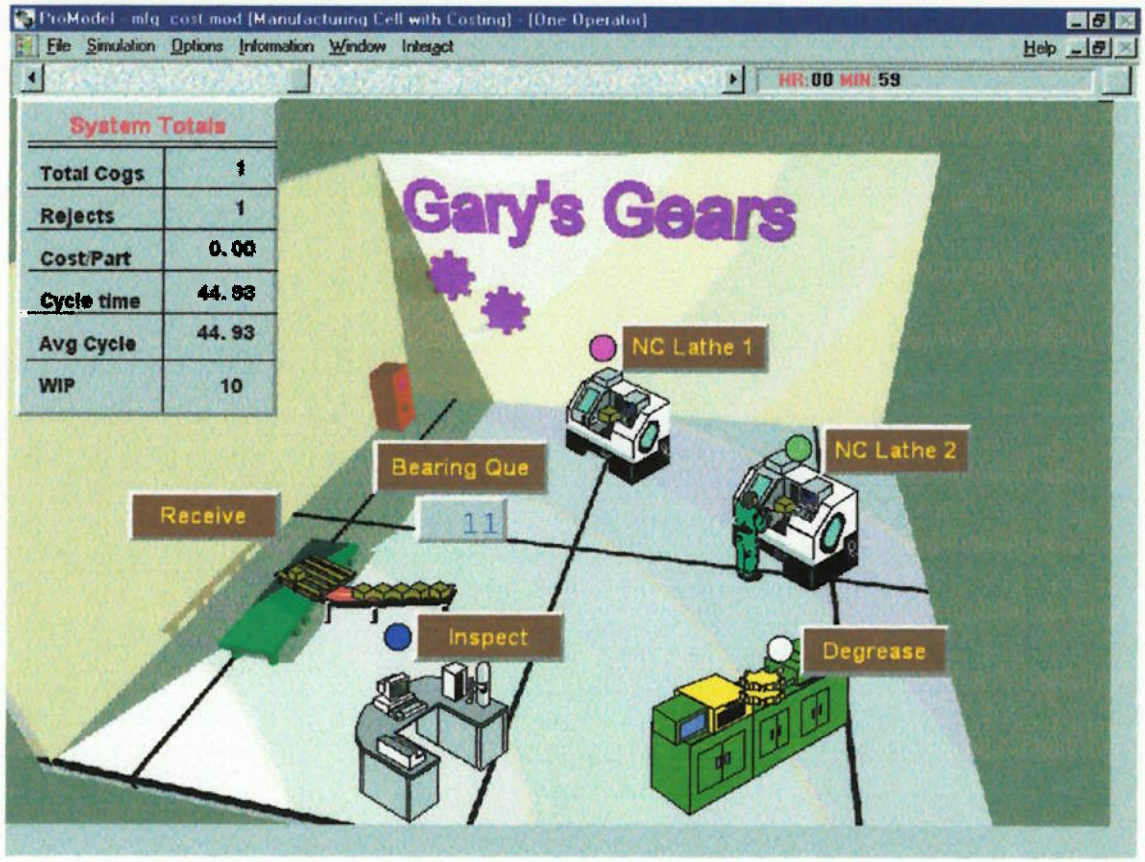
do ramo de simulação computacional. No Brasil, este programa é distribuído pela Belge Engenharia e Sistemas.

O Promodel 3.0 é compatível com os sistemas MS-Windows 3.1, 95 e NT. A Promodel Corporation disponibiliza uma versão demo deste software em seu servidor de FTP no seguinte endereço para login anônimo:

ftp://ftp.promodel.com/pub/pmwin/pminstall.exe

Este software foi desenvolvido especialmente para modelagem de sistemas de manufatura. Possui estruturas pré-programadas e sub-modelos que podem ser utilizados na modelagem de novos sistemas. Possui um sistema de *help* e treinamento *online*. Permite a criação de pacotes de simulação que não necessitam do Promodel para serem visualizados.

Abaixo pode-se ver a tela do programa com o modelo de simulação de uma pequena fábrica de engrenagens:



1.2.2 MATRIZ DE DECISÃO

Para a matriz de decisão não será considerado o software Automod, por não ter sido avaliado. Os critérios de decisão serão os seguintes:

Critério	Descrição	Peso
A	Conhecimento da Linguagem	5
B	Conhecimento do Software	3
C	Suporte de terceiros	4
D	Custo	2
E	Bibliografia da Linguagem	4
F	Bibliografia do Software	3
G	Biblioteca de Funções	4
H	Facilidade de Aprendizagem	4

Deve se observar que alguns critérios só se aplicam às linguagens de programação. Portanto, para determinar qual o

melhor programa a ser utilizado deve-se verificar a média ponderada.

Deste modo, a matriz de decisões fica da seguinte forma:

Critério	Visual Basic		Delphi		C++ Builder		Arena		Promodel	
	Pto	Peso	Pto	Peso	Pto	Peso	Pto	Peso	Pto	Peso
A	1	5	2	10	4	20	NA	NA	NA	NA
B	2	6	2	6	2	6	2	6	1	3
C	4	16	1	4	4	16	1	4	1	4
D	5	10	5	10	4	8	4	8	2	4
E	3	12	3	12	5	20	NA	NA	NA	NA
F	5	15	3	9	2	6	2	6	2	6
G	3	12	3	12	5	20	NA	NA	NA	NA
H	4	16	4	16	4	16	1	4	2	8
Total		92		79		112		28		25
Média		3.17		2.72		3.86		1.75		1.56

Pode-se verificar que o melhor programa a ser utilizado é o C++ Builder, da Borland International.

1.3 REQUISITOS BÁSICOS PARA O SISTEMA DE SIMULAÇÃO

Para que seja possível a simulação de uma célula de manufatura, é necessário que se estabelessa previamente quais são os elementos que constituem a mesma. No caso deste projeto, vai-se trabalhar com uma célula específica que é a célula que está sendo implementada no laboratório de CAD/CAM/CNC. Tendo em vista isso, concluiu-se que são necessários os seguintes elementos:

- Máquinas;
- Transportadores;
- Robôs;
- Sensores;
- Peças.

A partir destes elementos é possível criar um sistema que represente a célula de manufatura em questão e que estes possam interagir entre si de forma a gerar a simulação propriamente dita. A seguir serão apresentadas as características básicas que cada elemento deve possuir para que seja possível simular a célula em operação e gerar parâmetros (Ex.: Tempos, Custos, etc.) para que se possa analisar oos algoritmos de controle.

1.3.1 MÁQUINAS

O elemento máquina é o responsável por representar as operações de processo. Esta representação é feita através de métodos estatísticos, atribuindo a cada máquina um tempo médio de processo T_m com um desvio padrão σ_{Tm}. Os valores para estes parâmetros são baseados em tempos reais tomados do processo real.

Como características básicas necessárias para que o elemento represente uma máquina pode-se considerar as seguintes:

1. Tempo de Operação: Permite avaliar o tempo de operação para otimização do processo;
2. Capacidade de Peças: Há máquinas que tabalham com várias peças simultaneamente;
3. Status de Operação (Livre, Operando, Parada, Quebrada, Troca de Ferramenta): Estes são eventos comuns a uma máquina e que devem ser monitorados pelo computador gerente da célula para que ele possa tomar decisões quando da quebra de uma máquina;
4. Status de Ferramenta e Status de Dispositivo de Fixação: Estes status estão relacionados com os dispositivos presentes na máquina, como, por exemplo, dispositivo de fixação, sistema hidráulico, ferramenta, etc. A quebra de um destes componentes implica na parada da máquina e isto

deve ser informado ao computador gerente imediatamente.

Estas 4 características básicas permitem uma representação rasoável de uma máquina para um sistema de simulação

1.3.2 TRANSPORTADORES

No caso da célula em questão, o elemento transportador está diretamente relacionado com a esteira que interliga as máquinas da célula. São consideradas características importantes as listadas abaixo:

1. Velocidade: Permite analisar qual a melhor velocidade, dentre as possíveis, para operação do sistema de transporte. Também permite avaliar o tempo que a peça permanece no transportador;
2. Capacidade de Peças: Este parâmetro é necessário para saber quantas peças estão sendo transportadas e se é possível colocar outras no transportador;
3. Status de Operação: Como em uma máquina, deve-se saber quais as condições de operação do transportador(Quebrado, Livre, Com carga total, Manutenção, etc.);
4. Sensores Associados: Durante todo o tempo de operação é necessário saber em ponto do transportador está determinada peça. Para isso,

deve haver sensores capazes de identificar as peças e suas posições.

1.3.3 ROBÔ

O elemento robô foi não foi considerado como um transportador porque na célula que está sendo criada, sua função principal será a colocação e posicionamento de peças nas máquinas. Foram consideradas como características básicas as seguintes:

1. Capacidade de Transporte: Número de peças que podem ser transportadas por vez;
2. Posição: Localização espacial do mesmo na célula para permitir que o gerente de célula possa decidir qual o melhor caminho para se executar uma determinada tarefa;
3. Status de Operação: Quais as condições de trabalho (Carregado, Livre, Manutenção, Quebrado, etc.);
4. Velocidade: Permite otimizar o deslocamento de acordo com a posição e determinar tempo de trajeto;
5. Tempo de Operação: Verificar qual o tempo ocioso do robô, para verificar possíveis utiliações em outras máquinas.

1.3.4 SENSORES

Estes são os responsáveis por determinar os pontos do tranportador correspondentes a cada máquina e identificar a peça que está naquele ponto. Possui duas características básicas:

1. Status de Operação: Verifica a condição de operação do sensor;
2. Identificação de Peça: Permite identificar a peça que está naquele ponto naquele momento.

1.3.5 PEÇAS

As peças são os elementos mais importantes para o sistema de simulação, pois guardam as informações das operações e máquinas pelas quais devem passar. Abaixo tem-se as características básicas para se modelar o elemento peça:

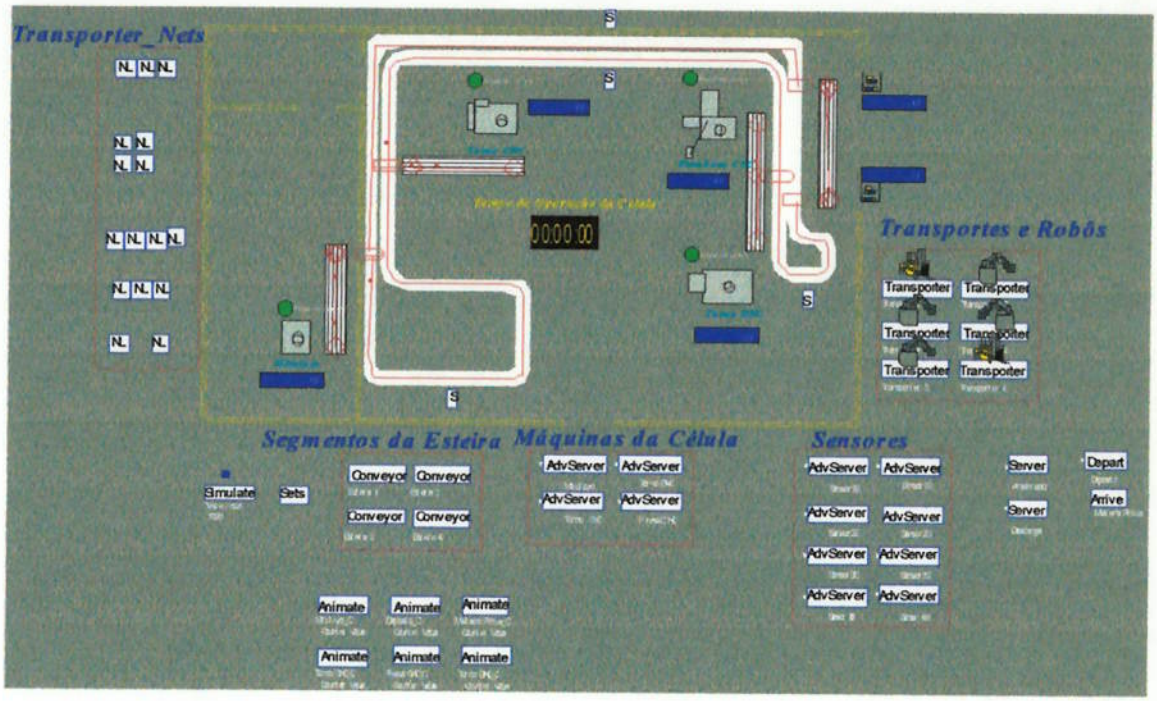
1. Identificador de Peça: Como a célula deve ser capaz de trabalhar com várias peças simultaneamente, cada peça deve possuir um identificador;
2. Sequência de Operações: Quando a peça entra na célula ela possui um sequência de operações pré-definidas que devem ser seguidas para que a peça possa ser produzida. Deve notar que neste trabalho não será simulada a classificação das peças, sendo que se está admitindo que as peças entrem na célula com as sequências de operações já definidas;

3. Máquinas Utilizadas: Com base na sequência de operações da peça e nas máquinas disponíveis, o computador gerente de célula deve decidir qual a melhor sequência de máquinas para a peça;
4. Tempo de Operação: Com base no tempo de operação e no tempo de transporte é possível analisar a eficiência do processo e conseqüentemente o custo da peça.

Estes são os requisitos básicos para cada elemento que deve compor o sistema de simulação para que este permita que a avaliação desejada seja feita. Evidentemente, à medida que o simulador for sendo implementado poderão surgir novas características que não são consideradas aqui, pois estas são o ponto de partida para o projeto.

1.4 MODELO BÁSICO IMPLEMENTADO COM O SOFTWARE ARENA V. 2.0

A partir das especificações acima construiu-se um modelo básico da célula com o programa ARENA V. 2.0, que pode ser observado na figura abaixo:



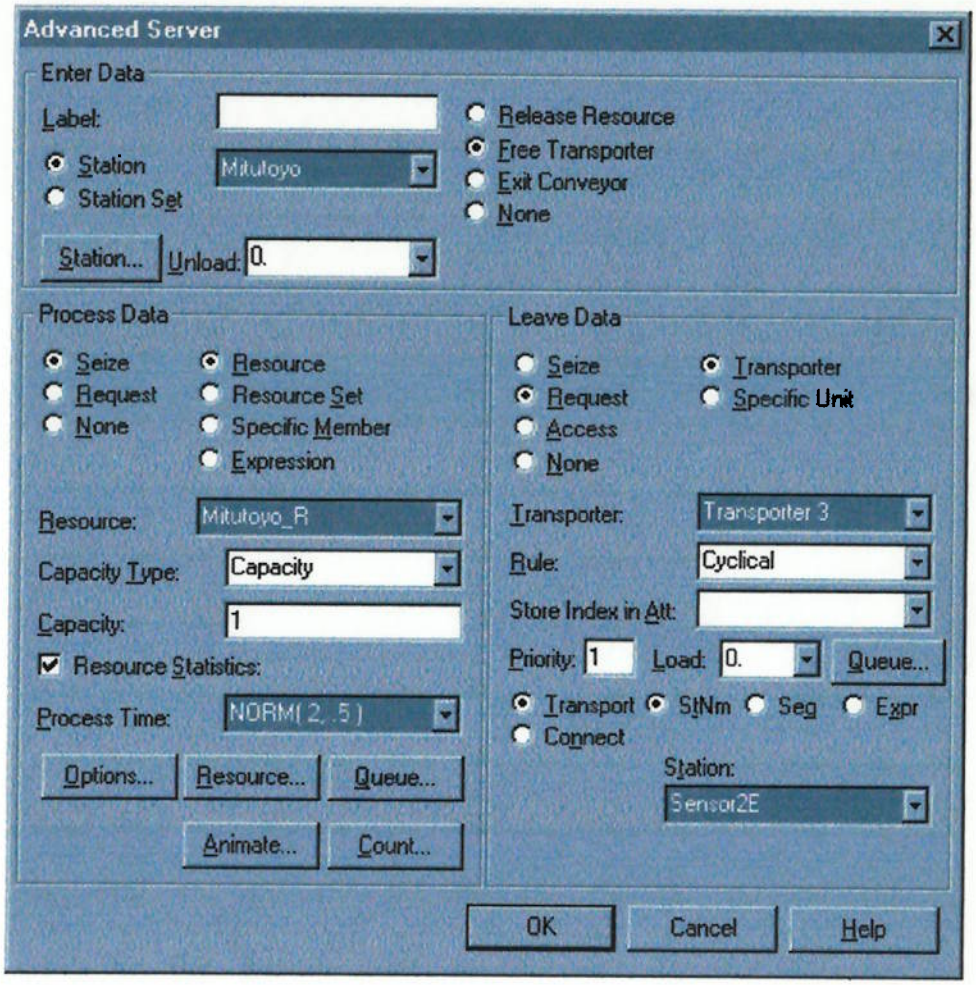
Neste modelo pode-se notar todos os elementos acima descritos: Máquinas, Robôs, Transportadores (esteira), Sensores e as peças.

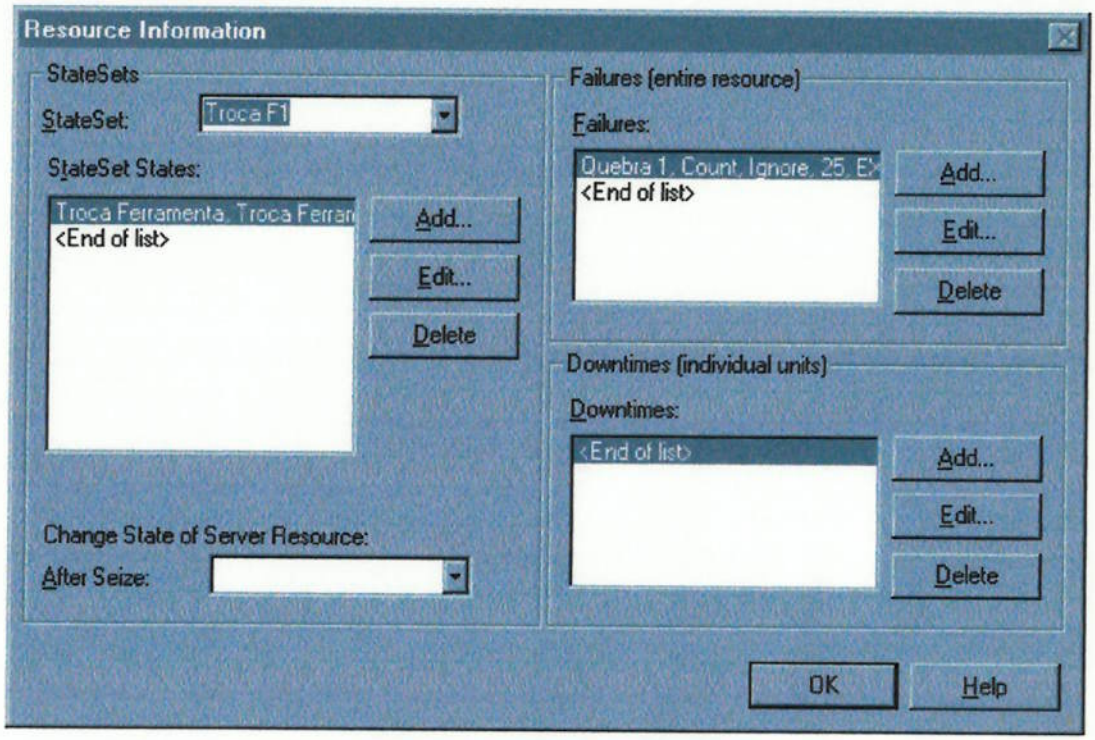
Abaixo, tem-se uma breve descrição de como estes elementos foram definidos no software:

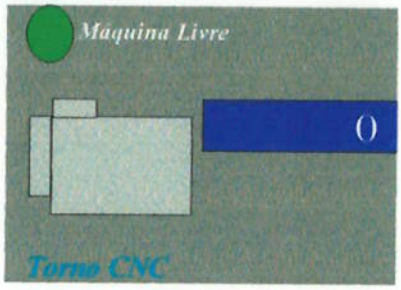
- Máquinas e Sensores

Uma máquina ou um Sensor é definido como um Advanced Server como pode ser visto na figura abaixo. Nesta janela definem-se o tempo de processo, capacidade da máquina,

transportes que acessam a máquina e recursos associados à máquina ou sensor. A figura seguinte mostra a tela para associar eventos à máquina como quebra, troca de ferramenta, etc. A última figura mostra a representação de uma máquina na tela durante a simulação, com o tipo de máquina, status de operação e número de peças trabalhadas.

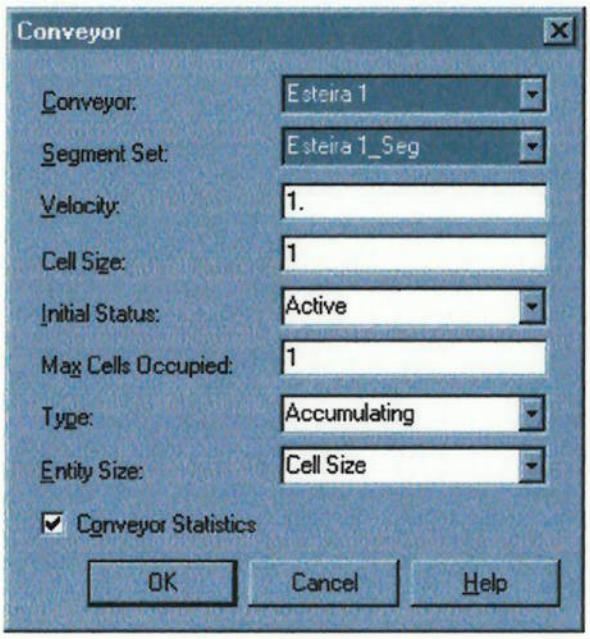






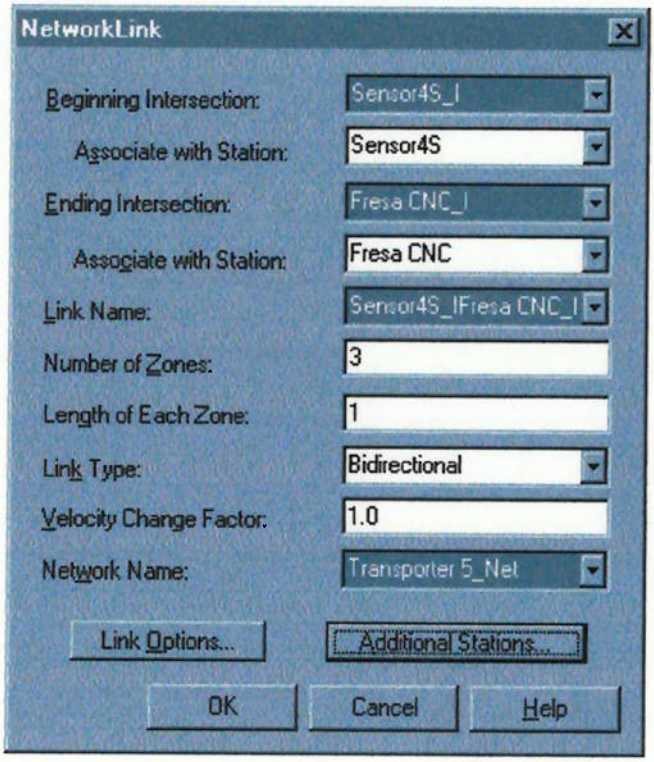
- Transportador

O transportador é definido através da janela mostrada abaixo e a ele são associados velocidade, variáveis estatísticas e os segmentos que definem o caminho das peças.



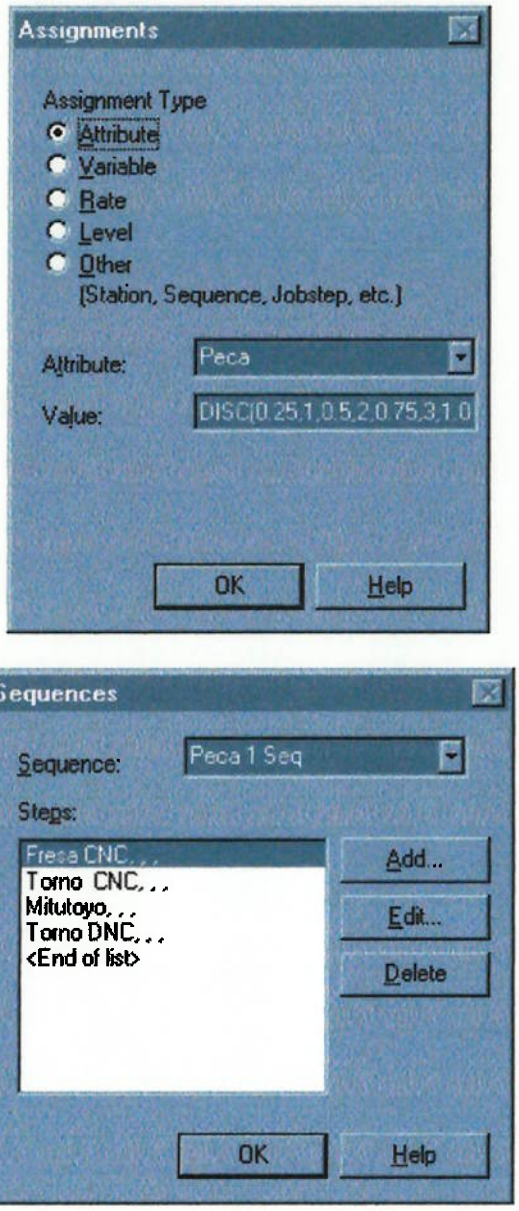
- Robôs

Um robô é definido como um transportador simples e não se pode atribuir um status de operação ao mesmo. Pode-se atribuir a ele uma velocidade de operação e deve-se especificar os caminhos entre máquinas com links como mostrado na figura a seguir.



- Peças

As peças são as entidades simuladas em processo e são definidas na janela mostrada abaixo, utilizando-se um número inteiro como identificador e as seqüências de máquinas são definidas a parte, como mostrada na última figura, para cada peça.



Embora seja um software desenvolvido especialmente para simulação, o Arena não atende às nossas necessidades, pois apresenta ligações muito rigidas entre elementos e que se tornam extremamente complexas quando se trabalha com váiros tipos de transportes interligados entre si, e não permite uma análise do transiente entre eventos, o que dificulta a simulação de células randômicas de manufatura.

Outro ponto contra é o material disponível para aprendizado do software, muito escasso e não muito amigável.

2. PARTE 2

Programa Final de Simulação

2.1 DIVISÃO DE ATIVIDADES

As atividades de desenvolvimento do programa de simulação foram divididas entre os alunos Humberto e Marcos, de acordo com o esquema a seguir:

2.1.1 ATIVIDADES EXECUTADAS EM CONJUNTO

As atividades desenvolvida em conjunto pelos alunos está listada a seguir:

- definição da estrutura interna do programa;
- definição da estrutura interna dos elementos de simulação;
- definição e implementação da estrutura de animação do programa.

2.1.2 ATIVIDADES EXECUTADAS POR HUMBERTO Y. MIYOSHI

Ficou responsável pela implementação e elaboração das rotinas de controle dos seguintes elementos:

- pallets;
- máquinas;
- esteira.

Foi responsável pela elaboração da estrutura interna do programa relacionados com:

- relógio interno do programa;
- sistema de unidades internas para simulação.

2.1.3 ATIVIDADES EXECUTADAS POR MARCOS A. PERRI

Ficou responsável pela implementação e elaboração das rotinas de controle dos seguintes elementos:

- robos;
- sensores;
- sistemas de entrada e saída de peças no processo.

Foi responsável pela elaboração da estrutura interna do programa relacionados com:

- sistema de coleta de dados do resultado da simulação;
- interface do programa de simulação.

2.2 SISTEMA DE ANIMAÇÃO

Existem dois métodos muito utilizados em animação computacional. Um deles é baseado em animação realizado por seqüência de quadros. Nele, cada instante da animação é composta por um quadro de imagem fixa. É a sucessão de quadros que dá a impressão de animação. O outro método é baseado em animação de elementos. Nele, apenas o objeto que é animado será movimentado sobre uma imagem de fundo estática.

Ambos os métodos de animação utilizam o mesmo princípio: a imagem é preparada na memória e depois é atualizada na tela. Este processo acelera a animação porque

as operações realizadas na memória do sistema são executadas muito mais rápidas do que em memórias de vídeo.

No programa de simulação da célula de manufatura será utilizado o segundo método, o de animação de elementos.

Para realizar a animação dos elementos é utilizado um sistema de máscaras para cada imagem. Nele é utilizado um sistema de operações booleanas com pixels. Na imagem do elemento, a região que não correspode ao objeto representado deve estar preenchida com a cor preta. Na máscara, esta mesma região deve estar preenchida com a cor branca, e a região que representa o objeto, com a cor preta:

Imagem do Objeto

Imagem da máscara

A lógica booleana utilizada com cores é a seguinte:

COR	OPERAÇÃO	COR	RESULTADO
PRETO	AND	QUALQUER	PRETO
BRANCO	AND	QUALQUER	QUALQUER
PRETO	OR	QUALQUER	QUALQUER
BRANCO	OR	QUALQUER	BRANCO

Desta forma, a seqüência de operações realizadas para colocar o objeto animado sobre a imagem de fundo é a seguinte:

((IMAGEM DE FUNDO) **AND** (MÁSCARA)) **OR** (IMAGEM DO OBJETO)

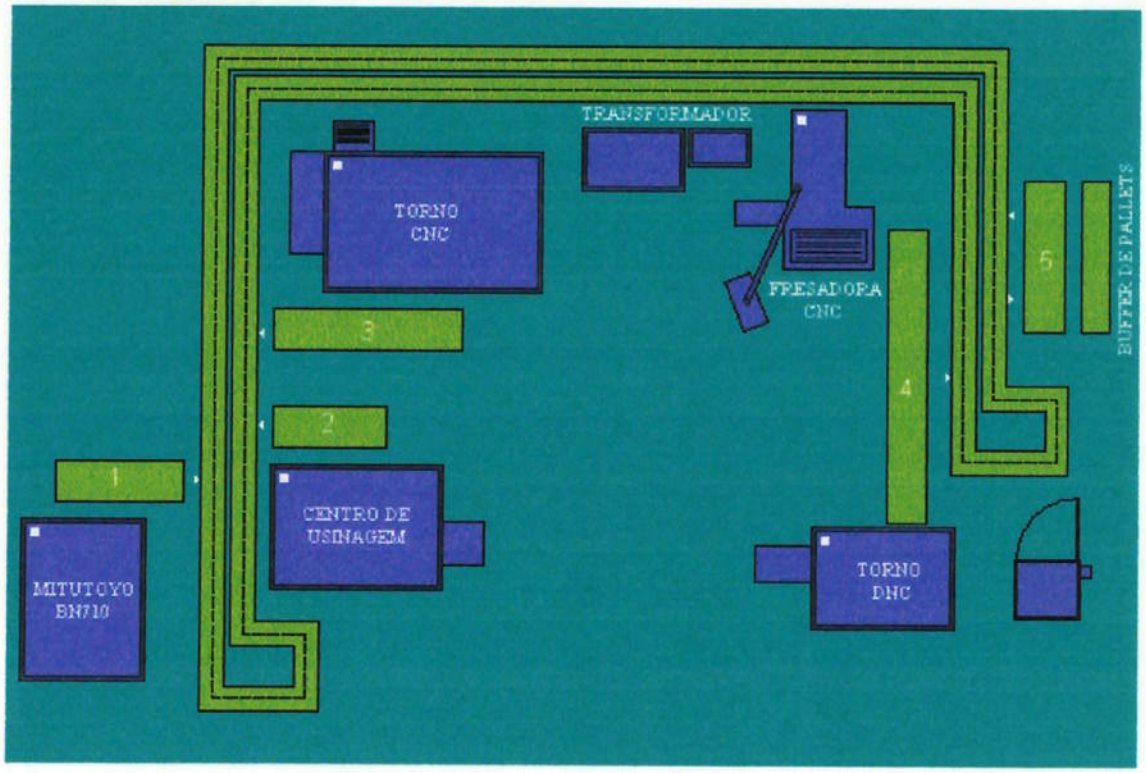
Desta forma, apenas a imagem do objeto representado é colocado sobre a imagem de fundo.

Para o efeito de animação, é importante que a imagem da região onde o objeto animado será colocado seja guardado para posteriormente ser reincorporado à imagem de fundo. Desta forma, a imagem de fundo é restaurada e a animação seguinte poderá ser processada sem a perda de partes da imagem de fundo. A seqüência de comandos será a seguinte:

- colocar a imagem do elemento sobra a imagem de fundo;
- atualizar a imagem da memória para a tela;
- retirar a imagem do elemento da imagem de fundo.

Deve-se tomar cuidado quando a animação for composta de vários elementos animados. Neste caso, a colocação e retirada das imagens dos elementos deve obedecer a lógica FILO (First In - Last Out), para evitar a perda de dados da imagem de fundo.

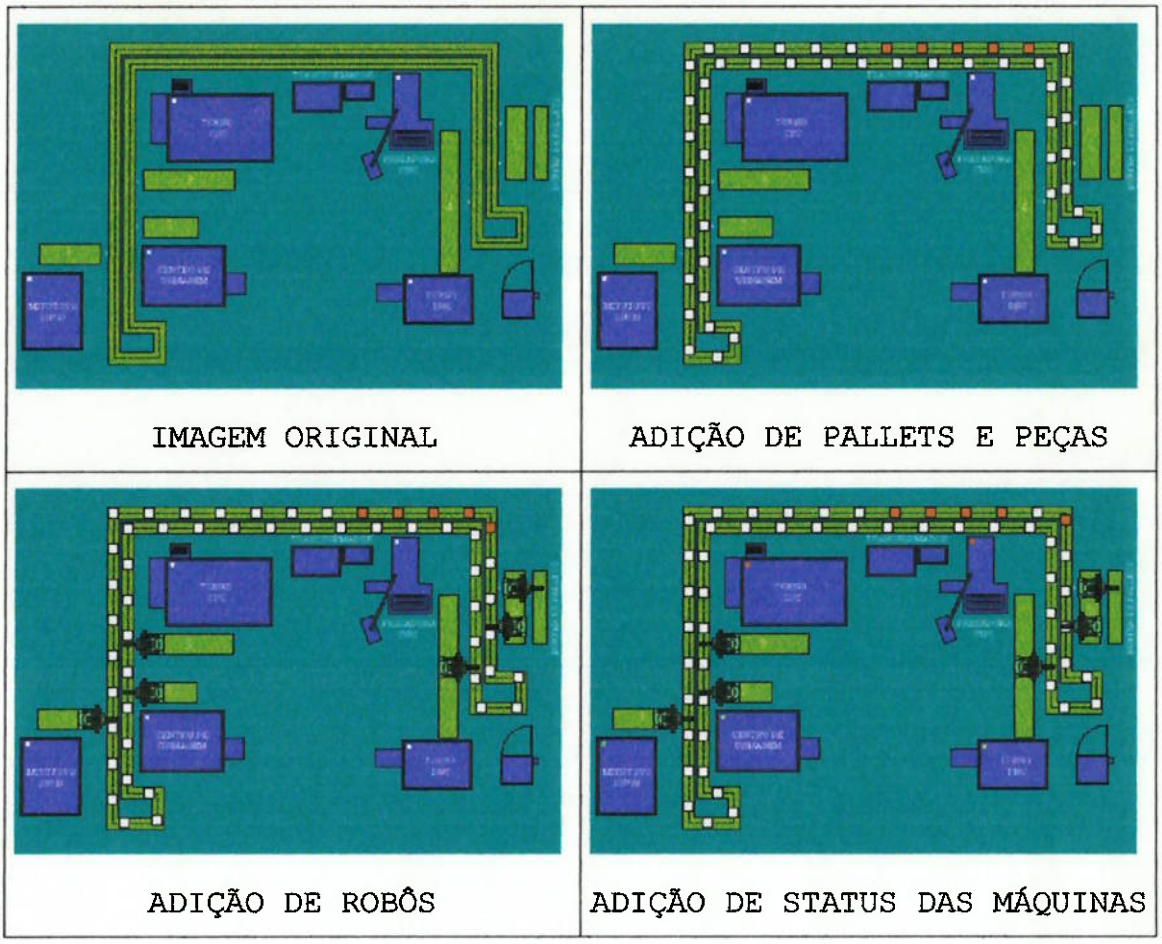
Como imagem de fundo, teremos a planta da célula de manufatura, com imagem da esteira e das máquinas:



Os elementos animados são os seguintes:

- pallets: □
- peças: ■
- robôs sem peça: 
- robôs com peça: 
- status das máquinas: ■

A seqüência utilizada para criar a animação pode ser vista a seguir. Está apresentada a seqüência de colocação dos elementos animados. A seqüência de retirada destes elementos é o inverso.



2.3 ESTRUTURA DO PROGRAMA PARA ANIMAÇÃO

Uma das primeiras versões do programa com estrutura de animação de pallets, robôs e máquinas ficou pronta no início de setembro. Nesta versão, a animação e a simulação eram realizadas em um mesmo bloco do programa.

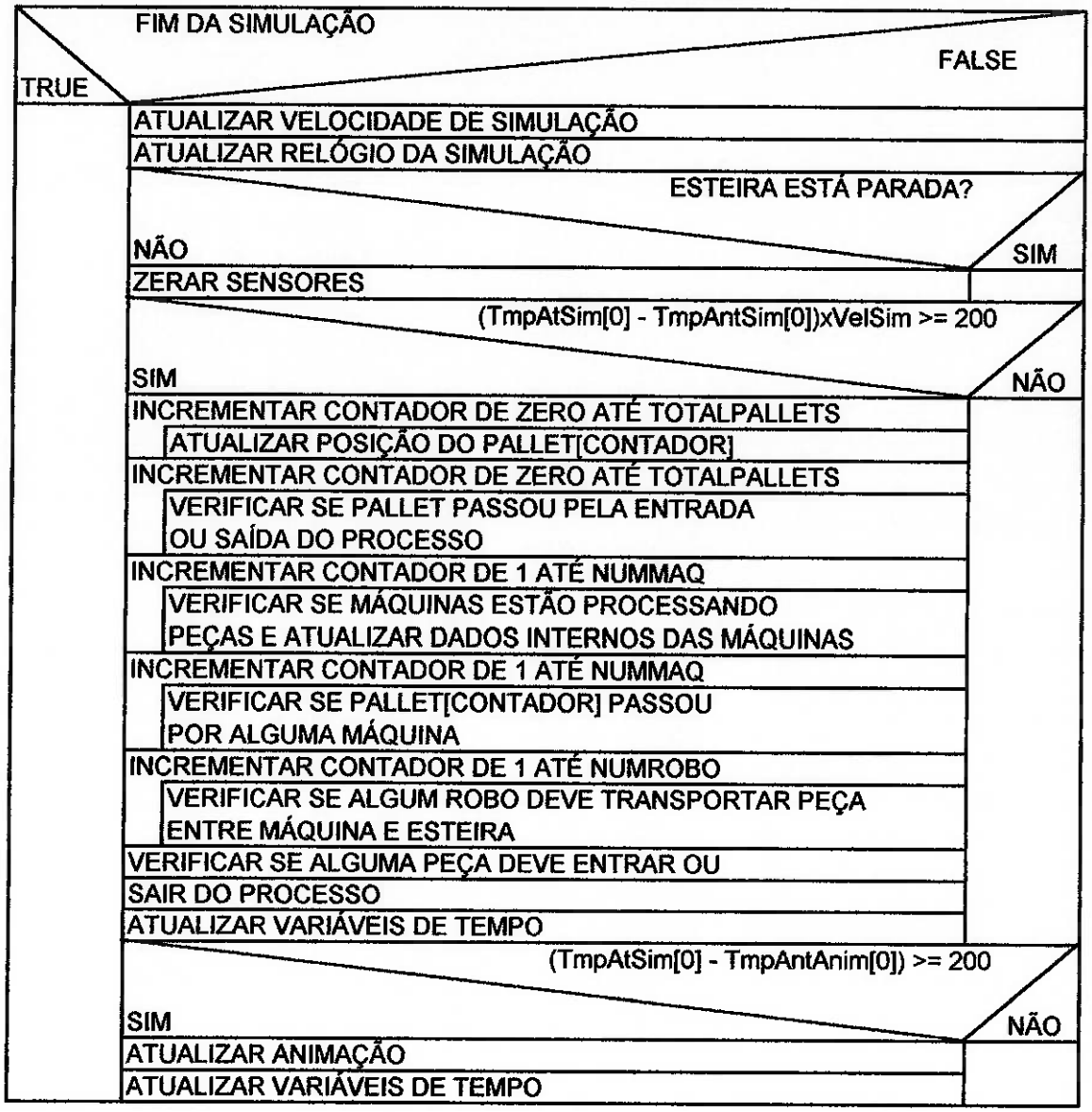
Esta estrutura mostrou-se boa para a animação em tempo real. Mas ao realizar a aceleração da animação, verificou-se que ocorria um retardamento na simulação. Isto ocorria porque o aumento da velocidade de simulação acarretava

também no aumento do número de atualizações da animação. Isto pode ser verificado na estrutura indicada a seguir:

ATUALIZAR VELOCIDADE DE SIMULAÇÃO
VERIFICAR POSICIONAMENTO DO ROBO
ATUALIZAR POSIÇÃO DOS PALLETS
ATUALIZAR POSIÇÃO DOS ROBOS
ATUALIZAR DESENHO DOS PALLETS NA IMAGEM QUE ESTÁ NA MEMÓRIA
ATUALIZAR DESENHO DOS ROBOS NA IMAGEM QUE ESTÁ NA MEMÓRIA
COLOCAR IMAGEM DA MEMÓRIA NO MONITOR
REMOVER DESENHO DOS ROBOS DA IMAGEM QUE ESTÁ NA MEMÓRIA
REMOVER DESENHO DOS PALLETS DA IMAGEM QUE ESTÁ NA MEMÓRIA

O retardamento da simulação ocorria devido ao alto índice de processamento exigido pela animação. Comparando o processamento necessário para a simulação e para a animação, verifica-se que a animação utiliza um grau de processamento muito mais elevado. Isto foi observado em um computador com processador Intel MMX 166 MHz, com 32 Mb de memória RAM EDO e 2 Mb de memória de vídeo. Em sistemas menos potentes, provavelmente este problema seria muito mais crítico.

Para contornar este problema, optou-se pela modificação da estrutura interna do programa, de modo a tornar independentes as rotinas de simulação e de animação. A estrutura adotada pode ser verificada no diagrama a seguir:



Nesta estrutura pode-se observar a existência de dois blocos distintos.

No primeiro bloco são realizados todas as operações referentes à simulação, tais como: atualizar posição de pallets e robôs, verificar máquinas e sensores, etc. Este bloco é executado normalmente em intervalos de 200 ms(mili-segundos). Ao aumentar a velocidade de simulação, o intervalo de execução diminui e a simulação é feita com maior velocidade.

No segundo bloco é realizado a animação do programa. Este bloco sempre é executado em intervalos de 200 ms, ou seja, 5 quadros por segundo. A velocidade de simulação não influi neste bloco.

Esta estrutura solucionou o problema verificado na estrutura anterior. O aumento na velocidade de simulação parou de prejudicar a animação.

2.4 ESTRUTURA INTERNA

A estrutura interna inicial foi implementada utilizando-se o pixel como unidade de comprimento. Esta estrutura apresentou problemas para a coleta de informações da simulação. Como o pixel permite trabalhar apenas com números inteiros, ocorria uma perda de precisão.

A estrutura do programa foi modificada de modo que se possa trabalhar internamente em metros e pixels. Desta forma, pode-se optar pela melhor unidade em cada caso. Na análise de posição dos pallets na esteira, é utilizado o posicionamento em metros, e a velocidade da esteira em metros por segundo. No momento de realizar a animação, o posicionamento é convertido em pixels.

Com esta modificação, a coleta de informações e o desenvolvimento de algoritmos de controle da célula se tornaram mais precisas.

2.5 Estrutura do Programa e Funções Criadas

A seguir são apresentados os arquivos criados, bem como todas as funções implementadas. Para cada função é feita uma descrição e, caso seja necessário, é apresentado um diagrama NS para melhor entendimento. A listagem completa do programa se encontra no Anexo 1.

Arquivo RMS.MAK

Este arquivo possui as informações que o programa Borland C++Builder necessita para compilar e linkar o programa de simulação.

Arquivo RMS.CPP

Este arquivo possui a declaração das units (bibliotecas) utilizadas pelo simulador:

- animacao
- auxiliar
- botao
- controle
- editor
- grafico
- inicio
- mainform
- robos
- sobre
- start

Arquivo ANIMACAO.CPP

Este arquivo contêm as rotinas para animação da simulação. A primeira rotina gerencia as outras relacionadas com a animação.

```
void TForm1::AtualizaAnimacao(void)
```

Esta rotina gerencia todas as outras rotinas relacionada com a animação da simulação.

ATUALIZAR DESENHO DOS PALLETS NA IMAGEM QUE ESTÁ NA MEMÓRIA
ATUALIZAR DESENHO DOS ROBOS NA IMAGEM QUE ESTÁ NA MEMÓRIA
ATUALIZAR DESENHO DOS STATUS DAS MÁQUINAS NA IMAGEM
QUE ESTÁ NA MEMÓRIA
COLOCAR IMAGEM DA MEMÓRIA NO MONITOR
REMOVER DESENHO DOS STATUS DA IMAGEM QUE ESTÁ NA MEMÓRIA
REMOVER DESENHO DOS ROBOS DA IMAGEM QUE ESTÁ NA MEMÓRIA
REMOVER DESENHO DOS PALLETS DA IMAGEM QUE ESTÁ NA MEMÓRIA

```
void TForm1::PatchBackground(void)
```

Esta rotina retira a imagem dos pallets do cenário da simulação.

```
void TForm1::DrawPalletOnBackground(void)
```

Esta rotina analisa qual o tipo de peça em cada pallet e depois coloca o desenho dos pallets e das peças no cenário da simulação.

```
void TForm1::MoveBitmapToScreen(void)
```

Esta rotina atualiza a imagem da simulação que está no monitor.

```
void TForm1::DrawStatusMaquina(void)
```

Esta rotina faz a verificação do status de cada máquina, atualiza a imagem do status e depois atualiza-o no cenário da simulação.

Arquivo ARQUIVO.CPP

Este arquivo possui funções auxiliares utilizadas para a debugação do programa. No programa final foram todas desativadas.

TForm1::CriarArquivo(void)

Esta rotina cria um arquivo de disco para escrita chamado "Saida1.txt", e associa-o com o ponteiro **fp**.

TForm1::Fechar Arquivo(void)

Esta rotina fecha o arquivo de disco associado ao ponteiro **fp**.

TForm1::GravarPeca1(void)

Esta rotina grava os dados referentes à peça do tipo 1 e índice 1 no arquivo associado ao ponteiro **fp**.

TForm1::GravarPeca(int Tipo, int Indice)

Esta rotina grava os dados referentes à peça do tipo **Tipo** e índice **Indice** no arquivo associado ao ponteiro **fp2**.

TForm1::GravarAll(void)

Esta rotina cria um arquivo de disco para escrita chamado "Saida2.txt", associando-o com o ponteiro **fp2**,

grava as informações de todas as peças processadas e depois fecha este arquivo.

Arquivo AUXILIAR.CPP

Este arquivo contêm funções que pertencem a classes criadas para o programa.

```
void clPallet::MetroParaPixel(void)
```

Esta rotina da classe **clPallet** analisa o valor da variável **Metros** em calcula o equivalente para a variável **Pixels**.

```
bool clPeca::ProcessosExecutados(void)
```

Esta rotina da classe **clPeca** verifica se todos os processos de determinada peça já foram executados. Caso a resposta seja positiva, retorna **true**. Caso contrário, retorna **false**.

```
cenario::cenario()
```

Esta é a rotina construtora da classe **cenario**, responsável pelo armazenamento da imagem da célula de manufatura.

```
cenario::~cenario()
```

Esta é a rotina destruidora da classe **cenario**. Ela apaga os dados da classe **cenario** da memória do computador assim que o programa é finalizado.

```
short int clMaquina::TirardaFilaEntrada(void)
```

Esta rotina da classe **clMaquina** pega uma peça da fila de entrada e coloca-a em processamento, além de atualizar a variável **TmpInc** (tempo de início de processamento) da peça processada. Caso o processo ocorra sem problemas, ela retorna o valor **1**(UM). Caso haja algum problema de fila vazia, ela retorna **0**(ZERO).

```
short int clMaquina::ColocarnaFilaEntrada(void)
```

Esta rotina da classe **clMaquina** pega uma peça do robô de transporte e coloca-a na fila de entrada, além de atualizar a variável **TmpIn** (tempo de entrada na máquina) da peça processada. Caso o processo ocorra sem problemas, ela retorna o valor **1**(UM). Caso haja algum problema de fila vazia, ela retorna **0**(ZERO).

```
short int clMaquina::TirardaFilaSaida(void)
```

Esta rotina da classe **clMaquina** pega uma peça da fila de saída e coloca-a no robô de transporte, além de atualizar a variável **TmpOut** (tempo de saída da máquina) da peça processada. Caso o processo ocorra sem problemas, ela

retorna o valor **1**(UM). Caso haja algum problema de fila vazia, ela retorna **0**(ZERO).

```
short int clMaquina::ColocarnaFilaSaida(void)
```

Esta rotina da classe **clMaquina** retira a peça do processamento da máquina e coloca-a na fila de saída, além de atualizar a variável **TmpFnl** (tempo final de processamento) da peça processada. Caso o processo ocorra sem problemas, ela retorna o valor **1**(UM). Caso haja algum problema de fila vazia, ela retorna **0**(ZERO).

```
short int clMaquina::FilaEntradaCheia(void)
```

Esta é uma rotina da classe **clMaquina** que verifica a fila de entrada da máquina. Caso ela esteja cheia, a rotina retorna o valor **1**(UM). Caso contrário, ela retorna **0**(ZERO).

```
short int clMaquina::FilaEntradaVazia(void)
```

Esta é uma rotina da classe **clMaquina** que verifica a fila de entrada da máquina. Caso ela esteja vazia, a rotina retorna o valor **1**(UM). Caso contrário, ela retorna **0**(ZERO).

```
short int clMaquina::FilaSaidaCheia(void)
```

Esta é uma rotina da classe **clMaquina** que verifica a fila de saida da máquina. Caso ela esteja cheia, a rotina retorna o valor **1**(UM). Caso contrário, ela retorna **0**(ZERO).

```
short int clMaquina::FilaSaidaEntradaVazia(void)
```

Esta é uma rotina da classe **clMaquina** que verifica a fila de saida da máquina. Caso ela esteja vazia, a rotina retorna o valor **1**(UM). Caso contrário, ela retorna **0**(ZERO).

```
void clSensor::Reiniciar(void)
```

Esta é uma rotina da classe **clSensor** que zera os flags de verificação desta classe.

Arquivo BOTAO.CPP

Este arquivo contém as rotinas associadas aos botões do programa de simulação. Somente as funções mais relevantes serão comentadas.

```
float CalcFloat(TEdit *Valor)
```

Esta rotina analisa a caixa de edição ***Valor** e retorna o valor nele contido, convertendo-o para o tipo ponto flutuante.

```
float CalcInt(TEdit *Valor)
```

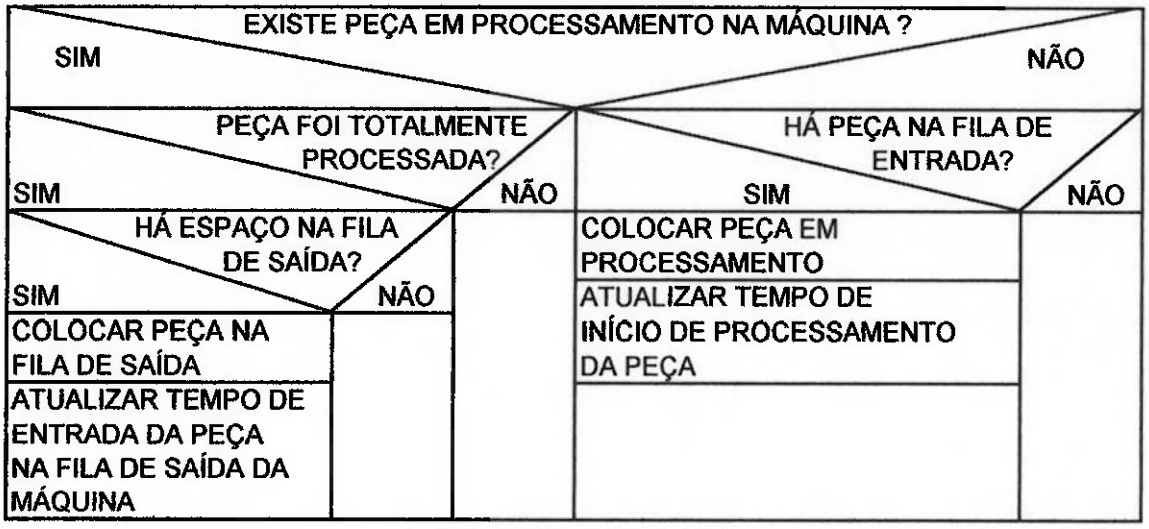
Esta rotina analisa a caixa de edição ***Valor** e retorna o valor nele contido, convertendo-o para o tipo inteiro.

Arquivo CONTROLE.CPP

Este arquivo contém as rotinas de controle dos elementos da célula de manufatura.

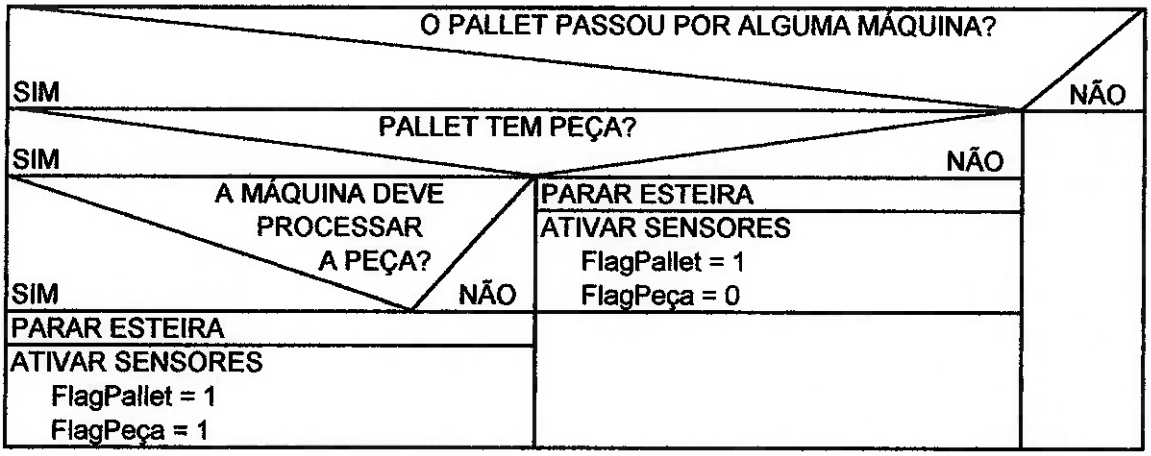
```
void TForm1::VerificaMaquina(int Indice)
```

Esta rotina verifica o funcionamento da máquina indicada pela variável Indice. Ela verifica a peça que está em processamento na máquina e também as filas de entrada e saída para verificar qual peça deve ser processada.



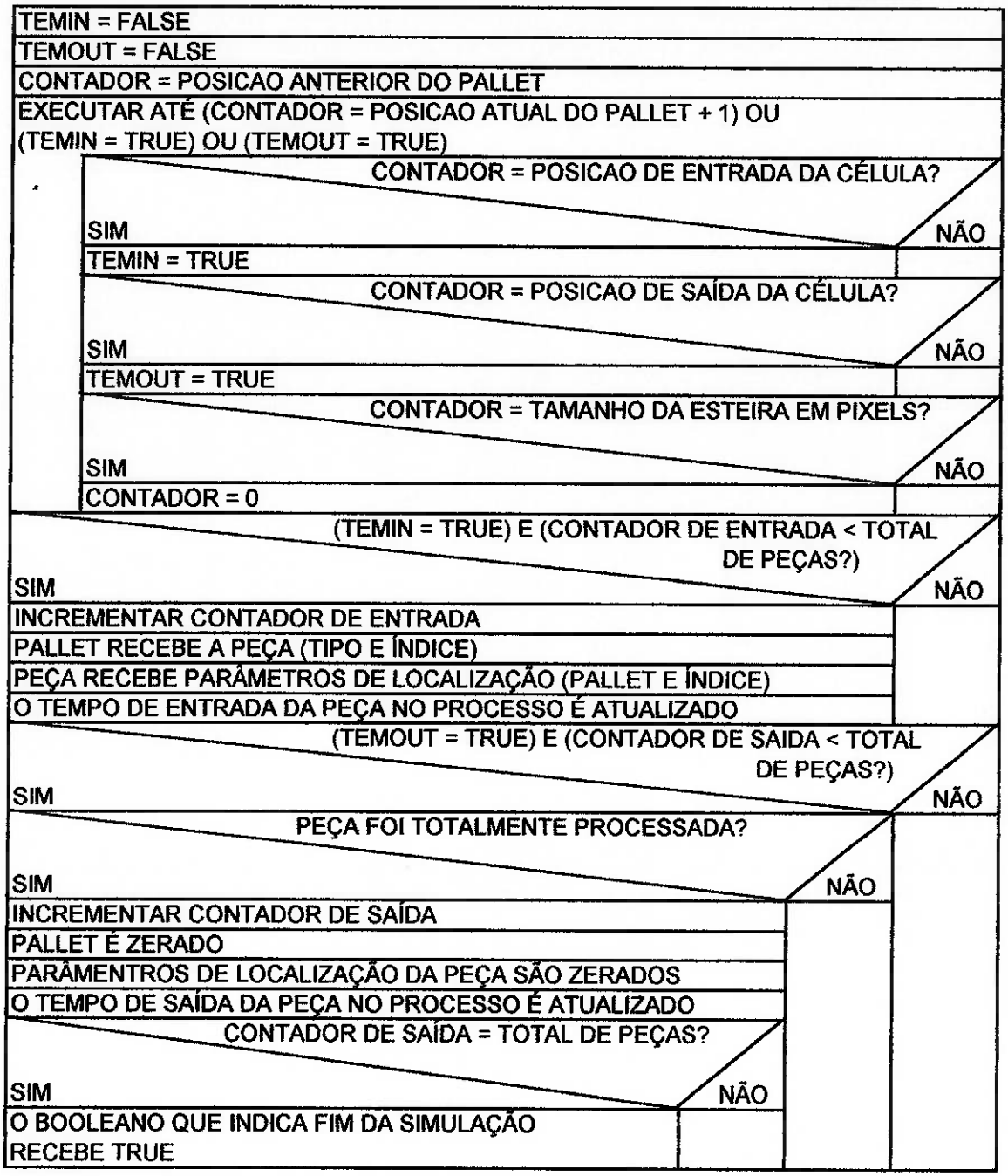

```
void TForm1::VerificaPallet(int Indice)
```

Esta rotina verifica o pallet indicado pela variável Indice. É verificado se o pallet analisado passou por alguma máquina, em determinado ciclo da simulação. Caso a resposta seja positiva, o sensor da respectiva máquina é acionado para uma verificação posterior.



```
void TForm1::VerificaInOut(int Indice)
```

Esta rotina verifica o pallet indicado pela variável Indice. É verificado se o pallet analisado passou por alguma entrada ou saída da célula, em determinado ciclo da simulação. Caso a resposta seja positiva, é executada a seqüência para colocar ou retirar a peça da célula.




```
int TForm1::TipoPecaIn(int Indice)
```

Esta rotina verifica a variável **Indice** e indica qual o tipo de peça que irá entrar no processo.

```
int TForm1::IndicePecaIn(int Indice)
```

Esta rotina verifica a variável **Indice** e indica qual o índice da peça que irá entrar no processo.

```
void TForm1::VerificaParadaEsteira(void)
```

Esta rotina verifica o tempo de parada da esteira. Caso ela ultrapasse o valor **TempoIOMaq**, ela faz com que a esteira volte a andar.

```
void TForm1::VerificaRobo(int i)
```

Esta rotina realiza o controle dos robôs de transporte. Ela faz várias verificações: sensores, filas de entrada e saída das máquinas, posição do robô, etc.

O controle do robô é feito em função da sua posição e de eventos que ocorrem. Existem três posições padronizadas para cada robô:

E: Esteira;

M: Máquina;

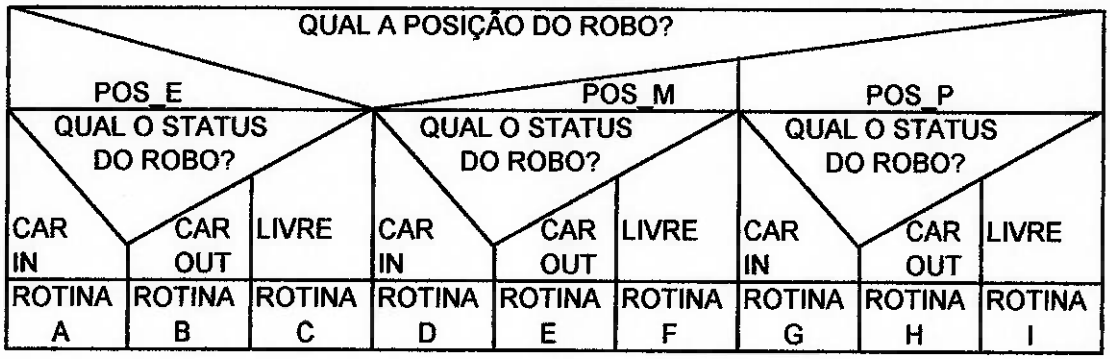
P: Qualquer posição entre Máquina - Esteira;

De acordo com estas posições, é feita a verificação dos eventos ocorridos e quais ações devem ser tomadas. Definiu-se como critério de otimização que o robô deve dar preferência sempre para liberar a esteira, pois esta é critica na célula. Qualquer outro critério que se queira acrescentar deve ser acrescentado nesta rotina do.

Os elementos que os robôs verificam são os sensores na esteira e as filas de entrada e saída das máquinas. De

acordo com os status destes elementos, o robô sabe qual a ação a ser tomada.

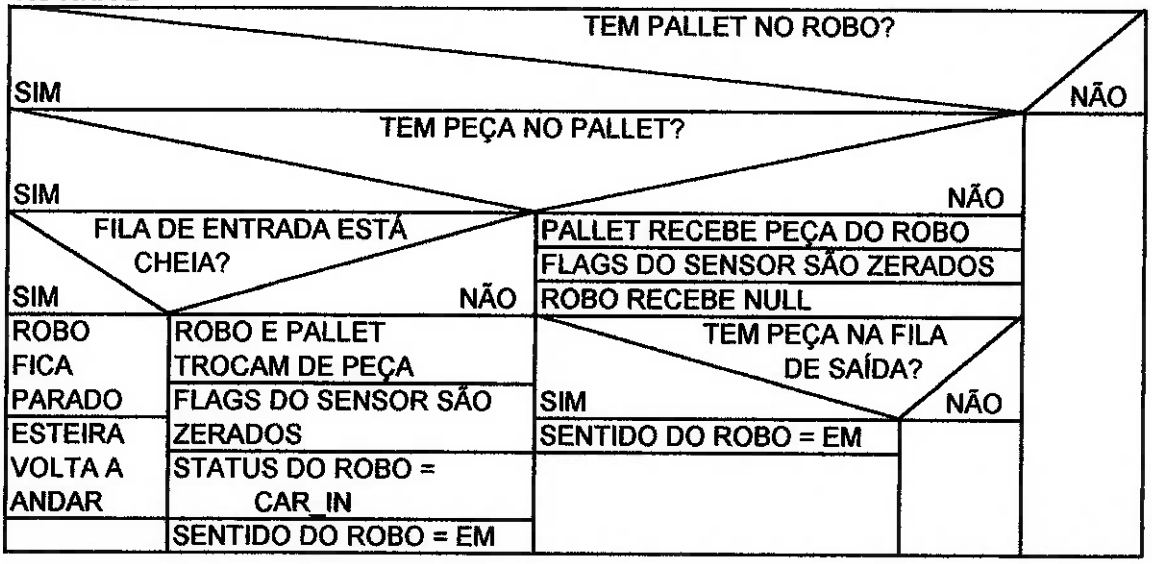
A seguir está um diagrama NS para os robôs da Mitutoyo, Centro de Usinagem e Torno CNC. O diagrama para o robô que atende a Fresadora CNC e o Torno DNC é semelhante.



ROTINA A

SENTIDO DO ROBO = EM

ROTINA B



ROTINA C

TEM PALLET NO ROBO?			
SIM			NÃO
TEM PEÇA NO PALLET?			
SIM			NÃO
FILA DE ENTRADA ESTÁ CHEIA?		TEM PEÇA NA FILA DE SAÍDA?	
SIM	NÃO	SIM	NÃO
ROBO FICA PARADO	ROBO RECEBE PEÇA DO PALLET	SENTIDO DO ROBO = EM	
ESTEIRA VOLTA A ANDAR	PALLET RECEBE NULL		
	FLAGS DO SENSOR SÃO ZERADOS		
	STATUS DO ROBO = CAR_IN		
	SENTIDO DO ROBO = EM		

ROTINA D

FILA DE ENTRADA ESTÁ CHEIA?		
NÃO		SIM
FILA DE SAÍDA ESTÁ VAZIA?		ROBO FICA PARADO
NÃO	SIM	
FILA DE ENTRADA RECEBE PEÇA DO ROBO	FILA DE ENTRADA RECEBE PEÇA DO ROBO	ESTEIRA VOLTA A ANDAR
ROBO RECEBE PEÇA DA FILA DE SAÍDA	ROBO RECEBE NULL	
STATUS DO ROBO = CAR_OUT	STATUS DO ROBO = LIVRE	
SENTIDO DO ROBO = ME	SENTIDO DO ROBO = ME	

ROTINA E

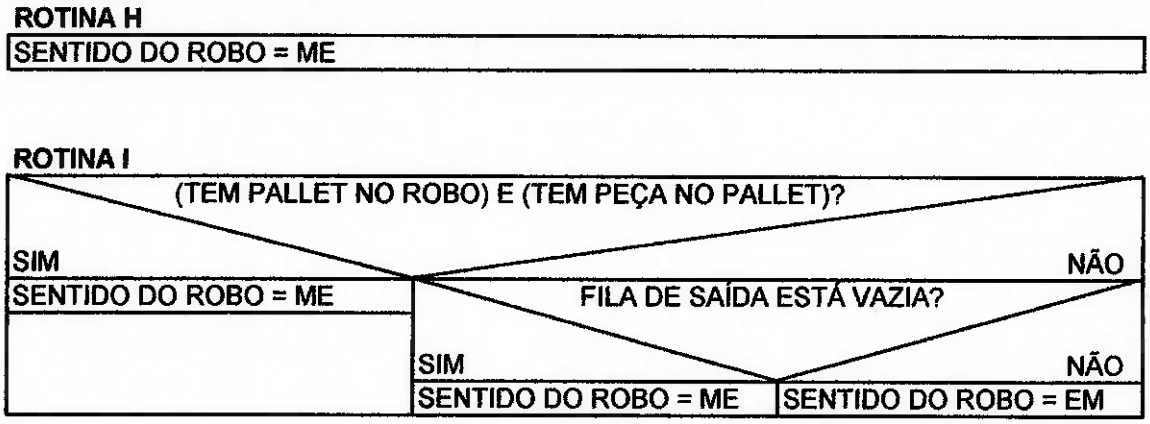
SENTIDO DO ROBO = ME

ROTINA F

FILA DE SAÍDA ESTÁ VAZIA?	
NÃO	SIM
ROBO RECEBE PEÇA DA FILA DE SAÍDA	SENTIDO DO ROBO = ME
STATUS DO ROBO = CAR_OUT	
SENTIDO DO ROBO = ME	

ROTINA G

SENTIDO DO ROBO = EM



Arquivo EDITOR.CPP

Este arquivo possui as rotinas que implementam o editor de textos utilizado para gerar o relatório da simulação. Abaixo têm-se a descriçào das funções relacionadas.

```
void __fastcall TForm1::abrir_arquivoClick(TObject *Sender)
```

Esta função abre um caixa de diálogo permitindo-se abrir um arquivo .TXT com o editor de texto.

```
void __fastcall TForm1::SalvarComoClick(TObject *Sender)
```

esta rotina cria uma caixa de diálogo para permitir ao usuário salvar o relatório gerado em um arquivo .TXT.

```
void __fastcall TForm1::imprimirClick(TObject *Sender)
```

Esta função abre um caixa de diálogo para imprimir o arquivo gerado.

```
void __fastcall TForm1::SelecionarTudo1Click(TObject
*Sender)
```

Esta função seleciona todo o texto que está no editor de textos e o armazena na área de transferência do Windows.

```
void __fastcall TForm1::Deletar1Click(TObject *Sender)
```

Esta função deleta o texto que está selecionado.

```
void __fastcall TForm1::copiarClick(TObject *Sender)
```

Está função copia para a área de transferência do Windows o texto selecionado.

```
void __fastcall TForm1::recortarClick(TObject *Sender)
```

Está função copia o texto selecionado para a área de transferência do Window e remove esta seleção do texto original.

```
void __fastcall TForm1::Colar2Click(TObject *Sender)
```

Está funão copia para o editor um texto que está na área de transferência.

```
void __fastcall TForm1::fonte1Click(TObject *Sender)
```

Esta função permite alterar a fonte do texto selecionado.

Arquivo GRAFICO.H

Neste arquivo estão definidas as funções que desenham os gráficos apresentados no programa. São quatro as funções, mas as quatro tem a mesma estrutura, só diferenciando uma da outra pelos ponteiros a que fazem referência. Abaixo têm-se a declaração das quatro funções:

```
void DesenhaGrafico1(float a1, float a2, float a3, float a4);
void DesenhaGrafico2(float a1, float a2, float a3, float a4);
void DesenhaGrafico3(float a1, float a2, float a3, float a4);
void DesenhaGrafico4(float a1, float a2, float a3, float a4);
```

Arquivo GRAFICO.CPP

Aqui estão implementadas as rotinas mencionadas acima. Por serem semelhantes, será feita uma explicação apenas da primeira.

```
void DesenhaGrafico1(float a1, float a2, float a3, float
a4);
```

Os parâmetros de entrada desta função são os valores que serão plotados. A partir destes valores e do tamanho da área disponível na tela, são calculadas as dimensões das barras e a escala em que os valores serão apresentados. A idéia básica consiste em desenhar o gráfico como um bitmap que é guardado por um ponteiro(no caso **Imagem1**), que é definido externamente e com as dimensões pré-definidas. Sendo assim, desenha-se as barras que indicam os valores, tomando-se como referência o maior dos quatro valores.

Arquivo INICIO.CPP

Este arquivo contêm todos os parâmetros de inicialização utilizados no programa de simulação. Contêm informações de elementos como robôs, pallets, máquinas, peças, etc.

Arquivo MAINFORM.H

Este arquivo contêm as declarações de constantes globais, de classes criadas e de funções utilizadas no programa de simulação.

Arquivo MAINFORM.CPP

Este arquivo contém a declaração da rotina responsável pela execução da simulação e da animação no tempo.

```
void __fastcall Tform1::Timer1Timer(Tobject *Sender)
```

Esta rotina é a responsável pela execução de todas as rotinas de simulação e de animação. Ela é executada periodicamente, de acordo com uma variável do programa.

Inicialmente ela verifica uma variável booleana para ver se a simulação está em andamento.

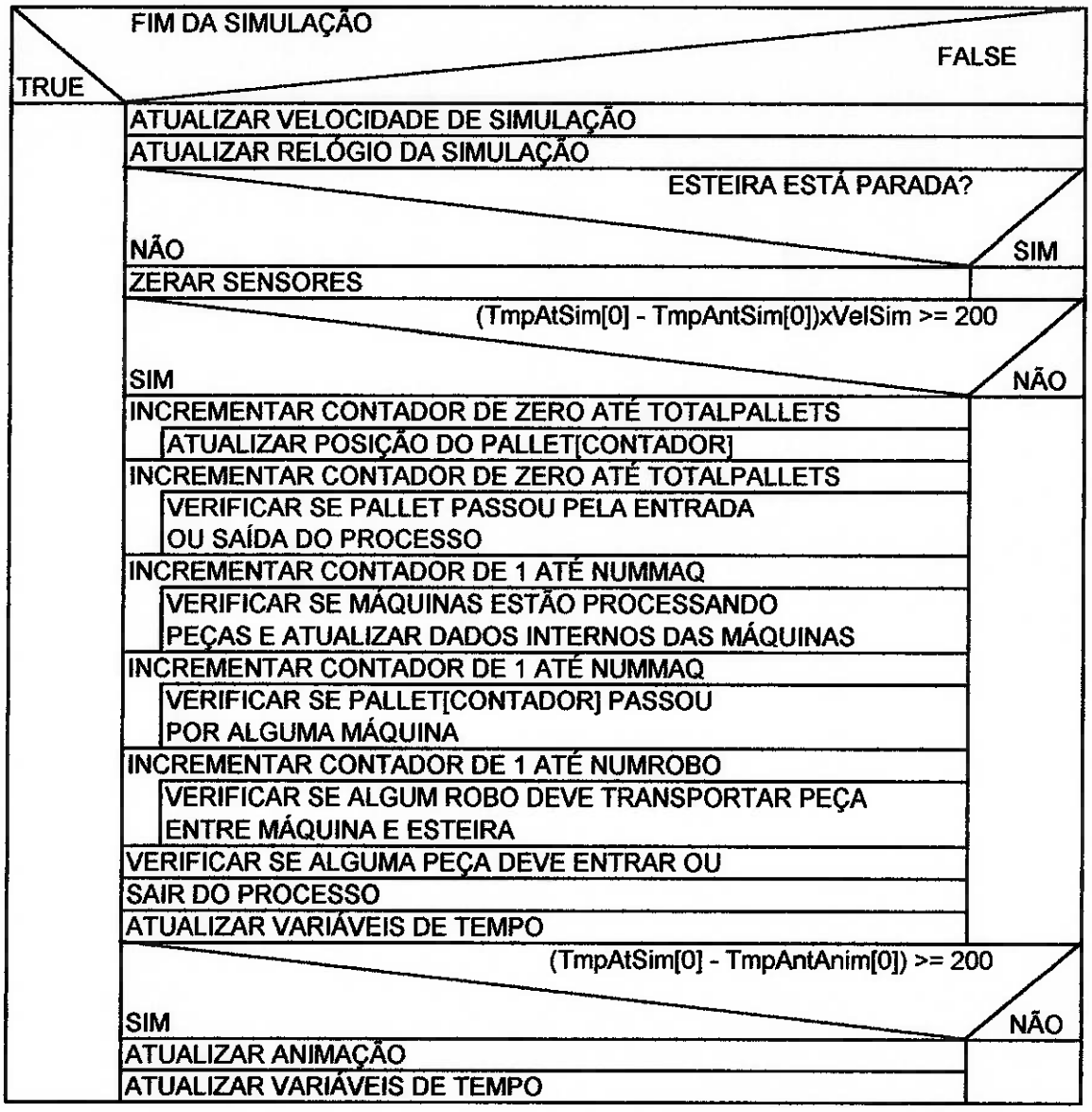
Caso a simulação esteja em andamento, são executados dois blocos distintos.

O primeiro bloco é responsável pela simulação. Nele são executados todas as rotinas de atualização de posição e

status dos pallets, robôs, sensores e máquinas, bem como a coletas de informações de tempo para cada peça.

O segundo bloco é responsável pela animação. Nele é executado uma função que gerencia as outras funções de animação.

A seguir está o diagrama NS desta rotina:




```
void TForm1::AtualizaTempo(void)
```

Esta rotina é a responsável pela atualização do relógio interno do simulador.

```
void TForm1::AtPosPallet(int Indice)
```

Esta rotina realiza a atualização da posição do pallet indicado pela variável **Indice**. Ela verifica o tempo decorrido, a posição anterior e a velocidade da esteira para realizar a atualização da posição em metros. Em seguida é realizada uma conversão da posição para pixels.

Arquivo ROBOS.CPP

Este arquivo contêm todas as rotinas referentes ao controle e animação dos robôs.

```
clRobo::clRobo()
```

Esta é a rotina construtora da classe **clRobo**, responsável pelo armazenamento de dados dos robôs.

```
clRobo::~clRobo()
```

Esta é a rotina destruidora da classe **clRobo**. Ela apaga os dados da classe **clRobo** da memória do computador assim que o programa é finalizado.

```
void clRobo::AssociaImagem(AnsiString imagem1, AnsiString
imagem2)
```

Esta rotina associa a imagem do robô e sua máscara, que serão utilizados na animação.

```
void clRobo::PatchBackground(void)
```

Esta rotina retira a imagem do robô do cenário da simulação.

```
void clRobo::MoveRoboLocation(void)
```

Esta rotina atualiza a posição dos robôs na simulação.

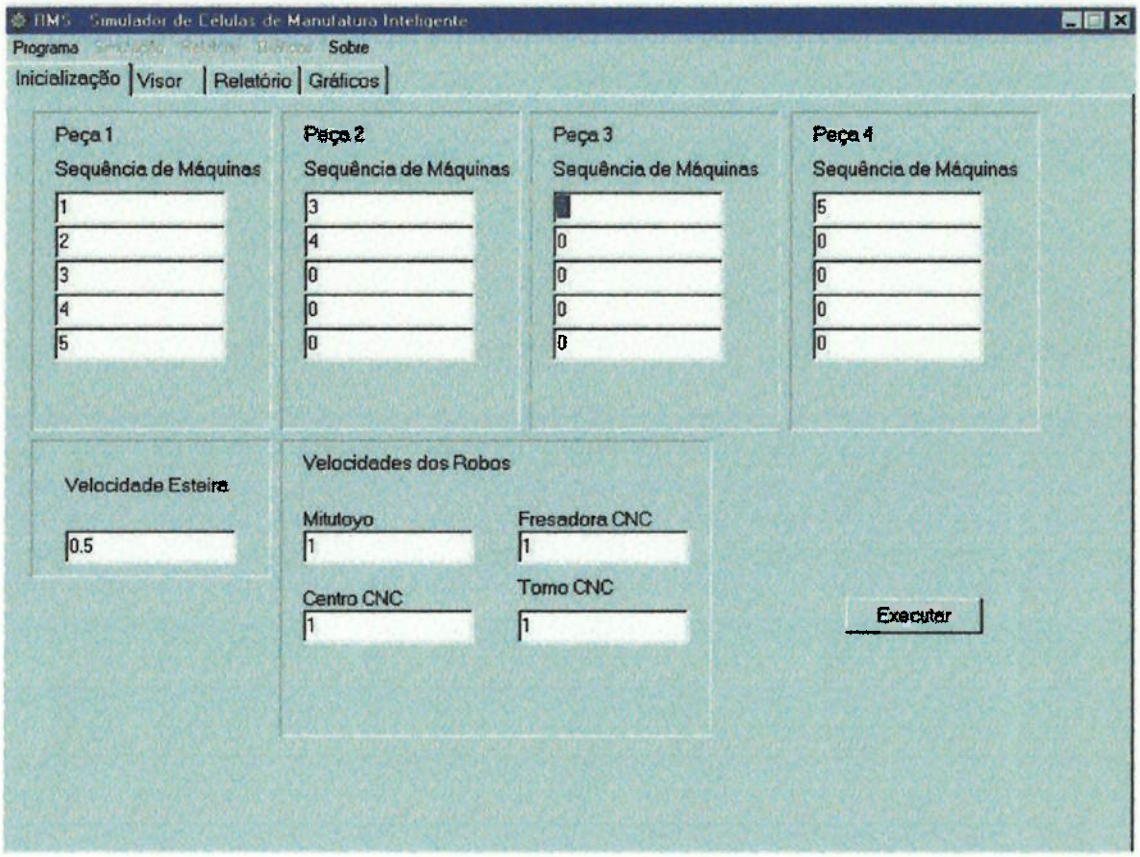
```
void clRobo::DrawRoboOnBackground(void)
```

Esta rotina coloca a imagem do robô no cenário da simulação.

2.6 Interface do Programa

O programa apresenta quatro telas básicas:

A primeira permite ao usuário a definição das seqüências de usinagem, bem como atribuir um valor para a velocidade dos robôs e da esteira. Esta tela é vista na figura a seguir.



A segunda apresenta a tela onde é mostrada a simulação E o tempo de simulação. Ainda nesta tela, é possível alterar a velocidade de simulação, que pode variar de 1 até 5 (Estes valores foram escolhidos arbitrariamente, pode-se mudá-los sem maiores problemas). Também está disponível ao usuário quatro botões que estão relacionados com a animação:

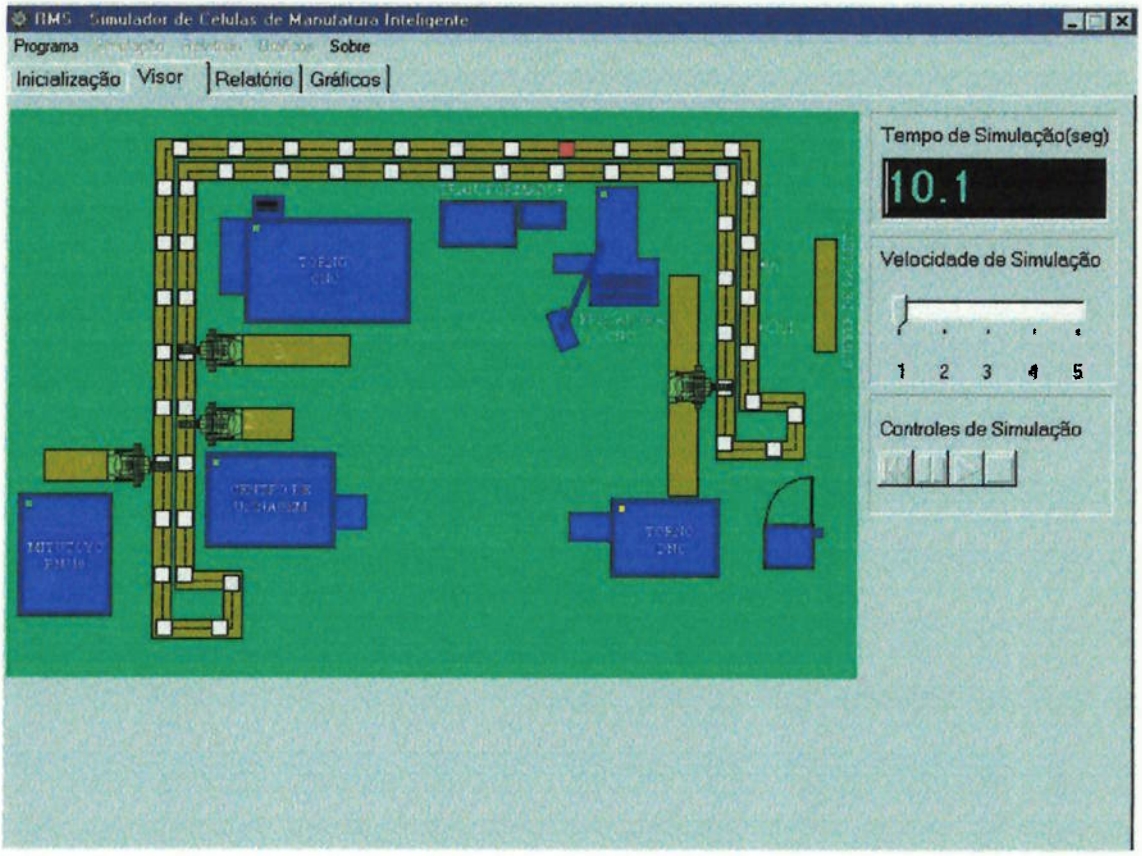
O primeiro reinicializa a simulação;

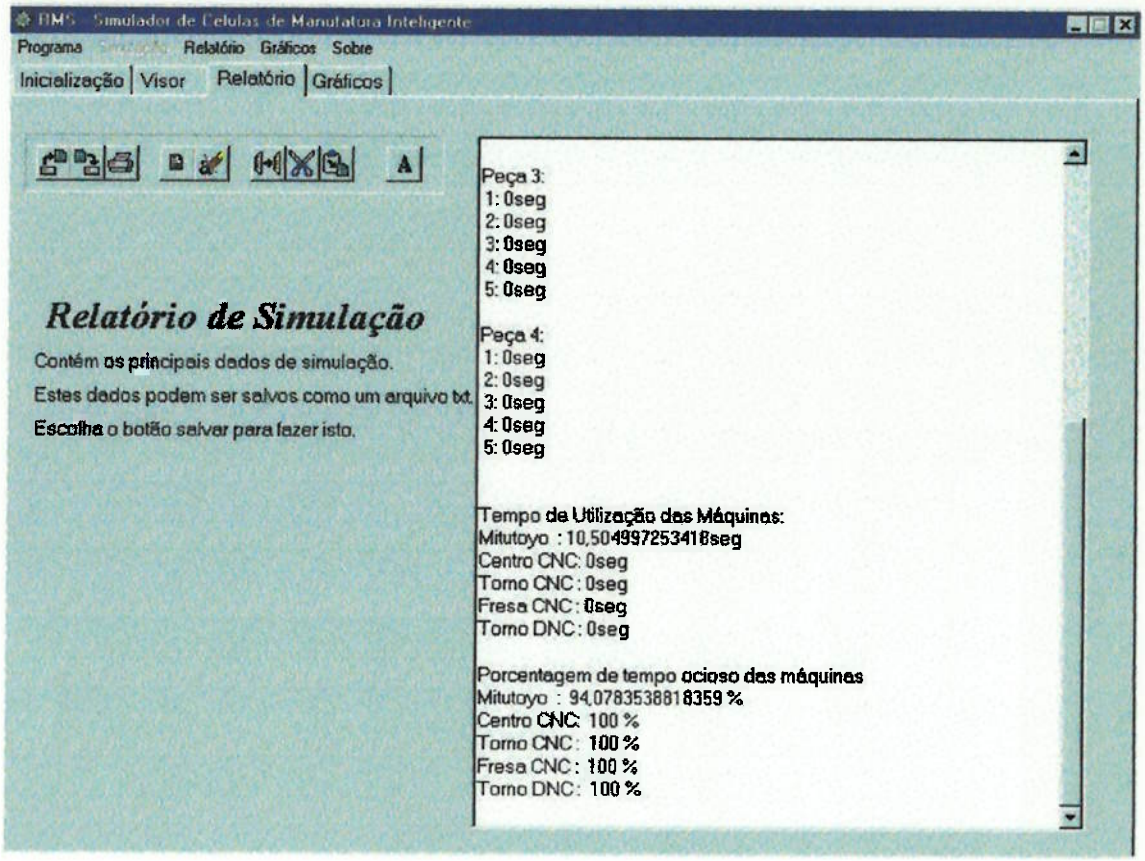
O segundo faz uma pausa da mesma;

O terceiro inicializa a simulação propriamente dita;

O quarto botão encerra a simulação e libera os itens "Relatório" e "Gráficos" do menu principal.

Abaixo tem-se as imagens destas telas:




Simulador de Células de Manufatura Inteligente
Programa
Relatório
Gráficos
Sobre
Inicialização
Visor
Relatório
Gráficos
Relatório de Simulação
Contém os principais dados de simulação.
Estes dados podem ser salvos como um arquivo txt.
Escolha o botão salvar para fazer isto.
Peça 3:
1: 0seg
2: 0seg
3: 0seg
4: 0seg
5: 0seg
Peça 4:
1: 0seg
2: 0seg
3: 0seg
4: 0seg
5: 0seg
Tempo de Utilização das Máquinas:
Mitutoyo : 10,504997253418seg
Centro CNC: 0seg
Torno CNC : 0seg
Fresa CNC : 0seg
Torno DNC : 0seg
Porcentagem de tempo ocioso das máquinas
Mitutoyo : 94,07835388 18359 %
Centro CNC: 100 %
Torno CNC : 100 %
Fresa CNC : 100 %
Torno DNC : 100 %


BMS Simulador de Celulas de Manufatura Inteligente
Programa Relatório Gráficos Sobre
Inicialização Visor Relatório Gráficos
Gráfico 1 Gráfico 2
Gráficos
Gráfico1 : Tempo de Produção
329.00
263.00
197.00
132.00
66.00

2.7 Conclusão

Chegou-se a um programa simples e que permite a simulação de produção de peças na célula modelo do Laboratório de Máquinas CNC da Escola Politécnica.

Trata-se de um programa específico para este fim e por isso mesmo é mais versátil e permite qualquer tipo de alteração uma vez que as rotinas de animação foram totalmente desenvolvidas e estão desvinculadas das funções de controle que foram ou que venham a ser implementadas.

Este programa apresenta amplas possibilidades de customização e alterações para futura implementação de simulação em tempo real. Ele não permite a inserção ou retirada de novas máquinas ou robôs diretamente no programa em execução, mas somente alterando a estrutura interna do programa, o que é bem simples.

O objetivo inicial foi atingido, pois chegou-se a um programa que supriu as principais dificuldades e empecilhos oferecidas pelo Arena estudado no início do projeto, tais como, dificuldade de adição de rotinas complexas de controle, definir seqüências de produção, adaptação para simulação em tempo real. Todas estas características estão disponíveis no software final, permitindo um futuro prosseguimento no estudo do controle de Células de Manufaturas Inteligentes.

3. REFERÊNCIAS BIBLIOGRÁFICAS

GROOVER, M.P. **Automation, Production Systems, and Computer Integrated Manufacturing.** s.e. New Jersey, Prentice-Hall, 1987.

NINA, N. **Visual Basic 4.0.** 1.ed. Rio de Janeiro, Brasport Livros e Multimídia Ltda, 1997.

KUAE, L.K.N.; BONESIO, M.C.M.; VILLELA, M.C.O. **Diretrizes Para Apresentação de Dissertações e Teses.** 1.ed. São Paulo, Escola Politécnica da USP, 1991.

LOPES, T. Borland C++ Builder: O Fim da Inveja. **Byte Brasil**, v.6, n.05, p.80-84, 1997.

Programe com a Força do Visual. **Informática Exame**, a.11, n.126, p.26-29, 1996.

SYSTEMS MODELING CORPORATION. **Arena User's Guide**, Systems Modeling Corporation, s.n.t.

MIANO, J.; CABANSKI, T.; HOWE, H. **Borland C++Builder How-to.** 1.ed. Corte Madera, Waite Group Press, 1997.

CALVERT'S, CHARLIE. **Borland C++Builder Unleashed.** 1.ed. , Sams Publishing, 1997.

MISCHEL, JIM; DUNTEMANN, JEFF **Borland C++Builder Programming Explorer.** 1.ed., The Coriolis Group, Inc., 1997.

POLITANO, P.R.; SCARPELLI, H. A. C.; PORTO, A. J. V. **Programação de Sistemas Flexíveis de Manufatura Baseada em lógica Fuzzy.** , Universidade Federal de São Carlos, p.287-292.

IWATA, K; ONOSATO, M. **Random Manufacturing System: a New Concept of Manufacturing Systems for Production to Order**, Osaka University and College of Industrial Technology, 1994.

4. ANEXO 1

Neste anexo é apresentada a listagem completa do programa, subdividida de acordo com os arquivos do programa.

RMS.MAK

```
# ---------------------------------------------------------------------------VERSION = BCB.01
# ---------------------------------------------------------------------------!ifndef BCB
BCB = $(MAKEDIR)\..
!endif
# ---------------------------------------------------------------------------PROJECT = RMS.exe
OBJFILES = RMS.obj MAINFORM.obj INICIO.obj BOTAO.obj ANIMACAO.obj AUXILIAR.obj \
   Controle.obj Arquivo.obj robos.obj Sobre.obj start.obj editor.obj grafico.obj
RESFILES = RMS.res
RESDEPEN = $(RESFILES) MAINFORM.dfm Sobre.dfm start.dfm
LIBFILES =
DEFFILE =
# ---------------------------------------------------------------------------CFLAG1 = -Od -Hc -w -k -r- -y -v -vi- -c -a4
-b- -w-par -w-inl -Vx -Ve -x
CFLAG2 = -Ic:\programacao\data;c:\programacao;$(BCB)\projects;$(BCB)\include;$(BCB)\include\vcl \
   -H=$(BCB)\lib\vcld.csm
PFLAGS = -AWinTypes=Windows;WinProcs=Windows;DbiTypes=BDE;DbiProcs=BDE;DbiErrs=BDE \
   -Uc:\programacao\data;c:\programacao;$(BCB)\projects;$(BCB)\lib\obj;$(BCB)\lib \
   -Ic:\programacao\data;c:\programacao;$(BCB)\projects;$(BCB)\include;$(BCB)\include\vcl \
   -v -$Y -$W -$O- -JPHNV -M
RFLAGS = -ic:\programacao\data;c:\programacao;$(BCB)\projects;$(BCB)\include;$(BCB)\include\vcl
LFLAGS = -Lc:\programacao\data;c:\programacao;$(BCB)\projects;$(BCB)\lib\obj;$(BCB)\lib \
   -aa -Tpe -x -v -V4.0
IFLAGS =
LINKER = ilink32
# ---------------------------------------------------------------------------ALLOBJ = c0w32.obj $(OBJFILES)
ALLRES = $(RESFILES)
ALLLIB = $(LIBFILES) vcl.lib import32.lib cp32mt.lib
# ---------------------------------------------------------------------------
.autodepend

$(PROJECT): $(OBJFILES) $(RESDEPEN) $(DEFFILE)
    $(BCB)\BIN\$(LINKER) @&&!
    $(LFLAGS) +
    $(ALLOBJ), +
    $(PROJECT),, +
    $(ALLLIB), +
    $(DEFFILE), +
    $(ALLRES)
!

.pas.hpp:
    $(BCB)\BIN\dcc32 $(PFLAGS) { $** }

.pas.obj:
    $(BCB)\BIN\dcc32 $(PFLAGS) { $** }

.cpp.obj:
    $(BCB)\BIN\bcc32 $(CFLAG1) $(CFLAG2) -o$* $*

.c.obj:
    $(BCB)\BIN\bcc32 $(CFLAG1) $(CFLAG2) -o$* $**

.rc.res:
    $(BCB)\BIN\brcc32 $(RFLAGS) $<
#---------------------------------------------------------------------------
```

RMS.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include "start.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("MAINFORM.cpp", Form1);
USERES("RMS.res");
USEUNIT("INICIO.cpp");
USEUNIT("BOTAO.cpp");
USEUNIT("ANIMACAO.cpp");
USEUNIT("AUXILIAR.cpp");
USEUNIT("Controle.cpp");
USEUNIT("Arquivo.cpp");
USEUNIT("robos.cpp");
USEFORM("Sobre.cpp", AboutBox);
USEFORM("start.cpp", SplashScreen);
USEUNIT("editor.cpp");
USEUNIT("grafico.cpp");
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
     Application->Initialize();
     SplashScreen=new TSplashScreen(Application);
     SplashScreen->Position=poScreenCenter;
```

```
    SplashScreen->Show();
    Application->CreateForm(__classid(TForm1), &Form1);
    Application->CreateForm(__classid(TAboutBox), &AboutBox);
    Application->CreateForm(__classid(TSplashScreen), &SplashScreen);
    Application->Run();
    SplashScreen->Hide();
    delete SplashScreen;
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
//---------------------------------------------------------------------------
```

ANIMACAO.H

```
//---------------------------------------------------------------------------
#ifndef ANIMACAOH
#define ANIMACAOH
//---------------------------------------------------------------------------
#endif
```

ANIMACAO.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
#include "ANIMACAO.h"
//---------------------------------------------------------------------------
void TForm1::AtualizaAnimacao(void)|
  register int i;
  DrawPalletOnBackground();
  for(i = 1;i <= NUM_ROBOS;i++)
    robos[i].DrawRoboOnBackground();
  DrawStatusMaquina();
  MoveBitmapToScreen();
  for(i = 1;i <= NUM_ROBOS;i++)
    robos[i].PatchBackground();
  PatchBackground();
}
//---------------------------------------------------------------------------
// ROTINAS DE ANIMACAO DA ESTEIRA
//---------------------------------------------------------------------------
void TForm1::PatchBackground(void)
{
  int Cont;
  TRect LocationRect[TotalPallets], RoboLocationRect;
  for (Cont = 0; Cont < TotalPallets; Cont++){
    // get coordinates of ship location on screen.
    LocationRect[Cont] = Rect(Pallet[Cont].PalletCoord.x,
      Pallet[Cont].PalletCoord.y,
      Pallet[Cont].PalletCoord.x+Pallet[Cont].ImgPallet->Width,
      Pallet[Cont].PalletCoord.y+Pallet[Cont].ImgPallet->Height);
  }
  RoboLocationRect = Rect(RoboLocation.x, RoboLocation.y,
  RoboLocation.x + Robo[CondAtual[0]]->Width,
  RoboLocation.y + Robo[CondAtual[0]]->Height);
  // need to copy (cmSrcCopy) pixels from the saved bknd bitmap
  // to the background. This restores the background bitmap to
  // its original state (no spaceship)
  Background.imagem->Canvas->CopyMode = cmSrcCopy;
  Background.imagem->Canvas->CopyRect(RoboLocationRect,
    RoboSaveBack->Canvas, RoboRect);
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Background.imagem->Canvas->CopyRect(LocationRect[Cont],
      Pallet[Cont].SavPallet->Canvas, Pallet[Cont].PalletRect);
  }
|
void TForm1::DrawPalletOnBackground(void)
{
  int Cont;
  TRect LocationRect[TotalPallets];
  for (Cont = 0; Cont < TotalPallets; Cont++){
    // associa a imagem da peca que estah no pallet
    Pallet[Cont].ImgPallet = ImgPeca[Pallet[Cont].Peca.TipoPeca];
    // pega as coordenadas do pallet relativas ao background
    LocationRect[Cont] = Rect(Pallet[Cont].PalletCoord.x,
      Pallet[Cont].PalletCoord.y,
      Pallet[Cont].PalletCoord.x + Pallet[Cont].ImgPallet->Width,
      Pallet[Cont].PalletCoord.y + Pallet[Cont].ImgPallet->Height);
  }
  // A regiao onde o Pallet vai se localizar serah
  // gravando num buffer para depois ser redesenhado
  // no Background
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Pallet[Cont].SavPallet->Canvas->CopyRect( Pallet[Cont].PalletRect,
      Background.imagem->Canvas, LocationRect[Cont]);
  }
  // O desenho do pallet sera gravado no background
  // Aqui serah aplicado a mascara do desenho do pallet
  Background.imagem->Canvas->CopyMode = cmSrcAnd;  // DEST = SRC AND DEST
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Background.imagem->Canvas->CopyRect(LocationRect[Cont],
      Pallet[Cont].MscPallet->Canvas, Pallet[Cont].PalletRect);
  }
  // O desenho do pallet sera gravado no background
  // Aqui serah aplicado o desenho do pallet
  Background.imagem->Canvas->CopyMode = cmSrcPaint;  // DEST = SRC OR DEST
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Background.imagem->Canvas->CopyRect(LocationRect[Cont],
      Pallet[Cont].ImgPallet->Canvas, Pallet[Cont].PalletRect);
  }
```

```
}
void TForm1::MoveBitmapToScreen(void){
  // move the image from the memory bitmap
  // to the PaintBox
  PaintBox1->Canvas->Draw(0,0,Background.imagem);
}
//---------------------------------------------------------------------------
// ROTINAS DE ANIMACAO DAS MAQUINAS
//---------------------------------------------------------------------------
void TForm1::DrawStatusMaquina(void){
  register int Indice;
  AnsiString InfoString;
  TRect RectAux[NumMaquina + 1], RectStatus;
  RectStatus = Rect(0, 0, ImagStatus[0]->Width, ImagStatus[0]->Height);
  for (Indice = 1; Indice <= NumMaquina; Indice++){
    // associa a imagem do status atual da maquina
    Maquina[Indice].Status.ImgStatus = ImagStatus[Maquina[Indice].Status.StatusAtual];
    RectAux[Indice] = Rect(Maquina[Indice].Status.PosStatus.x,
      Maquina[Indice].Status.PosStatus.y,
      Maquina[Indice].Status.PosStatus.x +
      Maquina[Indice].Status.ImgStatus->Width,
      Maquina[Indice].Status.PosStatus.y +
      Maquina[Indice].Status.ImgStatus->Width);
  }
  Background.imagem->Canvas->CopyMode = cmSrcAnd;
  for (Indice = 1; Indice <= NumMaquina; Indice++)
    Background.imagem->Canvas->CopyRect (RectAux[Indice],
      Maquina[Indice].Status.MscStatus->Canvas,RectStatus);
  Background.imagem->Canvas->CopyMode = cmSrcPaint;
  for (Indice = 1; Indice <= NumMaquina; Indice++)
    Background.imagem->Canvas->CopyRect (RectAux[Indice],
      Maquina[Indice].Status.ImgStatus->Canvas,RectStatus);
}
//---------------------------------------------------------------------------
```

ARQUIVO.H

```
//---------------------------------------------------------------------------
#ifndef ArquivoH
#define ArquivoH
//---------------------------------------------------------------------------
#endif
```

ARQUIVO.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
#include "Arquivo.h"
//---------------------------------------------------------------------------
FILE *fp;
FILE *fp2;
void TForm1::CriarArquivo(void){
  fp = fopen("Saida1.txt","w");
}
void TForm1::FecharArquivo(void){
  fclose(fp);
}
void TForm1::GravarPeca1(void){
  int Cont;
  int PcTipo, PcInd;
  PcTipo = 1;
  PcInd = 1;
  fprintf(fp,"Tempo = %lf\n",(float)TmpAtSim[0]/1000);
  fprintf(fp,"Posicao Atual: %d %d\n", Peca[PcTipo][PcInd].PosicaoAtual[0],Peca[PcTipo] [PcInd].PosicaoAtual[1]);
  if(Peca[PcTipo][PcInd].PosicaoAtual[0] == 1)
    fprintf(fp,"Posicao na esteira: %d Pixels  /  %6.2lf Metros\n", Pallet[Peca[PcTipo]
[PcInd].PosicaoAtual[1]].PosicaoAtual.Pixels,Pallet[Peca[PcTipo][PcInd].PosicaoAtual[1]]. PosicaoAtual.Metros);
  fprintf(fp,"Linha Processo: %d\n",Peca[PcTipo][PcInd].LinhaProcesso);
  fprintf(fp,"Sequencia de operacoes:  %d  %d  %d  %d  %d\n", Peca[PcTipo][PcInd]. Processos[0].Sequencia,
Peca[PcTipo][PcInd].Processos[1].Sequencia, Peca[PcTipo][PcInd]. Processos[2].Sequencia,
Peca[PcTipo][PcInd].Processos[3].Sequencia, Peca[PcTipo][PcInd]. Processos[4].Sequencia);
  fprintf(fp,"Tempo de entrada: %6.2lf\n",Peca[PcTipo][PcInd].TempoPeca[0]);
  fprintf(fp,"Tempo de saida:   %6.2lf\n",Peca[PcTipo][PcInd].TempoPeca[1]);
  fprintf(fp,"Tempos:          TmpInc     TmpFnl     TmpPrc      TmpIn      TmpOut\n");
  for (Cont=0; Cont < NumMaquina;Cont++){
    fprintf(fp,"Processo %d:     %6.2lf     %6.2lf     %6.2lf      %6.2lf     %6.2lf\n", Cont+1,
Peca[PcTipo][PcInd].Processos[Cont].TmpInc, Peca[PcTipo][PcInd].Processos[Cont]. TmpFnl,
Peca[PcTipo][PcInd].Processos[Cont].TmpPrc, Peca[PcTipo][PcInd].Processos[Cont]. TmpIn,
Peca[PcTipo][PcInd].Processos[Cont].TmpOut);
  }
  fprintf(fp,"-----------------------------------------------------------------\n");
}
void TForm1::GravarPeca(int Tipo, int Indice){
  int Cont;
  int PcTipo, PcInd;
  PcTipo = Tipo;
  PcInd = Indice;
  fprintf(fp2,"Peca (Tipo/Indice): %d / %d\n", PcTipo, PcInd);
  fprintf(fp2,"Posicao Atual: %d %d\n",Peca[PcTipo][PcInd].PosicaoAtual[0], Peca[PcTipo] [PcInd].PosicaoAtual[1]);
  fprintf(fp2,"Linha Processo: %d\n",Peca[PcTipo][PcInd].LinhaProcesso);
  fprintf(fp2,"Sequencia de operacoes:  %d  %d  %d  %d  %d\n", Peca[PcTipo][PcInd]. Processos[0].Sequencia,
Peca[PcTipo][PcInd].Processos[1].Sequencia, Peca[PcTipo][PcInd]. Processos[2].Sequencia,
Peca[PcTipo][PcInd].Processos[3].Sequencia, Peca[PcTipo][PcInd]. Processos[4].Sequencia);
  fprintf(fp2,"Tempo de entrada: %6.2lf\n",Peca[PcTipo][PcInd].TempoPeca[0]);
  fprintf(fp2,"Tempo de saida:   %6.2lf\n",Peca[PcTipo][PcInd].TempoPeca[1]);
  fprintf(fp2,"Tempos:          TmpInc     TmpFnl     TmpPrc      TmpIn      TmpOut\n");
  for (Cont=0; Cont < NumMaquina;Cont++){
    fprintf(fp2,"Processo %d:     %6.2lf     %6.2lf     %6.2lf      %6.2lf     %6.2lf\n", Cont+1,
Peca[PcTipo][PcInd].Processos[Cont].TmpInc, Peca[PcTipo][PcInd].Processos[Cont]. TmpFnl,
Peca[PcTipo][PcInd].Processos[Cont].TmpPrc, Peca[PcTipo][PcInd].Processos[Cont]. TmpIn,
Peca[PcTipo][PcInd].Processos[Cont].TmpOut);
  }
  fprintf(fp2,"-----------------------------------------------------------------\n");
}
```

```
void TForm1::GravarAll(void){
  int Cont1, Cont2;
  fp2 = fopen("Saida2.txt","w");
  for(Cont1 = 1; Cont1 <= NumPeca; Cont1++)
    for(Cont2 = 1; Cont2 <= TotalPecas[Cont1]; Cont2++)
      GravarPeca(Cont1, Cont2);
  fclose(fp2);
}
```

AUXILIAR.H

```
//---------------------------------------------------------------------------
#ifndef AUXILIARH
#define AUXILIARH
//---------------------------------------------------------------------------
#endif
```

AUXILIAR.CPP

```
//---------------------------------------------------------------------------
// Este modulo tem possui as declaracoes de funcoes que perentem
// a classes declaradas em MAINFORM.H
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
#include "AUXILIAR.h"
//---------------------------------------------------------------------------
// Declaracao de funcoes da classe clPallet
void clPallet::MetroParaPixel(void){
  int Saida;
  float Decimal, PosMetro;
  PosMetro = PosicaoAtual.Metros * TotalPontos / CompEstMetro;
  Saida = (int)PosMetro;
  Decimal = PosMetro - Saida;
  if(Decimal < 0.5)
    Saida--;
  if(Saida < 0)
    Saida = 0;
  if(Saida >= TotalPontos)
    Saida = TotalPontos - 1;
  PosicaoAtual.Pixels = Saida;
}
//---------------------------------------------------------------------------
// Declaracao de funcoes da classe clPeca
bool
clPeca::ProcessosExecutados(void){
  bool Saida;
  int Cont;
  Saida = true;
  for(Cont = 0; Cont < NumMaquina; Cont++){
    Saida = Saida && (Processos[Cont].Sequencia <= 0);
  }
  return(Saida);
}
//---------------------------------------------------------------------------
// Declaracao de funcoes da classe cenario
cenario::cenario(){
  imagem = new Graphics::TBitmap;
  imagem->LoadFromResourceName((int)HInstance, "PLANTA");
}
cenario::~cenario(){
  delete imagem;
}
//---------------------------------------------------------------------------
// Declaracao de funcoes da classe clMaquina
//Retira uma peca da fila de entrada da maquina
short int clMaquina::TirardaFilaEntrada(void){
  short int Saida, tipo, indice;
  if(rpos1 == spos1)
    Saida = 0;
  else{
    Saida = 1;
    rpos1++;
    tipo   = FilaEntrada[rpos1-1].TipoPeca;
    indice = FilaEntrada[rpos1-1].IndicePeca;
    PecaProcesso.TipoPeca = tipo;
    PecaProcesso.IndicePeca = indice;
    Form1->Peca[tipo][indice].Processos[Form1->Peca[tipo][indice].LinhaProcesso].TmpInc = (float)Form1-
>TmpAtSim[0]/1000;
    FilaEntrada[rpos1-1].TipoPeca = 0;
    FilaEntrada[rpos1-1].IndicePeca = 0;
    if(rpos1==TamanhoFila)
      rpos1 = 0;
  }
  return(Saida);
}
//---------------------------
//Coloca uma peca na fila de entrada da maquina
short int clMaquina::ColocarnaFilaEntrada(short int tipo, short int indice){
  short int Saida;
  if(((spos1 + 1) == rpos1)||((spos1 + 1) == TamanhoFila) && (rpos1 == 0)))
    Saida = 0;
  else{
    FilaEntrada[spos1].TipoPeca   = tipo;
    FilaEntrada[spos1].IndicePeca = indice;
    Form1->Peca[tipo][indice].Processos[Form1->Peca[tipo][indice].LinhaProcesso].TmpIn = (float)Form1-
>TmpAtSim[0]/1000;
    Form1->Peca[tipo][indice].PosicaoAtual[0] = 2;
    Form1->Peca[tipo][indice].PosicaoAtual[1] = Indice;
    spos1++;
    Saida = 1;
    if(spos1 == TamanhoFila)
      spos1 = 0;
  }
```

```
  return(Saida);
}
//---------------------------
//Retira uma peca da fila de saida da maquina
short int clMaquina::TirardaFilaSaida(void){
  short int Saida, tipo, indice;
  if(rpos2 == spos2)
    Saida = 0;
  else{
    rpos2++;
    tipo   = FilaSaida[rpos2-1].TipoPeca;
    indice = FilaSaida[rpos2-1].IndicePeca;
    if(Indice != 5){
      Form1->robos[Indice].TipoPeca   = tipo;
      Form1->robos[Indice].IndicePeca = indice;
    }
    else{
      Form1->robos[Indice-1].TipoPeca   = tipo;
      Form1->robos[Indice-1].IndicePeca = indice;
    }
    Form1->Peca[tipo][indice].Processos[Form1->Peca[tipo][indice].LinhaProcesso].TmpOut = (float)Form1-
>TmpAtSim[0]/1000;
    Form1->Peca[tipo][indice].Processos[Form1->Peca[tipo][indice].LinhaProcesso].Sequencia*=(-1);
    Form1->Peca[tipo][indice].PosicaoAtual[0] = 1;
    Form1->Peca[tipo][indice].PosicaoAtual[1] = Indice;
    Form1->Peca[tipo][indice].LinhaProcesso++;
    FilaSaida[rpos2-1].TipoPeca = 0;
    FilaSaida[rpos2-1].IndicePeca = 0;
    Saida = 1;
    if(rpos2==TamanhoFila)
      rpos2 = 0;
  }
  return(Saida);
}
//---------------------------
//Coloca uma peca na fila de saida da maquina
short int clMaquina::ColocarnaFilaSaida(void){
  short int Saida, tipo, indice;
  if(((spos2 + 1) == rpos2)||(((spos2 + 1) == TamanhoFila) && (rpos2 == 0)))
    Saida = 0;
  else{
    tipo   = PecaProcesso.TipoPeca;
    indice = PecaProcesso.IndicePeca;
    FilaSaida[spos2].TipoPeca   = tipo;
    FilaSaida[spos2].IndicePeca = indice;
    Form1->Peca[tipo][indice].Processos[Form1->Peca[tipo][indice].LinhaProcesso].TmpFnl = (float)Form1-
>TmpAtSim[0]/1000;
    PecaProcesso.TipoPeca       = 0;
    PecaProcesso.IndicePeca     = 0;
    spos2++;
    Saida = 1;
    if(spos2 == TamanhoFila)
      spos2 = 0;
  }
  return(Saida);
}
//---------------------------------------------------------------------------
short int clMaquina::FilaEntradaCheia(void){
  if(((spos1 + 1) == rpos1)||(((spos1 + 1) == TamanhoFila) && (rpos1 == 0)))
    return(1);
  return(0);
}
short int clMaquina::FilaEntradaVazia(void){
  if(rpos1 == spos1)
    return(1);
  return(0);
}
short int clMaquina::FilaSaidaCheia(void){
  if(((spos2 + 1) == rpos2)||(((spos2 + 1) == TamanhoFila) && (rpos2 == 0)))
    return(1);
  return(0);
}
short int clMaquina::FilaSaidaVazia(void){
  if(rpos2 == spos2)
    return(1);
  return(0);
}
//---------------------------------------------------------------------------
// Declaracao de funcoes da classe clSensor
//Reinicializa os sensores
void clSensor::Reiniciar(void){
  pallet = -1;
  FlagPallet = 0;
  FlagPeca = 0;
}
//---------------------------------------------------------------------------
```

BOTAO.H

```
//---------------------------------------------------------------------------
#ifndef BOTAOH
#define BOTAOH
//---------------------------------------------------------------------------
#endif
```

BOTAO.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
#include "BOTAO.h"
//---------------------------------------------------------------------------
// Declaracao de funcoes globais
float CalcFloat( TEdit *Valor){
```

```
    int Size;
    float Saida;
    Size=Valor->GetTextLen();
    char *Numero=new char[++Size];
    Valor->GetTextBuf(Numero,Size);
    Saida=atof(Numero);
    return (Saida);
}
float CalcInt( TEdit *Valor){
    int Size, Saida;
    Size=Valor->GetTextLen();
    char *Numero = new char[++Size];
    Valor->GetTextBuf(Numero,Size);
    Saida = atoi(Numero);
    return (Saida);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FecharArquivo();
  GravarAll();
  Close();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::VelEstControlChange(TObject *Sender, bool &AllowChange)
{
}
void __fastcall TForm1::VelEstChange(TObject *Sender)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ULClick(TObject *Sender)
{
  CondMov[PosicaoRobo] = Esquerda;
  CondMov[Movimento] = Subir;
}

void __fastcall TForm1::URClick(TObject *Sender)
{
  CondMov[PosicaoRobo] = Direita;
  CondMov[Movimento] = Subir;
}

void __fastcall TForm1::DLClick(TObject *Sender)
{
  CondMov[PosicaoRobo] = Esquerda;
  CondMov[Movimento] = Descer;
}

void __fastcall TForm1::DRClick(TObject *Sender)
{
  CondMov[PosicaoRobo] = Direita;
  CondMov[Movimento] = Descer;
}

void __fastcall TForm1::RStopClick(TObject *Sender)
{
  CondMov[Movimento] = CondAtual[Movimento];
}
//---------------------------------------------------------------------------
void __fastcall TForm1::VelRoboChange(TObject *Sender)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::VelRoboControlChange(TObject *Sender, bool &AllowChange)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::VelSysChange(TObject *Sender)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::VelSysControlChange(TObject *Sender, bool &AllowChange)
{
}
//---------------------------------------------------------------------------
// Esta funcao eh necessaria para evitar que a velocidade de
// simulacao seja alterada durante uma pausa.
void __fastcall TForm1::btPausaClick(TObject *Sender)
{
  if (PausaVar){
        PausaVar = false;
   // VelSys->Enabled = true;
  }
  else{
        PausaVar = true;
  //  VelSys->Enabled = false;
  }
}
//---------------------------------------------------------------------------
```

CONTROLE.H

```
//---------------------------------------------------------------------------
#ifndef ControleH
#define ControleH
//---------------------------------------------------------------------------
#endif
```

CONTROLE.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
```

```
#include "Controle.h"
//---------------------------------------------------------------------------
void TForm1::VerificaMaquina(int Indice){
  short int PosFila;
  short int IndPeca, TipoPeca;
  TipoPeca = Maquina[Indice].PecaProcesso.TipoPeca;
  if(TipoPeca != 0){ //  verifica se ha peca em processamento
    IndPeca = Maquina[Indice].PecaProcesso.IndicePeca;
    if(((float)TmpAtSim[0]/1000) - Peca[TipoPeca][IndPeca].Processos[Peca[TipoPeca][IndPeca] .LinhaProcesso].TmpInc)
>= Peca[TipoPeca][IndPeca].Processos[Peca[TipoPeca][IndPeca]. LinhaProcesso].TmpPrc){
      // Verificar influencia da velocidade de simulacao
      PosFila = Maquina[Indice].FilaSaidaCheia();
      if(PosFila == 0){  // executa se tiver espaco na fila de saida
        // Retira a peca da variavel PECAPROCESSO e encaminha para a FILA DE SAIDA
        Maquina[Indice].ColocarnaFilaSaida();
        Maquina[Indice].tempo[0] = Maquina[Indice].tempo[0] + ((float)TmpAtSim[0] / 1000 -Maquina[Indice].taux);
        Maquina[Indice].Status.StatusAtual = Espera; // Máquina esperando nova peca
      }  // if PosFila
    }  // if TmpAtSim
  }  // if Maquina
  else{
    PosFila = Maquina[Indice].FilaEntradaVazia();
    if(PosFila == 0){  // executa de tiver peca na fila de entrada
      // Retira a peca da a FILA DE ENTRADA e encaminha para a variavel PECAPROCESSO
      Maquina[Indice].TirardaFilaEntrada();
      Maquina[Indice].Status.StatusAtual = Ativo;          // Máquina operando
      Maquina[Indice].taux =(float)TmpAtSim[0]/1000;
    }
  }  // else
}
//---------------------------------------------------------------------------
void TForm1::VerificaPallet(int Indice){
  short int IndPeca, TipoPeca;
  int i, j, IndMaq, Anterior;
  bool TemMaq;
  IndMaq = -1;
  TemMaq = false;
  i = Pallet[Indice].PosicaoAnterior.Pixels + 1;
  if(i == Esteira.Comprimento.Pixels)
    i = 0;
  while((i != (Pallet[Indice].PosicaoAtual.Pixels + 1))&&(!TemMaq)){
    j = 1;
    while((j <= NumMaquina) && (!TemMaq)){
      if(Maquina[j].Posicao.Pixels == i){
        IndMaq = j;                       // Este laco WHILE serve para
        TemMaq = true;                    // determinar se exite alguma
      }  // if                           // maquina entre as posicoes atual
      j++;                                // e anterior do pallet
    }  // for while j
    i++;
    if(i == Esteira.Comprimento.Pixels)
      i = 0;
  }  // while
  //-------------------------
  if(TemMaq){                                  // Este IF foi acrescentado para
    Anterior = Maquina[IndMaq].PalletAnterior; // corrigir um bug que nao foi
    if (Anterior < 0)                          // possivel rastrear.
      Maquina[IndMaq].PalletAnterior = Indice;
    else{
      if (Anterior == 0)
        Anterior = TotalPallets;
      if (Indice == (Anterior - 1))
        Maquina[IndMaq].PalletAnterior = Indice;
      else
        TemMaq = false;
    }
  }                                            // Fim do IF de correcao do bug
  //-------------------------
  if(TemMaq){  // executa se houver maquina entre posicoes atual e anterior
    TipoPeca = Pallet[Indice].Peca.TipoPeca;
    if(TipoPeca > 0){  // executa se houver peca no pallet analisado
      IndPeca = Pallet[Indice].Peca.IndicePeca;
      // Verifica se a maquina é a proxima a processar a peca
      if(Peca[TipoPeca][IndPeca].Processos[Peca[TipoPeca][IndPeca].LinhaProcesso].Sequencia == IndMaq){
        // Parada da esteira
        Esteira.Velocidade.MetrosPorSeg = 0;
        Esteira.Velocidade.PixelsPorSeg = 0;
        Esteira.InicioParada = TmpAtSim[0];
        sensor[IndMaq].pallet     = (short int) Indice;
        sensor[IndMaq].FlagPallet = 1;
        sensor[IndMaq].FlagPeca   = 1;
      }
      if(IndMaq == 4)
        if(Peca[TipoPeca][IndPeca].Processos[Peca[TipoPeca][IndPeca].LinhaProcesso].Sequencia == IndMaq+1){
          Esteira.Velocidade.MetrosPorSeg = 0;
          Esteira.Velocidade.PixelsPorSeg = 0;
          Esteira.InicioParada = TmpAtSim[0];
          sensor[IndMaq+1].pallet     = (short int) Indice;
          sensor[IndMaq+1].FlagPallet = 1;
          sensor[IndMaq+1].FlagPeca   = 1;
        }
    } // if TipoPeca
    // executa se nao houver peca no pallet
    if((TipoPeca == 0) && ((robos[IndMaq].status == CAR_OUT) ||(robos[IndMaq].status == CAR_OUTT) ||
(robos[IndMaq].status == CAR_OUTF))/*  && robos[IndMaq].pos == POS_E */){
      Esteira.Velocidade.MetrosPorSeg = 0;
      Esteira.Velocidade.PixelsPorSeg = 0;
      Esteira.InicioParada = TmpAtSim[0];
      sensor[IndMaq].pallet     = (short int) Indice;
      sensor[IndMaq].FlagPallet = 1;
      sensor[IndMaq].FlagPeca   = 0;
      if(IndMaq == 4){
        Esteira.Velocidade.MetrosPorSeg = 0;
```

```
        Esteira.Velocidade.PixelsPorSeg = 0;
        Esteira.InicioParada            = TmpAtSim[0];
        sensor[IndMaq+1].pallet         = (short int) Indice;
        sensor[IndMaq+1].FlagPallet     = 1;
        sensor[IndMaq+1].FlagPeca       = 0;
      }
    }
  } // if TemMaq
}
void TForm1::VerificaInOut(int Indice){
  int Cont1, IndPeca, TipoPeca, Anterior;
  bool TemIn, TemOut;
  TemIn = false;
  TemOut = false;
  Cont1 = Pallet[Indice].PosicaoAnterior.Pixels;
  while((Cont1 != (Pallet[Indice].PosicaoAtual.Pixels + 1))&&(!TemIn)&&(!TemOut)){
    if(InProcesso.Posicao.Pixels == Cont1)
      TemIn  = true;
    if(OutProcesso.Posicao.Pixels == Cont1)
      TemOut = true;
    Cont1++;
    if(Cont1 == Esteira.Comprimento.Pixels)
      Cont1 = 0;
  }  // while
  if (TemIn){
    Anterior = InProcesso.PalletAnterior;
    if (Anterior < 0)
      InProcesso.PalletAnterior = Indice;
    else{
      if (Anterior == 0)
        Anterior = TotalPallets;
      if (Indice == (Anterior - 1))
        InProcesso.PalletAnterior = Indice;
      else
        TemIn = false;
    } // else
  } // if TemIn
  if((TemIn) && (InProcesso.Contador < TotalPecas[0])){
    InProcesso.Contador++;
    IndPeca  = IndicePecaIn(InProcesso.Contador);
    TipoPeca = TipoPecaIn(InProcesso.Contador);
    Pallet[Indice].Peca.TipoPeca   = (short int)TipoPeca;
    Pallet[Indice].Peca.IndicePeca = (short int)IndPeca;
    Peca[TipoPeca][IndPeca].TempoPeca[0]    = (float)TmpAtSim[0]/1000;
    Peca[TipoPeca][IndPeca].PosicaoAtual[0] = 1; // Peca estah no Pallet
    Peca[TipoPeca][IndPeca].PosicaoAtual[1] = (short int)Indice;
  }
  if (TemOut){
    Anterior = OutProcesso.PalletAnterior;
    if (Anterior < 0)
      OutProcesso.PalletAnterior = Indice;
    else{
      if (Anterior == 0)
        Anterior = TotalPallets;
      if (Indice == (Anterior - 1))
        OutProcesso.PalletAnterior = Indice;
      else
        TemOut = false;
    } // else
  } // if TemOut
  if(TemOut && (Pallet[Indice].Peca.TipoPeca > 0) && (OutProcesso.Contador <= TotalPecas[0])){
    TipoPeca = Pallet[Indice].Peca.TipoPeca;
    IndPeca = Pallet[Indice].Peca.IndicePeca;
    if(Peca[TipoPeca][IndPeca].SairProcesso || Peca[TipoPeca][IndPeca]. ProcessosExecutados()){
      OutProcesso.Contador++;
      Pallet[Indice].Peca.TipoPeca            = 0;
      Pallet[Indice].Peca.IndicePeca          = 0;
      Peca[TipoPeca][IndPeca].TempoPeca[1]    = (float)TmpAtSim[0] / 1000;
      Peca[TipoPeca][IndPeca].PosicaoAtual[0] = 0; // Peca estah fora do processo
      Peca[TipoPeca][IndPeca].PosicaoAtual[1] = 0;
      if(!(OutProcesso.Contador < TotalPecas[0]))
        FimDaSimulacao = true;
    } // if Peca || Peca
  } // it TemOut
}
//---------------------------------------------------------------------------
int
TForm1::TipoPecaIn(int Indice){
  int Saida, TotalAux;
  TotalAux = TotalPecas[1];
  if(Indice <= TotalAux){
    Saida = 1;
  }
  else{
    TotalAux = TotalAux + TotalPecas[2];
    if(Indice <= TotalAux){
      Saida = 2;
    }
    else{
      TotalAux = TotalAux + TotalPecas[3];
      if(Indice <= TotalAux){
        Saida = 3;
      }
      else{
        Saida = 4;
      }
    }
  }
  return(Saida);
}
//---------------------------------------------------------------------------
int
TForm1::IndicePecaIn(int Indice){
```

```
    int Saida, TotalAux;
    TotalAux = 0;
    if(Indice <= (TotalAux + TotalPecas[1])){
      Saida = Indice - TotalAux;
    }
    else{
      TotalAux = TotalAux + TotalPecas[1];
      if(Indice <= (TotalAux + TotalPecas[2])){
        Saida = Indice - TotalAux;
      }
      else{
        TotalAux = TotalAux + TotalPecas[2];
        if(Indice <= (TotalAux + TotalPecas[3])){
          Saida = Indice - TotalAux;
        }
        else{
          TotalAux = TotalAux + TotalPecas[3];
          Saida = Indice - TotalAux;
        }
      }
    }
    return(Saida);
}
//---------------------------------------------------------------------------
void TForm1::VerificaParadaEsteira(void){
  if ((Esteira.Velocidade.MetrosPorSeg == 0) && ((TmpAtSim[0] - Esteira.InicioParada) >= TempoIOMaq)){
    Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
    Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg *
      Esteira.Comprimento.Pixels / Esteira.Comprimento.Metros;
  }
}
//---------------------------------------------------------------------------
void TForm1::VerificaRobo(int i){
  short int tipopeca, indicepeca;
  if(robos[i].id <4){
    if(Maquina[i].FilaEntradaCheia() == 1){
      // este IF verifica a seguinte condicao em que a esteira nao para:
      //   - se o buffer de entrada estah cheio
      Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
      Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
      sensor[i].FlagPeca = 0;
      sensor[i].FlagPallet = 0;
      sensor[i].pallet = -1;
    }
    switch(robos[i].pos){
      case POS_E:
        switch(robos[i].status){
          case CAR_IN:  // SubRotina A
            robos[i].sentido = EM;
          break;
          case CAR_OUT: // SubRotina B
            if(sensor[i].FlagPallet == 1){
              if(sensor[i].FlagPeca == 1){
                if(Maquina[i].FilaEntradaCheia() == 1)
                  robos[i].sentido = PR;
                else{
                  tipopeca   = robos[i].TipoPeca;
                  indicepeca = robos[i].IndicePeca;
                  robos[i].TipoPeca   = Pallet[sensor[i].pallet].Peca.TipoPeca ;
                  robos[i].IndicePeca = Pallet[sensor[i].pallet].Peca.IndicePeca ;
                  Pallet[sensor[i].pallet].Peca.TipoPeca   = tipopeca;
                  Pallet[sensor[i].pallet].Peca.IndicePeca = indicepeca;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
                  Peca[tipopeca][indicepeca].PosicaoAtual[0] = 1;
                  Peca[tipopeca][indicepeca].PosicaoAtual[1] = sensor[i].pallet;
                  robos[i].status = CAR_IN;
                  robos[i].sentido = EM;
                  // verifica o robo da Mitutoyo
                  if(robos[i].id == 1){
                    if(robos[i].TipoPeca == 1)
                      robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 2)
                      robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 3)
                      robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 4)
                      robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                  } // if(robos[i].id == 1)
                  // verifica o robo do Centro de Usinagem e do Torno CNC
                  if(robos[i].id == 2 || robos[i].id == 3){
                    if(robos[i].TipoPeca == 1)
                      robos[i].AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
                    if(robos[i].TipoPeca == 2)
                      robos[i].AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
                    if(robos[i].TipoPeca == 3)
                      robos[i].AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
                    if(robos[i].TipoPeca == 4)
                      robos[i].AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
                  } // if(robos[i].id == 2 || robos[i].id == 3)
                  Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                  Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                } // else if(Maquina[i].FilaEntradaCheia == 1)
              } // if(sensor[i].FlagPeca == 1)
              else{
                Pallet[sensor[i].pallet].Peca.TipoPeca = robos[i].TipoPeca;
                Pallet[sensor[i].pallet].Peca.IndicePeca = robos[i].IndicePeca;
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 2;
```

```
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = sensor[i].pallet;
                robos[i].TipoPeca = 0;
                robos[i].IndicePeca = 0;
                robos[i].status = LIVRE;
                if(robos[i].id == 1)
                  robos[i].AssociaImagem("ROBO_RIGHT","ROBO_RIGHT_MASK");
                else
                  robos[i].AssociaImagem("ROBO_LEFT","ROBO_LEFT_MASK");
                Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
                if(Maquina[i].FilaSaidaVazia() == 1)
                  robos[i].sentido = PR;
                else
                  robos[i].sentido = EM;
              } // else if(sensor[i].FlagPeca == 1)
            } // if(sensor[i].FlagPallet == 1)
            else
              robos[i].sentido = PR;
          break;
          case LIVRE:   // SubRotina C
            if(sensor[i].FlagPallet == 1){
              if(sensor[i].FlagPeca == 1){
                if(Maquina[i].FilaEntradaCheia() == 1)
                  robos[i].sentido = PR;
                else{
                  robos[i].TipoPeca = Pallet[sensor[i].pallet].Peca.TipoPeca ;
                  robos[i].IndicePeca = Pallet[sensor[i].pallet].Peca.IndicePeca ;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
                  Pallet[sensor[i].pallet].Peca.TipoPeca = 0;
                  Pallet[sensor[i].pallet].Peca.IndicePeca = 0;
                  robos[i].status = CAR_IN;
                  robos[i].sentido = EM;
                  // verifica o robo da Mitutoyo
                  if(robos[i].id == 1){
                    if(robos[i].TipoPeca == 1)
                      robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 2)
                      robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 3)
                      robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 4)
                      robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                  }
                  // verifica o robo do Centro de Usinagem e do Torno CNC
                  if(robos[i].id == 2 || robos[i].id == 3){
                    if(robos[i].TipoPeca == 1)
                      robos[i].AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
                    if(robos[i].TipoPeca == 2)
                      robos[i].AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
                    if(robos[i].TipoPeca == 3)
                      robos[i].AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
                    if(robos[i].TipoPeca == 4)
                      robos[i].AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
                  }
                  Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                  Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                } // else if(Maquina[i].FilaEntradaCheia == 1)
              } // if(sensor[i].FlagPeca == 1)
              else{
                if(Maquina[i].FilaSaidaVazia() == 1)
                  robos[i].sentido = PR;
                else
                  robos[i].sentido = EM;
              } // else if(sensor[i].FlagPeca == 1)
            } // if(sensor[i].FlagPallet == 1)
            else{
              if(Maquina[i].FilaSaidaVazia() == 1)
                robos[i].sentido = PR;
              else
                robos[i].sentido = EM;
            }// else if(sensor[i].FlagPallet == 1)
          break;
        }
      break;
      case POS_M:
        switch(robos[i].status){
          case CAR_IN:  // SubRotina D
            if(Maquina[i].FilaEntradaCheia() == 1)
              robos[i].sentido = PR;
            else{
              if(Maquina[i].FilaSaidaVazia() == 1){
                Maquina[i].ColocarnaFilaEntrada(robos[i].TipoPeca, robos[i].IndicePeca);
                robos[i].TipoPeca = 0;
                robos[i].IndicePeca = 0;
                robos[i].status = LIVRE;
                robos[i].sentido = ME;
                if(robos[i].id == 1 || robos[i].id == 2)
                  robos[i].AssociaImagem("ROBO_DOWN","ROBO_DOWN_MASK");
                else
                  robos[i].AssociaImagem("ROBO_UP","ROBO_UP_MASK");
              } // if(Maquina[i].FilaSaidaVazia == 1)
              else{
                tipopeca = robos[i].TipoPeca;
                indicepeca = robos[i].IndicePeca;
```

```
            Maquina[i].TirardaFilaSaida();
            Maquina[i].ColocarnaFilaEntrada(tipopeca, indicepeca);
            robos[i].status = CAR_OUT;
            robos[i].sentido = ME;
            if(robos[i].id == 1){
              if(robos[i].TipoPeca == 1)
                robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
              if(robos[i].TipoPeca == 2)
                robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
              if(robos[i].TipoPeca == 3)
                robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
              if(robos[i].TipoPeca == 4)
                robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
            } // if(robos[i].id == 1)
            if(robos[i].id == 2 || robos[i].id == 3){
              if(robos[i].TipoPeca == 1)
                robos[i].AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
              if(robos[i].TipoPeca == 2)
                robos[i].AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
              if(robos[i].TipoPeca == 3)
                robos[i].AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
              if(robos[i].TipoPeca == 4)
                robos[i].AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
            } // if(robos[i].id == 2 || robos[i].id == 3)
            Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
            Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
          } // else if(Maquina[i].FilaSaidaVazia == 1)
        } // else if(Maquina[i].FilaEntradaCheia() == 1)
      break;
      case CAR_OUT: // SubRotina E
        robos[i].sentido = ME;
      break;
      case LIVRE:   // SubRotina F
        if(Maquina[i].FilaSaidaVazia() == 1)
          robos[i].sentido = ME;
        else{
          Maquina[i].TirardaFilaSaida();    // maquina e houver peca na fila de saida
          robos[i].status = CAR_OUT;
          robos[i].sentido = ME;
          if(robos[i].id == 1){
            if(robos[i].TipoPeca == 1)
              robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
            if(robos[i].TipoPeca == 2)
              robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
            if(robos[i].TipoPeca == 3)
              robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
            if(robos[i].TipoPeca == 4)
              robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
          }
          if(robos[i].id == 2 || robos[i].id == 3){
            if(robos[i].TipoPeca == 1)
              robos[i].AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
            if(robos[i].TipoPeca == 2)
              robos[i].AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
            if(robos[i].TipoPeca == 3)
              robos[i].AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
            if(robos[i].TipoPeca == 4)
              robos[i].AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
          }
          robos[i].sentido = ME;
          Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
          Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = (short int) i;
        } // else if(Maquina[i].FilaSaidaVazia() == 1)
      break;
    }
  break;
  case POS_P:
    switch(robos[i].status){
      case CAR_IN:  // SubRotina G
        robos[i].sentido = EM;
      break;
      case CAR_OUT: // SubRotina H
        robos[i].sentido = ME;
      break;
      case LIVRE:   // SubRotina I
        if((sensor[i].FlagPallet == 1)&&(sensor[i].FlagPeca == 1))
          robos[i].sentido=ME;
        else{
          if(Maquina[i].FilaSaidaVazia() == 1)
            robos[i].sentido = ME;
          else
            robos[i].sentido = EM;
        } // else if((sensor[i].FlagPallet == 1)&&(sensor[i].FlagPeca == 1))
      break;
    }
  break;
 } // switch(robos[i].pos)
}
//---------------------------------------------------------------------------
if(robos[i].id == 4){
  switch(robos[i].pos){
    case POS_E:
      switch(robos[i].status){
        case CAR_INF:  // SubRotina A
          robos[i].sentido = EF;
        break;
        case CAR_INT:  // SubRotina A
          robos[i].sentido = ET;
        break;
        case CAR_INFT:  // SubRotina A
          robos[i].sentido = FT;
        break;
        case CAR_INTF:  // SubRotina A
```

```
            robos[i].sentido = TF;
          break;
          case CAR_OUTF: // SubRotina B
            if(sensor[i].FlagPallet == 1){
              if(sensor[i].FlagPeca == 1){
                if(Maquina[i].FilaEntradaCheia() == 1)
                  robos[i].sentido = PR;
                else{
                  tipopeca   = robos[i].TipoPeca;
                  indicepeca = robos[i].IndicePeca;
                  robos[i].TipoPeca   = Pallet[sensor[i].pallet].Peca.TipoPeca ;
                  robos[i].IndicePeca = Pallet[sensor[i].pallet].Peca.IndicePeca ;
                  Pallet[sensor[i].pallet].Peca.TipoPeca   = tipopeca;
                  Pallet[sensor[i].pallet].Peca.IndicePeca = indicepeca;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
                  Peca[tipopeca][indicepeca].PosicaoAtual[0] = 1;
                  Peca[tipopeca][indicepeca].PosicaoAtual[1] = sensor[i].pallet;
                  robos[i].status = CAR_INF;
                  robos[i].sentido = EF;
                  if(robos[i].TipoPeca == 1)
                    robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                  if(robos[i].TipoPeca == 2)
                    robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
                  if(robos[i].TipoPeca == 3)
                    robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                  if(robos[i].TipoPeca == 4)
                     robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                  Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                  Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                } // else if(Maquina[i].FilaEntradaCheia == 1)
              } // if(sensor[i].FlagPeca == 1)
              else{
                Pallet[sensor[i].pallet].Peca.TipoPeca = robos[i].TipoPeca;
                Pallet[sensor[i].pallet].Peca.IndicePeca = robos[i].IndicePeca;
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] =  sensor[i].pallet;
                robos[i].TipoPeca = 0;
                robos[i].IndicePeca = 0;
                robos[i].status = LIVRE;
                robos[i].AssociaImagem("ROBO_RIGHT","ROBO_RIGHT_MASK");
                Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                if(Maquina[i].FilaSaidaVazia() == 1)
                  if(Maquina[i+1].FilaSaidaVazia() == 1)
                    robos[i].sentido = PR;
                  else{
                    sensor[i].FlagPeca = 0;
                    sensor[i].FlagPallet = 0;
                    sensor[i].pallet = -1;
                    sensor[i+1].FlagPeca = 0;
                    sensor[i+1].FlagPallet = 0;
                    sensor[i+1].pallet = -1;
                    robos[i].sentido = ET;
                  }
                else{
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                  robos[i].sentido = EF;
                }
              } // else if(sensor[i].FlagPeca == 1)
            } // if(sensor[i].FlagPallet == 1)
            else
              robos[i].sentido = PR;
            if(sensor[i+1].FlagPallet == 1){
              if(sensor[i+1].FlagPeca == 1){
                if(Maquina[i+1].FilaEntradaCheia() == 1)
                  robos[i].sentido = PR;
                else{
                  tipopeca   = robos[i].TipoPeca;
                  indicepeca = robos[i].IndicePeca;
                  robos[i].TipoPeca   = Pallet[sensor[i+1].pallet].Peca.TipoPeca ;
                  robos[i].IndicePeca = Pallet[sensor[i+1].pallet].Peca.IndicePeca ;
                  Pallet[sensor[i+1].pallet].Peca.TipoPeca   = tipopeca;
                  Pallet[sensor[i+1].pallet].Peca.IndicePeca = indicepeca;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
                  Peca[tipopeca][indicepeca].PosicaoAtual[0] = 1;
                  Peca[tipopeca][indicepeca].PosicaoAtual[1] = sensor[i+1].pallet;
                  robos[i].status = CAR_INT;
                  robos[i].sentido = ET;
                  if(robos[i].TipoPeca == 1)
                    robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                  if(robos[i].TipoPeca == 2)
                    robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
```

```
                if(robos[i].TipoPeca == 3)
                  robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                if(robos[i].TipoPeca == 4)
                  robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
              } // else if(Maquina[i+1].FilaEntradaCheia == 1)
            } // if(sensor[i+1].FlagPeca == 1)
            else{
              Pallet[sensor[i+1].pallet].Peca.TipoPeca = robos[i].TipoPeca;
              Pallet[sensor[i+1].pallet].Peca.IndicePeca = robos[i].IndicePeca;
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 2;
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = sensor[i+1].pallet;
              robos[i].TipoPeca = 0;
              robos[i].IndicePeca = 0;
              robos[i].status = LIVRE;
              robos[i].AssociaImagem("ROBO_RIGHT","ROBO_RIGHT_MASK");
              Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
              Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
              sensor[i+1].FlagPeca = 0;
              sensor[i+1].FlagPallet = 0;
              sensor[i+1].pallet = -1;
              sensor[i].FlagPeca = 0;
              sensor[i].FlagPallet = 0;
              sensor[i].pallet = -1;
              if(Maquina[i+1].FilaSaidaVazia() == 1)
                if(Maquina[i].FilaSaidaVazia() == 1)
                  robos[i].sentido = PR;
                else{
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                  robos[i].sentido = EF;
                }
              else{
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                robos[i].sentido = ET;
              }
            } // else if(sensor[i+1].FlagPeca == 1)
          } // if(sensor[i+1].FlagPallet == 1)
          else
            robos[i].sentido = PR;
        break;
        case CAR_OUTT: // SubRotina B
          if(sensor[i+1].FlagPallet == 1){
            if(sensor[i+1].FlagPeca == 1){
              if(Maquina[i+1].FilaEntradaCheia() == 1)
                robos[i].sentido = PR;
              else{
                tipopeca   = robos[i].TipoPeca;
                indicepeca = robos[i].IndicePeca;
                robos[i].TipoPeca   = Pallet[sensor[i+1].pallet].Peca.TipoPeca ;
                robos[i].IndicePeca = Pallet[sensor[i+1].pallet].Peca.IndicePeca ;
                Pallet[sensor[i+1].pallet].Peca.TipoPeca   = tipopeca;
                Pallet[sensor[i+1].pallet].Peca.IndicePeca = indicepeca;
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
                Peca[tipopeca][indicepeca].PosicaoAtual[0] = 1;
                Peca[tipopeca][indicepeca].PosicaoAtual[1] = sensor[i+1].pallet;
                robos[i].status = CAR_INT;
                robos[i].sentido = ET;
                if(robos[i].TipoPeca == 1)
                  robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                if(robos[i].TipoPeca == 2)
                  robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
                if(robos[i].TipoPeca == 3)
                  robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                if(robos[i].TipoPeca == 4)
                  robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
              } // else if(Maquina[i+1].FilaEntradaCheia == 1)
            } // if(sensor[i+1].FlagPeca == 1)
            else{
              Pallet[sensor[i+1].pallet].Peca.TipoPeca = robos[i].TipoPeca;
              Pallet[sensor[i+1].pallet].Peca.IndicePeca = robos[i].IndicePeca;
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 2;
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = sensor[i+1].pallet;
              robos[i].TipoPeca = 0;
              robos[i].IndicePeca = 0;
```

```
                robos[i].status = LIVRE;
                robos[i].AssociaImagem("ROBO_RIGHT","ROBO_RIGHT_MASK");
                Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
 Esteira.Comprimento.Metros;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
                if(Maquina[i+1].FilaSaidaVazia() == 1)
                  if(Maquina[i].FilaSaidaVazia() == 1)
                    robos[i].sentido = PR;
                  else{
                    sensor[i].FlagPeca = 0;
                    sensor[i].FlagPallet = 0;
                    sensor[i].pallet = -1;
                    sensor[i+1].FlagPeca = 0;
                    sensor[i+1].FlagPallet = 0;
                    sensor[i+1].pallet = -1;
                    robos[i].sentido = EF;
                  }
                else{
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                  robos[i].sentido = ET;
                }
              } // else if(sensor[i+1].FlagPeca == 1)
            } // if(sensor[i+1].FlagPallet == 1)
            else
              robos[i].sentido = PR;
              if(sensor[i].FlagPallet == 1){
                if(sensor[i].FlagPeca == 1){
                  if(Maquina[i].FilaEntradaCheia() == 1)
                    robos[i].sentido = PR;
                  else{
                    tipopeca    = robos[i].TipoPeca;
                    indicepeca = robos[i].IndicePeca;
                    robos[i].TipoPeca    = Pallet[sensor[i].pallet].Peca.TipoPeca ;
                    robos[i].IndicePeca = Pallet[sensor[i].pallet].Peca.IndicePeca ;
                    Pallet[sensor[i].pallet].Peca.TipoPeca    = tipopeca;
                    Pallet[sensor[i].pallet].Peca.IndicePeca = indicepeca;
                    Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
                    Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = (short int) i;
                    Peca[tipopeca][indicepeca].PosicaoAtual[0] = 1;
                    Peca[tipopeca][indicepeca].PosicaoAtual[1] = sensor[i].pallet;
                    robos[i].status = CAR_INF;
                    robos[i].sentido = EF;
                    if(robos[i].TipoPeca == 1)
                      robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 2)
                      robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 3)
                      robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                    if(robos[i].TipoPeca == 4)
                      robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                    Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                    Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                    sensor[i].FlagPeca = 0;
                    sensor[i].FlagPallet = 0;
                    sensor[i].pallet = -1;
                    sensor[i+1].FlagPeca = 0;
                    sensor[i+1].FlagPallet = 0;
                    sensor[i+1].pallet = -1;
                  } // else if(Maquina[i].FilaEntradaCheia == 1)
                } // if(sensor[i].FlagPeca == 1)
                else{
                  Pallet[sensor[i].pallet].Peca.TipoPeca = robos[i].TipoPeca;
                  Pallet[sensor[i].pallet].Peca.IndicePeca = robos[i].IndicePeca;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 2;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = sensor[i].pallet;
                  robos[i].TipoPeca = 0;
                  robos[i].IndicePeca = 0;
                  robos[i].status = LIVRE;
                  robos[i].AssociaImagem("ROBO_RIGHT","ROBO_RIGHT_MASK");
                  Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                  Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                  if(Maquina[i].FilaSaidaVazia() == 1)
                    if(Maquina[i+1].FilaSaidaVazia() == 1)
                      robos[i].sentido = PR;
                    else{
                      sensor[i].FlagPeca = 0;
                      sensor[i].FlagPallet = 0;
                      sensor[i].pallet = -1;
                      sensor[i+1].FlagPeca = 0;
                      sensor[i+1].FlagPallet = 0;
                      sensor[i+1].pallet = -1;
                      robos[i].sentido = ET;
                    }
                  else{
```

```
                    sensor[i].FlagPeca = 0;
                    sensor[i].FlagPallet = 0;
                    sensor[i].pallet = -1;
                    sensor[i+1].FlagPeca = 0;
                    sensor[i+1].FlagPallet = 0;
                    sensor[i+1].pallet = -1;
                    robos[i].sentido = EF;
                  }
                } // else if(sensor[i].FlagPeca == 1)
              } // if(sensor[i].FlagPallet == 1)
            else
              robos[i].sentido = PR;
          break;
          case LIVRE:   // SubRotina C
            if(sensor[i].FlagPallet == 1){
              if(sensor[i].FlagPeca == 1){
                if(Maquina[i].FilaEntradaCheia() == 1)
                  robos[i].sentido = PR;
                else{
                  robos[i].TipoPeca = Pallet[sensor[i].pallet].Peca.TipoPeca ;
                  robos[i].IndicePeca = Pallet[sensor[i].pallet].Peca.IndicePeca ;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
                  Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
                  Pallet[sensor[i].pallet].Peca.TipoPeca = 0;
                  Pallet[sensor[i].pallet].Peca.IndicePeca = 0;
                  robos[i].status = CAR_INF;
                  robos[i].sentido = EF;
                  if(robos[i].TipoPeca == 1)
                    robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                  if(robos[i].TipoPeca == 2)
                    robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
                  if(robos[i].TipoPeca == 3)
                    robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                  if(robos[i].TipoPeca == 4)
                    robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                  Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                  Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                } // else if(Maquina[i].FilaEntradaCheia == 1)
              } // if(sensor[i].FlagPeca == 1)
              else{
                if(Maquina[i].FilaSaidaVazia() == 1)
                  if(Maquina[i+1].FilaSaidaVazia() == 1)
                    robos[i].sentido = PR;
                  else{
                    sensor[i].FlagPeca = 0;
                    sensor[i].FlagPallet = 0;
                    sensor[i].pallet = -1;
                    sensor[i+1].FlagPeca = 0;
                    sensor[i+1].FlagPallet = 0;
                    sensor[i+1].pallet = -1;
                    robos[i].sentido = ET;
                  }
                else{
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                  robos[i].sentido = EF;
                }
              } // else if(sensor[i].FlagPeca == 1)
            } // if(sensor[i].FlagPallet == 1)
            else
              if(sensor[i+1].FlagPeca == 0){
                if(Maquina[i].FilaSaidaVazia() == 1)
                  if(Maquina[i+1].FilaSaidaVazia() == 1)
                    robos[i].sentido = PR;
                  else{
                    sensor[i].FlagPeca = 0;
                    sensor[i].FlagPallet = 0;
                    sensor[i].pallet = -1;
                    sensor[i+1].FlagPeca = 0;
                    sensor[i+1].FlagPallet = 0;
                    sensor[i+1].pallet = -1;
                    robos[i].sentido = ET;
                  }
                else{
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                  robos[i].sentido = EF;
                }
              }// else if(sensor[i].FlagPallet == 1)
              if(sensor[i+1].FlagPallet == 1){
                if(sensor[i+1].FlagPeca == 1){
                  if(Maquina[i+1].FilaEntradaCheia() == 1)
                    robos[i].sentido = PR;
                  else{
                    robos[i].TipoPeca = Pallet[sensor[i+1].pallet].Peca.TipoPeca ;
                    robos[i].IndicePeca = Pallet[sensor[i+1].pallet].Peca.IndicePeca ;
                    Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0]=1;
                    Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1]=(short int) i;
```

```
                Pallet[sensor[i+1].pallet].Peca.TipoPeca = 0;
                Pallet[sensor[i+1].pallet].Peca.IndicePeca = 0;
                robos[i].status = CAR_INT;
                robos[i].sentido = ET;
                if(robos[i].TipoPeca == 1)
                  robos[i].AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
                if(robos[i].TipoPeca == 2)
                  robos[i].AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
                if(robos[i].TipoPeca == 3)
                  robos[i].AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
                if(robos[i].TipoPeca == 4)
                  robos[i].AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
                Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
                Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg * Esteira.Comprimento.Pixels /
Esteira.Comprimento.Metros;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
              } // else if(Maquina[i+1].FilaEntradaCheia == 1)
            } // if(sensor[i+1].FlagPeca == 1)
            else{
              if(Maquina[i+1].FilaSaidaVazia() == 1)
                if(Maquina[i].FilaSaidaVazia() == 1)
                  robos[i].sentido = PR;
                else{
                  sensor[i].FlagPeca = 0;
                  sensor[i].FlagPallet = 0;
                  sensor[i].pallet = -1;
                  sensor[i+1].FlagPeca = 0;
                  sensor[i+1].FlagPallet = 0;
                  sensor[i+1].pallet = -1;
                  robos[i].sentido = EF;
                }
              else{
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                robos[i].sentido = ET;
              }
            } // else if(sensor[i+1].FlagPeca == 1)
          } // if(sensor[i+1].FlagPallet == 1)
          else{
            if(Maquina[i+1].FilaSaidaVazia() == 1)
              if(Maquina[i].FilaSaidaVazia() == 1)
                robos[i].sentido = PR;
              else{
                sensor[i].FlagPeca = 0;
                sensor[i].FlagPallet = 0;
                sensor[i].pallet = -1;
                sensor[i+1].FlagPeca = 0;
                sensor[i+1].FlagPallet = 0;
                sensor[i+1].pallet = -1;
                robos[i].sentido = EF;
              }
            else{
              sensor[i].FlagPeca = 0;
              sensor[i].FlagPallet = 0;
              sensor[i].pallet = -1;
              sensor[i+1].FlagPeca = 0;
              sensor[i+1].FlagPallet = 0;
              sensor[i+1].pallet = -1;
              robos[i].sentido = ET;
            }
          }// else if(sensor[i+1].FlagPallet == 1)
        break;
      }
    break;
    case POS_F:
      switch(robos[i].status){
        case CAR_INF:  // SubRotina D
          if(Maquina[i].FilaEntradaCheia() == 1)
            robos[i].sentido = PR;
          else{
            if(Maquina[i].FilaSaidaVazia() == 1){
              Maquina[i].ColocarnaFilaEntrada(robos[i].TipoPeca, robos[i].IndicePeca);
              robos[i].TipoPeca = 0;
              robos[i].IndicePeca = 0;
              robos[i].status = LIVRE;
              robos[i].sentido = FE;
              robos[i].AssociaImagem("ROBO_LEFT","ROBO_LEFT_MASK");
            } // if(Maquina[i].FilaSaidaVazia == 1)
            else{
              tipopeca = robos[i].TipoPeca;
              indicepeca = robos[i].IndicePeca;
              Maquina[i].TirardaFilaSaida();
              Maquina[i].ColocarnaFilaEntrada(tipopeca, indicepeca);
              if(Peca[robos[i].TipoPeca][robos[i].IndicePeca].Processos[Peca[robos[i]
.TipoPeca][robos[i].IndicePeca].LinhaProcesso].Sequencia == 5){
                robos[i].status = CAR_INFT;
                robos[i].sentido = FT;
              }
              else{
                robos[i].status = CAR_OUTF;
                robos[i].sentido = FE;
              }
              if(robos[i].TipoPeca == 1)
                robos[i].AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
```

```
                if(robos[i].TipoPeca == 2)
                  robos[i].AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
                if(robos[i].TipoPeca == 3)
                  robos[i].AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
                if(robos[i].TipoPeca == 4)
                  robos[i].AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] =(short int) i;
              } // else if(Maquina[i].FilaSaidaVazia == 1)
            | // else if(Maquina[i].FilaEntradaCheia() == 1)
          break;
          case CAR_INTF:  // SubRotina D
            if(Maquina[i].FilaEntradaCheia() == 1)
              robos[i].sentido = PR;
            else{
              if(Maquina[i].FilaSaidaVazia() == 1){
                Maquina[i].ColocarnaFilaEntrada(robos[i].TipoPeca, robos[i].IndicePeca);
                robos[i].TipoPeca = 0;
                robos[i].IndicePeca = 0;
                robos[i].status = LIVRE;
                robos[i].sentido = FE;
                robos[i].AssociaImagem("ROBO_LEFT","ROBO_LEFT_MASK");
              } // if(Maquina[i].FilaSaidaVazia == 1)
              else{
                tipopeca = robos[i].TipoPeca;
                indicepeca = robos[i].IndicePeca;
                Maquina[i].TirardaFilaSaida();
                Maquina[i].ColocarnaFilaEntrada(tipopeca, indicepeca);
                if(Peca[robos[i].TipoPeca][robos[i].IndicePeca].Processos[Peca[robos[i].
TipoPeca][robos[i].IndicePeca].LinhaProcesso].Sequencia == 5){
                  robos[i].status = CAR_INFT;
                  robos[i].sentido = FT;
                }
                else{
                  robos[i].status = CAR_OUTF;
                  robos[i].sentido = FE;
                }
                if(robos[i].TipoPeca == 1)
                  robos[i].AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
                if(robos[i].TipoPeca == 2)
                  robos[i].AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
                if(robos[i].TipoPeca == 3)
                  robos[i].AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
                if(robos[i].TipoPeca == 4)
                  robos[i].AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
                Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] =(short int) i;
              | // else if(Maquina[i].FilaSaidaVazia == 1)
            } // else if(Maquina[i].FilaEntradaCheia() == 1)
          break;
          case CAR_OUTF: // SubRotina E
            robos[i].sentido = FE;
          break;
          case LIVRE:   // SubRotina F
            if(Maquina[i].FilaSaidaVazia() == 1)
              robos[i].sentido = FE;
            else{
              Maquina[i].TirardaFilaSaida();    // maquina e houver peca na fila de saida
              if(Peca[robos[i].TipoPeca][robos[i].IndicePeca].Processos[Peca[robos[i].
[TipoPeca][robos[i].IndicePeca].LinhaProcesso].Sequencia == 5){
                robos[i].status = CAR_INFT;
                robos[i].sentido = FT;
              }
              else{
                robos[i].status = CAR_OUTF;
                robos[i].sentido = FE;
              }
              if(robos[i].TipoPeca == 1)
                robos[i].AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
              if(robos[i].TipoPeca == 2)
                robos[i].AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
              if(robos[i].TipoPeca == 3)
                robos[i].AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
              if(robos[i].TipoPeca == 4)
                robos[i].AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = (short int)i;
            } // else if(Maquina[i].FilaSaidaVazia() == 1)
          break;
        }
      break;
      case POS_T:
        switch(robos[i].status){
          case CAR_INT:  // SubRotina D
            if(Maquina[i+1].FilaEntradaCheia() == 1)
              robos[i].sentido = PR;
            else{
              if(Maquina[i+1].FilaSaidaVazia() == 1){
                Maquina[i+1].ColocarnaFilaEntrada(robos[i].TipoPeca, robos[i].IndicePeca);
                robos[i].TipoPeca = 0;
                robos[i].IndicePeca = 0;
                robos[i].status = LIVRE;
                robos[i].sentido = TE;
                robos[i].AssociaImagem("ROBO_DOWN","ROBO_DOWN_MASK");
              } // if(Maquina[i].FilaSaidaVazia == 1)
              else{
                tipopeca = robos[i].TipoPeca;
                indicepeca = robos[i].IndicePeca;
                Maquina[i+1].TirardaFilaSaida();
                Maquina[i+1].ColocarnaFilaEntrada(tipopeca, indicepeca);
                if(Peca[robos[i].TipoPeca][robos[i].IndicePeca].Processos[Peca[robos[i].
TipoPeca][robos[i].IndicePeca].LinhaProcesso].Sequencia == 4){
                  robos[i].status = CAR_INTF;
```

```
                robos[i].sentido = TF;
              }
              else{
                robos[i].status = CAR_OUTT;
                robos[i].sentido = TE;
              }
              if(robos[i].TipoPeca == 1)
                robos[i].AssociaImagem("ROBO_DOWN_P1","ROBO_DOWN_P_MASK");
              if(robos[i].TipoPeca == 2)
                robos[i].AssociaImagem("ROBO_DOWN_P2","ROBO_DOWN_P_MASK");
              if(robos[i].TipoPeca == 3)
                robos[i].AssociaImagem("ROBO_DOWN_P3","ROBO_DOWN_P_MASK");
              if(robos[i].TipoPeca == 4)
                robos[i].AssociaImagem("ROBO_DOWN_P4","ROBO_DOWN_P_MASK");
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] =(short int) i;
            } // else if(Maquina[i+1].FilaSaidaVazia == 1)
          } // else if(Maquina[i+1].FilaEntradaCheia() == 1)
        break;
        case CAR_INFT:  // SubRotina D
          if(Maquina[i+1].FilaEntradaCheia() == 1)
            robos[i].sentido = PR;
          else{
            if(Maquina[i+1].FilaSaidaVazia() == 1){
              Maquina[i+1].ColocarnaFilaEntrada(robos[i].TipoPeca, robos[i].IndicePeca);
              robos[i].TipoPeca = 0;
              robos[i].IndicePeca = 0;
              robos[i].status = LIVRE;
              robos[i].sentido = TE;
              robos[i].AssociaImagem("ROBO_DOWN","ROBO_DOWN_MASK");
            } // if(Maquina[i].FilaSaidaVazia == 1)
            else{
              tipopeca = robos[i].TipoPeca;
              indicepeca = robos[i].IndicePeca;
              Maquina[i+1].TirardaFilaSaida();
              Maquina[i+1].ColocarnaFilaEntrada(tipopeca, indicepeca);
              if(Peca[robos[i].TipoPeca][robos[i].IndicePeca].Processos[Peca[robos[i].
TipoPeca][robos[i].IndicePeca].LinhaProcesso].Sequencia == 4){
                robos[i].status = CAR_INTF;
                robos[i].sentido = TF;
              }
              else{
                robos[i].status = CAR_OUTT;
                robos[i].sentido = TE;
              }
              if(robos[i].TipoPeca == 1)
                robos[i].AssociaImagem("ROBO_DOWN_P1","ROBO_DOWN_P_MASK");
              if(robos[i].TipoPeca == 2)
                robos[i].AssociaImagem("ROBO_DOWN_P2","ROBO_DOWN_P_MASK");
              if(robos[i].TipoPeca == 3)
                robos[i].AssociaImagem("ROBO_DOWN_P3","ROBO_DOWN_P_MASK");
              if(robos[i].TipoPeca == 4)
                robos[i].AssociaImagem("ROBO_DOWN_P4","ROBO_DOWN_P_MASK");
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
              Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] =(short int) i;
            } // else if(Maquina[i+1].FilaSaidaVazia == 1)
          } // else if(Maquina[i+1].FilaEntradaCheia() == 1)
        break;
        case CAR_OUTT: // SubRotina E
          robos[i].sentido = TE;
        break;
        case LIVRE:   // SubRotina F
          if(Maquina[i+1].FilaSaidaVazia() == 1)
            robos[i].sentido = TE;
          else{
            Maquina[i+1].TirardaFilaSaida();    // maquina e houver peca na fila de saida
            if(Peca[robos[i].TipoPeca][robos[i].IndicePeca].Processos[Peca[robos[i].
TipoPeca][robos[i].IndicePeca].LinhaProcesso].Sequencia == 4){
              robos[i].status = CAR_INTF;
              robos[i].sentido = TF;
            }
            else{
              robos[i].status = CAR_OUTT;
              robos[i].sentido = TE;
            }
            if(robos[i].TipoPeca == 1)
              robos[i].AssociaImagem("ROBO_DOWN_P1","ROBO_DOWN_P_MASK");
            if(robos[i].TipoPeca == 2)
              robos[i].AssociaImagem("ROBO_DOWN_P2","ROBO_DOWN_P_MASK");
            if(robos[i].TipoPeca == 3)
              robos[i].AssociaImagem("ROBO_DOWN_P3","ROBO_DOWN_P_MASK");
            if(robos[i].TipoPeca == 4)
              robos[i].AssociaImagem("ROBO_DOWN_P4","ROBO_DOWN_P_MASK");
            Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[0] = 1;
            Peca[robos[i].TipoPeca][robos[i].IndicePeca].PosicaoAtual[1] = (short int)i;
          } // else if(Maquina[i+1].FilaSaidaVazia() == 1)
        break;
      }
    break;
    case POS_P:
      switch(robos[i].status){
        case CAR_INF:  // SubRotina G
          robos[i].sentido = EF;
        break;
        case CAR_INT:  // SubRotina G
          robos[i].sentido = ET;
        break;
        case CAR_OUTF: // SubRotina H
          robos[i].sentido = FE;
        break;
        case CAR_OUTT: // SubRotina H
          robos[i].sentido = TE;
        break;
```

```
          case LIVRE:   // SubRotina I
            if((sensor[i].FlagPallet == 1)&&(sensor[i].FlagPeca == 1) || (sensor[i+1]. FlagPallet == 1) &&
(sensor[i+1].FlagPeca == 1))
              if(robos[i].y < 178)
                robos[i].sentido = FE;
              else
                robos[i].sentido = TE;
            else{
              if(Maquina[i].FilaSaidaVazia() == 1){
                if(Maquina[i+1].FilaSaidaVazia() == 1)
                  if(robos[i].y < 178)
                    robos[i].sentido = FE;
                  else
                    robos[i].sentido = TE;
                else
                  robos[i].sentido = ET;
              }
              else
                robos[i].sentido = EF;
            } // else if((sensor[i].FlagPallet == 1)&&(sensor[i].FlagPeca == 1))
          break;
        }
      break;
    } // switch(robos[i].pos)
  }
}
//---------------------------------------------------------------------------
void TForm1::MostraFilaInOut(int Indice){
}
```

EDITOR.H

```
//---------------------------------------------------------------------------
#ifndef editorH
#define editorH
//---------------------------------------------------------------------------
#endif
```

EDITOR.CPP

```
#include "MAINFORM.h"
AnsiString NomeArq;
const AnsiString DefaultFileName = "Sim1";
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <vcl\clipbrd.hpp>
#include <vcl\printers.hpp>
#pragma hdrstop
#include "editor.h"
//---------------------------------------------------------------------------
void __fastcall TForm1::abrir_arquivoClick(TObject *Sender){
  OpenDialog1->FileName = NomeArq;
  if(OpenDialog1->Execute()){
    editor->Lines->LoadFromFile(OpenDialog1->FileName);
    NomeArq = OpenDialog1->FileName;
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::SalvarComoClick(TObject *Sender){
  SaveDialog1->Title = "Salvar Como";
  if (SaveDialog1->Execute()){
    editor->Lines->SaveToFile(SaveDialog1->FileName);
    editor->Modified = false;
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::imprimirClick(TObject *Sender){
  if(PrintDialog1->Execute())
  editor->Print(NomeArq);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::SelecionarTudo1Click(TObject *Sender){
  editor->SelectAll();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Deletar1Click(TObject *Sender){
  editor->ClearSelection();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::copiarClick(TObject *Sender){
  editor->CopyToClipboard();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::recortarClick(TObject *Sender){
  editor->CutToClipboard();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Colar2Click(TObject *Sender){
  editor->PasteFromClipboard();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::fonte1Click(TObject *Sender){
  FontDialog1->Font->Assign( editor->SelAttributes);
  if (FontDialog1->Execute() )
    editor->SelAttributes->Assign( FontDialog1->Font);
}
//---------------------------------------------------------------------------
```

GRAFICO.H

```
//---------------------------------------------------------------------------
#ifndef graficoH
#define graficoH
//---------------------------------------------------------------------------
void DesenhaGrafico1(float a1, float a2, float a3, float a4);
void DesenhaGrafico2(float a1, float a2, float a3, float a4);
```

```
void DesenhaGrafico3(float a1, float a2, float a3, float a4);
void DesenhaGrafico4(float a1, float a2, float a3, float a4);
#endif
```

GRAFICO.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop
#include "MAINFORM.h"
#include "grafico.h"
//---------------------------------------------------------------------------
void DesenhaGrafico1(float a1, float a2, float a3, float a4){
  char *s;
  int bi, bs, bd, be, lb, pos1, pos2, pos3, pos4, i, alt1, alt2, alt3, alt4,
    h, j, esc;
  float es, max = 0;
  if(a1>a2) max = a1;
  else max = a2;
  if(a3>max) max = a3;
  if(a4>max) max = a4;

  bs = 30;
  bi = Form1->Image1->ClientWidth - 10;
  be = 40;
  bd = be+320;
  lb = 40;
  pos1 = be+30; pos2 = be+100; pos3 = be+160;
  pos4 = be+220;
  h = bi - bs-lb-5;

  alt1 = bi - (a1/max*h);
  alt2 = bi - (a2/max*h);
  alt3 = bi - (a3/max*h);
  alt4 = bi - (a4/max*h);

  j = h / 5;

  Form1->Image1->Canvas->Pen->Color = clLtGray;
  Form1->Image1->Canvas->Brush->Color = clLtGray;
  Form1->Image1->Canvas->Rectangle(0,0,Form1->Image1->ClientWidth, Form1->Image1-> ClientHeight);
  Form1->Image1->Canvas->Pen->Color = clBlack;
  Form1->Image1->Canvas->Brush->Color = clDkGray;
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(be, bi),
    Point(be+lb, bi-lb), Point(be+lb,bs), Point(be,bs+lb))));
  Form1->Image1->Canvas->Brush->Color = clWhite;
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bs),
    Point(be+lb,bi-lb), Point(bd,bi-lb), Point(bd,bs))));
  Form1->Image1->Canvas->Brush->Color = clLtGray;
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bi-ib),
    Point(be,bi), Point(bd-lb,bi), Point(bd,bi-lb))));
  Form1->Image1->Canvas->TextOut(be-35,bi-5,"0,00");
  i = bi - j;
  while(i-lb > bs){
    es = ((bi - i)*a1)/(bi-alt1);
    es = ceil(es);
    sprintf(s, "%.2lf",es);
    Form1->Image1->Canvas->TextOut(be-35,i-5,s);
    Form1->Image1->Canvas->MoveTo(be, i);
    Form1->Image1->Canvas->LineTo(be+lb, i-lb);
    Form1->Image1->Canvas->LineTo(bd, i-lb);
    i = i - j;
  }
  //Desenha primeira barra do gráfico.
  Form1->Image1->Canvas->Pen->Style = psClear;
  Form1->Image1->Canvas->Brush->Color = clBlue;
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,bi),
    Point(pos1+lb,bi), Point(pos1+lb,alt1), Point(pos1,alt1))));
  Form1->Image1->Canvas->Brush->Color = RGB(0, 0, 150);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1+lb,bi),
    Point(pos1+lb+lb,bi-lb), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1))));
  Form1->Image1->Canvas->Brush->Color = RGB(0, 0, 200);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,alt1),
    Point(pos1+lb,alt1), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1-lb))));
  //Desenha segunda barra do gráfico.
  Form1->Image1->Canvas->Pen->Style = psClear;
  Form1->Image1->Canvas->Brush->Color = RGB(0,255,0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,bi),
    Point(pos2+lb,bi), Point(pos2+lb,alt2), Point(pos2,alt2))));
  Form1->Image1->Canvas->Brush->Color = RGB(0, 150, 0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2+lb,bi),
    Point(pos2+lb+lb,bi-lb), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2))));
  Form1->Image1->Canvas->Brush->Color = RGB(0, 200, 0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,alt2),
    Point(pos2+lb,alt2), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2-lb))));
  //Desenha terceira barra do gráfico.
  Form1->Image1->Canvas->Pen->Style = psClear;
  Form1->Image1->Canvas->Brush->Color = RGB(255,0,0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,bi),
    Point(pos3+lb,bi), Point(pos3+lb,alt3), Point(pos3,alt3))));
  Form1->Image1->Canvas->Brush->Color = RGB(150, 0, 0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3+lb,bi),
    Point(pos3+lb+lb,bi-lb), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3))));
  Form1->Image1->Canvas->Brush->Color = RGB(200, 0, 0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,alt3),
    Point(pos3+lb,alt3), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3-lb))));
  //Desenha quarta barra do gráfico.
  Form1->Image1->Canvas->Pen->Style = psClear;
  Form1->Image1->Canvas->Brush->Color = RGB(255, 255, 0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,bi),
    Point(pos4+lb,bi), Point(pos4+lb,alt4), Point(pos4,alt4))));
  Form1->Image1->Canvas->Brush->Color = RGB(150, 150, 0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4+lb,bi),
```

```
    Point(pos4+lb+lb,bi-lb), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4)}}};
  Form1->Image1->Canvas->Brush->Color = RGB(200, 200, 0);
  Form1->Image1->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,alt4),
    Point(pos4+lb,alt4), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4-lb)}});
  Form1->Image1->Canvas->Brush->Color = clLtGray;
  Form1->Image1->Canvas->TextOut(pos1,bi+10,"Peça 1");
  Form1->Image1->Canvas->TextOut(pos2,bi+10,"Peça 2");
  Form1->Image1->Canvas->TextOut(pos3,bi+10,"Peça 3");
  Form1->Image1->Canvas->TextOut(pos4,bi+10,"Peça 4");
  Form1->Image1->Canvas->TextOut(pos2,bs-15,"Gráfico1 : Tempo de Produção");
  Form1->Image1->Canvas->Pen->Style = psSolid;

}
//---------------------------------------------------------------------------
void DesenhaGrafico2(float a1, float a2, float a3, float a4){
  char *s;
  int bi, bs, bd, be, lb, pos1, pos2, pos3, pos4, i, alt1, alt2, alt3, alt4,
    h, j, esc;
  float es, max = 0;

  if(a1>a2) max = a1;
  else max = a2;
  if(a3>max) max = a3;
  if(a4>max) max = a4;

  bs = 30;
  bi = Form1->Image2->ClientWidth - 10;
  be = 40;
  bd = be+320;
  lb = 40;
  pos1 = be+30; pos2 = be+100; pos3 = be+160;
  pos4 = be+220;
  h = bi - bs-lb-5;

  alt1 = bi - (a1/max*h);
  alt2 = bi - (a2/max*h);
  alt3 = bi - (a3/max*h);
  alt4 = bi - (a4/max*h);

  j = h / 5;

  Form1->Image2->Canvas->Pen->Color = clLtGray;
  Form1->Image2->Canvas->Brush->Color = clLtGray;
  Form1->Image2->Canvas->Rectangle(0,0,Form1->Image2->ClientWidth, Form1->Image2-> ClientHeight);
  Form1->Image2->Canvas->Pen->Color = clBlack;
  Form1->Image2->Canvas->Brush->Color = clDkGray;
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(be, bi),
    Point(be+lb, bi-lb), Point(be+lb,bs), Point(be,bs+lb)}});
  Form1->Image2->Canvas->Brush->Color = clWhite;
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bs),
    Point(be+lb,bi-lb), Point(bd,bi-lb), Point(bd,bs)}});
  Form1->Image2->Canvas->Brush->Color = clLtGray;
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bi-lb),
    Point(be,bi), Point(bd-ib,bi), Point(bd,bi-lb)}});
  Form1->Image2->Canvas->TextOut(be-35,bi-5,"0,00");
  i = bi - j;

  while(i-lb > bs){
    es = ((bi - i)*a1)/(bi-alt1);
    es = ceil(es);
    sprintf(s, "%.2lf",es);
    Form1->Image2->Canvas->TextOut(be-35,i-5,s);
    Form1->Image2->Canvas->MoveTo(be, i);
    Form1->Image2->Canvas->LineTo(be+lb, i-lb);
    Form1->Image2->Canvas->LineTo(bd, i-lb);
    i = i - j;
  }

  //Desenha primeira barra do gráfico.
  Form1->Image2->Canvas->Pen->Style = psClear;
  Form1->Image2->Canvas->Brush->Color = clBlue;
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,bi),
    Point(pos1+lb,bi), Point(pos1+lb,alt1), Point(pos1,alt1)}});
  Form1->Image2->Canvas->Brush->Color = RGB(0, 0, 150);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1+lb,bi),
    Point(pos1+lb+lb,bi-lb), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1)}});
  Form1->Image2->Canvas->Brush->Color = RGB(0, 0, 200);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,alt1),
    Point(pos1+lb,alt1), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1-lb)}});

  //Desenha segunda barra do gráfico.
  Form1->Image2->Canvas->Pen->Style = psClear;
  Form1->Image2->Canvas->Brush->Color = RGB(0,255,0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,bi),
    Point(pos2+lb,bi), Point(pos2+lb,alt2), Point(pos2,alt2)}});
  Form1->Image2->Canvas->Brush->Color = RGB(0, 150, 0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2+lb,bi),
    Point(pos2+lb+lb,bi-lb), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2)}});
  Form1->Image2->Canvas->Brush->Color = RGB(0, 200, 0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,alt2),
    Point(pos2+lb,alt2), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2-lb)}});

  //Desenha terceira barra do gráfico.
  Form1->Image2->Canvas->Pen->Style = psClear;
  Form1->Image2->Canvas->Brush->Color = RGB(255,0,0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,bi),
    Point(pos3+lb,bi), Point(pos3+lb,alt3), Point(pos3,alt3)}});
  Form1->Image2->Canvas->Brush->Color = RGB(150, 0, 0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3+lb,bi),
    Point(pos3+lb+lb,bi-lb), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3)}});
  Form1->Image2->Canvas->Brush->Color = RGB(200, 0, 0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,alt3),
    Point(pos3+lb,alt3), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3-lb)}});
```

```
  //Desenha quarta barra do gráfico.
  Form1->Image2->Canvas->Pen->Style = psClear;
  Form1->Image2->Canvas->Brush->Color = RGB(255, 255, 0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,bi),
    Point(pos4+lb,bi), Point(pos4+lb,alt4), Point(pos4,alt4))));
  Form1->Image2->Canvas->Brush->Color = RGB(150, 150, 0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4+lb,bi),
    Point(pos4+lb+lb,bi-lb), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4))));
  Form1->Image2->Canvas->Brush->Color = RGB(200, 200, 0);
  Form1->Image2->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,alt4),
    Point(pos4+lb,alt4), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4-lb))));

  Form1->Image2->Canvas->Brush->Color = clLtGray;
  Form1->Image2->Canvas->TextOut(pos1,bi+10,"Máq. 1");
  Form1->Image2->Canvas->TextOut(pos2,bi+10,"Máq. 2");
  Form1->Image2->Canvas->TextOut(pos3,bi+10,"Máq. 3");
  Form1->Image2->Canvas->TextOut(pos4,bi+10,"Máq. 4");
  Form1->Image2->Canvas->TextOut(pos2,bs-15,"Gráfico2 : Desempenho das Máquinas");
  Form1->Image1->Canvas->Pen->Style = psSolid;
}
//---------------------------------------------------------------------------
void DesenhaGrafico3(float a1, float a2, float a3, float a4){
  char *s;
  int bi, bs, bd, be, lb, pos1, pos2, pos3, pos4, i, alt1, alt2, alt3, alt4,
    h, j, esc;
  float es, max = 0;

  if(a1>a2) max = a1;
  else max = a2;
  if(a3>max) max = a3;
  if(a4>max) max = a4;

  bs = 30;
  bi = Form1->Image3->ClientWidth - 10;
  be = 40;
  bd = be+320;
  lb = 40;
  pos1 = be+30; pos2 = be+100; pos3 = be+160;
  pos4 = be+220;
  h = bi - bs-lb-5;

  alt1 = bi - (a1/max*h);
  alt2 = bi - (a2/max*h);
  alt3 = bi - (a3/max*h);
  alt4 = bi - (a4/max*h);

  j = h / 5;

  Form1->Image3->Canvas->Pen->Color = clLtGray;
  Form1->Image3->Canvas->Brush->Color = clLtGray;
  Form1->Image3->Canvas->Rectangle(0,0,Form1->Image3->ClientWidth, Form1->Image3-> ClientHeight);
  Form1->Image3->Canvas->Pen->Color = clBlack;
  Form1->Image3->Canvas->Brush->Color = clDkGray;
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(be, bi),
    Point(be+lb, bi-lb), Point(be+lb,bs), Point(be,bs+lb))));
  Form1->Image3->Canvas->Brush->Color = clWhite;
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bs),
    Point(be+lb,bi-lb), Point(bd,bi-lb), Point(bd,bs))));
  Form1->Image3->Canvas->Brush->Color = clLtGray;
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bi-lb),
    Point(be,bi), Point(bd-lb,bi), Point(bd,bi-lb))));

  Form1->Image3->Canvas->TextOut(be-35,bi-5,"0,00");
  i = bi - j;

  while(i-lb > bs){
    es = ((bi - i)*a1)/(bi-alt1);
    es = ceil(es);
    sprintf(s, "%.2lf",es);
    Form1->Image3->Canvas->TextOut(be-35,i-5,s);
    Form1->Image3->Canvas->MoveTo(be, i);
    Form1->Image3->Canvas->LineTo(be+lb, i-lb);
    Form1->Image3->Canvas->LineTo(bd, i-lb);
    i = i - j;
  }

  //Desenha primeira barra do gráfico.
  Form1->Image3->Canvas->Pen->Style = psClear;
  Form1->Image3->Canvas->Brush->Color = clBlue;
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,bi),
    Point(pos1+lb,bi), Point(pos1+lb,alt1), Point(pos1,alt1))));
  Form1->Image3->Canvas->Brush->Color = RGB(0, 0, 150);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1+lb,bi),
    Point(pos1+lb+lb,bi-lb), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1))));
  Form1->Image3->Canvas->Brush->Color = RGB(0, 0, 200);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,alt1),
    Point(pos1+lb,alt1), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1-lb))));

  //Desenha segunda barra do gráfico.
  Form1->Image3->Canvas->Pen->Style = psClear;
  Form1->Image3->Canvas->Brush->Color = RGB(0,255,0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,bi),
    Point(pos2+lb,bi), Point(pos2+lb,alt2), Point(pos2,alt2))));
  Form1->Image3->Canvas->Brush->Color = RGB(0, 150, 0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2+lb,bi),
    Point(pos2+lb+lb,bi-lb), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2))));
  Form1->Image3->Canvas->Brush->Color = RGB(0, 200, 0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,alt2),
    Point(pos2+lb,alt2), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2-lb))));

  //Desenha terceira barra do gráfico.
  Form1->Image3->Canvas->Pen->Style = psClear;
```

```
  Form1->Image3->Canvas->Brush->Color = RGB(255,0,0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,bi),
    Point(pos3+lb,bi), Point(pos3+lb,alt3), Point(pos3,alt3))));
  Form1->Image3->Canvas->Brush->Color = RGB(150, 0, 0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3+lb,bi),
    Point(pos3+lb+lb,bi-lb), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3))));
  Form1->Image3->Canvas->Brush->Color = RGB(200, 0, 0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,alt3),
    Point(pos3+lb,alt3), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3-lb))));

  //Desenha quarta barra do gráfico.
  Form1->Image3->Canvas->Pen->Style = psClear;
  Form1->Image3->Canvas->Brush->Color = RGB(255, 255, 0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,bi),
    Point(pos4+lb,bi), Point(pos4+lb,alt4), Point(pos4,alt4))));
  Form1->Image3->Canvas->Brush->Color = RGB(150, 150, 0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4+lb,bi),
    Point(pos4+lb+lb,bi-lb), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4))));
  Form1->Image3->Canvas->Brush->Color = RGB(200, 200, 0);
  Form1->Image3->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,alt4),
    Point(pos4+lb,alt4), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4-lb))));

  Form1->Image3->Canvas->Brush->Color = clLtGray;
  Form1->Image3->Canvas->TextOut(pos1,bi+10,"Peça 1");
  Form1->Image3->Canvas->TextOut(pos2,bi+10,"Peça 2");
  Form1->Image3->Canvas->TextOut(pos3,bi+10,"Peça 3");
  Form1->Image3->Canvas->TextOut(pos4,bi+10,"Peça 4");
  Form1->Image3->Canvas->TextOut(pos2,bs-15,"Gráfico3 : Custo por Peça");
  Form1->Image1->Canvas->Pen->Style = psSolid;
}
//---------------------------------------------------------------------------
void DesenhaGrafico4(float a1, float a2, float a3, float a4){
  char *s;
  int bi, bs, bd, be, lb, pos1, pos2, pos3, pos4, i, alt1, alt2, alt3, alt4,
    h, j, esc;
  float es, max = 0;

  if(a1>a2) max = a1;
  else max = a2;
  if(a3>max) max = a3;
  if(a4>max) max = a4;

  bs = 30;
  bi = Form1->Image4->ClientWidth - 10;
  be = 40;
  bd = be+320;
  lb = 40;
  pos1 = be+30; pos2 = be+100; pos3 = be+160;
  pos4 = be+220;
  h = bi - bs-lb-5;

  alt1 = bi - (a1/max*h);
  alt2 = bi - (a2/max*h);
  alt3 = bi - (a3/max*h);
  alt4 = bi - (a4/max*h);

  j = h / 5;

  Form1->Image4->Canvas->Pen->Color = clLtGray;
  Form1->Image4->Canvas->Brush->Color = clLtGray;
  Form1->Image4->Canvas->Rectangle(0,0,Form1->Image4->ClientWidth, Form1->Image4-> ClientHeight);
  Form1->Image4->Canvas->Pen->Color = clBlack;
  Form1->Image4->Canvas->Brush->Color = clDkGray;
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(be, bi),
    Point(be+lb, bi-lb), Point(be+lb,bs), Point(be,bs+lb))));
  Form1->Image4->Canvas->Brush->Color = clWhite;
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bs),
    Point(be+lb,bi-lb), Point(bd,bi-lb), Point(bd,bs))));
  Form1->Image4->Canvas->Brush->Color = clLtGray;
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(be+lb,bi-lb),
    Point(be,bi), Point(bd-lb,bi), Point(bd,bi-lb))));
  Form1->Image4->Canvas->TextOut(be-35,bi-5,"0,00");
  i = bi - j;

  while(i-lb > bs){
    es = ((bi - i)*a1)/(bi-alt1);
    es = ceil(es);
    sprintf(s, "%.2lf",es);
    Form1->Image4->Canvas->TextOut(be-35,i-5,s);
    Form1->Image4->Canvas->MoveTo(be, i);
    Form1->Image4->Canvas->LineTo(be+lb, i-lb);
    Form1->Image4->Canvas->LineTo(bd, i-lb);
    i = i - j;
  }
  //Desenha primeira barra do gráfico.
  Form1->Image4->Canvas->Pen->Style = psClear;
  Form1->Image4->Canvas->Brush->Color = clBlue;
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,bi),
    Point(pos1+lb,bi), Point(pos1+lb,alt1), Point(pos1,alt1))));
  Form1->Image4->Canvas->Brush->Color = RGB(0, 0, 150);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1+lb,bi),
    Point(pos1+lb+lb,bi-lb), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1))));
  Form1->Image4->Canvas->Brush->Color = RGB(0, 0, 200);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos1,alt1),
    Point(pos1+lb,alt1), Point(pos1+lb+lb,alt1-lb), Point(pos1+lb,alt1-lb))));

  //Desenha segunda barra do gráfico.
  Form1->Image4->Canvas->Pen->Style = psClear;
  Form1->Image4->Canvas->Brush->Color = RGB(0,255,0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,bi),
    Point(pos2+lb,bi), Point(pos2+lb,alt2), Point(pos2,alt2))));
  Form1->Image4->Canvas->Brush->Color = RGB(0, 150, 0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2+lb,bi),
```

```
    Point(pos2+lb+lb,bi-lb), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2)}}));
  Form1->Image4->Canvas->Brush->Color = RGB(0, 200, 0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos2,alt2),
    Point(pos2+lb,alt2), Point(pos2+lb+lb,alt2-lb), Point(pos2+lb,alt2-lb))));

  //Desenha terceira barra do gráfico.
  Form1->Image4->Canvas->Pen->Style = psClear;
  Form1->Image4->Canvas->Brush->Color = RGB(255,0,0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,bi),
    Point(pos3+lb,bi), Point(pos3+lb,alt3), Point(pos3,alt3))));
  Form1->Image4->Canvas->Brush->Color = RGB(150, 0, 0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3+lb,bi),
    Point(pos3+lb+lb,bi-lb), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3))));
  Form1->Image4->Canvas->Brush->Color = RGB(200, 0, 0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos3,alt3),
    Point(pos3+lb,alt3), Point(pos3+lb+lb,alt3-lb), Point(pos3+lb,alt3-lb))));

  //Desenha quarta barra do gráfico.
  Form1->Image4->Canvas->Pen->Style = psClear;
  Form1->Image4->Canvas->Brush->Color = RGB(255, 255, 0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,bi),
    Point(pos4+lb,bi), Point(pos4+lb,alt4), Point(pos4,alt4))));
  Form1->Image4->Canvas->Brush->Color = RGB(150, 150, 0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4+lb,bi),
    Point(pos4+lb+lb,bi-lb), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4))));
  Form1->Image4->Canvas->Brush->Color = RGB(200, 200, 0);
  Form1->Image4->Canvas->Polygon (OPENARRAY(TPoint,(Point(pos4,alt4),
    Point(pos4+lb,alt4), Point(pos4+lb+lb,alt4-lb), Point(pos4+lb,alt4-lb))));

  Form1->Image4->Canvas->Brush->Color = clLtGray;
  Form1->Image4->Canvas->TextOut(pos1,bi+10,"Peça 1");
  Form1->Image4->Canvas->TextOut(pos2,bi+10,"Peça 2");
  Form1->Image4->Canvas->TextOut(pos3,bi+10,"Peça 3");
  Form1->Image4->Canvas->TextOut(pos4,bi+10,"Peça 4");
  Form1->Image4->Canvas->TextOut(pos2,bs-15,"Gráfico4 : Nível deProdução");
  Form1->Image1->Canvas->Pen->Style = psSolid;
}
```

INICIO.H

```
//---------------------------------------------------------------------------
#ifndef InicioH
#define InicioH
//---------------------------------------------------------------------------
#endif
```

INICIO.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
#include "Inicio.h"
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender){
  /*register int Cont, Cont1, Cont2;
  // Inicializacao da esteira
  Esteira.Comprimento.Metros      = CompEstMetro;
  Esteira.Comprimento.Pixels      = TotalPontos;
  Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
  Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg *
Esteira.Comprimento.Pixels / Esteira.Comprimento.Metros;
  // inicializacao do posicionamento dos pallets e o seu conteudo
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Pallet[Cont].PosicaoAtual.Metros = Cont * Esteira.Comprimento.Metros
                                       / TotalPallets;
    Pallet[Cont].MetroParaPixel();
    Pallet[Cont].PosicaoAnterior = Pallet[Cont].PosicaoAtual;
    Pallet[Cont].Peca.TipoPeca = 0;
    Pallet[Cont].Peca.IndicePeca = 0;
  }
  // inicializa o total de pecas
  TotalPecas[0] = TotalPeca1 + TotalPeca2 + TotalPeca3 + TotalPeca4;
  TotalPecas[1] = TotalPeca1;
  TotalPecas[2] = TotalPeca2;
  TotalPecas[3] = TotalPeca3;
  TotalPecas[4] = TotalPeca4;
  // inicializacao da matriz de posicao
  for (Cont = 0; Cont < TotalPontos; Cont++){
    if (Cont <= 83){
      MatCoord[Cont].x = 516;
      MatCoord[Cont].y = 104 - Cont;
    }
    else{
      if (Cont <= 496){
        MatCoord[Cont].x = 599 - Cont;
        MatCoord[Cont].y = 21;
      }
      else{
        if (Cont <= 834){
          MatCoord[Cont].x = 103;
          MatCoord[Cont].y = Cont - 475;
        }
        else{
          if (Cont <= 883){
            MatCoord[Cont].x = Cont - 731;
            MatCoord[Cont].y = 359;
          }
          else{
            if (Cont <= 916){
              MatCoord[Cont].x = 152;
              MatCoord[Cont].y = 1242 - Cont;
            }
            else{
              if (Cont <= 949){
```

```
            MatCoord[Cont].x = 1068 - Cont;
            MatCoord[Cont].y = 326;
          }
          else{
            if (Cont <= 1238){
              MatCoord[Cont].x = 119;
              MatCoord[Cont].y = 1275 - Cont;
            }
            else{
              if (Cont <= 1619){
                MatCoord[Cont].x = Cont - 1119;
                MatCoord[Cont].y = 37;
              }
              else{
                if (Cont <= 1815){
                  MatCoord[Cont].x = 500;
                  MatCoord[Cont].y = Cont - 1582;
                }
                else{
                  if (Cont <= 1865){
                    MatCoord[Cont].x = Cont - 1315;
                    MatCoord[Cont].y = 233;
                  }
                  else{
                    if (Cont <= 1898){
                      MatCoord[Cont].x = 550;
                      MatCoord[Cont].y = 2098 - Cont;
                    }
                    else{
                      if (Cont <= 1932){
                        MatCoord[Cont].x = 2448 - Cont;
                        MatCoord[Cont].y = 200;
                      }
                      else{
                        MatCoord[Cont].x = 516;
                        MatCoord[Cont].y = 2132 - Cont;
                      } // else Cont <= 1932
                    } // else Connt <= 1898
                  } // else Cont <= 1865
                } // else Cont <= 1815
              } // else Cont <= 1619
            } // else Cont <= 1238
          } // else Cont <= 949
        } // else Cont <= 916
      } // else Cont <= 883
    } // else Cont <= 834
  } // else Cont <= 496
 } // else Cont <= 83
} // for Cont
// inicializacao dos bitmaps
RoboSaveBack                        = new Graphics::TBitmap;
for (Cont = 0; Cont < TotalPallets; Cont++){
  if (Cont < NumStatus)
    ImagStatus[Cont]                = new Graphics::TBitmap;
  if (Cont < NumMaquina){
    Maquina[Cont+1].Status.ImgStatus = new Graphics::TBitmap;
    Maquina[Cont+1].Status.MscStatus = new Graphics::TBitmap;
    Maquina[Cont+1].Status.MscStatus->LoadFromResourceName((int)HInstance,"STATUSMASK");
  }
  if (Cont < NumPosicao){
    Robo[Cont]                  = new Graphics::TBitmap;
    RoboMask[Cont]                      = new Graphics::TBitmap;
  }
  if (Cont < (NumPeca+1))
    ImgPeca[Cont]                = new Graphics::TBitmap;
  Pallet[Cont].ImgPallet         = new Graphics::TBitmap;
  Pallet[Cont].MscPallet         = new Graphics::TBitmap;
  Pallet[Cont].SavPallet         = new Graphics::TBitmap;
}
// carregamento das imagens bitmaps
ImgPeca[0]   ->LoadFromResourceName((int)HInstance,"PALLET");
ImgPeca[1]   ->LoadFromResourceName((int)HInstance,"PECA1");
ImgPeca[2]   ->LoadFromResourceName((int)HInstance,"PECA2");
ImgPeca[3]   ->LoadFromResourceName((int)HInstance,"PECA3");
ImgPeca[4]   ->LoadFromResourceName((int)HInstance,"PECA4");
ImagStatus[0]->LoadFromResourceName((int)HInstance,"GREEN");
ImagStatus[1]->LoadFromResourceName((int)HInstance,"YELLOW");
ImagStatus[2]->LoadFromResourceName((int)HInstance,"RED");
for (Cont = 0; Cont < TotalPallets; Cont++){
  Pallet[Cont].ImgPallet = ImgPeca[0];
  Pallet[Cont].MscPallet->LoadFromResourceName((int)HInstance,"PALLETMASK");
}
//Associa imagens dos robos.
robos[1].AssociaImagem("ROBO_RIGHT", "ROBO_RIGHT_MASK");
robos[2].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
robos[3].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
robos[4].AssociaImagem("ROBO_RIGHT", "ROBO_RIGHT_MASK");
//robos[5].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
//SavedBackgroundRegion will store the portion of the background
//that gets destroyed when the ship is copied in. Make it the same
//size as the ship.
for (Cont = 0; Cont < TotalPallets; Cont++){
  Pallet[Cont].SavPallet->Width  = Pallet[Cont].ImgPallet->Width;
  Pallet[Cont].SavPallet->Height = Pallet[Cont].ImgPallet->Height;
}
for (Cont = 1; Cont <= 3; Cont++){
  robos[Cont].status = LIVRE;
        robos[Cont].pos = POS_E;
  robos[Cont].sentido = ME;
  robos[Cont].id =(short int)(Cont);
  robos[Cont].buffer_fundo->Width  = robos[Cont].imagem->Width;
  robos[Cont].buffer_fundo->Height = robos[Cont].imagem->Height;
}
```

```
    robos[4].status = LIVRE;
          robos[4].pos = POS_E;
    robos[4].sentido = PR;
    robos[4].id =(short int)4;
    robos[4].buffer_fundo->Width  = robos[4].imagem->Width;
    robos[4].buffer_fundo->Height = robos[4].imagem->Height;

   robos[5].status = LIVRE;
          robos[5].pos = POS_I;
    robos[5].sentido = OI;
    robos[5].id =(short int)5;
    robos[5].buffer_fundo->Width  = robos[5].imagem->Width;
    robos[5].buffer_fundo->Height = robos[5].imagem->Height;

//Define posicoes iniciais dos robos.
  robos[1].x = 48;
  robos[1].y = 234;
  robos[2].x = 117;
  robos[2].y = 205;
  robos[3].x = 117;
  robos[3].y = 153;
  robos[4].x = 444;
  robos[4].y = 178;
  robos[5].x = 513;
  robos[5].y = 84;
  //Guarda dimensoes das imagens
  for (Cont = 1; Cont <= NUM_ROBOS; Cont++)
    robos[Cont].roboRect = Rect(0,0,robos[Cont].buffer_fundo->Width,robos[Cont].buffer_ fundo->Height);

  for (Cont = 0; Cont < 7; Cont++){
    sensor[Cont].Id =(short int) Cont;
    sensor[Cont].pallet = -1;
    sensor[Cont].FlagPallet = 0;
    sensor[Cont].FlagPeca = 0;
  }
  // parametros de posicao e movimento do robo
  CondAtual[PosicaoRobo] = Esquerda;
  CondAtual[Movimento]   = Alto;
  CondMov[PosicaoRobo]   = Esquerda;
  CondMov[Movimento]     = Subir;
  // parametros de tempo
  TempoAnterior     = 0;
  TempoAnteriorComp = TempoInicial;
  TempoDecorrido = 0;
  // Inicializacao das maquinas
  for (Cont = 1; Cont <= NumMaquina; Cont++){
    Maquina[Cont].rpos1                    = 0;
    Maquina[Cont].spos1                    = 0;
    Maquina[Cont].rpos2                    = 0;
    Maquina[Cont].spos2                    = 0;
    Maquina[Cont].Indice                   = (short int) Cont;
    Maquina[Cont].vazio1                   = 1;
    Maquina[Cont].vazio2                   = 1;
    Maquina[Cont].cheio1                   = 0;
    Maquina[Cont].cheio2                   = 0;
    Maquina[Cont].PecaProcesso.TipoPeca    = 0;  // Note que 0(zero)indica s/peca
    Maquina[Cont].PecaProcesso.IndicePeca  = -1;
    Maquina[Cont].Status.StatusAtual       = Espera;
    Maquina[Cont].tempo[0]                 = 0;
    Maquina[Cont].tempo[1]                 = 0;
    Maquina[Cont].taux                     = 0;
    Maquina[Cont].PalletAnterior           = -1;
  }
  for (Cont = 1; Cont <= NumMaquina; Cont++)
    for(Cont1 = 0; Cont1 < TamanhoFila; Cont1++){
      Maquina[Cont].FilaEntrada[Cont1].TipoPeca = 0;
      Maquina[Cont].FilaSaida[Cont1].TipoPeca   = 0;
      Maquina[Cont].FilaEntrada[Cont1].IndicePeca = 0;
      Maquina[Cont].FilaSaida[Cont1].IndicePeca   = 0;
    }
  Maquina[1].Posicao.Metros     = 17.138;
  Maquina[1].Posicao.Pixels     = 720;
  Maquina[1].Status.PosStatus.x = 12;
  Maquina[1].Status.PosStatus.y = 276;
    Maquina[2].Posicao.Metros     = 25.208;
    Maquina[2].Posicao.Pixels     = 1059;
    Maquina[2].Status.PosStatus.x = 144;
    Maquina[2].Status.PosStatus.y = 247;
  Maquina[3].Posicao.Metros     = 26.374;
  Maquina[3].Posicao.Pixels     = 1108;
  Maquina[3].Status.PosStatus.x = 171;
  Maquina[3].Status.PosStatus.y = 81;
    Maquina[4].Posicao.Metros     = 42.156;
    Maquina[4].Posicao.Pixels     = 1771;
    Maquina[4].Status.PosStatus.x = 417;
    Maquina[4].Status.PosStatus.y = 56;
  Maquina[5].Posicao.Metros     = 42.156;
  Maquina[5].Posicao.Pixels     = 1771;
  Maquina[5].Status.PosStatus.x = 431;
  Maquina[5].Status.PosStatus.y = 279;
  // Inicializacao da Entrada e Saida do processo
  InProcesso.Posicao.Metros   = 0;
  InProcesso.Posicao.Pixels   = 0;
  InProcesso.Contador               = 0;
  InProcesso.PalletAnterior   = -1;
  OutProcesso.Posicao.Metros  = 47.321;
  OutProcesso.Posicao.Pixels  = 1988;
  OutProcesso.Contador              = 0;
  OutProcesso.PalletAnterior  = -1;
  // Inicializacao das pecas
  for(Cont2 = 1; Cont2 <= NumPeca; Cont2++){ // 0 eh lixo
    for(Cont = 1; Cont <= TotalPecas[Cont2]; Cont++){
      Peca[Cont2][Cont].PosicaoAtual[0] = 0;
```

```
        Peca[Cont2][Cont].PosicaoAtual[1] = 0;
        Peca[Cont2][Cont].TempoPeca[0]    = 0;
        Peca[Cont2][Cont].TempoPeca[1]    = 0;
        Peca[Cont2][Cont].SairProcesso    = false;
        Peca[Cont2][Cont].LinhaProcesso   = 0;
        for( Cont1 = 0; Cont1 < NumMaquina; Cont1++){
//          Peca[Cont2][Cont].Processos[Cont1].Ref    = 1;
          Peca[Cont2][Cont].Processos[Cont1].TmpInc = 0;
          Peca[Cont2][Cont].Processos[Cont1].TmpFnl = 0;
          Peca[Cont2][Cont].Processos[Cont1].TmpPrc = 10;
          Peca[Cont2][Cont].Processos[Cont1].TmpIn  = 0;
          Peca[Cont2][Cont].Processos[Cont1].TmpOut = 0;
    } } } // Finaliza os For Cont, Cont1 e Cont2
    // Configuracao da sequencia de operacao da peca 1
    for( Cont1 = 1; Cont1 <= TotalPecas[1]; Cont1++){
      Peca[1][Cont1].Processos[0].Sequencia    = 4;
      Peca[1][Cont1].Processos[1].Sequencia    = 0;
      Peca[1][Cont1].Processos[2].Sequencia    = 0;
      Peca[1][Cont1].Processos[3].Sequencia    = 0;
      Peca[1][Cont1].Processos[4].Sequencia    = 0;
    }
    // Configuracao da sequencia de operacao da peca 2
    for( Cont1 = 1; Cont1 <= TotalPecas[2]; Cont1++){
      Peca[2][Cont1].Processos[0].Sequencia    = 1;
      Peca[2][Cont1].Processos[1].Sequencia    = 2;
      Peca[2][Cont1].Processos[2].Sequencia    = 3;
      Peca[2][Cont1].Processos[3].Sequencia    = 4;
      Peca[2][Cont1].Processos[4].Sequencia    = 5;
    }
    // Configuracao da sequencia de operacao da peca 3
    for( Cont1 = 1; Cont1 <= TotalPecas[3]; Cont1++){
      Peca[3][Cont1].Processos[0].Sequencia    = 2;
      Peca[3][Cont1].Processos[1].Sequencia    = 0;
      Peca[3][Cont1].Processos[2].Sequencia    = 0;
      Peca[3][Cont1].Processos[3].Sequencia    = 0;
      Peca[3][Cont1].Processos[4].Sequencia    = 0;
    }
    // Configuracao da sequencia de operacao da peca 4
    for( Cont1 = 1; Cont1 <= TotalPecas[4]; Cont1++){
      Peca[4][Cont1].Processos[0].Sequencia    = 1;
      Peca[4][Cont1].Processos[1].Sequencia    = 2;
      Peca[4][Cont1].Processos[2].Sequencia    = 0;
      Peca[4][Cont1].Processos[3].Sequencia    = 0;
      Peca[4][Cont1].Processos[4].Sequencia    = 0;
    }

    // Inicializacao das variaveis de tempo
    TmpIncSim[0]  = 0;
    TmpIncSim[1]  = timeGetTime();
    TmpAtSim[0]   = 0;
    TmpAtSim[1]   = TmpIncSim[1];
    TmpPausa[0]   = 0;
    TmpPausa[1]   = 0;
    TmpPausa[2]   = 0;
    TmpAntSim[0]  = 0;
    TmpAntSim[1]  = TmpIncSim[1];
    TmpAntAnim[0] = 0;
    TmpAntAnim[1] = TmpIncSim[1];
    Pausa[0]      = false;
    Pausa[1]      = false;
    PausaVar                    = false;

    VelSim = VelocidadeInicial;
    InfoNumber = 0;
    AnimNumber = 0;

    FimDaSimulacao = true;

    // parameters are ready, draw the ship in its initial location
    // and move the image to the screen
    SysSpeedOp = SysSpeed;
    DrawPalletOnBackground();
    DrawRoboOnBackground();
    MoveBitmapToScreen();
    PatchBackground();
    AtualizaTempo();
    CriarArquivo();
    EditTempo->Text = "0.0";
    */
}
//---------------------------------------------------------------------------
```

MAINFORM.H

```
//---------------------------------------------------------------------------
#ifndef MAINFORMH
#define MAINFORMH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ExtCtrls.hpp>
#include <vcl\ComCtrls.hpp>
#include <stdio.h>
#include <vcl\Menus.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\Dialogs.hpp>

//---------------------------------------------------------------------------
// Declaracao de constantes globais
// Constantes referentes a pecas
#define TotalPeca                       10   // Total de pecas possiveis de cada tipo
#define TotalPeca1                      5    // Total de pecas possiveis
```

```
#define TotalPeca2                      5    // Total de pecas possiveis
#define TotalPeca3                      5    // Total de pecas possiveis
#define TotalPeca4                      5    // Total de pecas possiveis
#define NumPeca                         4    // Tipo de pecas disponiveis
// Constantes referentes a maquinas
#define TamanhoFila                     5    // Tam filas de entrada e saida das maqs
#define NumMaquina                      5    // Total de maquinas do processo
#define NumStatus                       3    // Numero de status possiveis das maquinas
#define Espera                          0    // Constantes referentes
#define Ativo                           1    //   aos status
#define Inop                            2    //   das maquinas
#define TempoIOMaq                      1000       // tempo para InOut na Maquina [ms]
// Constantes referentes a esteira
#define CompEstMetro          48.273 // Comprimento da esteira em metros
#define VelIncEsteira         0.5  // Velocidade inicial da esteira
#define TotalPontos                     2028 // Tamanho em pixels da esteira
#define TotalPallets                    52   // Numero de pallets na esteira
#define Passo                           39   // Distancia entre pallets
// Constantes referentes a simulacao
#define VelocidadeInicial     1
#define PosicaoRobo                     0    // Constantes de indexacao
#define Movimento                       1    //   das matrizes de dados
// Constantes referentes ao sensor
#define NumSensor               7
// Constantes referentes ao robo
#define NumPosicao                      2    // Numero de posicoes do robo
#define Esquerda                        0    // Constantes de posicionemento
#define Direita                         1    //   do robo
#define Alto                            0    // Constantes de
#define Meio                            1    //   posicionamento
#define Baixo                           2    //   do robo
#define Subir                           0    // Constantes de movomentacao
#define Descer                          2    //   do robo
#define Topo                            86   // coordenada de topo do robo
#define Fundo                           244  // coordenada de fundo do robo
#define NUM_ROBOS 5
//Definicao dos sentidos de movimentacao dos robos.
#define EM 0
#define ME 1
#define PR 2
#define ET 3
#define TE 5
#define TF 4
#define FT 7
#define IO 9
#define OI 10
#define EF 11
#define FE 12
//Definicao dos status dos robos.
#define CAR_IN 0
#define CAR_OUT 1
#define LIVRE 2
#define POS_M 0
#define POS_E 1
#define POS_P 2
#define POS_F 3
#define POS_T 4
#define CAR_INF 5
#define CAR_INT 6
#define POS_I 7
#define POS_O 8
#define GET_PECA_IN 9
#define PUT_PECA_IN  10
#define GET_PECA_OUT 11
#define PUT_PECA_OUT 12
#define CAR_OUTF 13
#define CAR_OUTT 14
#define CAR_INFT 15
#define CAR_INTF 16
//---------------------------------------------------------------------------
// Declaracao de Tipos
class tpPosicao{
  public:
    float Metros;          // Indica a posicao em metros do referencial
    int Pixels;            // Indica a posicao em pixels do referencial
};
class tpVelocidade{
  public:
    float MetrosPorSeg;    // Indica a velocidade em metros do referencial
    float PixelsPorSeg;    // Indica a velocidade em pixels do referencial
};
class tpProcesso{
  public:
    short int Sequencia;
    float TmpInc, TmpFnl, TmpPrc, TmpIn, TmpOut;
};
class tpFilaPeca{
  public:
    short int TipoPeca, IndicePeca;
};
//---------------------------------------------------------------------------
// Declaracao de Classes
class cenario{
public:
        cenario();
    ~cenario();
    Graphics::TBitmap *imagem;
};
class  clStatusMaq{
  public:
    TPoint PosStatus;              // Posicao da imagem do status
    Graphics::TBitmap *ImgStatus;  // Guarda a imagem do status
    Graphics::TBitmap *MscStatus;  // Guarda a imagem do status
```

```
    short int StatusAtual;          // Indica o status da maquina
};
class clPeca{
  public:
    int LinhaProcesso;                      // indica a linha da matriz de processos
    short int PosicaoAtual[2];       // armazena a posicao atual da peca
    float TempoPeca[2];              // armazena entrada e saida na celula
    tpProcesso Processos[NumMaquina];//armazena inform dos processos
    bool SairProcesso;               // True indica que esta peca deve sair do processo
    bool ProcessosExecutados(void);  // True indica que todos os processos foram finalizados
};
class clMaquina{
  public:
    short int Indice;                   // Indice da maq no processo
    tpPosicao Posicao;                  // Posicao da maq no processo
    tpFilaPeca FilaEntrada[TamanhoFila];// Fila de entrada de pecas na maquina
    tpFilaPeca FilaSaida[TamanhoFila];  // Fila de saida de pecas na maquina
    short int vazio1, vazio2, cheio1, cheio2;// Indicadores de fila
    tpFilaPeca PecaProcesso;            // Indica a peca processada no momento
    clStatusMaq Status;                 // Status da maquina
    int PalletAnterior;
    float tempo[2], taux;

    short int rpos1, spos1, rpos2, spos2;  // Ponteiros das filas de entrada e saida
    short int TirardaFilaEntrada(void);     // Tirar peca na fila de entrada
    short int ColocarnaFilaEntrada(short int tipo, short int indice); // Colocar peca da fila de entrada
    short int TirardaFilaSaida(void);       // Tirar peca na fila de saida
    short int ColocarnaFilaSaida(void);// Colocar peca da fila de saida
    short int FilaEntradaCheia(void);
    short int FilaEntradaVazia(void);
    short int FilaSaidaCheia(void);
    short int FilaSaidaVazia(void);
};
class clInOut{
  public:
    tpPosicao Posicao;                  // Posicao do InOut no Processo
    int Contador;                       // Contador
    int PalletAnterior;
};
class clEsteira{
  public:
    long InicioParada;              // Guarda o inicio da parada da esteira
    tpPosicao Comprimento;          // Guarda o tamanho da esteira
    tpVelocidade Velocidade;        // Guarda a velocidade da esteira
};
class clPallet{
  public:
    tpPosicao PosicaoAtual;         // Guarda a posicao atual do pallet
    tpPosicao PosicaoAnterior;      // Guarda a posicao anterior do pallet
    TPoint PalletCoord;             // Guarda as coordenadas do pallet
    TRect PalletRect;               // Guarda a dimensao da imagem
    tpFilaPeca Peca;                // Guarda inform da peca que esta no pallet
    Graphics::TBitmap *ImgPallet;   // Guarda a imagem do pallet
    Graphics::TBitmap *MscPallet;   // Guarda a mascara da imagem do pallet
    Graphics::TBitmap *SavPallet;   // Guarda a imagem anterior ao pallet
    void MetroParaPixel(void);      // Converte a posicao e metros para pixels
};
class clRobo{
  public:
    clRobo();
    ~clRobo();
    Graphics::TBitmap *imagem;
    Graphics::TBitmap *mascara;
    Graphics::TBitmap *buffer_fundo;
    short int id;
    short int pos;
    short int sentido;
    short int status;
    short int rotacao;
    short int flag_tempo;
    int x;
    int y;
    short int TipoPeca;
    short int IndicePeca;
    short int velocidade;
    TRect roboRect;
    void AssociaImagem(AnsiString imagem1, AnsiString imagem2);
    void PatchBackground(void);
    void MoveRoboLocation(void);
    void DrawRoboOnBackground(void);
};
class clSensor{
public:
    short int Id;
    short int pallet;
    short int FlagPallet;
    short int FlagPeca;
    void Reiniciar(void);
};
//---------------------------------------------------------------------------
// Declaracao de funcoes
//---------------------------------------------------------------------------
// Declaracao de variaveis
class TForm1 : public TForm
{
__published:        // IDE-managed Components
  TTimer *Timer1;
        TMainMenu *MenuPrincipal;
        TMenuItem *Programa1;
        TMenuItem *Sair;
        TMenuItem *Relatorio1;
        TMenuItem *Gerar1;
        TMenuItem *Graficos1;
```

```
TMenuItem *Criar1;
TMenuItem *Sobre1;
TMenuItem *SobreRMS1;
TPageControl *PageControl1;
TTabSheet *TabSheet1;
TTabSheet *TabSheet2;
TTabSheet *TabSheet3;
TTabSheet *TabSheet4;
TPaintBox *PaintBox1;
TBevel *Bevel1;
TEdit *EditTempo;
TLabel *Label11;
TBevel *Bevel2;
TLabel *Label2;
TTrackBar *TrackBar1;
TBevel *Bevel3;
TLabel *Label4;
TLabel *Label10;
TLabel *Label12;
TLabel *Label13;
TLabel *Label14;
TSpeedButton *pause;
TSpeedButton *Stop;
TSpeedButton *Play;
TSpeedButton *back;
TLabel *Label1;
TBevel *Bevel4;
TRichEdit *editor;
TSpeedButton *SelecionarTudo1;
TSpeedButton *Colar2;
TSpeedButton *Deletar1;
TSpeedButton *fonte1;
TSpeedButton *abrir_arquivo;
TSpeedButton *imprimir;
TSpeedButton *recortar;
TSpeedButton *copiar;
TSpeedButton *SalvarComo;
TBevel *Bevel5;
TLabel *Label3;
TLabel *Label5;
TLabel *Label6;
TLabel *Label7;
TOpenDialog *OpenDialog1;
TSaveDialog *SaveDialog1;
TFontDialog *FontDialog1;
TPrintDialog *PrintDialog1;
TPageControl *PageControl2;
TTabSheet *TabSheet5;
TTabSheet *TabSheet6;
TTabSheet *TabSheet7;
TTabSheet *TabSheet8;
TImage *Image1;
TImage *Image2;
TImage *Image3;
TImage *Image4;
TLabel *Label8;
TEdit *Edit3;
TEdit *Edit4;
TEdit *Edit5;
TEdit *Edit6;
TEdit *Edit7;
TLabel *Label9;
TLabel *Label16;
TLabel *Label18;
TLabel *Label19;
TEdit *Edit8;
TEdit *Edit9;
TEdit *Edit10;
TEdit *Edit11;
TEdit *Edit12;
TLabel *Label20;
TEdit *Edit13;
TEdit *Edit14;
TEdit *Edit15;
TEdit *Edit16;
TEdit *Edit17;
TLabel *Label21;
TLabel *Label22;
TLabel *Label23;
TLabel *Label24;
TLabel *Label25;
TLabel *Label26;
TLabel *Label27;
TLabel *Label28;
TBevel *Bevel6;
TBevel *Bevel7;
TBevel *Bevel8;
TBevel *Bevel9;
TLabel *Label31;
TEdit *Edit25;
TLabel *Label32;
TEdit *Edit26;
TBevel *Bevel10;
TBevel *Bevel11;
TEdit *Edit27;
TEdit *Edit28;
TEdit *Edit30;
TLabel *Label33;
TLabel *Label34;
TLabel *Label35;
TLabel *Label36;
TEdit *Edit1;
TEdit *Edit2;
```

```
        TEdit *Edit18;
        TEdit *Edit19;
        TEdit *Edit20;
        TSpeedButton *SpeedButton1;
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall Timer1Timer(TObject *Sender);
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall VelEstControlChange(TObject *Sender, bool &AllowChange);
    void __fastcall VelEstChange(TObject *Sender);
    void __fastcall ULClick(TObject *Sender);
    void __fastcall URClick(TObject *Sender);
    void __fastcall DLClick(TObject *Sender);
    void __fastcall DRClick(TObject *Sender);
    void __fastcall RStopClick(TObject *Sender);
    void __fastcall VelRoboChange(TObject *Sender);
    void __fastcall VelRoboControlChange(TObject *Sender, bool &AllowChange);
    void __fastcall VelSysChange(TObject *Sender);
    void __fastcall VelSysControlChange(TObject *Sender, bool &AllowChange);
    void __fastcall btPausaClick(TObject *Sender);
    void __fastcall SairClick(TObject *Sender);
    void __fastcall TrackBar1Change(TObject *Sender);
    void __fastcall pauseClick(TObject *Sender);
    void __fastcall StopClick(TObject *Sender);
    void __fastcall backClick(TObject *Sender);
    void __fastcall PlayClick(TObject *Sender);
    void __fastcall SobreRMS1Click(TObject *Sender);
    void __fastcall Iniciar1Click(TObject *Sender);
    void __fastcall Parar1Click(TObject *Sender);
    void __fastcall abrir_arquivoClick(TObject *Sender);
    void __fastcall SalvarComoClick(TObject *Sender);
    void __fastcall imprimirClick(TObject *Sender);
    void __fastcall SelecionarTudo1Click(TObject *Sender);
    void __fastcall Deletar1Click(TObject *Sender);
    void __fastcall copiarClick(TObject *Sender);
    void __fastcall recortarClick(TObject *Sender);
    void __fastcall Colar2Click(TObject *Sender);
    void __fastcall fonte1Click(TObject *Sender);
    void __fastcall Criar1Click(TObject *Sender);
    void __fastcall Gerar1Click(TObject *Sender);
    void __fastcall SpeedButton1Click(TObject *Sender);
    void __fastcall Edit8Change(TObject *Sender);
private:  // User declarations

public:             // User declarations
  Graphics::TBitmap *Robo[NumPosicao], *ImagStatus[NumStatus],
                    *ImgPeca[NumPeca+1], *RoboMask[NumPosicao], *RoboSaveBack;
  void PatchBackground(void);
  void DrawPalletOnBackground(void);
  void MoveBitmapToScreen(void);
  void DrawRoboOnBackground(void);
  TPoint Location[TotalPallets], RoboLocation;
  short int EstSpeed, RoboSpeed, SysSpeed, SysSpeedOp;
  TRect RoboRect;
  short int CondAtual[2]; // vetor que indica a situacao atual do robo
  short int CondMov[2];   // vetor que indica a condicao de movimento do robo
  TPoint MatCoord[TotalPontos];

// Variaveis do sistema
  long TmpIncSim[2];    // Tempo inicial da simulacao
  long TmpAtSim[2];     // Tempo atual da simulacao
  int VelSim;           // velocidade de simulacao
  long TmpPausa[3];     // Tempo de pausa
  long TmpAntSim[2];    // Tempo anterior da simulacao
  long TmpAntAnim[2];   // Tempo anterior da animacao

  cenario Background;                           //Imagem de fundo
  clRobo robos[NUM_ROBOS + 1];
  clSensor sensor[NumSensor];
  clPallet Pallet[TotalPallets];                // representa todos os pallets da esteira
  clMaquina Maquina[NumMaquina + 1];            // representa todas as maquinas
  int TotalPecas[NumPeca+1];
  clPeca Peca[NumPeca + 1][TotalPeca + 1];      // representa as pecas
  clEsteira Esteira;                            // representa a esteira
  clInOut InProcesso;                           // representa a entrada do processo
  clInOut OutProcesso;                          // representa a saida do processo
  bool Pausa[2];                                // indica se a simulacao esta em pausa
  bool PausaVar;
  bool FimDaSimulacao;                          // indica o fim da droga de simulacao

  void VerificaPausa(void);
  void VerificaInOut(int Indice);
  void DrawStatusMaquina(void);
  void VerificaParadaEsteira(void);
  void VerificaMaquina(int Indice);
  void VerificaPallet(int Indice);
  void AtualizaTempo(void);
  void AtPosPallet(int Indice);
  void AtualizaAnimacao(void);
  int TipoPecaIn(int Indice);
  int IndicePecaIn(int Indice);
  void MostraFilaInOut(int Indice);
  void VerificaRobo(int i);

  void CriarArquivo(void);
  void FecharArquivo(void);
  void GravarPeca1(void);
  void GravarPeca(int Tipo, int Indice);
  void GravarAll(void);

  int InfoNumber;
  int AnimNumber;

  __fastcall TForm1(TComponent* Owner);
```

```
};
//---------------------------------------------------------------------------
extern TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

MAINFORM.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
#include <mmsystem.hpp>
#include "Arquivo.cpp"
#include "BOTAO.cpp"
#include "Sobre.cpp"
#include "grafico.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
#pragma resource "PICTURES.res"
int play = 0;
AnsiString NomeArq;
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
  : TForm(Owner)
{
          NomeArq = "RMS";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender){
  AnsiString InfoString;
  int Cont, Indice;
  Indice = 1;
  if(play == 1){
    if(!(Esteira.Velocidade.MetrosPorSeg == 0))
      for (Cont = 0; Cont < NumSensor; Cont++)
        sensor[Cont].Reiniciar();
    if(!FimDaSimulacao){
      VelSim = SysSpeed;
      VerificaPausa();
      AtualizaTempo();
      if ((TmpAtSim[0] - TmpAntSim[0])*VelSim >= 50){
        for (Cont = 0; Cont < TotalPallets; Cont++)
          AtPosPallet(Cont);
        for(Cont = 1;Cont <= NUM_ROBOS;Cont++)
          robos[Cont].MoveRoboLocation();
        for (Cont = 0; Cont < TotalPallets; Cont++)
          VerificaInOut(Cont);
        for (Cont = 1; Cont <= NumMaquina; Cont++)
          VerificaMaquina(Cont);
        for (Cont = 0; Cont < TotalPallets; Cont++)
          VerificaPallet(Cont);
        for (Cont = 1; Cont <= NUM_ROBOS; Cont++)
          VerificaRobo(Cont);
        TmpAntSim[0] = TmpAtSim[0];
        TmpAntSim[1] = TmpAtSim[1];
        InfoNumber = InfoNumber + 1;
      }  // if simulacao
      if ((TmpAtSim[0] - TmpAntAnim[0]) >= 200){
        AtualizaAnimacao();
        TmpAntAnim[0] = TmpAtSim[0];
        TmpAntAnim[1] = TmpAtSim[1];
        AnimNumber = AnimNumber + 1;
      }
      GravarPeca1();
      MostraFilaInOut(Indice);
    }
  }
}
//---------------------------------------------------------------------------
void TForm1::AtualizaTempo(void){
  char s[20];
  long TempoCiclo = timeGetTime();
  TmpAtSim[1] = TempoCiclo;
  if(Pausa[0]){
    if(Pausa[1]){   // eh executado durante a pausa
    }
    else{                       // eh executado apenas no inicio da pausa
      TmpPausa[1] = TempoCiclo;
    }
  }  // if Pausa[0]
  else{
    if(Pausa[1]){   // eh executado no final da pausa
      TmpPausa[2] = TempoCiclo;
      TmpAntSim[1] = TmpAntSim[1] + TmpPausa[2] - TmpPausa[1];
    }
    else{                       // eh executado quando nao ha pausa
      TmpAtSim[0] = (TmpAtSim[1] - TmpAntSim[1]) * VelSim + TmpAntSim[0];
    }
  }  // else Pausa[0]
  sprintf(s,"%.1lf",(float)TmpAtSim[0]/1000);
  EditTempo->Text = s;
}
//---------------------------------------------------------------------------
void TForm1::AtPosPallet(int Indice){
  float DeltaT;
  DeltaT = (float)(TmpAtSim[0] - TmpAntSim[0])/1000;
  Pallet[Indice].PosicaoAnterior=Pallet[Indice].PosicaoAtual;
  Pallet[Indice].PosicaoAtual.Metros = Pallet[Indice].PosicaoAtual.Metros +
    DeltaT * Esteira.Velocidade.MetrosPorSeg;
  if (Pallet[Indice].PosicaoAtual.Metros >= CompEstMetro)
    Pallet[Indice].PosicaoAtual.Metros = Pallet[Indice].PosicaoAtual.Metros
                                       - CompEstMetro;
```

```
  Pallet[Indice].MetroParaPixel();
  Pallet[Indice].PalletCoord = MatCoord(Pallet[Indice].PosicaoAtual.Pixels);
}
//---------------------------------------------------------------------------
void TForm1::VerificaPausa(void){
  Pausa[1] = Pausa[0];
  Pausa[0] = PausaVar;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::SairClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::TrackBar1Change(TObject *Sender)
{
  SysSpeed =(short int) TrackBar1->Position;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::pauseClick(TObject *Sender)
{
  if (PausaVar){
    PausaVar = false;
   // VelSys->Enabled = true;
  }
  else{
    PausaVar = true;
   // VelSys->Enabled = false;
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::StopClick(TObject *Sender)
{
  FimDaSimulacao = true;
  play = 0;
  Form1->Relatorio1->Enabled = true;
  Form1->Graficos1->Enabled = true;
  Form1->TabSheet4->Enabled = true;
  Form1->TabSheet3->Enabled = true;
  Form1->TabSheet3->TabVisible = true;
  Form1->TabSheet4->TabVisible = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::backClick(TObject *Sender)
{
  int Cont, Cont1, Cont2;
  // carregamento das imagens bitmaps
  ImgPeca[0]   ->LoadFromResourceName((int)HInstance,"PALLET");
  ImgPeca[1]   ->LoadFromResourceName((int)HInstance,"PECA1");
  ImgPeca[2]   ->LoadFromResourceName((int)HInstance,"PECA2");
  ImgPeca[3]   ->LoadFromResourceName((int)HInstance,"PECA3");
  ImgPeca[4]   ->LoadFromResourceName((int)HInstance,"PECA4");
  ImagStatus[0]->LoadFromResourceName((int)HInstance,"GREEN");
  ImagStatus[1]->LoadFromResourceName((int)HInstance,"YELLOW");
  ImagStatus[2]->LoadFromResourceName((int)HInstance,"RED");
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Pallet[Cont].ImgPallet = ImgPeca[0];
    Pallet[Cont].MscPallet->LoadFromResourceName((int)HInstance,"PALLETMASK");
  }
  //Associa imagens dos robos.
  robos[1].AssociaImagem("ROBO_RIGHT", "ROBO_RIGHT_MASK");
  robos[2].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
  robos[3].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
  robos[4].AssociaImagem("ROBO_RIGHT", "ROBO_RIGHT_MASK");
  //robos[5].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
  //SavedBackgroundRegion will store the portion of the background
  //that gets destroyed when the ship is copied in. Make it the same
  //size as the ship.
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Pallet[Cont].SavPallet->Width  = Pallet[Cont].ImgPallet->Width;
    Pallet[Cont].SavPallet->Height = Pallet[Cont].ImgPallet->Height;
  }
  for (Cont = 1; Cont <= 3; Cont++){
    robos[Cont].status = LIVRE;
    robos[Cont].pos = POS_E;
    robos[Cont].sentido = ME;
    robos[Cont].id =(short int)(Cont);
    robos[Cont].buffer_fundo->Width  = robos[Cont].imagem->Width;
    robos[Cont].buffer_fundo->Height = robos[Cont].imagem->Height;
  }
  robos[4].status = LIVRE;
  robos[4].pos = POS_E;
  robos[4].sentido = PR;
  robos[4].id =(short int)4;
  robos[4].buffer_fundo->Width  = robos[4].imagem->Width;
  robos[4].buffer_fundo->Height = robos[4].imagem->Height;

  robos[5].status = LIVRE;
  robos[5].pos = POS_I;
  robos[5].sentido = OI;
  robos[5].id =(short int)5;
  robos[5].buffer_fundo->Width  = robos[5].imagem->Width;
  robos[5].buffer_fundo->Height = robos[5].imagem->Height;

  // program starts with the ship in the upper left hand corner
  // of the screen, initialize members, and draw the ship there.
  for (Cont = 1; Cont <= NUM_ROBOS; Cont++)
    robos[Cont].velocidade = 3;
  EstSpeed  = 1;
  RoboSpeed = 1;
  SysSpeed  = 1;
  // Obtem as coordenadas de cada pallet
  for (Cont = 0; Cont < TotalPallets; Cont++)
```

```
    Pallet[Cont].PalletCoord = MatCoord[Pallet[Cont].PosicaoAtual.Pixels];
  RoboLocation.x = 444;
  RoboLocation.y = 86;
  // Guarda as dimensoes das imagens
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Pallet[Cont].PalletRect = Rect(0,0, Pallet[Cont].SavPallet->Width,
      Pallet[Cont].SavPallet->Height);
  }
  //Define posicoes iniciais dos robos.
  robos[1].x = 48;
  robos[1].y = 234;
  robos[2].x = 117;
  robos[2].y = 205;
  robos[3].x = 117;
  robos[3].y = 153;
  robos[4].x = 444;
  robos[4].y = 178;
  robos[5].x = 513;
  robos[5].y = 84;
  //Guarda dimensoes das imagens
  for (Cont = 1; Cont <= NUM_ROBOS; Cont++)
    robos[Cont].roboRect = Rect(0,0,robos[Cont].buffer_fundo->Width,robos[Cont].buffer_ fundo->Height);
  for (Cont = 0; Cont < 7; Cont++){
    sensor[Cont].Id =(short int) Cont;
    sensor[Cont].pallet = -1;
    sensor[Cont].FlagPallet = 0;
    sensor[Cont].FlagPeca = 0;
  }
  // parametros de posicao e movimento do robo
  CondAtual[PosicaoRobo] = Esquerda;
  CondAtual[Movimento]   = Alto;
  CondMov[PosicaoRobo]   = Esquerda;
  CondMov[Movimento]     = Subir;
  // parametros de tempo
/*  TempoInicial      = timeGetTime();
  TempoAnterior     = 0;
  TempoAnteriorComp = TempoInicial;
  TempoDecorrido = 0;*/
  // Inicializacao das maquinas
  for (Cont = 1; Cont <= NumMaquina; Cont++){
    Maquina[Cont].rpos1                    = 0;
    Maquina[Cont].spos1                            = 0;
    Maquina[Cont].rpos2                    = 0;
    Maquina[Cont].spos2                            = 0;
    Maquina[Cont].Indice                   = (short int) Cont;
    Maquina[Cont].vazio1                   = 1;
    Maquina[Cont].vazio2                   = 1;
    Maquina[Cont].cheio1                   = 0;
    Maquina[Cont].cheio2                   = 0;
    Maquina[Cont].PecaProcesso.TipoPeca    = 0;  // Note que 0(zero)indica s/peca
    Maquina[Cont].PecaProcesso.IndicePeca = -1;
    Maquina[Cont].Status.StatusAtual       = Espera;
    Maquina[Cont].PalletAnterior           = -1;
    Maquina[Cont].tempo[0]       = 0;
    Maquina[Cont].tempo[1]     = 0;
    Maquina[Cont].taux           = 0;
  }

  for (Cont = 1; Cont <= NumMaquina; Cont++)
    for(Cont1 = 0; Cont1 < TamanhoFila; Cont1++){
      Maquina[Cont].FilaEntrada[Cont1].TipoPeca = 0;
      Maquina[Cont].FilaSaida[Cont1].TipoPeca   = 0;
      Maquina[Cont].FilaEntrada[Cont1].IndicePeca = 0;
      Maquina[Cont].FilaSaida[Cont1].IndicePeca   = 0;
    }
  Maquina[1].Posicao.Metros     = 17.138;
  Maquina[1].Posicao.Pixels     = 720;
  Maquina[1].Status.PosStatus.x = 12;
  Maquina[1].Status.PosStatus.y = 276;
    Maquina[2].Posicao.Metros     = 25.208;
    Maquina[2].Posicao.Pixels     = 1059;
    Maquina[2].Status.PosStatus.x = 144;
    Maquina[2].Status.PosStatus.y = 247;
  Maquina[3].Posicao.Metros     = 26.374;
  Maquina[3].Posicao.Pixels     = 1108;
  Maquina[3].Status.PosStatus.x = 171;
  Maquina[3].Status.PosStatus.y = 81;
    Maquina[4].Posicao.Metros     = 42.156;
    Maquina[4].Posicao.Pixels     = 1771;
    Maquina[4].Status.PosStatus.x = 417;
    Maquina[4].Status.PosStatus.y = 56;
  Maquina[5].Posicao.Metros     = 42.156;
  Maquina[5].Posicao.Pixels     = 1771;
  Maquina[5].Status.PosStatus.x = 431;
  Maquina[5].Status.PosStatus.y = 279;
  // Inicializacao da Entrada e Saida do processo
  InProcesso.Posicao.Metros   = 0;
  InProcesso.Posicao.Pixels   = 0;
  InProcesso.Contador                = 0;
  InProcesso.PalletAnterior   = -1;
  OutProcesso.Posicao.Metros  = 47.321;
  OutProcesso.Posicao.Pixels  = 1988;
  OutProcesso.Contador               = 0;
  OutProcesso.PalletAnterior  = -1;
  // Inicializacao das pecas
  for(Cont2 = 1; Cont2 <= NumPeca; Cont2++){ // 0 eh lixo
    for(Cont = 1; Cont <= TotalPecas[Cont2]; Cont++){
      Peca[Cont2][Cont].PosicaoAtual[0] = 0;
      Peca[Cont2][Cont].PosicaoAtual[1] = 0;
      Peca[Cont2][Cont].TempoPeca[0]    = 0;
      Peca[Cont2][Cont].TempoPeca[1]    = 0;
      Peca[Cont2][Cont].SairProcesso    = false;
      Peca[Cont2][Cont].LinhaProcesso   = 0;
```

```
      for( Cont1 = 0; Cont1 < NumMaquina; Cont1++){
//        Peca[Cont2][Cont].Processos[Cont1].Ref    = 1;
        Peca[Cont2][Cont].Processos[Cont1].TmpInc = 0;
        Peca[Cont2][Cont].Processos[Cont1].TmpFnl = 0;
        Peca[Cont2][Cont].Processos[Cont1].TmpPrc = 10;
        Peca[Cont2][Cont].Processos[Cont1].TmpIn  = 0;
        Peca[Cont2][Cont].Processos[Cont1].TmpOut = 0;
  } } } // Finaliza os For Cont, Cont1 e Cont2
  // Configuracao da sequencia de operacao da peca 1
  for( Cont1 = 1; Cont1 <= TotalPecas[1]; Cont1++){
    Peca[1][Cont1].Processos[0].Sequencia    = 1;
    Peca[1][Cont1].Processos[1].Sequencia    = 2;
    Peca[1][Cont1].Processos[2].Sequencia    = 3;
    Peca[1][Cont1].Processos[3].Sequencia    = 5;
    Peca[1][Cont1].Processos[4].Sequencia    = 0;
  }
  // Configuracao da sequencia de operacao da peca 2
  for( Cont1 = 1; Cont1 <= TotalPecas[2]; Cont1++){
    Peca[2][Cont1].Processos[0].Sequencia    = 1;
    Peca[2][Cont1].Processos[1].Sequencia    = 2;
    Peca[2][Cont1].Processos[2].Sequencia    = 3;
    Peca[2][Cont1].Processos[3].Sequencia    = 4;
    Peca[2][Cont1].Processos[4].Sequencia    = 0;
  }
  // Configuracao da sequencia de operacao da peca 3
  for( Cont1 = 1; Cont1 <= TotalPecas[3]; Cont1++){
    Peca[3][Cont1].Processos[0].Sequencia    = 0;
    Peca[3][Cont1].Processos[1].Sequencia    = 2;
    Peca[3][Cont1].Processos[2].Sequencia    = 3;
    Peca[3][Cont1].Processos[3].Sequencia    = 4;
    Peca[3][Cont1].Processos[4].Sequencia    = 5;
  }
  // Configuracao da sequencia de operacao da peca 4
  for( Cont1 = 1; Cont1 <= TotalPecas[4]; Cont1++){
    Peca[4][Cont1].Processos[0].Sequencia    = 1;
    Peca[4][Cont1].Processos[1].Sequencia    = 2;
    Peca[4][Cont1].Processos[2].Sequencia    = 3;
    Peca[4][Cont1].Processos[3].Sequencia    = 0;
    Peca[4][Cont1].Processos[4].Sequencia    = 0;
  }

  // Inicializacao das variaveis de tempo
  TmpIncSim[0]  = 0;
  TmpIncSim[1]  = timeGetTime();
  TmpAtSim[0]   = 0;
  TmpAtSim[1]   = TmpIncSim[1];
  TmpPausa[0]   = 0;
  TmpPausa[1]   = 0;
  TmpPausa[2]   = 0;
  TmpAntSim[0]  = 0;
  TmpAntSim[1]  = TmpIncSim[1];
  TmpAntAnim[0] = 0;
  TmpAntAnim[1] = TmpIncSim[1];
  Pausa[0]      = false;
  Pausa[1]      = false;
  PausaVar                    = false;

  VelSim = VelocidadeInicial;
  InfoNumber = 0;
  AnimNumber = 0;

  FimDaSimulacao = false;

  // parameters are ready, draw the ship in its initial location
  // and move the image to the screen
  SysSpeedOp = SysSpeed;
  DrawPalletOnBackground();
  DrawRoboOnBackground();
  MoveBitmapToScreen();
  PatchBackground();
  AtualizaTempo();
  play = 0;
  EditTempo->Text = "0.0";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::PlayClick(TObject *Sender){
  play = 1;
  FimDaSimulacao = false;
  Form1->Iniciar1->Checked = true;
  Form1->Parar1->Checked = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::SobreRMS1Click(TObject *Sender){
  AboutBox->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Iniciar1Click(TObject *Sender){
  play = 1;
  Iniciar1->Checked = true;
  Parar1->Checked = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Parar1Click(TObject *Sender){
  play = 0;
  Parar1->Checked = true;
  Iniciar1->Checked = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Criar1Click(TObject *Sender){
  float a1 = 0, a2 = 0, a3 = 0, a4 = 0, a5 = 0;
  int i;

  for(i = 1; i<NumMaquina; i++)
```

```
    a1 = a1 +(Peca[1][i].TempoPeca[1]-Peca[1][i].TempoPeca[0]);
  a1 = a1/NumMaquina;
  for(i = 1; i<NumMaquina; i++)
    a2 = a2 +(Peca[2][i].TempoPeca[1]-Peca[2][i].TempoPeca[0]);
  a2 = a2/NumMaquina;
  for(i = 1; i<NumMaquina; i++)
    a3 = a3 +(Peca[3][i].TempoPeca[1]-Peca[3][i].TempoPeca[0]);
  a3 = a3/NumMaquina;
  for(i = 1; i<NumMaquina; i++)
    a4 = a4 +(Peca[4][i].TempoPeca[1]-Peca[4][i].TempoPeca[0]);
  a4 = a4/NumMaquina;

  DesenhaGrafico1(a1,a2,a3,a4);

  a1 = CalcFloat(EditTempo);
  a1 =(a1-(float)Maquina[1].tempo[0])/a1*100;
  a2 = CalcFloat(EditTempo);
  a2 =(a2-(float)Maquina[2].tempo[0])/a2*100;
  a3 = CalcFloat(EditTempo);
  a3 =(a3-(float)Maquina[3].tempo[0])/a3*100;
  a4 = CalcFloat(EditTempo);
  a4 =(a4-(float)Maquina[4].tempo[0])/a4*100;
  a5 = CalcFloat(EditTempo);
  a5 =(a5-(float)Maquina[5].tempo[0])/a5*100;

  DesenhaGrafico2(a1, a2, a3, a4);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Gerar1Click(TObject *Sender){
  AnsiString S,s1,s2;
  float a;
  int i;

  S = "       Relatório de Simulação";
  editor->Lines->Add(S);
  S = "                              ";
  editor->Lines->Add(S);
  S = "Total de peças processadas:       4";
  editor->Lines->Add(S);
  S = "Tempo total de simulação:         " + EditTempo->Text+ "segundos";
  editor->Lines->Add(S);
  S = "                              ";
  editor->Lines->Add(S);
  S = "Tempo de produção para cada peça: ";
  editor->Lines->Add(S);
  S = "                              ";
  editor->Lines->Add(S);
  S = "Peça 1: " ;
  editor->Lines->Add(S);
  for(i=1; i<=NumMaquina; i++){
    s1 = Peca[1][i].TempoPeca[1]-Peca[1][i].TempoPeca[0];
    s2 = i;
    s1 = " " + s2 + ": " + s1 + "seg";
    editor->Lines->Add(s1);
  }
  S = "                          ";
  editor->Lines->Add(S);
  S = "Peça 2: ";
  editor->Lines->Add(S);
  for(i=1; i<=NumMaquina; i++){
    s1 = Peca[2][i].TempoPeca[1]-Peca[2][i].TempoPeca[0];
    s2 = i;
    s1 = " " +s2+": " + s1 + "seg";
    editor->Lines->Add(s1);
  }
  S = "                  ";
  editor->Lines->Add(S);
  S = "Peça 3: ";
  editor->Lines->Add(S);
  for(i=1; i<=NumMaquina; i++){
    s1 = Peca[3][i].TempoPeca[1]-Peca[3][i].TempoPeca[0];
    s2 = i;
    s1 = " " +s2+": " + s1 + "seg";
    editor->Lines->Add(s1);
  }
  S = "                  ";
  editor->Lines->Add(S);
  S = "Peça 4: ";
  editor->Lines->Add(S);
  for(i=1; i<=NumMaquina; i++){
    s1 = Peca[4][i].TempoPeca[1]-Peca[4][i].TempoPeca[0];
    s2 = i;
    s1 = " " + s2+": " + s1 + "seg";
    editor->Lines->Add(s1);
  }
  S = "                                ";
  editor->Lines->Add(S);
  S = "                                ";
  editor->Lines->Add(S);
  S = "Tempo de Utilização das MÁquinas: ";
  editor->Lines->Add(S);
  S = "Mitutoyo  : ";
  s1 = Maquina[1].tempo[0];
  S = S + s1 + "seg";
  editor->Lines->Add(S);
  S = "Centro CNC: ";
  s1 = Maquina[2].tempo[0];
  S = S + s1 + "seg";
  editor->Lines->Add(S);
  S = "Torno CNC : ";
  s1 = Maquina[3].tempo[0];
  S = S + s1 + "seg";
  editor->Lines->Add(S);
```

```
  S = "Fresa CNC : ";
  s1 = Maquina[4].tempo[0];
  S = S + s1 + "seg";
  editor->Lines->Add(S);
  S = "Torno DNC : ";
  s1 = Maquina[5].tempo[0];
  S = S + s1 + "seg";
  editor->Lines->Add(S);
  S = "                              ";
  editor->Lines->Add(S);
  S = "Porcentagem de tempo ocioso das máquinas";
  editor->Lines->Add(S);
  S = "Mitutoyo  :  ";
  a = CalcFloat(EditTempo);
  a =(a-(float)Maquina[1].tempo[0])/a*100;
  s1 = a;
  S = S + s1 + " %";
  editor->Lines->Add(S);
  S = "Centro CNC:  ";
  a = CalcFloat(EditTempo);
  a =(a-(float)Maquina[2].tempo[0])/a*100;
  s1 = a;
  S = S + s1 + " %";
  editor->Lines->Add(S);
  S = "Torno CNC :  ";
  a = CalcFloat(EditTempo);
  a =(a-(float)Maquina[3].tempo[0])/a*100;
  s1 = a;
  S = S + s1 + " %";
  editor->Lines->Add(S);
  S = "Fresa CNC :  ";
  a = CalcFloat(EditTempo);
  a =(a-(float)Maquina[4].tempo[0])/a*100;
  s1 = a;
  S = S + s1 + " %";
  editor->Lines->Add(S);
  S = "Torno DNC :  ";
  a = CalcFloat(EditTempo);
  a =(a-(float)Maquina[5].tempo[0])/a*100;
  s1 = a;
  S = S + s1 + " %";
  editor->Lines->Add(S);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::SpeedButton1Click(TObject *Sender){
  register int Cont, Cont1, Cont2;
  int a;
  // Inicializacao da esteira
  Esteira.Comprimento.Metros      = CompEstMetro;
  Esteira.Comprimento.Pixels      = TotalPontos;
  Esteira.Velocidade.MetrosPorSeg = VelIncEsteira;
  Esteira.Velocidade.PixelsPorSeg = Esteira.Velocidade.MetrosPorSeg *
      Esteira.Comprimento.Pixels / Esteira.Comprimento.Metros;
  // inicializacao do posicionamento dos pallets e o seu conteudo
  for (Cont = 0; Cont < TotalPallets; Cont++){
    Pallet[Cont].PosicaoAtual.Metros = Cont * Esteira.Comprimento.Metros
                                       / TotalPallets;
    Pallet[Cont].MetroParaPixel();
    Pallet[Cont].PosicaoAnterior = Pallet[Cont].PosicaoAtual;
    Pallet[Cont].Peca.TipoPeca = 0;
    Pallet[Cont].Peca.IndicePeca = 0;
  }
  // inicializa o total de pecas
  TotalPecas[0] = TotalPeca1 + TotalPeca2 + TotalPeca3 + TotalPeca4;
  TotalPecas[1] = TotalPeca1;
  TotalPecas[2] = TotalPeca2;
  TotalPecas[3] = TotalPeca3;
  TotalPecas[4] = TotalPeca4;
  // inicializacao da matriz de posicao
  for (Cont = 0; Cont < TotalPontos; Cont++){
    if (Cont <= 83){
      MatCoord[Cont].x = 516;
      MatCoord[Cont].y = 104 - Cont;
    }
    else{
      if (Cont <= 496){
        MatCoord[Cont].x = 599 - Cont;
        MatCoord[Cont].y = 21;
      }
      else{
        if (Cont <= 834){
          MatCoord[Cont].x = 103;
          MatCoord[Cont].y = Cont - 475;
        }
        else{
          if (Cont <= 883){
            MatCoord[Cont].x = Cont - 731;
            MatCoord[Cont].y = 359;
          }
          else{
            if (Cont <= 916){
              MatCoord[Cont].x = 152;
              MatCoord[Cont].y = 1242 - Cont;
            }
            else{
              if (Cont <= 949){
                MatCoord[Cont].x = 1068 - Cont;
                MatCoord[Cont].y = 326;
              }
              else{
                if (Cont <= 1238){
                  MatCoord[Cont].x = 119;
                  MatCoord[Cont].y = 1275 - Cont;
```

```
            }
            else{
              if (Cont <= 1619){
                MatCoord[Cont].x = Cont - 1119;
                MatCoord[Cont].y = 37;
              }
              else{
                if (Cont <= 1815){
                  MatCoord[Cont].x = 500;
                  MatCoord[Cont].y = Cont - 1582;
                }
                else{
                  if (Cont <= 1865){
                    MatCoord[Cont].x = Cont - 1315;
                    MatCoord[Cont].y = 233;
                  }
                  else{
                    if (Cont <= 1898){
                      MatCoord[Cont].x = 550;
                      MatCoord[Cont].y = 2098 - Cont;
                    }
                    else{
                      if (Cont <= 1932){
                        MatCoord[Cont].x = 2448 - Cont;
                        MatCoord[Cont].y = 200;
                      }
                      else{
                        MatCoord[Cont].x = 516;
                        MatCoord[Cont].y = 2132 - Cont;
                      } // else Cont <= 1932
                    } // else Connt <= 1898
                  } // else Cont <= 1865
                } // else Cont <= 1815
              } // else Cont <= 1619
            } // else Cont <= 1238
          } // else Cont <= 949
        } // else Cont <= 916
      } // else Cont <= 883
    } // else Cont <= 834
  } // else Cont <= 496
 } // else Cont <= 83
} // for Cont
// inicializacao dos bitmaps
RoboSaveBack                        = new Graphics::TBitmap;
for (Cont = 0; Cont < TotalPallets; Cont++){
  if (Cont < NumStatus)
    ImagStatus[Cont]                = new Graphics::TBitmap;
  if (Cont < NumMaquina){
    Maquina[Cont+1].Status.ImgStatus = new Graphics::TBitmap;
    Maquina[Cont+1].Status.MscStatus = new Graphics::TBitmap;
    Maquina[Cont+1].Status.MscStatus->LoadFromResourceName((int)HInstance,"STATUSMASK");
  }
  if (Cont < NumPosicao){
    Robo[Cont]                    = new Graphics::TBitmap;
    RoboMask[Cont]                        = new Graphics::TBitmap;
  }
  if (Cont < (NumPeca+1))
    ImgPeca[Cont]                 = new Graphics::TBitmap;
  Pallet[Cont].ImgPallet          = new Graphics::TBitmap;
  Pallet[Cont].MscPallet          = new Graphics::TBitmap;
  Pallet[Cont].SavPallet          = new Graphics::TBitmap;
}
// carregamento das imagens bitmaps
ImgPeca[0]    ->LoadFromResourceName((int)HInstance,"PALLET");
ImgPeca[1]    ->LoadFromResourceName((int)HInstance,"PECA1");
ImgPeca[2]    ->LoadFromResourceName((int)HInstance,"PECA2");
ImgPeca[3]    ->LoadFromResourceName((int)HInstance,"PECA3");
ImgPeca[4]    ->LoadFromResourceName((int)HInstance,"PECA4");
ImagStatus[0]->LoadFromResourceName((int)HInstance,"GREEN");
ImagStatus[1]->LoadFromResourceName((int)HInstance,"YELLOW");
ImagStatus[2]->LoadFromResourceName((int)HInstance,"RED");
for (Cont = 0; Cont < TotalPallets; Cont++){
  Pallet[Cont].ImgPallet = ImgPeca[0];
  Pallet[Cont].MscPallet->LoadFromResourceName((int)HInstance,"PALLETMASK");
}
//Associa imagens dos robos.
robos[1].AssociaImagem("ROBO_RIGHT", "ROBO_RIGHT_MASK");
robos[2].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
robos[3].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
robos[4].AssociaImagem("ROBO_RIGHT", "ROBO_RIGHT_MASK");
//robos[5].AssociaImagem("ROBO_LEFT", "ROBO_LEFT_MASK");
//SavedBackgroundRegion will store the portion of the background
//that gets destroyed when the ship is copied in. Make it the same
//size as the ship.
for (Cont = 0; Cont < TotalPallets; Cont++){
  Pallet[Cont].SavPallet->Width  = Pallet[Cont].ImgPallet->Width;
  Pallet[Cont].SavPallet->Height = Pallet[Cont].ImgPallet->Height;
}
for (Cont = 1; Cont <= 3; Cont++){
  robos[Cont].status = LIVRE;
  robos[Cont].pos = POS_E;
  robos[Cont].sentido = ME;
  robos[Cont].id =(short int)(Cont);
  robos[Cont].buffer_fundo->Width  = robos[Cont].imagem->Width;
  robos[Cont].buffer_fundo->Height = robos[Cont].imagem->Height;
}
  robos[4].status = LIVRE;
  robos[4].pos = POS_E;
  robos[4].sentido = PR;
  robos[4].id =(short int)4;
  robos[4].buffer_fundo->Width  = robos[4].imagem->Width;
  robos[4].buffer_fundo->Height = robos[4].imagem->Height;
```

```
    /*robos[5].status = LIVRE;
    robos[5].pos = POS_L;
    robos[5].sentido = OI;
    robos[5].id =(short int)5;
    robos[5].buffer_fundo->Width  = robos[5].imagem->Width;
    robos[5].buffer_fundo->Height = robos[5].imagem->Height;*/

  // program starts with the ship in the upper left hand corner
  // of the screen, initialize members, and draw the ship there.
  //for (Cont = 1; Cont <= NUM_ROBOS; Cont++)
  a= CalcInt(Edit26);
  robos[1].velocidade = a;
  a= CalcInt(Edit28);
  robos[2].velocidade = a;
  a= CalcInt(Edit30);
  robos[3].velocidade = a;
  a= CalcInt(Edit27);
  robos[4].velocidade = a;

  a= CalcInt(Edit25);
  EstSpeed  = a;
  RoboSpeed = 1;
  SysSpeed  = 1;
  // Obtem as coordenadas de cada pallet
  for (Cont = 0; Cont < TotalPallets; Cont++)
    Pallet[Cont].PalletCoord = MatCoord[Pallet[Cont].PosicaoAtual.Pixels];
  RoboLocation.x = 444;
  RoboLocation.y = 86;
  // Guarda as dimensoes das imagens
  for (Cont = 0; Cont < TotalPallets; Cont++){Pallet[Cont].PalletRect = Rect(0,0,
               Pallet[Cont].SavPallet->Width, Pallet[Cont].SavPallet->Height);
  }
  //Define posicoes iniciais dos robos.
  robos[1].x = 48;
  robos[1].y = 234;
  robos[2].x = 117;
  robos[2].y = 205;
  robos[3].x = 117;
  robos[3].y = 153;
  robos[4].x = 444;
  robos[4].y = 178;
  robos[5].x = 513;
  robos[5].y = 84;
  //Guarda dimensoes das imagens
  for (Cont = 1; Cont <= NUM_ROBOS; Cont++)
    robos[Cont].roboRect = Rect(0,0,robos[Cont].buffer_fundo->Width,robos[Cont].buffer_ fundo->Height);
  for (Cont = 0; Cont < 7; Cont++){
    sensor[Cont].Id =(short int) Cont;
    sensor[Cont].pallet = -1;
    sensor[Cont].FlagPallet = 0;
    sensor[Cont].FlagPeca = 0;
  }
  // parametros de posicao e movimento do robo
  CondAtual[PosicaoRobo] = Esquerda;
  CondAtual[Movimento]   = Alto;
  CondMov[PosicaoRobo]   = Esquerda;
  CondMov[Movimento]     = Subir;
  // parametros de tempo
  TempoAnterior     = 0;
  TempoAnteriorComp = TempoInicial;
  TempoDecorrido = 0;*/
  // Inicializacao das maquinas
  for (Cont = 1; Cont <= NumMaquina; Cont++){
    Maquina[Cont].rpos1                    = 0;
    Maquina[Cont].spos1                            = 0;
    Maquina[Cont].rpos2                    = 0;
    Maquina[Cont].spos2                            = 0;
    Maquina[Cont].Indice                   = (short int) Cont;
    Maquina[Cont].vazio1                   = 1;
    Maquina[Cont].vazio2                   = 1;
    Maquina[Cont].cheio1                   = 0;
    Maquina[Cont].cheio2                   = 0;
    Maquina[Cont].PecaProcesso.TipoPeca    = 0;  // Note que 0(zero)indica s/peca
    Maquina[Cont].PecaProcesso.IndicePeca  = -1;
    Maquina[Cont].Status.StatusAtual       = Espera;
    Maquina[Cont].tempo[0]       = 0;
    Maquina[Cont].tempo[1]     = 0;
    Maquina[Cont].taux           = 0;
    Maquina[Cont].PalletAnterior           = -1;
  }
  for (Cont = 1; Cont <= NumMaquina; Cont++)
    for(Cont1 = 0; Cont1 < TamanhoFila; Cont1++){
      Maquina[Cont].FilaEntrada[Cont1].TipoPeca = 0;
      Maquina[Cont].FilaSaida[Cont1].TipoPeca   = 0;
      Maquina[Cont].FilaEntrada[Cont1].IndicePeca = 0;
      Maquina[Cont].FilaSaida[Cont1].IndicePeca   = 0;
    }
  Maquina[1].Posicao.Metros      = 17.138;
  Maquina[1].Posicao.Pixels      = 720;
  Maquina[1].Status.PosStatus.x = 12;
  Maquina[1].Status.PosStatus.y = 276;
    Maquina[2].Posicao.Metros      = 25.208;
    Maquina[2].Posicao.Pixels      = 1059;
    Maquina[2].Status.PosStatus.x = 144;
    Maquina[2].Status.PosStatus.y = 247;
  Maquina[3].Posicao.Metros      = 26.374;
  Maquina[3].Posicao.Pixels      = 1108;
  Maquina[3].Status.PosStatus.x = 171;
  Maquina[3].Status.PosStatus.y = 81;
    Maquina[4].Posicao.Metros      = 42.156;
    Maquina[4].Posicao.Pixels      = 1771;
    Maquina[4].Status.PosStatus.x = 417;
    Maquina[4].Status.PosStatus.y = 56;
```

```
  Maquina[5].Posicao.Metros     = 42.156;
  Maquina[5].Posicao.Pixels     = 1771;
  Maquina[5].Status.PosStatus.x = 431;
  Maquina[5].Status.PosStatus.y = 279;
  // Inicializacao da Entrada e Saida do processo
  InProcesso.Posicao.Metros   = 0;
  InProcesso.Posicao.Pixels   = 0;
  InProcesso.Contador               = 0;
  InProcesso.PalletAnterior   = -1;
  OutProcesso.Posicao.Metros  = 47.321;
  OutProcesso.Posicao.Pixels  = 1988;
  OutProcesso.Contador              = 0;
  OutProcesso.PalletAnterior  = -1;
  // Inicializacao das pecas
  for(Cont2 = 1; Cont2 <= NumPeca; Cont2++){ // 0 eh lixo
    for(Cont = 1; Cont <= TotalPecas[Cont2]; Cont++){
      Peca[Cont2][Cont].PosicaoAtual[0] = 0;
      Peca[Cont2][Cont].PosicaoAtual[1] = 0;
      Peca[Cont2][Cont].TempoPeca[0]    = 0;
      Peca[Cont2][Cont].TempoPeca[1]    = 0;
      Peca[Cont2][Cont].SairProcesso    = false;
      Peca[Cont2][Cont].LinhaProcesso   = 0;
      for( Cont1 = 0; Cont1 < NumMaquina; Cont1++){
//        Peca[Cont2][Cont].Processos[Cont1].Ref    = 1;
        Peca[Cont2][Cont].Processos[Cont1].TmpInc = 0;
        Peca[Cont2][Cont].Processos[Cont1].TmpFnl = 0;
        Peca[Cont2][Cont].Processos[Cont1].TmpPrc = 10;
        Peca[Cont2][Cont].Processos[Cont1].TmpIn  = 0;
        Peca[Cont2][Cont].Processos[Cont1].TmpOut = 0;
  } } } // Finaliza os For Cont, Cont1 e Cont2
  // Configuracao da sequencia de operacao da peca 1
  for( Cont1 = 1; Cont1 <= TotalPecas[1]; Cont1++){
    a= CalcInt(Edit3);
    Peca[1][Cont1].Processos[0].Sequencia    = a;
    a= CalcInt(Edit4);
    Peca[1][Cont1].Processos[1].Sequencia    = a;
    a= CalcInt(Edit5);
    Peca[1][Cont1].Processos[2].Sequencia    = a;
    a= CalcInt(Edit6);
    Peca[1][Cont1].Processos[3].Sequencia    = a;
    a= CalcInt(Edit7);
    Peca[1][Cont1].Processos[4].Sequencia    = a;
  }
  // Configuracao da sequencia de operacao da peca 2
  for( Cont1 = 1; Cont1 <= TotalPecas[2]; Cont1++){
    a= CalcInt(Edit8);
    Peca[2][Cont1].Processos[0].Sequencia    = a;
    a= CalcInt(Edit9);
    Peca[2][Cont1].Processos[1].Sequencia    = a;
    a= CalcInt(Edit10);
    Peca[2][Cont1].Processos[2].Sequencia    = a;
    a= CalcInt(Edit11);
    Peca[2][Cont1].Processos[3].Sequencia    = a;
    a= CalcInt(Edit12);
    Peca[2][Cont1].Processos[4].Sequencia    = a;
  }
  // Configuracao da sequencia de operacao da peca 3
  for( Cont1 = 1; Cont1 <= TotalPecas[3]; Cont1++){
    a= CalcInt(Edit13);
    Peca[3][Cont1].Processos[0].Sequencia    = a;
    a= CalcInt(Edit14);
    Peca[3][Cont1].Processos[1].Sequencia    = a;
    a= CalcInt(Edit15);
    Peca[3][Cont1].Processos[2].Sequencia    = a;
    a= CalcInt(Edit16);
    Peca[3][Cont1].Processos[3].Sequencia    = a;
    a= CalcInt(Edit17);
    Peca[3][Cont1].Processos[4].Sequencia    = a;
  }
  // Configuracao da sequencia de operacao da peca 4
  for( Cont1 = 1; Cont1 <= TotalPecas[4]; Cont1++){
    a= CalcInt(Edit1);
    Peca[4][Cont1].Processos[0].Sequencia    = a;
    a= CalcInt(Edit2);
    Peca[4][Cont1].Processos[1].Sequencia    = a;
    a= CalcInt(Edit20);
    Peca[4][Cont1].Processos[2].Sequencia    = a;
    a= CalcInt(Edit18);
    Peca[4][Cont1].Processos[3].Sequencia    = a;
    a= CalcInt(Edit19);
    Peca[4][Cont1].Processos[4].Sequencia    = a;
  }
  // Inicializacao das variaveis de tempo
  TmpIncSim[0]  = 0;
  TmpIncSim[1]  = timeGetTime();
  TmpAtSim[0]   = 0;
  TmpAtSim[1]   = TmpIncSim[1];
  TmpPausa[0]   = 0;
  TmpPausa[1]   = 0;
  TmpPausa[2]   = 0;
  TmpAntSim[0]  = 0;
  TmpAntSim[1]  = TmpIncSim[1];
  TmpAntAnim[0] = 0;
  TmpAntAnim[1] = TmpIncSim[1];
  Pausa[0]      = false;
  Pausa[1]      = false;
  PausaVar      = false;

  VelSim = VelocidadeInicial;
  InfoNumber = 0;
  AnimNumber = 0;

  FimDaSimulacao = true;
```

```
  // parameters are ready, draw the ship in its initial location
  // and move the image to the screen
  SysSpeedOp = SysSpeed;
  DrawPalletOnBackground();
  DrawRoboOnBackground();
  MoveBitmapToScreen();
  PatchBackground();
  AtualizaTempo();
  CriarArquivo();
  EditTempo->Text = "0.0";
  play = 1;
}
```

RELOGIO.H

```
//---------------------------------------------------------------------------
#ifndef relogioH
#define relogioH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------------
class TForm4 : public TForm{
__published:        // IDE-managed Components
        TEdit *EditTempo;
private:  // User declarations
public:             // User declarations
        __fastcall TForm4(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TForm4 *Form4;
//---------------------------------------------------------------------------
#endif
```

RELOGIO.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "relogio.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TForm4 *Form4;
//---------------------------------------------------------------------------
__fastcall TForm4::TForm4(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

ROBOS.H

```
//---------------------------------------------------------------------------
#ifndef robosH
#define robosH
//---------------------------------------------------------------------------
#endif
```

ROBOS.CPP

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "MAINFORM.h"
//---------------------------------------------------------------------------
clRobo::clRobo(){
  imagem       = new Graphics::TBitmap;
  mascara      = new Graphics::TBitmap;
  buffer_fundo = new Graphics::TBitmap;
}
//---------------------------------------------------------------------------
clRobo::~clRobo(){
    delete buffer_fundo;
  delete mascara;
  delete imagem;
}
//---------------------------------------------------------------------------
void clRobo::AssociaImagem(AnsiString imagem1, AnsiString imagem2){
  imagem     ->LoadFromResourceName((int)HInstance, imagem1);
  mascara    ->LoadFromResourceName((int)HInstance, imagem2);
}
//---------------------------------------------------------------------------
void clRobo::PatchBackground(void){
  TRect LocationRect = Rect(x, y, x+imagem->Width, y+imagem->Height);
  Form1->Background.imagem->Canvas->CopyMode = cmSrcCopy;
  Form1->Background.imagem->Canvas->CopyRect(LocationRect, buffer_fundo->Canvas, roboRect);
}
//---------------------------------------------------------------------------
void clRobo::MoveRoboLocation(void){
  if (id == 1){
    if(x==20 && y==219){
      x = 1;
      y = 234;
    }
    if(sentido == ME && x<=48){
      x = x + velocidade * Form1->VelSim;
      if(x>=48){
        pos = POS_E;
        x = 48;
      }
    }
    if(sentido ==EM && x>=1){
      x = x - velocidade * Form1->VelSim;
      if(x<=1){
```

```
        pos = POS_M;
        x = 1;
      }
    }
    if(x>1 && x<48)
      pos = POS_P;
    if(x==1){
      if(TipoPeca == 0)
        AssociaImagem("ROBO_DOWN","ROBO_DOWN_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_DOWN_P1","ROBO_DOWN_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_DOWN_P2","ROBO_DOWN_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_DOWN_P3","ROBO_DOWN_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_DOWN_P4","ROBO_DOWN_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
      x=20;
      y=219;
    }
    else{
      if(TipoPeca == 0)
        AssociaImagem("ROBO_RIGHT","ROBO_RIGHT_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
    }
  }
  if (id == 2){
    if(x==171 && y==190){
      x = 153;
      y = 205;
    }
    if(sentido == ME && x>=117){
      x = x - velocidade * Form1->VelSim;
      if(x<=117){
        pos = POS_E;
        x = 117;
      }
    }
    if(sentido ==EM && x<=153){
      x = x + velocidade * Form1->VelSim;
      if(x>=153){
        pos = POS_M;
        x = 153;
      }
    }
    if(x>117 && x<153)
      pos = POS_P;
    if(x==153){
      if(TipoPeca == 0)
        AssociaImagem("ROBO_DOWN","ROBO_DOWN_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_DOWN_P1","ROBO_DOWN_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_DOWN_P2","ROBO_DOWN_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_DOWN_P3","ROBO_DOWN_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_DOWN_P4","ROBO_DOWN_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
      x=171;
      y=190;
    }
    else{
      if(TipoPeca == 0)
        AssociaImagem("ROBO_LEFT","ROBO_LEFT_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
    }
  }
  if (id == 3){
    if(x==211 && y==135){
      x = 193;
      y = 153;
    }
    if(sentido == ME && x>=117){
      x = x - velocidade * Form1->VelSim;
      if(x<=117){
        pos = POS_E;
        x = 117;
```

```
      }
    }
    if(sentido ==EM && x<=193){
      x = x + velocidade * Form1->VelSim;
      if(x>=193){
        pos = POS_M;
        x = 193;
      }
    }
    if(x>117 && x<193)
      pos = POS_P;
    if(x==193){
      if(TipoPeca == 0)
        AssociaImagem("ROBO_UP","ROBO_UP_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_UP_P1","ROBO_UP_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_UP_P2","ROBO_UP_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_UP_P3","ROBO_UP_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_UP_P4","ROBO_UP_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
      x=211;
      y=135;
    }
    else{
      if(TipoPeca == 0)
        AssociaImagem("ROBO_LEFT","ROBO_LEFT_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
    }
  }
  if (id == 4){
    if(x==459 && y==229){
      x = 444;
      y = 244;
    }
    if(sentido == EF && y>=110){
      y = y - velocidade * Form1->VelSim;
      if(y<=110){
        //sentido = FE;
        pos = POS_F;
        y = 110;
      }
    }
    if(sentido == FE && y<=178){
      y = y + velocidade * Form1->VelSim;
      if(y>=178){
        //sentido = EF;
        pos = POS_E;
        y = 178;
      }
    }
    if(sentido == ET && y<=244){
      y = y + velocidade * Form1->VelSim;
      if(y>=244){
        //sentido = TE;
        pos = POS_T;
        y = 244;
      }
    }
    if(sentido == TE && y>=178){
      y = y - velocidade * Form1->VelSim;
      if(y<=178){
        //sentido = ET;
        pos = POS_E;
        y = 178;
      }
    }
    if(sentido == FT && y<=244){
      y = y + velocidade * Form1->VelSim;
      if(y>=244){
        //sentido = TF;
        pos = POS_T;
        y = 244;
      }
    }
    if(sentido == TF && y>=110){
      y = y - velocidade * Form1->VelSim;
      if(y<=110){
        //sentido = FT;
        pos = POS_F;
        y = 110;
      }
    }
    if(y>110 && y<244 && y!=178)
      pos = POS_P;
    if(y == 244){
      if(TipoPeca == 0)
        AssociaImagem("ROBO_DOWN","ROBO_DOWN_MASK");
      if(TipoPeca == 1)
```

```
        AssociaImagem("ROBO_DOWN_P1","ROBO_DOWN_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_DOWN_P2","ROBO_DOWN_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_DOWN_P3","ROBO_DOWN_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_DOWN_P4","ROBO_DOWN_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
      x = 459;
      y = 229;
    }
    else{
      if(TipoPeca == 0)
        AssociaImagem("ROBO_RIGHT","ROBO_RIGHT_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_RIGHT_P1","ROBO_RIGHT_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_RIGHT_P2","ROBO_RIGHT_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_RIGHT_P3","ROBO_RIGHT_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_RIGHT_P4","ROBO_RIGHT_P_MASK");
      buffer_fundo->Width  = imagem->Width;
      buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
    }
    if(y == 110){
      if(TipoPeca == 0)
        AssociaImagem("ROBO_LEFT","ROBO_LEFT_MASK");
      if(TipoPeca == 1)
        AssociaImagem("ROBO_LEFT_P1","ROBO_LEFT_P_MASK");
      if(TipoPeca == 2)
        AssociaImagem("ROBO_LEFT_P2","ROBO_LEFT_P_MASK");
      if(TipoPeca == 3)
        AssociaImagem("ROBO_LEFT_P3","ROBO_LEFT_P_MASK");
      if(TipoPeca == 4)
        AssociaImagem("ROBO_LEFT_P4","ROBO_LEFT_P_MASK");
      buffer_fundo->Width  = imagem->Width;
        buffer_fundo->Height = imagem->Height;
      roboRect = Rect(0,0,buffer_fundo->Width,buffer_fundo->Height);
      x=444;
      y=110;
    }
  }
}
//---------------------------------------------------------------------------
void clRobo::DrawRoboOnBackground(void){
  TRect LocationRect = Rect(x, y, x+imagem->Width, y+imagem->Height);
  buffer_fundo->Canvas->CopyRect( roboRect, Form1->Background.imagem->Canvas, LocationRect);
  Form1->Background.imagem->Canvas->CopyMode = cmSrcAnd;
  Form1->Background.imagem->Canvas->CopyRect(LocationRect, mascara->Canvas, roboRect);
  Form1->Background.imagem->Canvas->CopyMode = cmSrcPaint;
  Form1->Background.imagem->Canvas->CopyRect(LocationRect, imagem->Canvas, roboRect);
}
//---------------------------------------------------------------------------
```

SOBRE.H

```
//---------------------------------------------------------------------------
#ifndef SobreH
#define SobreH
//---------------------------------------------------------------------------
#include <vcl\System.hpp>
#include <vcl\Windows.hpp>
#include <vcl\SysUtils.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Graphics.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TAboutBox : public TForm
{
__published:
        TPanel *Panel1;
        TImage *ProgramIcon;
        TLabel *ProductName;
        TLabel *Version;
        TLabel *Copyright;
        TLabel *Comments;
        TButton *OKButton;
        TBevel *Bevel1;
        TBevel *Bevel2;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TBevel *Bevel3;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
private:
public:
        virtual __fastcall TAboutBox(TComponent* AOwner);
};
//---------------------------------------------------------------------------
extern TAboutBox *AboutBox;
//---------------------------------------------------------------------------
#endif
```

SOBRE.CPP

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Sobre.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TAboutBox *AboutBox;
//---------------------------------------------------------------------
__fastcall TAboutBox::TAboutBox(TComponent* AOwner)
        : TForm(AOwner)
{
}
//---------------------------------------------------------------------
#ifndef startH
#define startH
//---------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------
class TSplashScreen : public TForm
{
__published:        // IDE-managed Components

        TPanel *Panel1;
        TImage *Image1;
        void __fastcall FormPaint(TObject *Sender);
private:  // User declarations
public:             // User declarations
        virtual __fastcall TSplashScreen(TComponent* Owner);
};
//---------------------------------------------------------------------
extern TSplashScreen *SplashScreen;
//---------------------------------------------------------------------
#endif

//---------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "start.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TSplashScreen *SplashScreen;
//---------------------------------------------------------------------
__fastcall TSplashScreen::TSplashScreen(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TSplashScreen::FormPaint(TObject *Sender)
{
        double a;
    while (a<1000000000) a++;
}
//---------------------------------------------------------------------
```