UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GABRIELL HENRIQUE MARQUES HOTT

SUPERVISÓRIO E CONTROLADOR DE REABASTECIMENTO DE UM ALTO-FORNO

VIÇOSA
2022

GABRIELL HENRIQUE MARQUES HOTT

SUPERVISÓRIO E CONTROLADOR DE REABASTECIMENTO DE UM ALTO-FORNO

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 402 – Projeto de Engenharia II – e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. André Gomes Tôrres.

VIÇOSA
2022

GABRIELL HENRIQUE MARQUES HOTT

SUPERVISÓRIO E CONTROLADOR DE REABASTECIMENTO DE UM ALTO-FORNO

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 402 – Projeto de Engenharia II – e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovada em 09 de agosto de 2022.

COMISSÃO EXAMINADORA

Prof. Dr. André Gomes Tôrres - Orientador
Universidade Federal de Viçosa

Prof. Dr. Denilson Eduardo Rodrigues
Universidade Federal de Viçosa

Me. Lucas Johys Ribeiro Silva
Universidade de São Paulo

“No one is you and that is your biggest power.”

(Dave Grohl)

Agradecimentos

Inicialmente gostaria de agradecer ao meu pai Lúcio e minha mãe Henny por todo o auxílio e suporte, além de me proporcionarem essa oportunidade de realizar esse objetivo com muito carinho e ímpeto. Aos meus avós Hélio e Lourival por toda a ajuda que deram nesta caminhada também, vocês são eternos em nossos corações.

Agradeço também aos grandes amigos que fiz nessa caminhada, especialmente eles que nunca mediram esforços e sempre trouxeram infinitas risadas, o meu Swing formado por Sossai, Rabelo, Danilo, Alexandre, Terê, Timo, Faccioli e Gui. Por último e mais importante sim, meu irmão que me acompanha desde o início Ricardo Ferreira.

Agradeço também a minha amada Lateria e banda, vocês marcaram muito essa etapa e levo vocês comigo para sempre. Aos grandes amigos que fiz no curso Erick, Ramon, Jonys e Mathias. A todos professores do DEL especialmente Rodolpho e André que sempre estiveram dispostos a ajudar e ensinar.

Resumo

Processos automatizados estão cada vez mais em evidência no mundo moderno e seu estudo é relevante na Engenharia Elétrica, portanto, este projeto visa a criação de um sistema de supervisão de um alto-forno em processo de combustão. Utilizando a linguagem de programação *C#* e técnicas de controle, será desenvolvido um supervisório que manterá o Alto-Forno sempre abastecido e sua escória constantemente descartada, além de um sistema de comunicação para monitoramento remoto online. O programa visa abranger técnicas de programação paralela de maneira que *threads* interajam entre si e com o sistema remoto a fim de representar um processo totalmente automático e funcional a fim de trazer uma alternativa para gerenciamento deste processo que compõe uma importante etapa na produção metalúrgica.

Palavras-chaves: Alto-forno, Ferro Gusa, Automação, Automação em Tempo Real, Threads, Metalurgia.

Abstract

Automated processes are increasingly in evidence in the modern world and their study is relevant in Electrical Engineering, therefore, this project aims to create a system for supervising a blast furnace in the combustion process. Using the C # programming language and control techniques, a supervisory will be developed that will keep the oven always filled and its slag constantly discarded, as well as a communication system for remote monitoring via website. The program aims to cover parallel programming techniques so that hreads interact with each other and with the remote system in order to represent a fully automatic and functional process in order to present a management alternative in an important metallurgical process.

Keywords: Blast furnace, Pig Iron, Automation, Real-Time Automation, Threads, Metallurgy.

Sumário

1 Introdução 10
1.1 O Alto-forno 11
1.2 Visual Studio e C# 13
1.3 Threads e Programação Paralela 14
1.4 Objetivos 15

2 Materiais e Métodos 17
2.1 Materiais 17
2.2 Metodologia 17

3 Resultados e Discussão 25

4 Conclusões 28

5 Referências Bibliográficas 29

Apêndice A – Código base da aplicação 30

Apêndice B – Código do servidor HTTP 39

Lista de Figuras

Figura 1 - Matérias-primas e produtos do Alto-forno. 12
Figura 2 - Principais componentes do Alto-forno. 12
Figura 3 - Alto-forno real. 13
Figura 4 - Blocos de tarefas sendo executados por um thread. 14
Figura 5 - Dois conjuntos de tarefas gerenciados por duas threads. 15
Figura 6 - Rotina de desenho de carros. 18
Figura 7 - Rotina do botão de Novo Carrinho. 19
Figura 8 - Rotina do botão Salvar Relatório. 20
Figura 9 - Rotina do botão Manual/Automático. 21
Figura 10 - Saída do Visual Studio. 22
Figura 11 - Rotina de controle de posições na zona de exclusão. 23
Figura 12 - Saída do programa após 30 minutos de operação. 25
Figura 13 - Servidor Http operando. 26
Figura 14 - Teste computacional em Notebook. 27
Figura 15 – Teste computacional em Desktop 27

1 Introdução

Os primeiros Altos-fornos conhecidos datam do século XV *(RIZZO, 2009)* e são instrumentos essenciais no processamento de metais. Para que se mantenham em aquecimento e deem continuidade aos processos metalúrgicos, devem ser constantemente abastecidos com insumos combustíveis. Baseado nesses pressupostos, uma boa alternativa para controle e monitoramento desse tipo de processo é a aplicação da Automação em Tempo Real que por sua vez trabalha com programação multitarefas e da utilização de *threads* para execução de diferentes tarefas simultaneamente.

As *threads* permitem execuções em paralelo de diversos processos e são unidades que podem ser controladas de maneira independente *(MACHADO e MAIA, 2002)* por diferentes processadores e, portanto, são um recurso adequado para construção de um *software* com intuito de controlar um Alto-Forno, visto que a interação de tarefas é essencial para o bom funcionamento de um controle automatizado.

O processo de automação por sua vez consiste em mecanismos que verificam seu próprio funcionamento, efetuando medições e introduzindo correções, sem a necessidade da interferência do homem *(LIMA, 2003)*. Diante de tal conceito e das disponibilidades de recursos fornecida pela linguagem *C#* é possível construir tal *software* supracitado com riqueza de detalhes para uma aplicação enfática de um supervisório funcional. Nesse sentido o presente trabalho utiliza de técnicas de programação *multitask* para construção de um *software* para controle e supervisão do abastecimento de insumos de um Alto-Forno em processo corrente.

1.1 O Alto-forno

Em processos metalúrgicos, para obter-se aço em sua forma final, são necessárias duas etapas. A primeira consiste na obtenção de um material chamado Ferro Gusa que posteriormente será transformado em aço, sendo que este último processo foge os escopos deste texto. Para a primeira etapa, é possível obter o material em questão por meio do aquecimento em Altos-fornos ou uma redução direta do minério de ferro, sendo assim, o Alto-forno é uma das maneiras essenciais de obtenção de Ferro Gusa. *(RIZZO, 2009)*

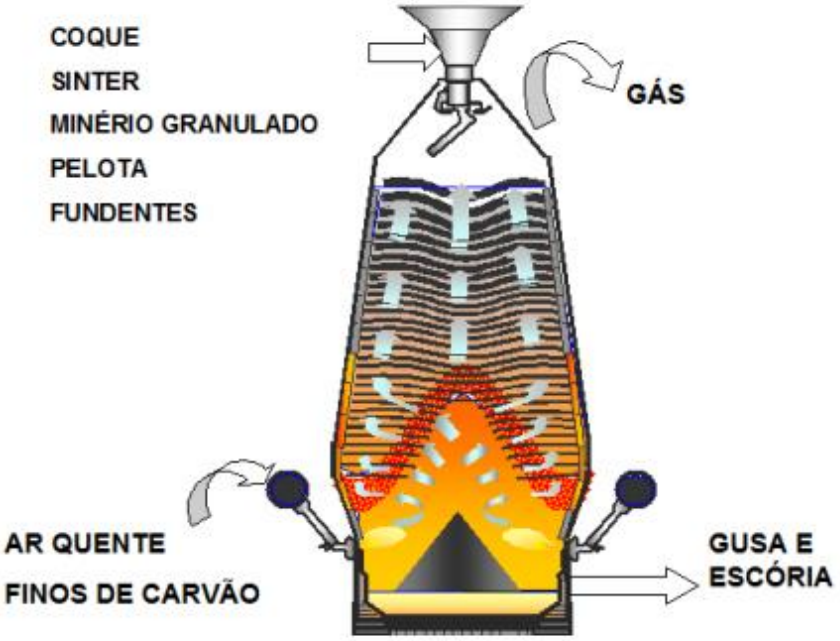
Um AF ou Alto-Forno é uma espécie de reator metalúrgico e que ao alimentar o forno com insumos de minério de ferro aglomerado em forma de Sínter ou Pelotas por meio de guindastes, será obtido na saída abaixo o Ferro na forma líquida (Gusa) além de escórias que poderão ser descartadas deste processo e utilizada em outros como na produção de cimento. *(INFOMET, 1998)*

Através de um processo chamado de fusão redutora de minérios de ferro em presença de carvão vegetal ou coque e fundentes, estes insumos são adicionados ao AF e passam pelo topo em um movimento de descida de maneira que gases resultantes da combustão no fundo do equipamento transformam o material. O fundo possui temperaturas elevadas que podem chegar a 2000ºC.

As Figuras 1 e 2 a seguir representam desenhos esquemáticos do alto-forno ressaltando suas principais estruturas matérias-primas as quais estão apresentadas as

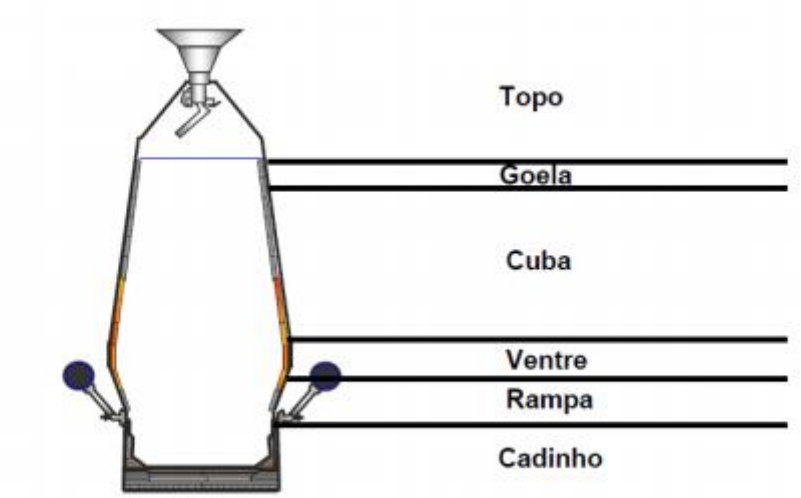
formas de insumos e combustíveis de minério e carvão, além de todos componentes que atuam nas etapas de fundição.

Figura 1 - Matérias-primas e produtos do Alto-forno.



Fonte: (ARAÚJO, 1997)

Figura 2 - Principais componentes do Alto-forno.



Fonte: (ARAÚJO, 1997)

A Figura 3 por sua vez retrata um Alto-forno real em funcionamento.

Figura 3 - Alto-forno real.

Fonte: (COSIAÇO, 2016)

Dadas as características e principais componentes dos Altos-fornos, na próxima seção serão apresentadas algumas das técnicas de programação que serão utilizadas como ferramentas de desenvolvimento.

1.2 Visual Studio e C#

O *Visual Studio* é um *software* gratuito produzido pela *Microsoft* e consiste num ambiente de desenvolvimento integrado que por meio das ferramentas do .NET Framework permite trabalhar com algumas linguagens de programação, a se citar a que será abordada neste trabalho, chamada *C# ("C Sharp").*

Criada para tornar mais acessível a produção de *softwares* aos públicos interessados *(DEITAL, 2003)* conta com uma estrutura capaz de criar janelas e adicionar componentes como *timer*, caixas de texto, fazer desenhos via código, trabalhar com importação de inúmeras bibliotecas úteis, baseando-se na versão .NET framework utilizada, que por sua vez é uma plataforma unificada de ferramentas criada pela

Microsoft para que o código gerado possa ser executado em qualquer dispositivo que possua um *framework.*

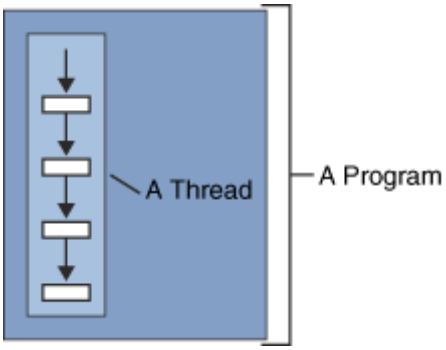
1.3 Threads e Programação Paralela

Um recurso útil para ganho de performance por meio do desafogamento do núcleo do processador de um computador é a programação paralela *(DOMINGOS, 2011)*, que consiste em programar dividindo os recursos computacionais de maneira que cada tarefa seja executada durante um período de tempo, de forma que vários servidores atendam diferentes clientes ao mesmo tempo, reduzindo o tempo de resposta para cada cliente *(DOMINGOS, 2011).*

Baseada nos conceitos do paralelismo, os *threads* são uma ferramenta muito útil para trabalhar com múltiplos processos de maneira otimizada e simultânea. *Threads* permitem que múltiplas execuções ocorram no mesmo ambiente do aplicativo com um elevado grau de independência uma da outra, portanto, se temos muitos *threads* executando em paralelo no sistema é análogo a múltiplos aplicativos executando em paralelo em um computador. *(MEDEIROS, 2007).*

A Figura 4 mostra um esquema simplificado do funcionamento de *threads*.

Figura 4 - Blocos de tarefas sendo executados por um thread.

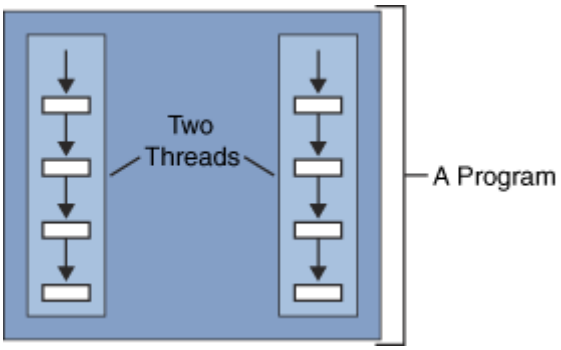


Fonte: (UFCG)

Cada retângulo branco representa uma tarefa. A *thread* fica responsável por definir a sequência e tempo de execução de cada tarefa para otimizar o gerenciamento.

Na Figura 5 nota-se dois conjuntos distintos de tarefas sendo gerenciados por duas *threads.*

Figura 5 - Dois conjuntos de tarefas gerenciados por duas threads.



Fonte: (UFCG)

Ao passo que cada *thread* possui a autonomia de gerenciar e otimizar a sequência de processos, o *software* recebe um aspecto mais funcional e dinâmico.

1.4 Objetivos

Este trabalho tem por objetivo desenvolver um programa em *C#* capaz de monitorar e representar um processo totalmente automatizado de abastecimento e evacuação de escórias de um alto-forno em combustão. Para tanto, uma interface gráfica com um alto-forno, duas estradas que se cruzam e carros de cores aleatórias serão desenvolvidas e utilizando *threads* o sistema deverá evitar colisão entre os carros e deverá manter o forno sempre em combustão com abastecimento constante.

Por fim, para tratar do tópico de comunicação entre sistemas, todo o projeto terá interação para controle remoto a partir de um servidor *Http* de forma que o usuário possa ter um relatório em tempo real da saída e possa ativar/desativar o processo.

Dito isso, os objetivos específicos deste trabalho são:

- Desenvolver um *software* interativo e funcional.
- Simular o processo de abastecimento e descarte de escórias.
- Trabalhar com técnicas de paralelismo e programação com *threads.*
- Desenvolver um *website* para que o processo possa ser acompanhado remotamente.
- Trazer um *software* com baixa demanda de recursos computacionais e que possa ser utilizado em qualquer dispositivo conectado à internet.

2 Materiais e Métodos

2.1 Materiais

Para o desenvolvimento deste projeto será utilizado o *software Microsoft Visual Studio* como compilador e ambiente de programação em C# e o navegador *Google Chrome* para todos os testes e aplicações *Web*.

Todos os códigos foram desenvolvidos utilizando o Sistema Operacional *Windows 10 Home* em um *desktop* que conta com um processador Intel® Core™ i3 – 9100F de 3.6Ghz e 16GB de memória *RAM*.

Para os testes de conexão e controle remoto foi utilizado um *Notebook* Lenovo G480 contendo um processador Intel® Core™ i5 – 5200U de 2.7Ghz e 6GB de memória *RAM* com o propósito de efetuar testes em um equipamento mais antigo e com menos capacidade computacional.

2.2 Metodologia

O passo inicial foi desenvolver via *Visual Studio* a interface gráfica contendo uma estrada com cruzamento interligando a entrada de reabastecimento de insumos de minério de ferro e a saída de escória do AF, além do próprio forno simulando seu nível que decai à medida que o tempo passa e a combustão ocorre. Para esta etapa serão utilizadas as bibliotecas *drawning* e *drawning 2D* do C#. Tais bibliotecas contém as funções de desenho os quais foram implementadas para compor a simulação.

Foi escolhido um formulário que utiliza o .NET *framework 4.5* para o desenvolvimento do projeto por se tratar de uma versão que possui todas as funções necessárias, entretanto, versões posteriores poderiam ter sido utilizadas sem perda de informações e funcionalidades.

Um método nomeado “desenhista” foi criado e recebe como parâmetros a posição atual de cada *thread* relativa a um carro, uma cor advinda de um arranjo de cores aleatórias e a posição de uma *thread* relativa por sua vez a um carro de escória. A Figura

6 mostra a rotina de desenho de cada carrinho e a maneira como ela é atualizada a cada segundo.

Figura 6 - Rotina de desenho de carros.

```
for (int q = 0; q < qntcarrinhos; q++)

{

    // Selecionar cor
    Brush pincel = new SolidBrush(c[q]);

    desenhador2.FillRectangle(pincel, new Rectangle((int)(pixel[q]), 95, 5, 5));

}

for (int h = 0; h < 500; h++)
{
    // Selecionar cor
    Brush pincel = new SolidBrush(c[h]);

    if(posicaoesc[h]>=0 && posicaoesc[h]<=225)
    desenhador2.FillRectangle(pincel, new Rectangle(500-(int)(posicaoesc[h]), 160, 5, 5));

    if(posicaoesc[h] > 225 && posicaoesc[h] <= 355)
    desenhador2.FillRectangle(pincel, new Rectangle(275, (160 - (int)(posicaoesc[h]) + 225), 5, 5));

    if (posicaoesc[h] > 355 && posicaoesc[h] <= 630)
    desenhador2.FillRectangle(pincel, new Rectangle(630-(int)(posicaoesc[h]), 30, 5, 5));

}

//Apresenta a imagem final
return imgback2;
```

Fonte: Autor.

É importante ressaltar que os valores de entrada inicialmente inseridos são baseados nas coordenadas em relação ao próprio desenho de base da estrada. Uma vez que existem curvas no trajeto foram necessárias variações ora no eixo x, ora no eixo y. A rotina inicia verificando em um *loop* se um novo carrinho deve ser adicionado (via botão ou de forma automática como será explicado a seguir). Em caso positivo, é criado um pincel com a cor aleatória recebida o qual vai preencher um quadrado enquanto ele estiver no alcance da estrada (de 0 a 500 no eixo x). Por fim é retornada a imagem formada e atualizada a cada variação de um valor "*tick*".

Uma variável sentinela nomeada "*tick"* será responsável por atualizar a cada segundo passado seu valor somando uma unidade representativa de tempo e atualizar em tempo real os relógios, posições e imprimir estes fatores em tela. Todas as posições dos carrinhos serão atualizadas e salvas em um arranjo contendo suas coordenadas x e y.

Para controle e operação foram adicionados 5 botões com as seguintes funcionalidades:

1. **Start**: Botão responsável pelo início da simulação.

2. **Stop:** Botão responsável por encerrar a simulação.

3. **Novo Carrinho**: Botão que efetua a liberação de um representativo de carrinho contendo 10 toneladas de insumos.

4. **Salvar Relatório:** Botão responsável por salvar na memória do dispositivo (computador, *smartphone* e afins) um relatório contendo a quantidade de carrinhos liberados e seus respectivos horários de saída e entrega.

5. **Manual/Automático:** Botão que ativa e desativa o modo de funcionamento automático do *software*.

Os botões *Start* e *Stop* estão indicados no Apêndice A, entretanto, iremos destacar as rotinas de cada um dos outros botões. A Figura 7 relata a lógica utilizada no botão 3.

Figura 7 - Rotina do botão de Novo Carrinho.

```
if (!manual)
{
    if (resultado <= 31)
    {
        //Adicionar um carrinho
        if (i != qntcarrinhos)
        {

            i++;

        }

        resultado = num_seg - novo_carrinho;


        if (NC == 0)
        {

            timer_esteira.Start();

        }

        countcar++;
        NC++;
        numcarrinhos.Text = ("Quantidade de carrinhos: " + NC + " carrinhos.");

        //Cria uma thread no array na posição i
        threadsArray[i] = new Thread(k1);

        //inicia a thread i
        threadsArray[i].Start(i);

        //Adicionar uma unidade à variável i

    }
}
```

Fonte: Autor.

Inicialmente verifica-se se o estado automático está ativado, caso contrário, é possível a liberação manual de carros. Verifica-se qualquer alteração na quantidade de carros via variável sentinela e atualiza-se o resultado final com um novo carro e reservando uma *thread* ao mesmo.

Já a Figura 8 mostra a lógica implementada para salvar o relatório atual de funcionamento no diretório do próprio programa.

Figura 8 - Rotina do botão Salvar Relatório.

```
private void Salvar_Click(object sender, EventArgs e)
{
    //Rotina para salvar arquivos do Listbox

    const string sPath = "Relatorio.txt";

    System.IO.StreamWriter SaveFile = new System.IO.StreamWriter(sPath);
    foreach (var item in listBox1.Items)
    {
        SaveFile.WriteLine(item);
    }

    SaveFile.Close();

    MessageBox.Show("Relatório salvo!");
}
```

Fonte: Autor.

De posse de uma variável com a lista dos comandos efetuados foi utilizado o "*StreamWriter"* diretamente da biblioteca "*System IO"* o qual efetuará a criação de um arquivo de texto contendo o relatório operacional e exibirá em tela uma mensagem dizendo "Relatório salvo!" para confirmar que o mesmo foi salvo com sucesso.

Finalmente o botão que alterna entre os estados manual e automático está explicitado na Figura 9.

Figura 9 - Rotina do botão Manual/Automático.

```
private void button3_Click(object sender, EventArgs e)
{
    if (manual == true)
    {
        manual = false;
        timer10seg.Start();
        timer8seg.Start();
    }
    else
    {
        manual = true;
        timer10seg.Stop();
        timer8seg.Stop();
    }
}
```

Fonte: Autor.

Verifica-se o estado atual da variável de monitoramento chamada manual e em caso de estar desativada, seu estado se tornará ativado, sendo válida a sentença inversa. Temporizadores serão ativados passados 10 segundos e 8 segundos de operação para manter o fluxo constante de carros.

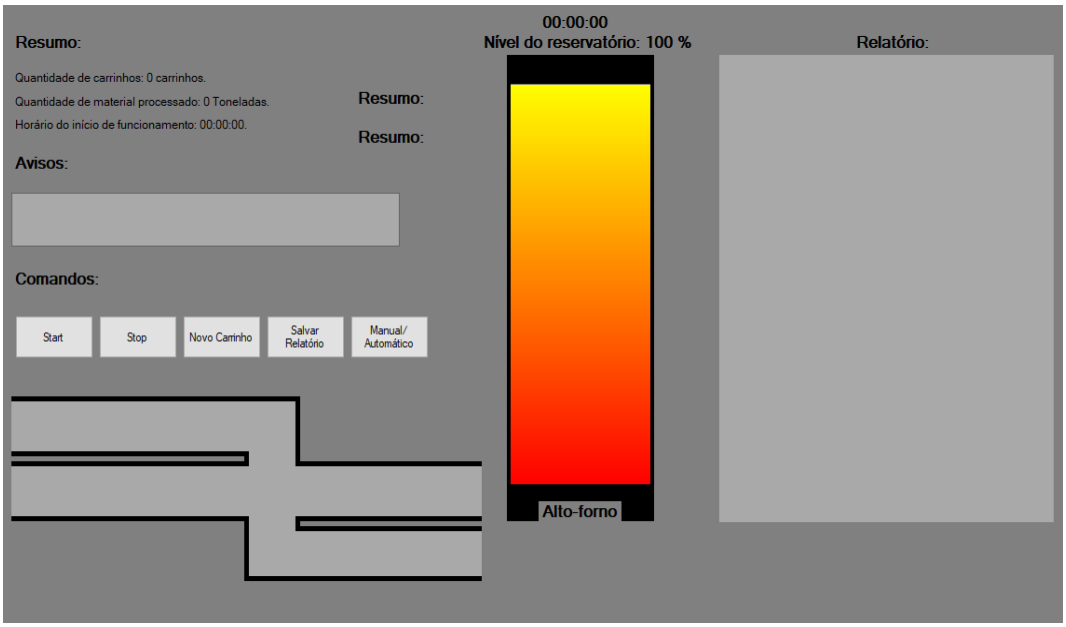
O operador tem a sua disposição um resumo com detalhes sobre a operação a se citar a quantidade de carrinhos liberados, quantidade de material processado e horário de início do funcionamento, além de um quadro de avisos indicando a situação atual do reservatório. Para valores de insumos acima de 67 toneladas será exibido um aviso:

“Nível do reservatório dentro do desejável.”. Níveis entre 33 e 66 toneladas receberão o aviso: "Nível do reservatório abaixo do esperado, adicione mais material.".

Para valores abaixo de 33 toneladas o aviso será crítico com a mensagem: "Nível do reservatório muito abaixo do esperado, adicione mais material urgentemente.".

A saída gráfica do *software* obtida foi representada pela Figura 10 a seguir:

Figura 10 - Saída do Visual Studio.



Fonte: Autor.

De posse da interface gráfica, a lógica condicional e implementação das *threads* foi iniciada. Utilizando as bibliotecas “*Threading”* e “*Threading.tasks”* um arranjo de 999 *threads* será responsável por cada carro de insumo. Os carros desenhados pelas funções utilizando *drawning 2D* supracitado serão representados por quadrados e cada um possuirá uma cor aleatória para distinção.

Cada unidade que atingir o final de seu curso em direção ao AF irá somar ao seu reservatório 10 toneladas de insumos. A posição atual de cada carrinho será dada em valores cartesianos, sendo esta afirmativa válida tanto para os insumos quanto as escórias a serem descartadas. A cada 10 toneladas de insumo utilizados na combustão, um carro de escória será liberado.

Mediante estas informações a região onde ocorre o cruzamento dos carros será considerada uma região de exclusão mútua na qual apenas uma unidade pode percorrer para que não ocorram colisões. Utilizando a função "*mutex"* cada carro irá conferir se existe alguma *thread* operando na área de exclusão e em caso positivo seu movimento deverá cessar, caso contrário, continuará sua rota normalmente. Tal rotina está explicitada na Figura 11.

Figura 11 - Rotina de controle de posições na zona de exclusão.

```
public void k1(object data)
{
    //Recebe a hora de inicio
    H_start[(int)data] = DateTime.Now;


    for (int h = 0; h < 500; h++)
    {

        //Controla posição
        pixel[(int)data]++;

        if (h == 250)
        {
            m.WaitOne();

        }

        if (h == 300)
        {
            m.ReleaseMutex();
            Thread.Sleep(100);

        }
```

Fonte: Autor.

No início obtém-se a hora atual e durante todo alcance da estrada irá variar a posição do carro, tendo que aguardar antes da posição central caso exista algum carro presente na mesma. Ao atingir valores entre 250 e 300 o *mutex* é ativado impedindo o acesso que apenas será liberado quando a posição do carro chegar a valores acima de 300.

De posse deste sistema, o passo final é definir as coordenadas limite para o controle de velocidade de cada carrinho, onde serão definidas referências de valores que representam alto e baixo nível de necessidade de insumos no forno. A partir de quantidades abaixo de 20 toneladas deve-se estabelecer velocidade máxima possível nos

trilhos para abastecimento. Em casos acima de 80%, mínima prioridade de velocidade e consequentemente desaceleração.

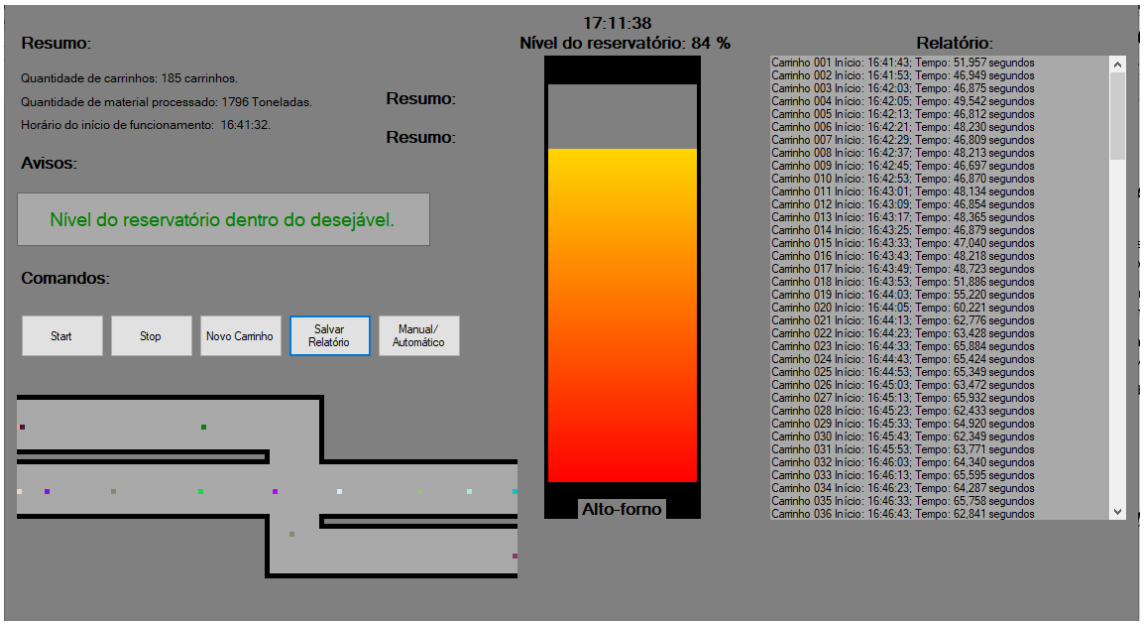
Com este método de controle já estabelecido inicia-se a etapa de desenvolvimento em Http de um website que se conecte ao servidor do programa como cliente e exponha os dados em tela. Para todas as etapas finais, desde a implementação do controle de velocidade e conexão com servidor, além do Desktop utilizado para programação base, será necessário um dispositivo com conexão a internet.

3 *Resultados e Discussão*

Nesta seção serão apresentadas todas as saídas finais, relatórios e interface do *website* desenvolvidos. Todos os códigos utilizados estarão disponíveis nos apêndices A e B ao final do texto.

Inicialmente na Figura 12 a seguir será apresentada a interface do programa após decorridos 30 minutos de operação automática.

Figura 12 - Saída do programa após 30 minutos de operação.



Fonte: Autor.

Nota-se que o AF manteve-se operando em regiões próximas de 80% de insumos e um nível dentro do desejável, indicando bom funcionamento da simulação.

Na sequência foi utilizado o botão de salvar relatório e um arquivo "*txt"* foi criado no diretório do *software*.

No controle automático apenas ao atingir valores abaixo de 78% o primeiro carrinho será liberado. Este valor se encontra na região em que deverá ocorrer a elevação de velocidade dos carrinhos e, portanto, é esperado um tempo menor para que seja percorrido todo o trajeto. Isso é comprovado com uma análise breve dos tempos indicados no relatório, os quais estão numa média de 45,35 segundos nos primeiros 18 carros. À

medida que a situação começa a se estabilizar os tempos decorridos sobem para uma média de 63,55 segundos, indicando sucesso no controle da velocidade.

Por fim a saída do código *html* está exposto na Figura 13 a seguir:

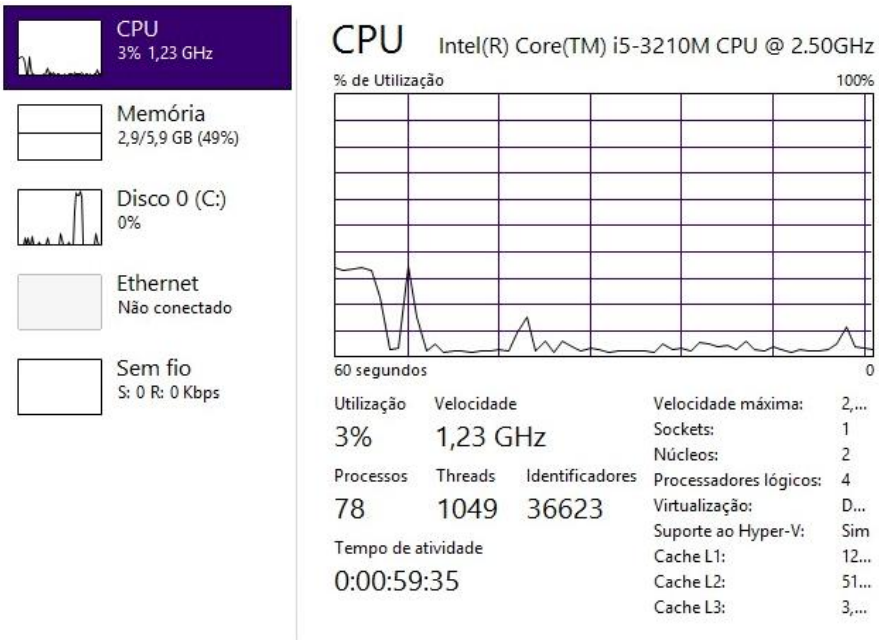
Figura 13 - Servidor *Http* operando.

Fonte: Autor.

Ao efetuar a conexão o operador tem acesso às informações básicas sobre o nível do AF e o tempo de operação decorrido, além de botões capazes de parar e iniciar o processo de maneira remota.

Em razão de explicitar o baixo uso de recursos computacionais as Figuras 14 e 15 trazem capturas do estado de uso de CPU e memória *RAM* de ambos os dispositivos de teste, tendo apenas o *Visual Studio* e o controlador em funcionamento.

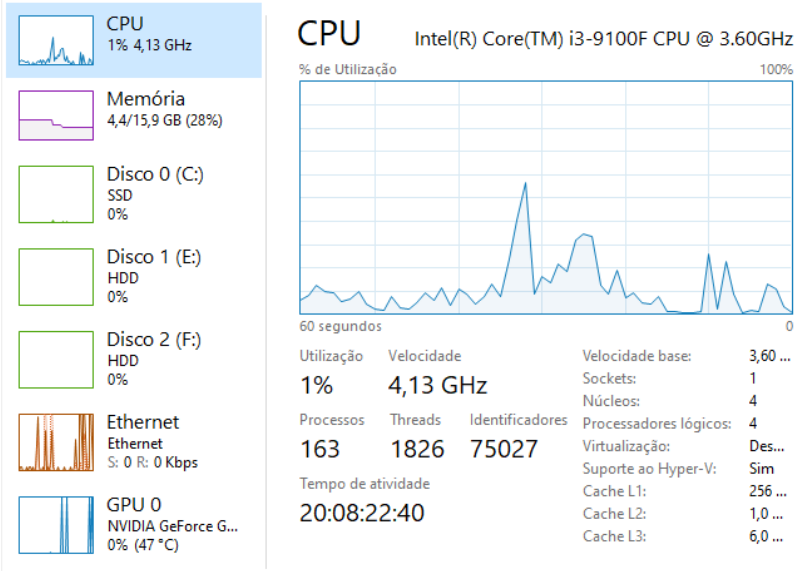
Figura 14 - Teste computacional em Notebook.



Fonte: Autor.

Nota-se um uso de 3% do processador de 1,23 GHz além de cerca de 49% dos 5,9 GB de memória *RAM* disponível no dispositivo.

Figura 15 – Teste computacional em Desktop



Fonte: Autor.

Percebe-se um uso em torno de 1% relativo a um processador de 3.6 GHz e 28% de uma memória *RAM* de 15,9 GB disponível.

4 Conclusões

Este texto tratou do desenvolvimento de um supervisório de reabastecimento de alto-forno, abordando uma simulação simples e sem custos utilizando a linguagem C#, além da utilização de um servidor *Http* para controle remoto do programa.

O *Microsoft Visual Studio* se mostrou uma ferramenta muito prática e eficaz na etapa de desenvolvimento, uma vez que seus *frameworks* trouxeram uma vasta combinação de possibilidades gráficas para exibição em tela e inúmeros recursos de programação que facilitaram o processo de simulação.

É imperativo ressalvar que com o decorrer do tempo a quantidade de insumos se manteve em faixas satisfatórias mesmo tendo início em quantidades menores presentes no AF, atendendo bem ao objetivo de um controle automático independente.

Não obstante, é notório o fato de que o simulador não demanda muitos recursos computacionais, indicando enorme facilidade de utilização mesmo em computadores menos robustos, sendo assim um ótimo ponto de partida para o desenvolvimento de um *software* controlador real e eficaz a níveis industriais, ficando assim como sugestão ao leitor.

O servidor *Http* se mostrou uma ferramenta muito útil para a comunicação remota com o *software* e abrange diversas possíveis melhorias a se citar maior exposição de dados em tela, novos botões e recursos a fim de tornar cada vez mais prática a utilização do mesmo e o acesso ao *software* base seja mínima e feita somente quando necessário.

Por fim este trabalho foi conseguiu cumprir com seus objetivos, trazendo um controlador completamente automático e efetivo para o reabastecimento de Alto-fornos, com baixíssima demanda de recursos computacionais e uma possibilidade de controle remoto via *WEB*.

5 Referências Bibliográficas

ARAÚJO, L. **Manual de Siderurgia**. São Paulo: Editora São Paulo, v. 1, 1997.

COSIAÇO. Cosiaço. **Cosiaço**, 21 nov. 2016. Disponivel em: <https://www.cosiaco.com.br/blog/como-de-aco-e-feito-um-breve-resumo-do-alto-forno/>. Acesso em: 25 maio 2021.

DEITAL, H. M. **C# - Como Programar**. São Paulo: Pearson Education, 2003.

DOMINGOS, M. DEVMEDIA. **DEVMEDIA**, 2011. Disponivel em: <https://www.devmedia.com.br/programacao-paralela/21405>. Acesso em: 25 maio 2021.

INFOMET. **Aços & Ligas | Aço: Processos de Fabricação | Processo Siderúrgico**. **Infomet**, 1998. Disponivel em: <https://www.infomet.com.br/site/acos-e-ligas-conteudo-ler.php?codConteudo=234#:~:text=O%20alto%20forno%20%C3%A9%20um,com%20C%20no%20estado%20l%C3%ADquido.>. Acesso em: 25 maio 2021.

LIMA, W. **Um breve histórico da automação industrial e redes para automação industrial**. [S.l.]. 2003.

MACHADO, F. B.; MAIA, L. P. **Arquitetura de Sistemas Operacionais**. Rio de Janeiro: LTC, v. 3.ed., 2002. p.83-94 p.

MEDEIROS, H. DEVMEDIA. **DEVMEDIA**, 2007. Disponivel em: <https://www.devmedia.com.br/programacao-com-threads/6152>. Acesso em: 25 maio 2021.

PAZOS, F. **Automação de sistemas & robótica:** Fernando Pazos. Rio de Janeiro: Axcel Books, 2002.

RIZZO, E. M. D. S. **Processo de fabricação de ferro gusa em alto-forno**. São Paulo: Associação Brasileira de Metalurgia, v. Materiais e, 2009.

UFCG. **O que é um Thread?**. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/threads/threads1.html>. Acesso em: 25 jul. 2021.

Apêndice A – Código base da aplicação

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace Trabalho___ELT_402
{
    public partial class Form1 : Form
    {
        static Mutex m = new Mutex(false, "Sample_Mutex");

        public Form1()
        {
            InitializeComponent();

            //Responsável por criar cores aleatorias
            c = new Color[qntcarrinhos];
            Random r = new Random();

            for (int a=0; a < qntcarrinhos; a++)
            {

                c[a] = Color.FromArgb(r.Next(0, 255), r.Next(0, 255), r.Next(0, 255));

//listBox1.Items.Add(c[a]);
            }
        }

//Inicializadores
        int resultado;

        //Sentinela para carrinho de escoria
        bool sent_escoria = false;

        //Hora início
        DateTime[] H_start = new DateTime[999];

        //Hora fim
        DateTime[] H_end = new DateTime[999];

        //Diferença de hora
        DateTime[] H_dif = new DateTime[999];

        //Criar cor aleatoria
        Color[] c;

        //Desenhar carro

        Graphics[] desenhadorcar = new Graphics[999];

        //zeros para relatório
        int zerosesquerda = 3;

        //Mudar qntcarrinho/thread/pixel pra alterar o numero de carrinho
        int qntcarrinhos = 999;

        //Criar arrays para thread
        int i = -1;
        Thread[] threadsArray = new Thread[999];

        //Criar array thread para escoria
        int n_esc = -1;
        Thread[] threadesc = new Thread[500];

        //movimentação do carrinho
        float[] pixel = new float[999];

        //movimentação carrinho de escoria
        float[] posicaoesc = new float[500];

        //Conta segundos
        int num_seg = 0;

        //Contador de carrinhos
        int novo_carrinho = 0;


        //Contador para controlar carrinhos thread
        int countcar = 0;

        //Contador para numero de vezes apertado o botão NC
        int NC = 0;

        //Sentinela botão start
        bool startpress = false;
```

```
        //Sentinela para modo
manual/automático
        bool manual = true;

      //Inciar desenhos originais
private void Form1_Load(object
sender, EventArgs e)
        {

//Parte Relacionada ao Alto-Forno

//criar a folha em branco
            Bitmap imgback = new
Bitmap(PicB_Altoforno.Width,
PicB_Altoforno.Height);
            Graphics desenhador =
Graphics.FromImage(imgback);

            //Atribuir um cor de
fundo

desenhador.Clear(Color.Gray);

            //Criar lápis
            Pen lapis = new
Pen(Color.Black, 4);

            //Preenchimento
            Brush pincel = new
SolidBrush(Color.Black);
            Brush pincel1 = new
LinearGradientBrush(new Point(30,
430),

new Point(30, 29),

Color.Red,

Color.Yellow

);



            //Plotagem

            //Desenhar altoforno

desenhador.DrawRectangle(lapis, new
Rectangle(8, 27, 154, 405));

//Preencher altoforno

desenhador.FillRectangle(pincel1, new
Rectangle(10, 30, 150, 400));

//Detalhes sup e inf

desenhador.FillRectangle(pincel, new
Rectangle(6, 0, 158, 25));

desenhador.FillRectangle(pincel, new
Rectangle(6, 430, 158, 50));

//Apresenta a imagem final do alto-
forno

PicB_Altoforno.BackgroundImage =
imgback;

//Desenhar trilho/carrinho

            Bitmap imgback2 = new
Bitmap(PicB_Trilho.Width,
PicB_Trilho.Height);
            Graphics desenhador2 =
Graphics.FromImage(imgback2);
            //Criar pincel

            Brush pincel11 = new
SolidBrush(Color.Black);
            Brush pincel21 = new
SolidBrush(Color.DarkGray);
            //Desenhar Estrada

            //Estrada 1
            //Primeira
metade(contornos)

desenhador2.FillRectangle(pincel11,
0, 65, 255, 5);

desenhador2.FillRectangle(pincel11,
0, 120, 255, 5);

            //meio(estrada)

desenhador2.FillRectangle(pincel21,
0, 70, 505, 50);

            //Segunda
metade(contornos)

desenhador2.FillRectangle(pincel11,
305, 65, 200, 5);

desenhador2.FillRectangle(pincel11,
305, 120, 200, 5);



            //Estrada 2
            //Primeira
metade(contornos)

desenhador2.FillRectangle(pincel11,
0, 0, 310, 5);

desenhador2.FillRectangle(pincel11,
0, 55, 255, 5);

desenhador2.FillRectangle(pincel11,
305, 0, 5, 65);
```

```
desenhador2.FillRectangle(pincel11,
250, 60, 5, 5);

//meio(estrada)

desenhador2.FillRectangle(pincel21,
0, 5, 305, 50);

desenhador2.FillRectangle(pincel21,
250, 135, 255, 50);

desenhador2.FillRectangle(pincel21,
255, 55, 50, 15);

desenhador2.FillRectangle(pincel21,
255, 120, 50, 15);

//Segunda metade(contornos)

desenhador2.FillRectangle(pincel11,
0, 0, 310, 5);

desenhador2.FillRectangle(pincel11,
0, 55, 255, 5);

desenhador2.FillRectangle(pincel11,
250, 125, 5, 55);

desenhador2.FillRectangle(pincel11,
305, 125, 5, 5);

desenhador2.FillRectangle(pincel11,
305, 130, 200, 5);

desenhador2.FillRectangle(pincel11,
250, 180, 255, 5);



PicB_Trilho.BackgroundImage =
imgback2; ;

        }

//Função para desenhar o carrinho

        public Bitmap
desenhista(float[] pixel, Color[] c,
float[] posicaoesc)
        {

            Bitmap imgback2 = new
Bitmap(PicB_Trilho.Width,
PicB_Trilho.Height);
            Graphics desenhador2 =
Graphics.FromImage(imgback2);



//Atribuir um cor de fundo

desenhador2.Clear(Color.Gray);
```

```
//Criar pincel
            Brush pincel11 = new
SolidBrush(Color.Black);
            Brush pincel21 = new
SolidBrush(Color.DarkGray);


//Desenhar Estrada

//Estrada 1
//Primeira metade(contornos)

desenhador2.FillRectangle(pincel11,
0, 65, 255, 5);

desenhador2.FillRectangle(pincel11,
0, 120, 255, 5);

//meio(estrada)

desenhador2.FillRectangle(pincel21,
0, 70, 505, 50);

//Segunda metade(contornos)

desenhador2.FillRectangle(pincel11,
305, 65, 200, 5);

desenhador2.FillRectangle(pincel11,
305, 120, 200, 5);


//Estrada 2
//Primeira metade(contornos)

desenhador2.FillRectangle(pincel11,
0, 0, 310, 5);

desenhador2.FillRectangle(pincel11,
0, 55, 255, 5);

desenhador2.FillRectangle(pincel11,
305, 0, 5, 65);

desenhador2.FillRectangle(pincel11,
250, 60, 5, 5);

 //meio(estrada)

desenhador2.FillRectangle(pincel21,
0, 5, 305, 50);

desenhador2.FillRectangle(pincel21,
250, 135, 255, 50);

desenhador2.FillRectangle(pincel21,
255, 55, 50, 15);

desenhador2.FillRectangle(pincel21,
255, 120, 50, 15);
```

```
//Segunda metade(contornos)

desenhador2.FillRectangle(pincel11,
0, 0, 310, 5);

desenhador2.FillRectangle(pincel11,
0, 55, 255, 5);

desenhador2.FillRectangle(pincel11,
250, 125, 5, 55);

desenhador2.FillRectangle(pincel11,
305, 125, 5, 5);

desenhador2.FillRectangle(pincel11,
305, 130, 200, 5);

desenhador2.FillRectangle(pincel11,
250, 180, 255, 5);



            for (int q = 0; q <
qntcarrinhos; q++)

            {

 // Selecionar cor
                Brush pincel = new
SolidBrush(c[q]);


desenhador2.FillRectangle(pincel, new
Rectangle((int)(pixel[q]), 95, 5,
5));

            }

            for (int h = 0; h < 500;
h++)
            {
// Selecionar cor
                Brush pincel = new
SolidBrush(c[h]);

                if(posicaoesc[h]>=0
&& posicaoesc[h]<=225)

desenhador2.FillRectangle(pincel, new
Rectangle(500-(int)(posicaoesc[h]),
160, 5, 5));

                if(posicaoesc[h] >
225 && posicaoesc[h] <= 355)

desenhador2.FillRectangle(pincel, new
Rectangle(275, (160 -
(int)(posicaoesc[h]) + 225), 5, 5));

                if (posicaoesc[h] >
355 && posicaoesc[h] <= 630)

desenhador2.FillRectangle(pincel, new
Rectangle(630-(int)(posicaoesc[h]),
30, 5, 5));

            }

            //Apresenta a imagem
final
            return imgback2;

        }

public void k1(object data)
        {
            //Recebe a hora de inicio
            H_start[(int)data] =
DateTime.Now;


            for (int h = 0; h < 500;
h++)
            {

                //Controla posição
                pixel[(int)data]++;

                if (h == 250)
                {
                    m.WaitOne();

                }

                if (h == 300)
                {
                    m.ReleaseMutex();

Thread.Sleep(100);

                }

                if (h == 500-1)
                {
//Adiciona material ao alto-forno
                    novo_carrinho +=
10;

                    //Recebe a hora
final
                    H_end[(int)data]
= DateTime.Now;


                    if
(listBox1.InvokeRequired)

listBox1.BeginInvoke((MethodInvoker)
                                delegate
{


//Calcula tempo total
```

```
TimeSpan horaTotal = new
TimeSpan(H_end[(int)data].Ticks -
H_start[(int)data].Ticks);


                            //Escreve
o relatório no listbox

listBox1.Items.Add("Carrinho " +
((int)data+1).ToString("D"+zerosesque
rda.ToString()) + " Início: " +
H_start[(int)data].ToString("T") + ";
Tempo: "+
horaTotal.TotalSeconds.ToString("F" +
zerosesquerda.ToString()) + "
segundos" );


                            });

                }

//Controla a velocidade do carrinho
                if(resultado>=20)
                {
                    Thread.Sleep(90);
                }
                else
                {

Thread.Sleep(190);
                }

                //Thread.Sleep(10);

            }



        }

        public void escoria(object
data)
        {
            for (int h = 0; h < 630;
h++)
            {

posicaoesc[(int)data]++;
                Thread.Sleep(90);
                //Thread.Sleep(10);

                if (h == 265)
                {
                    m.WaitOne();

                }

                if (h == 315)
                {
                    m.ReleaseMutex();

Thread.Sleep(100);

                }
            }
        }

private void pictureBox1_Click(object
sender, EventArgs e)
        {

        }

//Timer para o relógio
        private void
timer1_Tick(object sender, EventArgs
e)
        {
            //Enviar hora atual para
o relogio
            Relogio.Text =
DateTime.Now.ToString("T");

        }


//Parte responsável pelo alto-forno
        private void
timer_operação_Tick(object sender,
EventArgs e)
        {

//criar a folha em branco
            Bitmap imgback = new
Bitmap(PicB_Altoforno.Width,
PicB_Altoforno.Height);
Graphics desenhador =
Graphics.FromImage(imgback);

//Atribuir um cor de fundo

desenhador.Clear(Color.Gray);

//Criar lápis
Pen lapis = new Pen(Color.Black, 4);

//Preenchimento
Brush pincel = new
SolidBrush(Color.Black);
            Brush pincel1 = new
LinearGradientBrush(new Point(30,
430),

new Point(30, 29),

Color.Red,

Color.Yellow

);


//Plotagem
```

```
//Desenhar altoforno

desenhador.DrawRectangle(lapis, new
Rectangle(8, 27, 154, 405));

//Preencher altoforno

desenhador.FillRectangle(pincel1, new
Rectangle(10, (30+4*resultado), 150,
(400-4*resultado)));

//Detalhes sup e inf

desenhador.FillRectangle(pincel, new
Rectangle(6, 0, 158, 25));

desenhador.FillRectangle(pincel, new
Rectangle(6, 430, 158, 50));

//Apresenta a imagem final

PicB_Altoforno.BackgroundImage =
imgback;

//Texto indicador de volume no
reservatório

            nv_reserv.BackColor =
Color.Transparent;
            nv_reserv.Text = "Nível
do reservatório: "+(100-resultado) +
" %";


//Mostrar quantidade de material
processado
            Qnt_processada.Text =
"Quantidade de material processado: "
+ num_seg + " Toneladas.";


// Avisos para o operador
            if (resultado <= 33)
            {
                Avisos.ForeColor =
System.Drawing.Color.Green;
                Avisos.Text = "Nível
do reservatório dentro do
desejável.";
            }
            else if (resultado > 33
&& resultado <= 66)
            {
                Avisos.ForeColor =
System.Drawing.Color.Yellow;
                Avisos.Text = "Nível
do reservatório abaixo do esperado,
adicione mais material.";
            }
            else
            {
                Avisos.ForeColor =
System.Drawing.Color.Red;
                Avisos.Text = "Nível
do reservatório muito abaixo do
esperado, adicione mais material
urgentemente.";
            }

            if (resultado < 100)
            {
//Contador de tempo
                num_seg++;
            }
        }

//Botão Start
        private void
button1_Click(object sender,
EventArgs e)
        {
            int resultado = num_seg -
novo_carrinho;
            if (resultado >= 100)
            {
                if
(MessageBox.Show("Reservatóio vazio,
adicione mais material para ligar o
sistema.",

"Erro",

MessageBoxButtons.OK,

MessageBoxIcon.Error) ==
DialogResult.OK)
                {

}
}

else
{

timer_operação.Start();

timer_relogio.Start();
                inic_func.Text =
"Horário do início de funcionamento:
" + DateTime.Now.ToString("T") + ".";

timer_esteira.Start();

start_residuo.Start();
startpress = true;
}
}

//Botão Novo carrinho
private void btn_NC_Click(object
sender, EventArgs e)
{
if (manual)
 {
```

```
//Adicionar um carrinho
if (i != qntcarrinhos)
                {

                    i++;

                }


                    if (NC == 0)
                    {


timer_esteira.Start();

                    }

                    countcar++;
                    NC++;
                    numcarrinhos.Text
= ("Quantidade de carrinhos: " + NC +
" carrinhos.");

//Cria uma thread no array na posição
i
                    threadsArray[i] =
new Thread(k1);

//inicia a thread i

threadsArray[i].Start(i);

//Adicionar uma unidade à variável i


                }
           // }

            else
            {
                if
(MessageBox.Show("Modo automático
habilitado",

"Erro",

MessageBoxButtons.OK,

MessageBoxIcon.Error) ==
DialogResult.OK)
                {

                }
            }



        }

        private void
label1_Click(object sender, EventArgs
e)
        {

        }

        private void
nv_reserv_Click(object sender,
EventArgs e)
        {

        }

        private void
nv_reserv_Click_1(object sender,
EventArgs e)
        {

        }

        private void
label1_Click_1(object sender,
EventArgs e)
        {

        }

        private void
button2_Click(object sender,
EventArgs e)
        {
            timer_operação.Stop();
//timer_esteira.Stop();

        }

        private void
label2_Click(object sender, EventArgs
e)
        {

        }

//Timer que controla a esteira
        private void
timer_esteira_Tick(object sender,
EventArgs e)
        {


PicB_Trilho.BackgroundImage =
desenhista(pixel, c,posicaoesc);
//Rotina para salvar arquivos do
Listbox

            const string sPath =
"Relatorio.txt";

            System.IO.StreamWriter
SaveFile = new
System.IO.StreamWriter(sPath);
            foreach (var item in
listBox1.Items)
            {
```

```
SaveFile.WriteLine(item);
            }

            SaveFile.Close();


 // Atualiza a vaíável "resultado"
mais rápido, fiz essa mudança para
evitar problemas com os carrinhos de
residuo
            resultado = num_seg -
novo_carrinho;

        }
        private void
timer1_Tick_1(object sender,
EventArgs e)
        {

        }

        private void
timer2_Tick(object sender, EventArgs
e)
        {

        }

        private void
label4_Click(object sender, EventArgs
e)
        {

        }

        private void
label4_Click_1(object sender,
EventArgs e)
        {

        }

        private void
Salvar_Click(object sender, EventArgs
e)
        {
//Rotina para salvar arquivos do
Listbox

            const string sPath =
"Relatorio.txt";

            System.IO.StreamWriter
SaveFile = new
System.IO.StreamWriter(sPath);
            foreach (var item in
listBox1.Items)
            {

SaveFile.WriteLine(item);
            }

            SaveFile.Close();


MessageBox.Show("Relatório salvo!");
        }

        private void
button3_Click(object sender,
EventArgs e)
        {
            if (manual == true)
            {
                manual = false;
                timer10seg.Start();
                timer8seg.Start();
            }
            else
            {
                manual = true;
                timer10seg.Stop();
                timer8seg.Stop();
            }
        }

        private void
timer1seg_Tick(object sender,
EventArgs e)
        {
            if (!manual)
            {
                if (resultado <= 31)
                {
//Adicionar um carrinho
                    if (i !=
qntcarrinhos)
                    {

                        i++;

                    }

                    resultado =
num_seg - novo_carrinho;


                    if (NC == 0)
                    {


timer_esteira.Start();

                    }

                    countcar++;
                    NC++;
                    numcarrinhos.Text
= ("Quantidade de carrinhos: " + NC +
" carrinhos.");

                    //Cria uma thread
no array na posição i
```

```
                    threadsArray[i] =
new Thread(k1);

//inicia a thread i

threadsArray[i].Start(i);

 //Adicionar uma unidade à variável i

                }
            }
        }

private void timer2seg_Tick(object
sender, EventArgs e)
        {
            if (!manual)
            {
                if (resultado > 31)
                {
//Adicionar um carrinho
                    if (i !=
qntcarrinhos)
                    {

                        i++;

                    }



                    if (NC == 0)
                    {


timer_esteira.Start();

                    }

                    countcar++;
                    NC++;
                    numcarrinhos.Text
= ("Quantidade de carrinhos: " + NC +
" carrinhos.");

                    //Cria uma thread
no array na posição i
                    threadsArray[i] =
new Thread(k1);

                    //inicia a thread
i

threadsArray[i].Start(i);

                    //Adicionar uma
unidade à variável i


                }
            }
        }

        private void
start_residuo_Tick(object sender,
EventArgs e)
        {
            if (num_seg % 20 == 0 &&
num_seg != 0 )
            {
                if (resultado == 100)
                {
                    if (sent_escoria
== false)
                    {
                        n_esc++;

//Cria uma thread no array na posição
i

threadesc[n_esc] = new
Thread(escoria);

//inicia a thread i

threadesc[n_esc].Start(n_esc);
sent_escoria = true;
                    }

                }

                else
                {
                    n_esc++;
//Cria uma thread no array na posição
i
                    threadesc[n_esc]
= new Thread(escoria);

//inicia a thread i

threadesc[n_esc].Start(n_esc);

                    sent_escoria =
false;

                }
            }
        }
    }
}
```

Apêndice B – Código do servidor HTTP

```
using System;
using System.IO;
using System.Text;
using System.Net;
using System.Threading.Tasks;


namespace HttpServer
{
    class HttpServer
    {
        public static HttpListener
listener;
        public static string url =
"http://localhost:8000/";
        public static int pageViews =
0;
        public static int
requestCount = 0;
        public static string pageData
=
            "<!DOCTYPE>" +
            "<html>" +
            "<html lang =\"pt-br\">"+
            "  <head>" +
            "    <title>Supervisorio
Alto-forno</title>" +
            "  </head>" +
            "  <body>" +
            "    <h1>Controlador
Remoto</h1>" +
            "       <hr>" +
            "    <p>Quantidade de
insumos: 100</p>" +
            "    <p>Tempo decorrido:
01:53 horas</p>" +
            "    <form
method=\"post\" action=\"start\">" +
            "      <input
type=\"submit\" value=\"Start\" {1}>"
+
            "    </form>" +
            "    <form
method=\"post\" action=\"shutdown\">"
+
            "      <input
type=\"submit\" value=\"Shutdown\"
{1}>" +
            "    </form>" +
            "  </body>" +
            "</html>";


        public static async Task
HandleIncomingConnections()
        {
            bool runServer = true;

            // While a user hasn't
visited the `shutdown` url, keep on
handling requests
            while (runServer)
            {
                // Will wait here
until we hear from a connection
                HttpListenerContext
ctx = await
listener.GetContextAsync();

                // Peel out the
requests and response objects
                HttpListenerRequest
req = ctx.Request;
                HttpListenerResponse
resp = ctx.Response;

                // Print out some
info about the request

Console.WriteLine("Request #: {0}",
++requestCount);

Console.WriteLine(req.Url.ToString())
;

Console.WriteLine(req.HttpMethod);

Console.WriteLine(req.UserHostName);

Console.WriteLine(req.UserAgent);
                Console.WriteLine();

                // If `shutdown` url
requested w/ POST, then shutdown the
server after serving the page
                if ((req.HttpMethod
== "POST") && (req.Url.AbsolutePath
== "/shutdown"))
                {

Console.WriteLine("Shutdown
requested");
                    runServer =
false;
                }

                // Make sure we don't
increment the page views counter if
`favicon.ico` is requested
```

```
                if
(req.Url.AbsolutePath !=
"/favicon.ico")
                    pageViews += 1;

                // Write the response
info
                string disableSubmit
= !runServer ? "disabled" : "";
                byte[] data =
Encoding.UTF8.GetBytes(String.Format(
pageData, pageViews, disableSubmit));
                resp.ContentType =
"text/html";
                resp.ContentEncoding
= Encoding.UTF8;
                resp.ContentLength64
= data.LongLength;

                // Write out to the
response stream (asynchronously),
then close it
                await
resp.OutputStream.WriteAsync(data, 0,
data.Length);
                resp.Close();
            }
        }
```

```
        public static void
Main(string[] args)
        {
            // Create a Http server
and start listening for incoming
connections
            listener = new
HttpListener();

listener.Prefixes.Add(url);
            listener.Start();

Console.WriteLine("Listening for
connections on {0}", url);

            // Handle requests
            Task listenTask =
HandleIncomingConnections();

listenTask.GetAwaiter().GetResult();

            // Close the listener
            listener.Close();
        }
    }
}
```